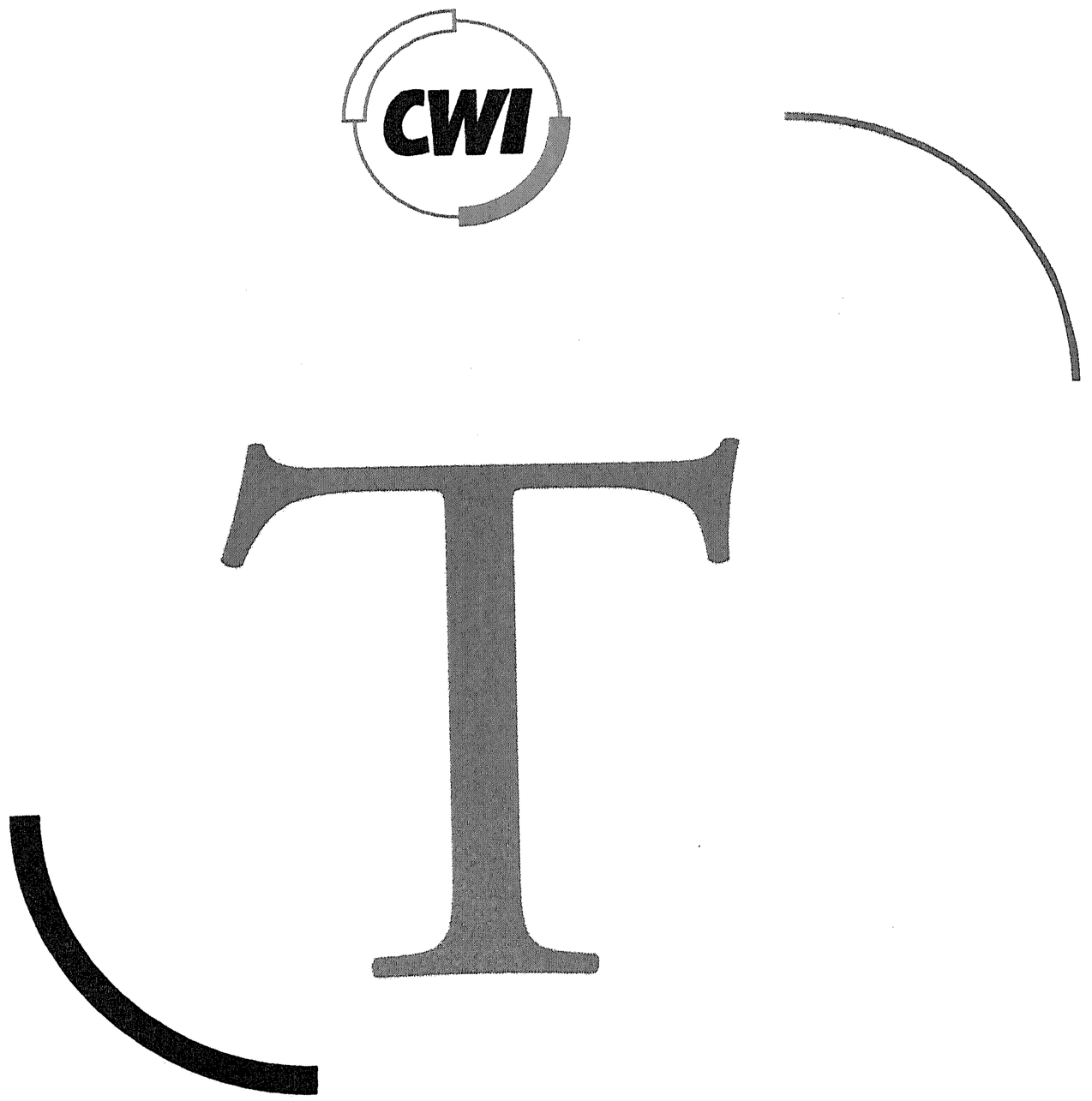


Selected publications of Eugene L. Lawler

edited by
K. Aardal
J.K. Lenstra
F. Maffioli
D.B. Shmoys

126

Centrum voor Wiskunde en Informatica
CWI TRACT

CWI Tracts

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Telephone + 31 - 20 592 9333
Telefax + 31 - 20 592 4199
WWW page http://www.cwi.nl

CWI is the nationally funded Dutch institute for research in Mathematics and Computer Science.

Selected publications of Eugene L. Lawler

edited by

K. Aardal
J.K. Lenstra
F. Maffioli
D.B. Shmoys

AMS Subject Classification: 01A75 (selected works), 90C27 (combinatorial optimization).

ISBN 90 6196 484 9
NUGI-code: 811

Preface

Eugene L. Lawler died on September 2, 1994, aged 61, after an eight-month battle with cancer. He is survived by his wife Marijke, son Stephen, and daughter, son-in-law and granddaughter, Susan, Matthew and Janna Rose Surprise. He is also dearly missed by his students, colleagues, and friends.

Gene obtained an A.M. at Harvard University in 1957, was a Senior Electrical Engineer at Sylvania Electric Products in Needham, Massachusetts, from 1959 until 1961, and went back to Harvard to obtain a Ph.D. in 1962. He taught at the University of Michigan in Ann Arbor from 1962 until 1970 and at the University of California at Berkeley since 1971. He combined an illustrious career of highly influential research with a history of dedicated service to both universities. Throughout his career, Gene was an active member of the theoretical computer science community. He served on the editorial boards of the *SIAM Journal on Applied Mathematics* (1968–1972) and the *SIAM Journal on Computing* (1972–1980). In 1992, he was elected a Fellow of the American Association for the Advancement of Science. He was posthumously awarded the A. Nico Habermann Award from the Computer Research Association, "in recognition of his outstanding contributions to the advancement of underrepresented groups in computing research."

For more than thirty years, Gene Lawler studied algorithmic issues in combinatorial optimization. His work has been fundamental in giving the discipline the breadth and depth it has now attained. Of all of his publications, his textbook *Combinatorial Optimization: Networks and Matroids* (1976) has had the most pronounced impact. It brought the most important results in the area together, and is notable for its lucid writing style. It gave new clarity to well-understood and less understood results, brought the reader to the forefront of the field, and made the challenges of the future both apparent and accessible. It is one of the classics of the area, and is as useful today as the day it was written. The book *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (1985), which he edited with three younger colleagues, also became a benchmark reference.

It is hard to separate Gene's role as an expositor and as a researcher. His great gift in investigating a computational approach to a problem was his ability to extract the essential difficulty, achieve a deeper insight, and then solve a more general problem in a simpler way. To some extent, his expository talent came from the relative difficulty he had in absorbing new ideas. In order for him to understand other people's work, especially when it was written in a complicated way, he often had to wrestle with it in order to arrive at a better understanding and a simplification of the result.

Gene's papers on branch-and-bound (1966, with D.E. Wood) and dynamic programming (1969, with J.M. Moore) are classics; the former, in fact, was selected as a citation classic in 1987. Both papers, rather than introducing radically new techniques, brought a new level of usefulness and understanding

to important algorithmic paradigms. His paper on matroid intersection (1975) had a similar influence. Since the mid-1970's, Gene was particularly interested in sequencing and scheduling. Prior to his work, the area was a rather unmathematical hodgepodge, with little systematic understanding of the types of methods and techniques that could be used most effectively. Gene stimulated and unified the area greatly. His main unfinished project is the completion of a graduate textbook on scheduling. He made significant contributions to a wide variety of issues in combinatorial optimization, which were influential in both operations research and computer science. Most recently, he had turned his attention to combinatorial problems in computational biology, an area whose importance he was among the first to recognize.

As this summary of Gene's research might suggest, he was a phenomenal educator. He could provide the intuition that made difficult results easily accessible. His appreciation of the difficulties in absorbing ideas to the point that one can go beyond them in some original way helped to make him a great advisor. He was constantly available for every new idea and always ready to interest his students in whatever he was currently thinking about.

Gene had an enormous influence on the atmosphere of the Computer Science Division at Berkeley. He never lost sight of the mission of a university and never backed away from difficult tasks. Gene was the social conscience of the division. He helped the individual student fight the bureaucracy, reformed what the university taught and to whom it taught, and made the university a more humane and more stimulating place to study. Upon his retirement in 1994, he was awarded the Berkeley Citation, the campus's highest accolade.

Gene Lawler was a remarkable man, who was ready to discuss intelligently nearly any current issue and did so in a thought-provoking way. We all miss him very much.

* * *

This volume is intended as both a scientific and a personal memorial. It contains a complete list of his publications, and a selection of twenty-six technical and expository papers. We have tried to give a representative picture of the areas Gene was interested in and of the kind of work he did, keeping in mind its impact at the time as well as its relevance today. The majority of the papers included are original research contributions. We have also selected a fair amount of his writings for a more general audience, such as "The great mathematical Sputnik of 1979" and a speech on the implications of technology for society that he gave at the 1994 UC Berkeley Computer Science commencement ceremony.

Gene's writing was imbued with a style that was distinctively his own. As we read through his papers, we often felt that we could hear him explaining the particular result at hand, with his typical enthusiasm. We hope that everyone who had the privilege to know him and to learn from him will feel similarly, when going through these pages.

We mention two related publications. The paper "The mystical power of twoness: in memoriam Eugene L. Lawler" (*Journal of Scheduling 1*, pp. 3–14 (1998)) is a review of his work, with an emphasis of his research in scheduling.

A double issue of *Mathematical Programming* (*B82*, pp. 1–290 (1998)) contains fourteen papers dedicated to his memory.

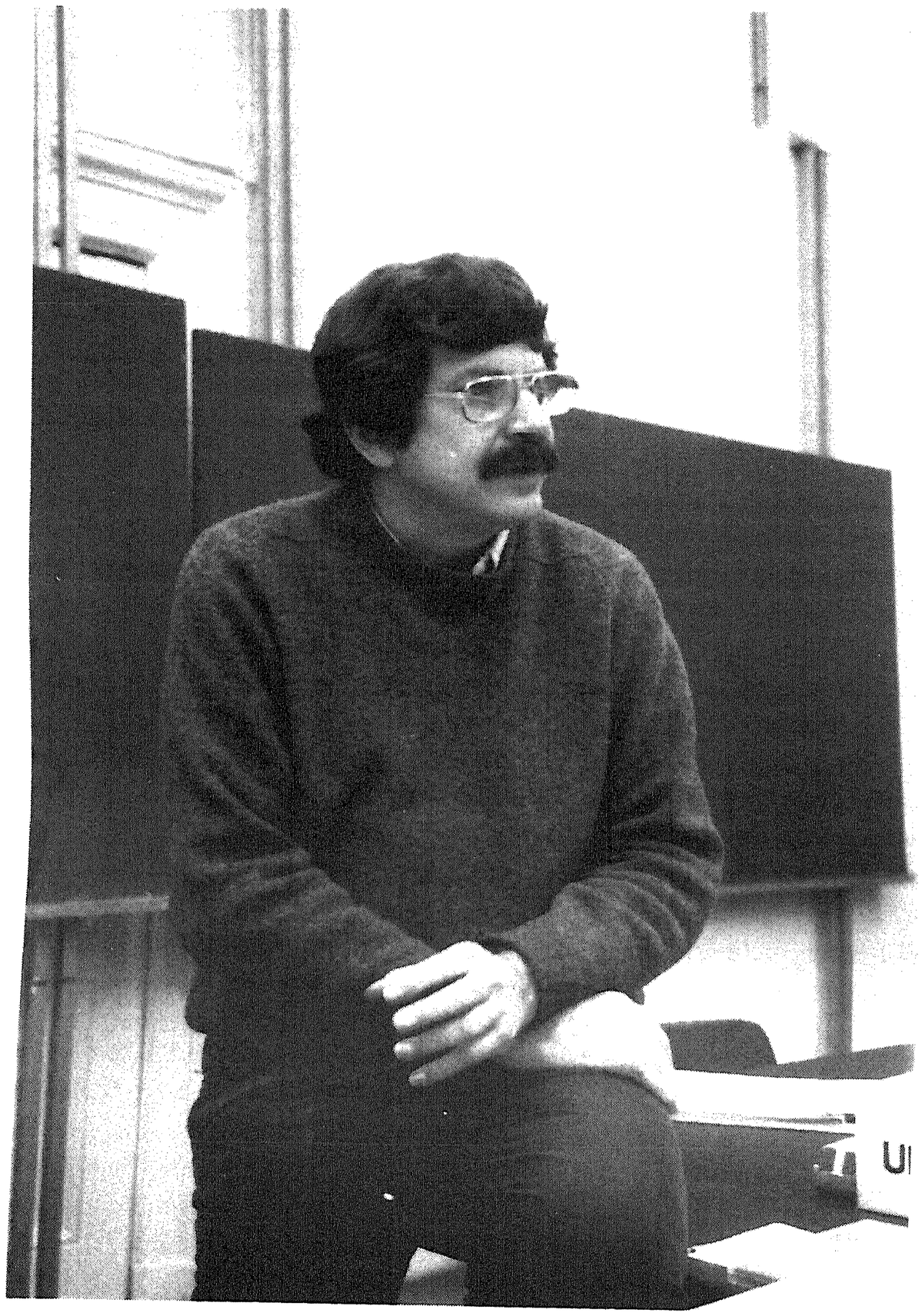
We are grateful to several people for their help in producing the present volume. Ann DiFruscia helped us to sort out Gene's publications, in the same caring way that she supported him as secretary over the years. Arjen Vestjens checked the bibliographic details of some of the earlier papers, Milan Vlach provided the details about the translation of the Sputnik paper into Czech, and Yu Wenci printed the title of Gene's paper in Chinese. Gene's picture on page x was taken by Peter van Emde Boas on December 8, 1981, at the University of Amsterdam. We thank Gene's co-authors and the copyright owners of the selected papers for their permission to reprint them. Our editorial work was supported by the National Science Foundation through Grants IRI-9120074, CCR-9700029, DMS-9505155 and CCR-9120008, by the Office for Naval Research through Grant N00014-96-1-00500, and by NATO through Collaborative Research Grant 971550. Finally, we acknowledge the support of CWI for making this publication possible.

Karen Aardal
Jan Karel Lenstra
Francesco Maffioli
David Shmoys

January 1999

Table of contents

The codes between brackets refer to the list of publications on pages [1]–[10].



Eugene L. Lawler

1933-1994

Publications of Eugene L. Lawler

A Books

A1 E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, NY (1976).

A2 E.L. Lawler. *Kombinatorikus Optimalizálás: Hálózatok és Matroidok.* Műszaki Könyvkiadó, Budapest (1982). [Translation of A1 into Hungarian, by A. Frank.]

A3 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (editors). *The Traveling Salesman Problem; A Guided Tour of Combinatorial Optimization.* Wiley, Chichester (1985).

B Papers in journals

B1 E.L. Lawler, G.A. Salton. The use of parenthesis-free notation for the automatic design of switching circuits. *IRE Transactions on Electronic Computers EC-9* (1960), 342-352.

B2 E.L. Lawler. Minimization of switching circuits subject to reliability conditions. *IRE Transactions on Electronic Computers EC-10* (1961), 781-782.

B3 E.L. Lawler. Electrical assemblies with a minimum number of interconnections. *IRE Transactions on Electronic Computers EC-11* (1962), 86-88.

B4 E.L. Lawler. The quadratic assignment problem. *Management Science 9* (1963), 586-599.

B5 E.L. Lawler. A comment on minimum feedback arc sets. *IEEE Transactions on Circuit Theory CT-11* (1964), 296-297.

B6 E.L. Lawler. An approach to multilevel Boolean minimization. *Journal of the Association for Computing Machinery 11* (1964), 283-295. [Revised version of C1.]

B7 E.L. Lawler. On scheduling problems with deferral costs. *Management Science 11* (1964), 280-288.

B8 E.L. Lawler, D.E. Wood. Branch-and-bound methods: a survey. *Operations Research 14* (1966), 699-719.

B9 E.L. Lawler. Covering problems: duality relations and a new method of solution. *SIAM Journal on Applied Mathematics 14* (1966), 1115-1132.

B10 E.L. Lawler, M.D. Bell. A method for solving discrete optimization problems. *Operations Research 11* (1966), 1098-1112.

B11 E.L. Lawler, K.N. Levitt, J. Turner. Module clustering to minimize delay in digital networks. *IEEE Transactions on Computers C-18* (1969), 47-57.

B12 E.L. Lawler, J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science 16* (1969), 77-84.

B13 E.L. Lawler. A solvable case of the traveling salesman problem. *Mathematical Programming 1* (1971), 267-269.

B14 E.L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science 18* (1972), 401-405.

B15 E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science 19* (1973), 544-546.

B16 E.L. Lawler. A matroid generalization of a theorem of Mendelsohn and Dulmage. *Discrete Mathematics 4* (1973), 159-163.

B17 E.L. Lawler. Cutsets and partitions of hypergraphs. *Networks 3* (1973), 275-285.

B18 E.L. Lawler. Matroid intersection algorithms. *Mathematical Programming 9* (1975), 31-56. [Full version of results announced in C9.]

B19 E.L. Lawler. Algorithms, graphs, and complexity. *Networks 5* (1975), 89-92.

B20 E.L. Lawler. Sequencing to minimize the weighted number of tardy jobs. *Revue Française d'Automatique, Informatique, Recherche Opérationnelle 10.5 Supplément* (1976), 27-33.

B21 E.L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters 5* (1976), 66-67.

B22 E.L. Lawler. Comment on computing the k shortest paths in a graph. *Communications of the ACM 20* (1977), 603-604.

B23 E.L. Lawler, B.D. Sivazlian. Minimization of time-varying costs in single-machine scheduling. *Operations Research 26* (1978), 563-569.

B24 E.L. Lawler, J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery 25* (1978), 612-619.

B25 E.L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research 4* (1979), 339-356. [Revised version of C24.]

B26 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing 9* (1980), 558-565.

B27 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Minimizing maximum lateness in a two-machine open shop. *Mathematics of Operations Research 6* (1981), 153-158. Erratum. *Mathematics of Operations Research 7* (1982), 635.

B28 E.L. Lawler, C.U. Martel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters 12* (1981), 9-12.

B29 J. Valdes, R.E. Tarjan, E.L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing 11* (1982), 298-313. [Revised version of C29.]

B30 E.L. Lawler, C.U. Martel. Computing maximal "polymatroidal" network flows. *Mathematics of Operations Research 7* (1982), 334-347.

B31 E.L. Lawler, M.G. Luby, V.V. Vazirani. Scheduling open shops with parallel machines. *Operations Research Letters 1* (1982), 161-164.

B32 B.J. Lageweg, J.K. Lenstra, E.L. Lawler, A.H.G. Rinnooy Kan. Computer-aided complexity classification of combinatorial problems. *Communications of the ACM 25* (1982), 817-822.

B33 E.L. Lawler. A fully polynomial approximation scheme for the total tardiness problem. *Operations Research Letters 1* (1982), 207-208.

B34 K.R. Baker, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research 31* (1983), 381-386.

B35 P. Tong, E.L. Lawler. A faster algorithm for finding edge-disjoint branchings. *Information Processing Letters 17* (1983), 73-76.

B36 B.J. Lageweg, J.K. Lenstra, E.L. Lawler, A.H.G. Rinnooy Kan. 计算机辅助组合问题的复杂性分类 [Jisuanji fuzhu zuhe wenti de fuzaxing fenlei]. *Chinese Journal of Operations Research 4* (1985), 73-80, 32, 21. [Translation of B32 into Chinese.]

B37 E.L. Lawler, C.U. Martel. Polymatroidal flows with lower bounds. *Discrete Applied Mathematics 15* (1986), 291-313.

B38 M.W. Bern, E.L. Lawler, A.L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms 8* (1987), 216-235. [Revised version of C42.]

B39 W.I. Chang, E.L. Lawler. Edge coloring of hypergraphs and a conjecture of Erdős, Faber, Lovász. *Combinatorica 8* (1988), 293-295.

B40 E.L. Lawler, C.U. Martel. Preemptive scheduling of two uniform machines to minimize the number of late jobs. *Operations Research 37* (1989), 314-318.

B41 T. Gonzalez, E.L. Lawler, S. Sahni. Optimal preemptive scheduling of two unrelated processors. *ORSA Journal on Computing 2* (1990), 219-224.

B42 E.L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the "tower of sets" property. *Mathematical and Computer Modelling 20* (1994), 91-106.

B43 W.I. Chang, E.L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica 12* (1994), 327-344. [Revised version of C46.]

B44 E.L. Lawler, S. Sarkissian. An algorithm for "Ulam's game" and its application to error correcting codes. *Information Processing Letters 56* (1995), 89-93. [Full version of C54.]

B45 S.K. Kannan, E.L. Lawler, T.J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms 21* (1996), 26-50. [Revised version of C45.]

B46 L. Wang, T. Jiang, E.L. Lawler. Approximation algorithms for tree alignment with a given phylogeny. *Algorithmica 16* (1996), 302-315. [Revised version of C52.]

B47 T.A. Varvarigou, V.P. Roychowdhury, T. Kailath, E.L. Lawler. Scheduling in and out forests in the presence of communication delays. *IEEE Transactions on Parallel and Distributed Systems 7* (1996), 1065-1074.

B48 V. Bafna, E.L. Lawler, P.A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science 182* (1997), 233-244. [Revised version of C53.]

C Papers in books and proceedings

C1 E.L. Lawler. Minimal Boolean expressions with more than two levels of sums and products. *Switching Circuit Theory and Logical Design; Proceedings of the Third Annual Symposium.* AIEE, New York, NY (1962), 49-59.

C2 R.F. Arnold, E.L. Lawler. On the analysis of functional symmetry. *Switching Circuit Theory and Logical Design; Proceedings of the Fourth Annual Symposium.* IEEE, New York, NY (1963), 53-62.

C3 E.L. Lawler. The minimal synthesis of tree structures. *Switching Circuit Theory and Logical Design; Proceedings of the Fourth Annual Symposium.* IEEE, New York, NY (1963), 63-82.

C4 E.L. Lawler. An analysis of Roth's methods of synthesis. *Seventh Midwest Symposium on Circuit Theory; Summary of Papers.* College of Engineering, University of Michigan, Ann Arbor, MI (1964), 223-233.

C5 E.L. Lawler. Combinatorial aspects of variable-length encoding. *The Summaries of Papers of International Conference on Microwaves, Circuit Theory and Information Theory; Part 3, Information Theory.* The Institute of Electrical Communication Engineers of Japan, Tokyo (1964), 27-28. [Condensed version of D2.]

C6 R. Gonzalez, E.L. Lawler. Two-level threshold minimization. *1965 IEEE Conference Record on Switching Circuit Theory and Logical Design; Papers Presented at the Sixth Annual Symposium.* IEEE, New York, NY (1965), 41-44.

C7 E.L. Lawler. Optimal cycles in doubly weighted directed linear graphs. In: P. Rosenstiehl (editor). *Theory of Graphs, International Symposium; Théorie des Graphes, Journées Internationales d'Etude.* Gordon and Breach, New York, NY, and Dunod, Paris (1967), 209-213.

C8 E.L. Lawler, T.F. Piatkowski. Generalized state identification problems. *Conference Record of 1967 Eighth Annual Symposium on Switching and Automata Theory.* IEEE, New York, NY (1967), 252-254.

C9 E.L. Lawler. Optimal matroid intersections. In: R.K. Guy, H. Hanani, N. Sauer, J. Schonheim (editors). *Combinatorial Structures and Their Applications.* Gordon and Breach, New York, NY (1970), 233-234.

C10 E.L. Lawler. The complexity of combinatorial computations: a survey. In: J. Fox (editor). *Proceedings of the Symposium on Computers and Automata; Microwave Research Institute Symposia Series XXI.* Polytechnic Press of the Polytechnic Institute of Brooklyn, New York, NY (1971), 305-311.

C11 E.L. Lawler. Optimal cycles in graphs and the minimal cost-to-time ratio problem. In: A. Marzollo (editor). *Periodic Optimization; CISM Courses and Lectures No. 135, Volume I.* Springer, Vienna (1972), 37-60.

C12 E.L. Lawler. Polynomial-bounded and (apparently) non-polynomial-bounded matroid computations. In: R. Rustin (editor). *Combinatorial Algorithms; Courant Computer Science Symposium 9.* Algorithmics Press, New York, NY (1973), 49-57.

C13 E.L. Lawler. Discussion of "On the set representation and the set covering problem." In: S.E. Elmaghraby (editor). *Symposium on the Theory of Scheduling and Its Applications; Lecture Notes in Economics and Mathematical Systems 86.* Springer, Berlin (1973), 164-166.

C14 E.L. Lawler. The quadratic assignment problem: a brief review. In: B. Roy (editor). *Combinatorial Programming: Methods and Applications.* Reidel, Dordrecht (1975), 351-360.

C15 E.L. Lawler. Computing shortest paths in networks. In: S. Rinaldi (editor). *Topics in Combinatorial Optimization; CISM Courses and Lectures No. 175.* Springer, Vienna (1975), 1-5.

C16 E.L. Lawler. Complexity of combinatorial computations. In: S. Rinaldi (editor). *Topics in Combinatorial Optimization; CISM Courses and Lectures No. 175.* Springer, Vienna (1975), 87-95.

C17 E.L. Lawler. Overview of network flow theory. In: S. Rinaldi (editor). *Topics in Combinatorial Optimization; CISM Courses and Lectures No. 175.* Springer, Vienna (1975), 97-107.

C18 E.L. Lawler. Some aspects of duality on combinatorial optimization. In: S. Rinaldi (editor). *Topics in Combinatorial Optimization; CISM Courses and Lectures No. 175.* Springer, Vienna (1975), 109-116.

C19 E.L. Lawler. An introduction to matroid optimization. In: S. Rinaldi (editor). *Topics in Combinatorial Optimization; CISM Courses and Lectures No. 175.* Springer, Vienna (1975), 181-186.

C20 E.L. Lawler. Graphical algorithms and their complexity. In: K.R. Apt, J.W. de Bakker (editors). *Foundations of Computer Science II, Part 1; Mathematical Centre Tracts 81.* Mathematisch Centrum, Amsterdam (1976), 3-32.

C21 E.L. Lawler. Introduction to the complexity of algorithms. In: R.T. Yeh (editor). *Applied Computation Theory: Analysis, Design, Modeling.* Prentice-Hall, Englewood Cliffs, NJ (1976), Chapter 2, 61-81.

C22 E.L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In: P.L. Hammer, E.L. Johnson, B.H. Korte, G.L. Nemhauser (editors). *Studies in Integer Programming; Annals of Discrete Mathematics 1.* North-Holland, Amsterdam (1977), 331-342.

C23 E.L. Lawler. Fast approximation algorithms for knapsack problems. *18th Annual IEEE Symposium on Foundations of Computer Science.* IEEE, New York, NY (1977), 206-213.

C24 E.L. Lawler. Fast approximation algorithms for knapsack problems. In: J.K. Lenstra, A.H.G. Rinnooy Kan, P. van Emde Boas (editors). *Interfaces between Computer Science and Operations Research; Mathematical Centre Tracts 99.* Mathematisch Centrum, Amsterdam (1978), 109-139. [Revised and extended version of C23.]

C25 R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: J.K. Lenstra, A.H.G. Rinnooy Kan, P. van Emde Boas (editors). *Interfaces between Computer Science and Operations Research; Mathematical Centre Tracts 99.* Mathematisch Centrum, Amsterdam (1978), 169-214.

C26 E.L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In: B. Alspach, P. Hell, D.J. Miller (editors). *Algorithmic Aspects of Combinatorics; Annals of Discrete Mathematics 2.* North-Holland, Amsterdam (1978), 75-90.

C27 E.L. Lawler. Shortest path and network flow algorithms. In: P.L. Hammer, E.L. Johnson, B.H. Korte (editors). *Discrete Optimization I; Annals of Discrete Mathematics 4.* North-Holland, Amsterdam (1979), 251-263.

C28 R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: P.L. Hammer, E.L. Johnson, B.H. Korte (editors). *Discrete Optimization II; Annals of Discrete Mathematics 5.* North-Holland, Amsterdam (1979), 287-326. [Revised version of C25.]

C29 J. Valdes, R.E. Tarjan, E.L. Lawler. The recognition of series parallel digraphs. *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing.* ACM, New York, NY (1979), 1-12.

C30 E.L. Lawler. An introduction to polymatroidal network flows. In: G. Ausiello, M. Lucertini (editors). *Analysis and Design of Algorithms in Combinatorial Optimization; CISM Courses and Lectures No. 266.* Springer, Vienna (1981), 129-145.

C31 E.L. Lawler, C.U. Martel. Flow network formulations of polymatroid optimization problems. In: A. Bachem, M. Grötschel, B. Korte (editors). *Bonn Workshop on Optimization; Annals of Discrete Mathematics 16.* North-Holland, Amsterdam (1982), 189-200.

C32 E.L. Lawler, J.K. Lenstra. Machine scheduling with precedence constraints. In: I. Rival (editor). *Ordered Sets.* Reidel, Dordrecht (1982), 655-675.

C33 B.J. Lageweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Een geautomatiseerde complexiteitsclassificatie van combinatorische problemen. In: P.M.B. Vitányi, J. van Leeuwen, P. van Emde Boas (editors). *Colloquium Complexiteit en Algoritmen, Deel 2; MC Syllabus 48.2.* Mathematisch Centrum, Amsterdam (1982), 159-171. [Translation of B32 into Dutch.]

C34 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Recent developments in deterministic sequencing and scheduling: a survey. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (editors). *Deterministic and Stochastic Scheduling.* Reidel, Dordrecht (1982), 35-73.

C35 E.L. Lawler. Preemptive scheduling of precedence-constrained jobs on parallel machines. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (editors). *Deterministic and Stochastic Scheduling.* Reidel, Dordrecht (1982), 101-123.

C36 E.L. Lawler. Recent results in the theory of machine scheduling. In: A. Bachem, M. Grötschel, B. Korte (editors). *Mathematical Programming; The State of the Art; Bonn 1982.* Springer, Berlin (1983), 202-234.

C37 E.L. Lawler, M.G. Luby, B. Parker. Finding shortest paths in very large networks. In: M. Nagle, J. Perl (editors). *Proceedings of the WG'83; International Workshop on Graphtheoretic Concepts in Computer Science.* Trauner, Linz (1983), 184-199.

C38 J. Labetoulle, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In: W.R. Pulleyblank (editor). *Progress in Combinatorial Optimization.* Academic Press, New York, NY (1984), 245-261.

C39 P. Tong, E.L. Lawler, V.V. Vazirani. Solving the weighted parity problem for gammoids by reduction to graphic matching. In: W.R. Pulleyblank (editor). *Progress in Combinatorial Optimization.* Academic Press, New York, NY (1984), 363-374.

C40 E.L. Lawler, P.J. Slater. A linear time algorithm for finding an optimal dominating subforest of a tree. In: Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, C. E. Wall (editors). *Graph Theory with Applications to Algorithms and Computer Science.* Wiley, New York, NY (1985), 501-506.

C41 E.L. Lawler. Submodular functions and polymatroid optimization. In: M. O'hEigeartaigh, J.K. Lenstra, A.H.G. Rinnooy Kan (editors). *Combinatorial Optimization: Annotated Bibliographies.* Wiley, Chichester (1985), 32-38.

C42 M.W. Bern, E.L. Lawler, A.L. Wong. Why certain subgraph computations require only linear time. *26th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Washington, DC (1985), 117-125.

C43 P.C. Gilmore, E.L. Lawler, D.B. Shmoys. Well-solved special cases. In: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (editors). *The Traveling Salesman Problem; A Guided Tour of Combinatorial Optimization.* Wiley, Chichester (1985), Chapter 4, 87-143.

C44 E.L. Lawler. Combinatorial structures and combinatorial optimization. In: B. Simeone (editor). *Combinatorial Optimization; Lecture Notes in Mathematics 1403.* Springer, Berlin (1989), 162-197.

C45 S.K. Kannan, E.L. Lawler, T.J. Warnow. Determining the evolutionary tree. *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms.* ACM, New York, NY, and SIAM, Philadelphia, PA (1990), 475-484.

C46 W.I. Chang, E.L. Lawler. Approximate string matching in sublinear expected time. *Proceedings 31st Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, CA (1990), 116-124.

C47 E.L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. In: M. Queyranne (editor). *Production Planning and Scheduling; Annals of Operations Research 26.* Baltzer, Basel (1990), 125-133.

C48 E.L. Lawler. Computing shortest paths in networks derived from recurrence relations. In: J. MacGregor Smith, P. Winter (editors). *Topological Network Design; Annals of Operations Research 33.* Baltzer, Basel (1991), 363-377.

C49 P. Hilfinger, E.L. Lawler, G. Rote. Flattening a rooted tree. In: P. Gritzmann, B. Sturmfels (editors). *Applied Geometry and Discrete Mathematics;*

The Victor Klee Festschrift; DIMACS Series in Discrete Mathematics and Theoretical Computer Science 4. AMS, Providence, RI, and ACM, Baltimore, MD (1991), 335-340.

C50 E.L. Lawler. On preference orders for sequencing problems or, what has Smith wrought? In: M. Akgül, H. W. Hamacher, S. Tüfekçi (editors). *Combinatorial Optimization; New Frontiers in Theory and Practice; NATO ASI Series F82.* Springer, Berlin (1992), 133-159.

C51 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In: S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin (editors). *Logistics of Production and Inventory; Handbooks in Operations Research and Management Science, Volume 4.* North-Holland, Amsterdam (1993), 445-522.

C52 T. Jiang, E.L. Lawler, L. Wang. Aligning sequences via an evolutionary tree: complexity and approximation. *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing.* ACM, New York, NY (1994), 760-769.

C53 V. Bafna, E.L. Lawler, P.A. Pevzner. Approximation algorithms for multiple sequence alignment. In: M. Crochemore, D. Gusfield (editors). *Combinatorial Pattern Matching; Lecture Notes in Computer Science 807.* Springer, Berlin (1994), 43-53.

C54 E.L. Lawler, S. Sarkissian. Adaptive error correcting codes based on cooperative play of the game of "twenty questions with a liar"; abstract. In: J.A. Storer, M. Cohn (editors). *Proceedings; DCC '95: Data Compression Conference.* IEEE Computer Society Press, Los Alamitos, CA (1995), 464.

D Reports of unpublished papers

D1 E.L. Lawler. *Some aspects of discrete mathematical programming; A thesis presented by Eugene Leighton Lawler to The Division of Engineering and Applied Physics in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the subject of Applied Mathematics, Harvard University, Cambridge, Massachsetts, August 1962.* Report No. BL-31, The Computation Laboratory, Harvard University, Cambridge, MA (1963).

D2 E.L. Lawler. Combinatorial aspects of variable-length encoding. Manuscript (1964). [Complete version of C5.]

D3 E.L. Lawler. *Preemptive scheduling of uniform parallel machines to minimize the weighted number of late jobs.* Report BW 105/79, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1979).

D4 E.L. Lawler. *Efficient implementation of dynamic programming algorithms for sequencing problems.* Report BW 106/79, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1979).

D5 B.J. Lageweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. *Computer aided complexity classification of deterministic scheduling problems.* Report BW 138/81, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1981).

D6 E.L. Lawler, O. Vornberger. *The partial order dimension problem is NP-complete.* Memorandum UCB/ERL M81/46, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA (1981).

D7 E.L. Lawler. *Generalizations of the polymatroidal network flow model.* Report BW 158/82, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1982).

D8 E.L. Lawler. *Scheduling a single machine to minimize the number of late jobs.* Report No. UCB/CSD 83/139, Computer Science Division, University of California, Berkeley, CA (1983).

D9 E.L. Lawler, J.K. Lenstra, C.U. Martel, B. Simons, L.J. Stockmeyer. *Pipeline scheduling: a survey.* Computer Science Research Report RJ 5738 (57717), IBM Research Division, San José, CA (1987).

D10 P. Horton, E.L. Lawler. *An analysis of the efficiency of the A* algorithm for multiple sequence alignment.* Manuscript (1994).

E Lighter fare

E1 E.L. Lawler. F. Weinberg (editor), Einführung in die Methode Branch and Bound, Springer, Berlin (1968); Book review. *Econometrica 40* (1972), 211-212.

E2 B.J. Lageweg, E.L. Lawler, J.K. Lenstra. *Machine scheduling problems: computations, complexity and classification; in honour of A.H.G. Rinnooy Kan upon the occasion of the defense of his doctoral thesis, January 28, 1976.* Report BN 30/76, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1976).

E3 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. An algorithm for finding this paper. In: P. van Emde Boas, J.K. Lenstra, F. Oort, A.H.G. Rinnooy Kan, T. J. Wansbeek (editors). *Een Pak met een Korte Broek; Papers Presented to H.W. Lenstra, Jr. on the Occasion of the Publication of his "Euclidische Getallenlichamen".* Amsterdam (1977), 4 unnumbered pages.

E4 E.L. Lawler. The great mathematical sputnik of 1979. *The Sciences* (September 1980), 12-15, 34-35.

E5 E.L. Lawler. The great mathematical sputnik of 1979. *The Mathematical Intelligencer 2* (1980), 190-198. [Reprinted version of E4.]

E6 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. At play in the fields of scheduling theory. *Optima 7* (1982), 1-3.

E7 E.L. Lawler. Velký matematický sputnik roku 1979. *Pokroky Matematiky, Fyziky a Astronomie 26* (1983), 38-47. [Translation of E4 into Czech, by O. Kowalski.]

E8 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys. Are you all salesman, here? *Optima 19* (1986), 1-2.

E9 E.L. Lawler. This week's citation classic. *Current Contents (Arts & Humanities; Social & Behavioral Sciences; Engineering, Technology & Applied Sciences; Physical, Chemical & Earth Sciences) 2* (January 12, 1987), 16. [The citation classic in question is B8.]

E10 E.L. Lawler. Old stories. In: J.K. Lenstra, A.H.G. Rinnooy Kan, A. Schrijver (editors). *History of Mathematical Programming; A Collection of Personal Reminiscences.* CWI, Amsterdam, and North-Holland, Amsterdam (1991), 97-106.

E11 E.L. Lawler. Algorithmic challenges of the human genome initiative. *Newsletter SIAM Activity Group on Discrete Mathematics 2.2* (1991-2), 1-3.

E12 E.L. Lawler. Combinatorics. In: S.I. Gass, C.M. Harris (editors). *Encyclopedia of Operations Research and Management Science.* Kluwer Academic, Boston, MA (1996), 83-85.

E13 E.L. Lawler. "If a thing is not worth doing, it is not worth doing right"; commencement address to the Computer Science Majors, College of Letters & Sciences, UC Berkeley, May 1994. Not published before.

The Use of Parenthesis-Free Notation for the Automatic Design of Switching Circuits*

E. L. LAWLER†, MEMBER, IRE, AND G. A. SALTON‡, MEMBER, IRE

Summary—**A parenthesis-free notation is introduced for the representation of series-parallel switching networks. The notation facilitates the calculation of circuit parameters and permits an unambiguous characterization of the circuit topology.**

Given certain criteria for feasibility of a switching network related to the circuit parameter values, it is shown how an infeasible series-parallel network can be transformed into an equivalent feasible network by "cascading" operations applied to the two-terminal subnetworks of the original network. A systematic method is developed, resulting in an optimum choice of cascading operations such that the number of switching elements required to implement the transformed circuit is minimized relative to cascading.

I. Introduction

COMPUTERS are being used increasingly in the design and development of digital equipment. In particular, a number of techniques are available for the minimization of Boolean functions using computer programs [1]–[3]. Since Boolean addition and multiplication correspond respectively to parallel and series connections of switching elements, it is always possible to transform a Boolean equation into a corresponding series-parallel switching network.

Unfortunately, circuit theoretical considerations may restrict the direct implementation of minimized functions. It then becomes necessary to revise the original circuit configurations to satisfy circuit restrictions. In general, equivalent "feasible" configurations can be substituted for configurations that do not obey circuit rules; this is usually achieved at the cost of a number of additional circuit elements.[1]

In order to program a digital computer to make these substitutions, it is necessary to use a notation that unambiguously exhibits the structure of each configuration. Such a notation is the parenthesis-free notation first introduced by Lukasiewicz [4]. A parenthesis-free notation can be used effectively to represent series-parallel switching networks, to calculate certain dimensional parameters of the network, and to isolate all the two-terminal subnetworks.

For purposes of demonstration, direct coupled transistor logic will be used in some of the examples. In particular, unless otherwise specified, it is assumed that the circuits are implemented by transistor inverter

* Received by PGEC, February 2, 1960; revised manuscript received, June 18, 1960.

† Sylvania Electric Products, Inc., Needham, Mass.
‡ Harvard University, Cambridge, Mass., and Sylvania Electric Products, Inc., Needham, Mass.

[1] It might be more elegant conceptually to incorporate rules for the satisfaction of circuit constraints in the original process of circuit synthesis, instead of generating circuits which must later be modified if infeasible.

switches (inverters) connected either in series or in parallel. The notation and method are, however, adaptable to other technologies.

II. Abstract Network Theory

A *two-terminal switching network* is defined abstractly as a finite collection of two types of entities, *elements* $a, b, c, \cdots$ and *vertices* $1, 2, 3, \cdots$ with the following properties:

1) With each element are associated two of the vertices, the combination forming a *branch*. A branch consisting of the element e and the vertices α and β will be symbolized by $e\ (\alpha, \beta)$.
2) The collection contains two special vertices called the *terminals* of the network. The two terminals will be labelled 1 and 2 respectively.

A *two-terminal subnetwork* of a network is a subnetwork that has exactly two vertices in common with the remainder of the network. The two common vertices constitute the terminals of the subnetwork.

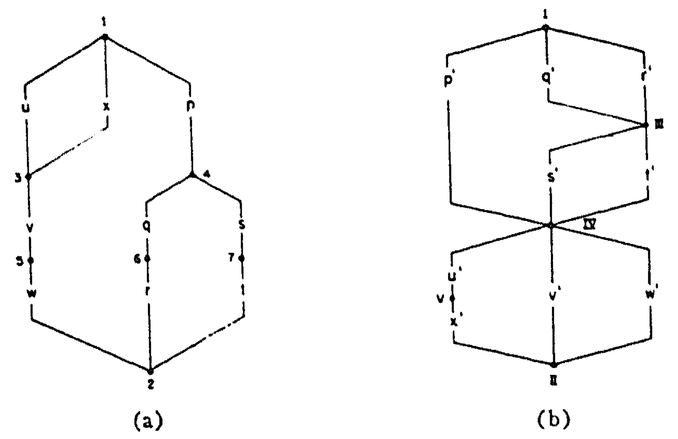
A *path* is defined as a set of branches that can be ordered in the form $e_1(\alpha, \beta), e_2(\beta, \gamma), e_3(\gamma, \delta), \cdots, e_k(\epsilon, \zeta)$; the path is said to *join* the vertices α and ζ. If none of the vertices $\beta, \gamma, \delta, \cdots, \epsilon$ appear in any other branches of the network, the elements $e_1, e_2, \cdots, e_k$ are said to be connected in *series*. A *loop* is a set of branches that can be ordered in the form $e_1(\alpha, \beta), e_2(\beta, \gamma), e_3(\gamma, \delta), \cdots, e_k(\zeta, \alpha)$, the vertices $\alpha, \beta, \gamma, \cdots, \zeta$ being distinct as in the case of the path. A set of elements, every two of which (with their associated vertices) form a loop, are said to be connected in *parallel*. A two-terminal switching network is *series-parallel*, if it is a series or parallel combination of two series-parallel networks; a single element is a two-terminal series-parallel switching network [5].

To an abstract network as defined above, there corresponds a physical network. To each of the vertices $1, 2, 3, \cdots$ there corresponds a connection point of the network, and to each of the elements $a, b, c, \cdots$ there corresponds a switch. The switches may be bilateral, in which case $e(\alpha, \beta) = e(\beta, \alpha)$, or they may be unilateral, in which case a unique direction is assumed between the vertices α and β. A special element including a potential source in series with a current detector is often associated with the two terminals of the network.

Two abstract two-terminal series-parallel networks are shown in Fig. 1(a) and 1(b). It may be noted that all series (parallel) connections in the network of Fig. 1(a) are replaced by parallel (series) connections respectively, in Fig. 1(b) and that the polarity of all switches is reversed. The networks are thus *duals* of each other.

III. The Parenthesis-Free Notation

Consider a language composed of two types of characters: operators P_i and S_i of degree i, and variables $\{a, b, c, \cdots\}$. Let the two operators represent respectively a parallel and a series connection of i two-terminal subnetworks of a network, and let the lower case letters represent the variables assigned to the switches.[2] A two-terminal subnetwork can be a single branch or a more complicated array of several branches.



(a) (b)

Fig. 1—Dual abstract two-terminal series-parallel networks.

Since a series-parallel network is defined as a series or parallel connection of switching elements, the P-S language can clearly be used to represent any series-parallel switching network. For example, $P_3(x, y, z)$ represents a parallel connection of the three elements x, y, and z, while $P_2(S_2(a, b), c)$ represents the parallel connection of $S_2(a, b)$ and c, where $S_2(a, b)$ is itself a series connection of elements a and b.

A language whose words consist of operators of degree i, each followed by i operands, has been called a "simple" language [6]. It has been shown that for such languages the usual parentheses indicating grouping are redundant. Expressions can thus be written without parentheses or commas. Moreover, algorithms can be devised for transforming ordinary Boolean expressions into the corresponding parenthesis-free expressions, and vice versa [7].

As an example, the abstract two-terminal series-parallel network shown in Fig. 1(b) is implemented with transistor inverters as shown in Fig. 2. Terminal 1 is connected to a resistor leading to a potential source and to an output terminal f; terminal 2 is connected to ground. The network of Fig. 2 generates the function $f = p(qr + st) + (u + x)vw$.[3]

The parenthesis-free expression corresponding to the inverter array of Fig. 2 is given by

$$f = S_2P_2p'S_2P_2q'r'P_2s't'P_2S_2u'x'v'w'. \qquad (1)$$

[2] Thanks are given to a referee for pointing out that a similar notation was previously suggested by G. W. Patterson in the "Sixth Quarterly Progress Report on the Theory of Switching" of the Moore School of Electrical Engineering on the Theory of Switching (ASTIA No. AD 155-004, December, 1957).

[3] The configuration of Fig. 2, corresponding to that of Fig. 1(b), generates the function $f = p(qr + st) + (u + x)vw$, since direct-coupled transistor logic is used. If more familiar relay-like logic were used, the required configuration would be that of Fig. 1(a).

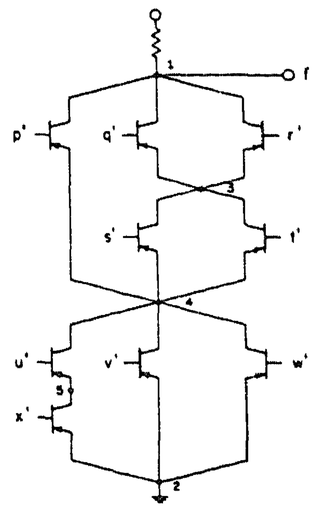


Fig. 2—Transistor inverter network for the function $p(qr+st)+(u+x)vw$.

A set of n adjacent characters starting with the left-most character in a parenthesis-free expression and proceeding to the right is called a *head* of length n. Similarly, the right-most n characters are defined as a *tail* of length n. In expression (1), $S_2P_2p'S_2$ is a head of length 4, and $u'x'v'w'$ is a tail of length 4.

A system of weights is assigned to the characters to distinguish legitimate parenthesis-free expressions from illegitimate ones and to aid in identifying the switching networks corresponding to specified expressions.[4] In particular, let the weight of the characters be specified as shown in Table I, and let the weight of an expression be the sum obtained by adding the character weights of all characters in the expression.

TABLE I
CHARACTER WEIGHTS

Character	Weight
Operators of degree i	$1-i$
Variables	1

A *well-formed formula* is then defined as an expression of weight +1 whose minimum tail weight is positive [8]. As an example, S_2xP_2yz is well-formed; S_2xyz has positive tail weight but is not well-formed since its total weight is +2; S_2xP_2y has a minimum tail weight of 0 and is thus not well-formed. It may be verified that the expression (1) is a well-formed formula.

A well-formed formula Δ is of the form

$$\Delta = [\delta, \Delta_{D(\delta)}, \cdots, \Delta_1]$$

[4] The unique identification of a series-parallel switching network corresponding to a given parenthesis-free expression depends on a convention associating the order in which various operators and variables are listed in the parenthesis-free expression, with the physical ordering of sub-networks within the two-terminal network. The uniqueness does not hold in the reverse direction, in that several distinct but equivalent parenthesis-free expressions can sometimes be found to represent a given switching network.

where δ is of length 1; $D(\delta)$ represents the degree of δ, and each of the Δ_k is well formed. Given an operator δ, it is possible to find subnetwork $\Delta_{D(\delta)}$ by summing the weights of the characters from *left to right* starting with the character immediately to the right of the operator δ, and continuing until the count reaches +1. Subnetwork $\Delta_{D(\delta)-1}$, associated with the same operator δ, is found similarly by resetting the count to zero, and summing the character weights from left to right starting with the first character to the right of the subnetwork $\Delta_{D(\delta)}$, and continuing until the new count reaches +1. The remaining subnetworks associated with δ are found by similar counting procedures. Each well-formed formula Δ_k is thus represented by a shortest head of positive weight.

The set obtained by taking the network associated with a given parenthesis-free expression, as well as the aggregate of all subnetworks associated with each of the operators in the expression, will be called the set of *constituent two-terminal subnetworks*. The constituent two-terminal subnetworks of expression (1) are isolated by the counting procedure shown in Table II.

Corresponding to each operator of degree $i>2$ occurring in a parenthesis-free expression, it is possible to

TABLE II
TWO-TERMINAL SUBNETWORKS FOR NETWORK OF FIG. 2

Operator	Sub-network Number	Subnetwork
—	1	$S_2\ P_2\ p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_2\ S_2\ u'\ x'\ v'\ w'$ $-1-2-1-2-3-2-1-2-10-2-3-2-10+1$
S_2	2	$P_2\ p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'$ $-1\ 0\ -1-2-1\ 0\ -1\ 0\ +1$
S_2	3	$P_2\ S_2\ u'\ x'\ v'\ w'$ $-2-3-2-10+1$
P_2	4	p' $+1$
P_2	5	$S_2\ P_2\ q'\ r'\ P_2\ s'\ t'$ $-1-2-10-10+1$
S_2	6	$P_2\ q'\ r'$ $-1\ 0+1$
S_2	7	$P_2\ s'\ t'$ $-10+1$
P_2	8	q' $+1$
P_2	9	r' $+1$
P_2	10	s' $+1$
P_2	11	t' $+1$
P_2	12	$S_2\ u'\ x'$ $-10+1$
P_2	13	v' $+1$
P_2	14	w' $+1$
S_2	15	u' $+1$
S_2	16	x' $+1$

define *compound* two-terminal subnetworks by taking the constituent subnetworks corresponding to each operator, first two at a time, then three at a time, and, finally, $i-1$ at a time, and joining them by an operator of appropriate degree. Consider, as an example, the P_3 operator and the three associated elementary subnetworks w', v', and $S_2u'x'$ of Table II. By taking the subnetworks two at a time and prefixing them by a P_2 operator, three additional compound two-terminal subnetworks are obtained as follows:

⑰ $P_2 S_2 u' x' v'$,

⑱ $P_2 S_2 u' x' w'$,

⑲ $P_2 v' w'$.

The total number of subnetworks that are isolated by the weight counting procedure depends on the form of the parenthesis-free expression being considered. In order to obtain *all* subnetworks of a given network, it is necessary to operate upon a unique *reduced* form. The reduced form of a parenthesis-free expression may be defined as that form which represents the corresponding series-parallel network with a minimum number of operators. To obtain the reduced form, an attempt is made to replace a set of P (S) operators by a single P (S) operator of larger degree. Specifically, since a parallel (series) array of elements which are themselves in parallel (series) with other elements is equivalent to a larger parallel (series) array, it is possible to use the following equivalences to reduce a parenthesis-free expression:

$$\left.\begin{aligned} P_i\, P_j\, x_1 x_2 \cdots x_{i+j-1} &= P_{i+j-1}\, x_1 x_2 \cdots x_{i+j-1} \\ S_i\, S_j\, x_1 x_2 \cdots x_{i+j-1} &= S_{i+j-1}\, x_1 x_2 \cdots x_{i+j-1} \end{aligned}\right\}. \quad (2)$$

An expression in reduced form has associated with it the maximum number of compound subnetworks, since the degree of its operators is maximized.

Consider an expression $\delta, \Delta_{D(\delta)}, \cdots, \Delta_1$ where each of the Δ_k is well-formed and δ is an operator of degree $D(\delta)$ as before. If the left-most character of any of the Δ_k is an operator of the same type as δ, that operator can be removed from the expression and $D(\delta)$ can be increased in accordance with the formulas (2). Each time a reduction is achieved, the subnetworks corresponding to the "incremented" operator are determined and the process is repeated, until no further reductions are possible. When all operators in the expression have been treated and no further reductions are possible, the reduced form is obtained.

The process is illustrated in Table III for the expression $P_3S_3aP_2bcS_2deP_2fS_2ghP_2iP_2ikl$. The operator being operated upon is underlined and the corresponding subnetworks are shown by brackets. The elimination of an operator from one line of Table III to the next is indicated by a circle.

All expressions shown in Table III correspond to the same networks. The last expression represents the required reduced form.

TABLE III

REDUCTION PROCESS FOR PARENTHESIS-FREE EXPRESSIONS

P_3	S_3	a	P_2	b	c	S_2	d	e	ⓟ$_3$	f	S_2	g	h	P_2	i	P_2	j	k	l
P_3	S_3	a	P_2	b	c	S_2	d	e		f	S_2	g	h	ⓟ$_2$	i	P_2	j	k	l
P_6	S_3	a	P_2	b	c	S_2	d	e		f	S_2	g	h		i	ⓟ$_2$	j	k	l
P_7	S_3	a	P_2	b	c	S_2	d	e		f	S_2	g	h		i		j	k	l
P_7	S_3	a	P_2	b	c	ⓢ$_2$	d	e		f	S_2	g	h		i		j	k	l
P_7	S_4	a	P_2	b	c		d	e		f	S_2	g	h		i		j	k	l

IV. Circuit Parameters

A number of important parameters characterize a series-parallel switching network. Among these,

E = number of elements in the circuit;
Q = number of paths, not including any loops, which join the two terminals of the network;
C = number of connection points, or vertices, of the network;
V = vertical dimension defined as the largest number of branches in any path which joins the two terminals and does not include a loop;
H = horizontal dimension defined as the vertical dimension of the dual network.

Because of the recursive definition of a series-parallel network, the parameter evaluation techniques are based on the fact that each parameter value for a complete network is a function of the parameter values of the constituent subnetworks. Thus, if the parameter values for the individual elements are known, the value for the complete expression can be determined recursively. In particular, given a series-parallel network, it is sufficient to know the value of each parameter for a single branch, for a parallel array of branches, and for a series array of branches. In the parenthesis-free expression this corresponds to the values for a single variable, for a set of well-formed formulas prefixed by a P operator, and for a set of well-formed formulas prefixed by an S operator. The recursive definition of the previously given parameters is shown in Table IV. In each case, X_k refers to the value of X for the kth constituent subnetwork operated upon by the P_i or S_i operator.

Consider, for example, the vertical and horizontal dimensions V and H. The values given reflect the fact that V and H are equal to 1 for a single element, and that a set of networks connected in series will have a vertical dimension equal to the sum of the vertical dimensions of the networks, and a horizontal dimension equal to the maximum of the horizontal dimensions of the networks. The values are reversed for networks connected in parallel.

All parameter values for a given network can now be calculated by a *single linear scan* from right to left through the characters of the corresponding parenthesis-

TABLE IV

RECURSIVE DEFINITION OF NETWORK PARAMETERS

Parameter	Single Element	Formula Prefixed by P_i Operator	Formula Prefixed by S_i Operator
E	1	$\sum_{k=1}^{i} E_k$	$\sum_{k=1}^{i} E_k$
Q	1	$\sum_{k=1}^{i} Q_k$	$\prod_{k=1}^{i} Q_k$
C	2	$\sum_{k=1}^{i} C_k - 2(i-1)$	$\sum_{k=1}^{i} C_k - (i-1)$
V	1	$\max_k V_k$ $k = 1, 2, \cdots, i$	$\sum_{k=1}^{i} V_k$
H	1	$\sum_{k=1}^{i} H_k$	$\max_k H_k$ $k = 1, 2, \cdots, i$

TABLE V

COMPUTATION OF VERTICAL DIMENSION FOR NETWORK OF FIG. 2

S_2	P_2	p'	S_2	P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	V
															w'	(1)
														v'	w'	(1)(1)
													x'	v'	w'	(1)(1)(1)
												u'	x'	v'	w'	(1)(1)(1)(1)
											S_2	u'	x'	v'	w'	(2)(1)(1)
										P_2	S_2	u'	x'	v'	w'	(2)
									t'	P_2	S_2	u'	x'	v'	w'	(1)(2)
								s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(1)(2)
							P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(2)
						r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(1)(2)
					q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(1)(1)(2)
				P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(1)(2)
			S_2	P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(2)(2)
		p'	S_2	P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(1)(2)(2)
	P_2	p'	S_2	P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(2)(2)
S_2	P_2	p'	S_2	P_2	q'	r'	P_2	s'	t'	P_2	S_2	u'	x'	v'	w'	(4)

free expression. For each character in the expression, a parameter value is computed in accordance with the formulas of Table IV. When the left-most character is treated, the correct parameter value for the complete network will have been obtained.

As an example, the complete computation for the vertical dimension V of the network of Fig. 2 is detailed in Table V. Each time a variable is scanned, a (1) is added to the accumulated string of dimensions of constituent subnetworks. Each time a P_i or S_i operator is scanned, the i dimensions of its constituents are removed from the string, and their sum or maximum is substituted in accordance with the formulas of Table IV. The vertical dimension computed when the left-most operator is scanned is 4. It may be seen from Fig. 2 that this is indeed the correct dimension of the network. This value corresponds to one of the paths including the connection points 1, 3, 4, 5, and 2. The other parameter values may be similarly ascertained. For the network of Fig. 2 corresponding to expression (1), one obtains respectively $E=9$, $Q=15$, $C=5$, and $H=3$.

Depending upon the switching elements used, certain additional parameters may be related to the feasibility of a network. For example, if transistor inverters are connected in series and parallel, the number of distinct, simultaneously conducting transistors that lie between any given transistor and the output terminal is a limiting factor. Accordingly, the *collector loading of a switching element* may be defined abstractly as the number of distinct elements that lie on all paths, excluding loops, between the given transistor and the output terminal. In Fig. 2 the collector loading of the s' transistor is 2, since the transistors q' and r' lie between the s' transistor and the output terminal. The *collector loading of a network* may be defined as the maximum of the collector loadings of its elements. Thus, the collector loading of the network in Fig. 2 is 6 corresponding to the loading of the x' transistor.[5]

It is apparent that the collector loading of a network is affected by the ordering of its constituent subnetworks. Consider a series connection of i subnetworks. The collector loading of the complete network is

$$L = \sum_{k=1}^{i} E_k + L_j, \qquad k \neq j \tag{3}$$

where L_j is the collector loading of the constituent subnetwork located closest to terminal 2, *i.e.*, closest to ground. Eq. (3) is minimized to give

$$L_{\min} = \sum_{k=1}^{i} E_k + \min_k \{L_k - E_k\}. \tag{4}$$

Thus, for a set of subnetworks connected in series, that subnetwork should be placed closest to ground for which $L_k - E_k$ is smallest. If a set of subnetworks is connected in parallel, the collector loading of the complete network is simply the maximum collector loading of the subnetworks.

As before, the collector loading of a given network can be computed recursively in terms of the collector loadings of the constituent subnetworks. The relevant formulas are shown in Table VI, where L is taken to be the minimum value of the collector loading over any reordering of constituent subnetworks, and L' is the collector loading for the dual network. The formulas for the element count E are repeated from Table IV for completeness.

To compute the collector loading, the characters in the parenthesis-free expression are again scanned one at

[5] The definition of the collector loading given above for an abstract network is somewhat more restrictive than the actual restriction on the corresponding physical network. Loading is actually determined only by those elements which may be simultaneously in a conducting state.

a time from right to left. It is necessary, however, to store for each elementary subnetwork a pair of digits (L, E) corresponding to the collector loading and the element count. Each time a variable is treated, the pair (0, 1) is added to the string; for P_i and S_i operators, the i ordered pairs corresponding to the i constituent subnetworks are used to compute a new ordered pair in accordance with the formulas of Table VI. The computation of the collector loading for the network of Fig. 2 is shown in Table VII. It may be seen that the collector loading is found to be equal to 6 and the transistor count equal to 9.

TABLE VI
RECURSIVE DEFINITION OF COLLECTOR LOADING

Parameter	Single Element	Formula Prefixed by P_i Operator	Formula Prefixed by S_i Operator
E	1	$\sum_{k=1}^{i} E_k$	$\sum_{k=1}^{i} E_k$
L	0	$\max_k L_k$ $k = 1, 2, \cdots, i$	$E + \min_k \{L_k - E_k\}$
L'	0	$E + \min_k \{L_k' - E_k\}$	$\max_k L_k'$ $k = 1, 2, \cdots, i$

TABLE VII
COMPUTATION OF COLLECTOR LOADING FOR NETWORK OF FIG. 2

$S_2\ P_2\ p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(L, E)
w'	(0, 1)
$v'\ w'$	(0, 1)(0, 1)
$x'\ v'\ w'$	(0, 1)(0, 1)(0, 1)
$u'\ x'\ v'\ w'$	(0, 1)(0, 1)(0, 1)(0, 1)
$S_2\ u'\ x'\ v'\ w'$	(1, 2)(0, 1)(0, 1)
$P_3\ S_2\ u'\ x'\ v'\ w'$	(1, 4)
$t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 1)(1, 4)
$s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 1)(0, 1)(1, 4)
$P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 2)(1, 4)
$r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 1)(0, 2)(1, 4)
$q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 1)(0, 1)(0, 2)(1, 4)
$P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 2)(0, 2)(1, 4)
$S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(2, 4)(1, 4)
$p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(0, 1)(2, 4)(1, 4)
$P_2\ p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(2, 5)(1, 4)
$S_2\ P_2\ p'\ S_2\ P_2\ q'\ r'\ P_2\ s'\ t'\ P_3\ S_2\ u'\ x'\ v'\ w'$	(6, 9)

V. CIRCUIT RESTRICTIONS

The parameters defined in the preceding section are related to actual restrictions on series-parallel switching circuits. For example, for transistor circuits, the values of H, L, and V must generally not exceed certain critical values H_0, L_0, and V_0. Restrictions of this type may be classified as *dimensional restrictions*.

Another type of restriction is imposed by the unavailability of certain input polarities. If input signal x is available, but not x', the circuitry must be designed to use only x. Restrictions of this type may be classified as *polarity restrictions*.

Both types of restrictions can be satisfied by using the *cascading operation*. The cascading operation is based on the fact that the output of a switching circuit is unaffected if any two-terminal subnetwork is removed, and a functionally equivalent network is substituted. In the case of a transistor inverter network, any subnetwork can be replaced by a single transistor, and the dual of the subnetwork cascaded into this transistor. Such a substitution is illustrated in Fig. 3.

To dualize a cascaded subnetwork, it is sufficient to alter the corresponding parenthesis-free expression by replacing P by S, S by P, and primed (unprimed) vari-

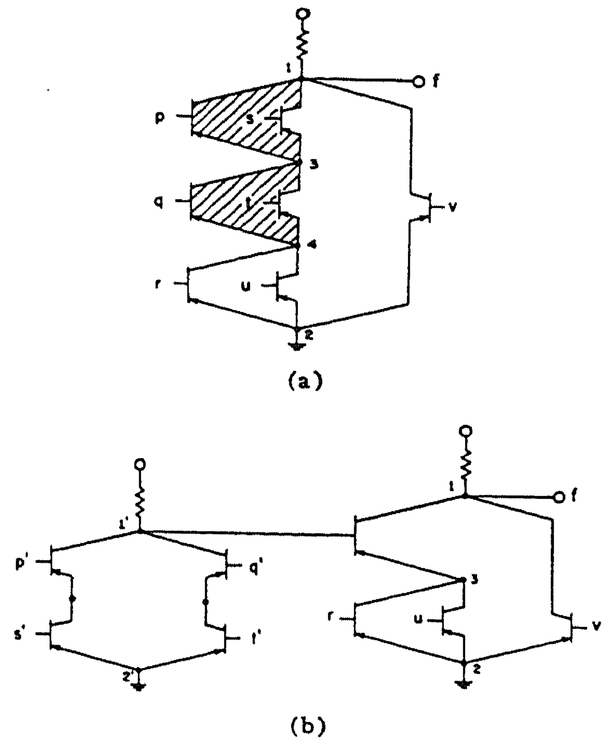


Fig. 3—(a) Transistor inverter network prior to cascading. (b) Equivalent network resulting from cascading operation.

ables by unprimed (primed) ones. The cascading operation itself can be indicated by prefixing the expression of the dualized submatrix with a C_1 operator. The cascade operator stands for the transistor which replaces the dualized subnetwork in the original network. Consider, for example, the network of Fig. 3. The expression for the subnetwork being cascaded is $S_2\ P_2\ p\ s\ P_2\ q\ t$; the dualized input to the cascade operator C_1 will then be $P_2\ S_2\ p'\ s'\ S_2\ q'\ t'$ and the expression for the complete array of Fig. 3(b) is $P_2\ S_2\ C_1\ \{P_2 S_2 p' s' S_2 q' t'\} P_2\ r\ u\ v$.

The cascading operation can be defined similarly for non-inverting switches, such as transistor emitter followers. However, whereas for an inverter network a two-terminal subnetwork must be replaced by a single element whose input is the dual of the function generated by the subnetwork, no dualization is required for emitter followers.

A circuit resulting from the repeated application of cascading operations is a tree-like structure of cascaded subnetworks, as shown schematically in Fig. 4. Each cascading operation generates a new network and increases the cost of the complete circuit by one additional circuit element.

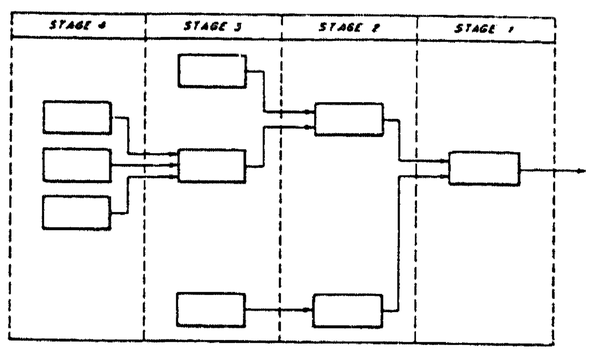


Fig. 4—Circuit resulting from repeated application of cascading operations.

Since each dimensional parameter of a network is a monotonically nondecreasing function of the parameter values of the constituent subnetworks, cascading operations tend to decrease the parameter value for the entire network, by substituting single switching elements for larger constituents. As an example, the network of Fig. 3(a) has dimensional parameter values equal to $V=3$, $H=3$, and $L=4$. The same parameters are equal to $V=2$, $H=3$, and $L=1$ for the two-terminal subnetwork on the right-hand side of Fig. 3(b), and to $V=2$, $H=2$, and $L=2$ for the cascaded subnetwork on the left-hand side of Fig. 3(b).

The cascading operation is also useful in satisfying polarity restrictions. If an inverter network contains a transistor whose input, x, is not available, then some subnetwork containing x can be cascaded, and the available polarity x' can be used to form the dual.

While it is always possible to find a combination of cascading operations to transform an infeasible network into one satisfying the dimensional restrictions, each cascaded subnetwork may have to be transformed in turn by further cascading operations. In fact, several physical stages may have to be created before a satisfactory circuit is obtained. Each cascading operation places the cascaded subnetwork one stage further from the output of the complete circuit. A limit for the number of stages is set by *delay restrictions*.

A nominal delay D can be assigned to the output of a circuit if the delay associated with each input is known. If D_i is the delay relative to a clock pulse, associated with the ith input of the circuit, and s_i is the stage at which the input enters (counting the stage closest to the final output as 1, the next stage as 2, etc., as shown in Fig. 4), then the delay D associated with the output of the circuit is

$$D = \operatorname{Max}_i (D_i + s_i\tau),$$

where τ is the nominal delay assigned to each cascaded network. Clearly, the delay is lessened if the more-delayed inputs are entered at the stages closest to the output of the total circuit.

If the delay of the final output of a circuit is to be less than some critical value D_0, the delay associated with a network cascaded into the jth stage must be no more than $D_j = D_0 - j\tau$. Similarly, the variable x may be an input to a network at the jth stage if and only if $D_x \leq D_0 - j\tau = D_j$.

A *feasible two-terminal network* can now be defined as one whose dimensional parameters do not exceed certain critical values, whose inputs are available in the required polarities, and whose output is delayed by no more than some critical value, depending upon the stage at which the network is located in a larger circuit. A *feasible circuit* is either a feasible two-terminal network, or a set of cascaded networks, each of which is feasible. Since the output of each cascaded network leads to only one other network, output loading is not a factor in determining feasibility.

The problem of generating feasible circuits by cascading operations in the most efficient way can now be formulated as follows, given

1) a single network N,
2) a set of critical values for the height, width, and possibly collector loading,
3) a list of unavailable polarities for all input variables,
4) delay values D_0 and τ, and the delays associated with each polarity of each input,

to transform the network N into a feasible circuit using a minimum number of cascading operations.

VI. The Minimization Procedure, Exclusive of Delay Restrictions

It is evident that by generating all possible combinations of cascading operations, the feasible circuit or circuits with the fewest cascading operations may be exhaustively determined. A systematic procedure will now be described for obtaining the same result in a non-exhaustive fashion. Specifically, given an expression in parenthesis-free notation corresponding to a switching network, the set of all subnetworks of the complete network is generated by the reduction and weight-counting procedure described in Section III. Each of these subnetworks is then examined for feasibility using the parameter calculation techniques described in Section IV. The set of feasible subnetworks thus found is used to determine feasible circuits for those subnetworks that can be rendered feasible with a single cascading operation. Feasible circuits generated by two, three, $\cdots$, n cascading operations are successively determined until a feasible circuit is generated for the entire network. This is done by using at each step feasible circuits formed by 0, 1, 2, $\cdots$, i cascading operations to find feasible circuits with $i+1$ cascading operations.

The procedure will now be formalized. Let the *order* of a network be defined as the least number of cascading operations required to transform the network into a feasible circuit. In the present development, delay is not assumed to be a factor affecting the feasibility of a network. Let the sets R_0, R_1, R_2, $\cdots$, R_{i-1} contain all subnetworks of a given network that are of order 0, 1, 2, $\cdots$, $i-1$. Similarly, let the sets R_0', R_1', $\cdots$, R_{i-1}' contain the corresponding subnetworks for the dual of the given network. If inverter elements are used as circuit elements the sets R_0, R_1, $\cdots$, R_{i-1} can be used to determine all the members of R_i'. Similarly, the sets R_0', R_1', $\cdots$, R_{i-1}' can be used to determine all the members of R_i.[6]

Specifically, the set R_i is formed by considering all the subnetworks that are not members of R_0, R_1, $\cdots$, R_{i-1}. For each of those subnetworks, an attempt is made to demonstrate that i cascading operations will transform it into a feasible circuit. In the case of an inverter array, this can be done by finding some subnetwork whose dual is of order $i-1$, such that when the subnetwork is cascaded, the given network is rendered feasible. Alternatively, it can be done by taking i subnetworks, whose duals are feasible, such that when all the subnetworks are cascaded, the given network is rendered feasible. More generally, it is necessary to find sets of j subnetworks such that

$$\sum_{k=1}^{j} r_k + j = i,$$

where r_k is the order of the dual of the kth subnetwork. To ascertain that a given set of j cascading operations render the given network feasible, it is possible to substitute a single element for each cascaded subnetwork in the parenthesis-free expression, to recalculate the dimensional parameters as previously described, and to verify that the required polarities for all input variables are available.

Consider a given set of j subnetworks from the sets R_0', R_1', $\cdots$, R_{i-1}' which may be cascaded to render a given network feasible. A particular subnetwork is either included in the given combination of j subnetworks, or it is not included. If a particular subnetwork is included, then no other subnetwork is included that has circuit elements in common with it, since it is not possible to cascade overlapping subnetworks. If a particular subnetwork is not included, it may be necessary to include others, in order to render the network feasible; hence not all possible combinations of subnetworks need be considered.

A convenient, graphical means for examining the logical alternatives is to lay out the problem as a tree composed of nodes and arcs [2], [9]. Each node corresponds to a set of cascaded subnetworks, and each arc stands for the inclusion into the cascaded set or exclusion from the cascaded set of a given subnetwork.[7]

The significant information associated with each node is

1) a set A of j subnetworks that have been included in the combination of cascading operations,
2) a set B of subnetworks that have yet to be either included or excluded,
3) the cost $\sum r_k + j$ associated with the cascading operations already included in set A.

Two arcs may extend from a given node. One arc, called the I-arc, leads to a node, at which a subnetwork contained in set B has been *included* in the combination of cascading operations. The other arc, called the E-arc, leads to a node at which the *same* subnetwork has been *excluded*.

No arcs lead from a node that has been *terminated*. A node is terminated if

1) no combination of cascading operations drawn from the subnetworks that remain in B would be sufficient to render the given network feasible, at any cost;
2) the cost associated with the node is i or greater, where i is the cost of the feasible circuit to be generated.

It is evident that if a suitable combination of subnetworks exists, it will be associated with one of the terminal nodes of the tree.

[6] If circuit elements are used that do not require dualization of the cascaded subnetworks, R_0, R_1, $\cdots$, R_{i-1} could be used directly to determine R_i.

[7] In the actual computer procedure, the tree can be replaced by a listing of subnetworks with a defined lexicographic ordering.

The branching procedure for finding a circuit of cost i for a given network may be outlined as follows. At the initial node of the tree, the set A is empty; the cost is zero; and the set B consists of all the subnetworks of the given network that are members of R_0', R_1', $\cdots$, or R_{i-1}'. In actual practice, many of these sets may be empty and thus contain no subnetworks. The number of branching operations may be lessened if the subnetworks in B are ordered by number of included elements. Each branching operation is then performed with respect to that subnetwork of set B which includes the largest number of circuit elements. The information associated with the node terminating an E-arc is the same as the information associated with the previous node, except that one member of B has been eliminated. The I-node, on the other hand, is created by transferring the included member to A, augmenting the cost, and eliminating from B those members that have elements in common with the included subnetwork.

Before a new branch is created at a node, a *sufficiency test* is performed to determine if the set B is sufficient to render the given network feasible, at any cost. If the sufficiency test fails, it is fruitless to continue branching from the node in question.

A simple test of sufficiency consists in attempting to cascade in turn all of the subnetworks remaining in set B. If two or more subnetworks have an element in common, a single cascading operation is performed for the combination. The resulting network is examined for feasibility and if it is infeasible the sufficiency test fails.

As an example of the complete branching procedure, consider the expression

$$S_2\ P_2\ S_2\ t\ u\ x\ P_2\ v\ w\ S_2\ y\ z. \tag{5}$$

Let the dimensions V, H, and L be restricted to $V \leq 3$, $H \leq 2$, and $L \leq 2$ and let inputs u and y be available only in the positive polarity and inputs x and v only in the negative polarity. A computation of the dimensions indicates $V = 4$, $H = 3$, and $L = 4$. The complete network is thus not of order 0 and one or more cascading operations are necessary to render the network feasible.

The network has 14 subnetworks (excluding the complete network). The duals of six of these subnetworks are feasible; these six dualized subnetworks thus constitute the set R_0'. The duals of the remaining eight subnetworks are feasible at the cost of one additional circuit element; they thus constitute R_1'. The fourteen subnetworks and their feasible duals are shown in Table VIII.

It may be ascertained that the cascading of any member of R_0' is not sufficient to render the complete network feasible. The complete network is thus not of order 1. For purposes of illustration, the computations are thus started using the subnetworks of Table VIII.

At the origin of the tree, the cost is zero and the set

TABLE VIII
SUBNETWORKS AVAILABLE FOR CASCADING FOR NETWORK OF FIG. 6(a)

Subnetworks		Feasible Duals	
①	t	R_0'	t'
②	v		v'
③	w		w'
④	x		x'
⑤	z		z'
⑥	$P_2\ v\ w$		$S_2\ v'\ w'$
⑦	u	R_1'	$C_1\ u$
⑧	y		$C_1\ y$
⑨	$S_2\ t\ u$		$P_2\{C_1\ u\}t'$
⑩	$S_2\ y\ z$		$P_2\{C_1\ y\}z'$
⑪	$P_2\ S_2\ t\ u\ x$		$S_2\{C_1\ S_2\ t\ u\}x'$
⑫	$P_2\ v\ S_2\ y\ z$		$S_2\ v'\{C_1\ S_2\ y\ z\}$
⑬	$P_2\ w\ S_2\ y\ z$		$S_2\ w'\{C_1\ S_2\ y\ z\}$
⑭	$P_2\ v\ w\ S_2\ y\ z$		$S_2\ v'\{C_1\ P_2\ w\ S_2\ y\ z\}$

of networks available for cascading is numbered 1–14. An attempt is first made to cascade subnetwork 14. At node E14 (exclude 14), the cost is still zero and the subnetworks still available are 1–13. At node I14 (include 14) on the other hand, the cost is 2 since the dual of subnetwork 14 is of order 1 and one additional circuit element is required to cascade the subnetwork. Furthermore, only subnetworks 1, 4, 7, 9, and 11 are still available for cascading, since any subnetwork including elements v, w, y, or z must be eliminated. The expression for the complete network at node I14 becomes

$$S_2\ P_2\ S_2\ t\ u\ x\{C_1\ S_2\ v'[C_1\ P_2\ w\ S_2\ y\ z]\}. \tag{6}$$

It may be verified that the complete network is still infeasible since input x is unavailable in the positive polarity. At least one additional subnetwork including element x must thus be cascaded to render the network feasible.

The branching operations are illustrated by the tree of Fig. 5. Next to each node are shown the cost, the I-(included) subnetworks, the E-(excluded) subnetworks, and the set of subnetworks still available for cascading (in square brackets). The network obtained at node I14 14 is feasible at a cost of 3. The corresponding expression is

$$S_2\ P_2\ S_2\ t\ u\{C_1\ x'\}\{C_1\ S_2\ v'[C_1\ P_2\ w\ S_2\ y\ z]\}. \tag{7}$$

It may be noted that while the complete network is of order 3, the duals of all subnetworks of the network are of order 0 or 1. Sets R_2', R_3', $\cdots$, are therefore empty and need not be considered in the branching process for the example. All the open nodes of the tree which have a

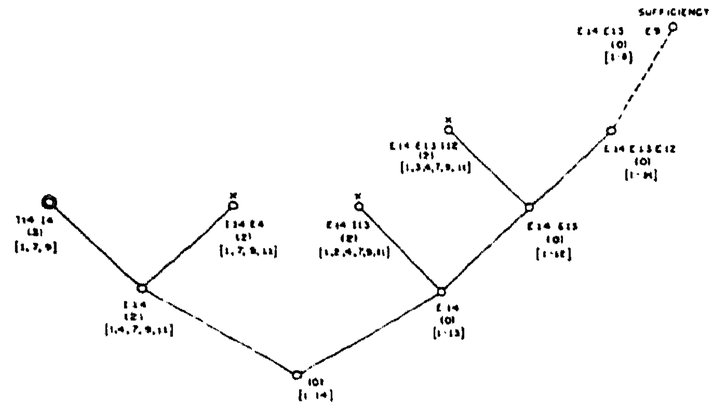


Fig. 5—Branching procedure for subnetworks of Table VIII.

cost of 2 and do not correspond to a feasible solution can be terminated, since they cannot possibly yield a feasible solution at a cost of less than 3. This is indicated by an *X* above the corresponding node in Fig. 5. Node E14 E13 E12 E11 E10 E9 can be terminated because of failure of the sufficiency test.

The original infeasible network, as well as the infeasible network at node I14 and the final feasible network at node I14 I4 are illustrated in Fig. 6(a)–6(c), respectively. The networks of Fig. 6(b) and 6(c) require respectively two and three more circuit elements than the original network of Fig. 6(a).

VII. The Minimization Procedure, Including Delay Restrictions

Delay considerations may restrict some inputs to certain physical stages. For example, input x may be available at stages 1, 2, and 3, and not at stages 4, 5, 6, $\cdots$, and x' may be available at stages 1, 2, 3, and 4, and not at stages 5, 6, 7, $\cdots$. Such restrictions may prevent a given subnetwork from being cascaded at stage $j+1$, although it may be cascaded at stage j. In general, it is seen that the number of possible cascading operations becomes smaller at stages further from the output of the circuit. Because possible cascading operations are more restricted, it may become more costly to render a given subnetwork feasible at these stages.

It is convenient to designate the order of a network at each stage. In particular, one may define the set $R_i(j)$ as the set of networks which, when located at stage j, require a minimum of i cascading operations to transform them into feasible circuits.

Again assuming transistor inverters as circuit elements, the sets $R_0(j), R_1(j), \cdots, R_{i-1}(j)$ can be used to determine all the members of $R_i'(j-1)$. Conversely, the sets $R_0'(j), R_1'(j), \cdots, R_{i-1}'(j)$ can be used to determine all members of $R_i(j-1)$.

Since every feasible network of cost i at stage $j-1$ results from a cascading of subnetworks drawn from the sets $R_0(j), R_1(j), \cdots, R_{i-1}(j)$ or $R_0'(j), R_1'(j), \cdots, R_{i-1}'(j)$, the members of these sets constitute the set B. The branching procedure is then identical to that previously described. The sequence in which the various sets are determined is as follows:

1) $R_0(1)$
2) $R_0'(2)$
3) $R_1(1)$
4) $R_0(3)$
5) $R_1'(2)$
6) $R_2(1)$, etc.

This sequence may be represented graphically, as shown in Fig. 7. The figure is swept diagonally, starting with the set $R_0(1)$ in the upper left-hand corner, then going to the sets on the next diagonal, $R_0'(2)$, $R_1(1)$ and so on.

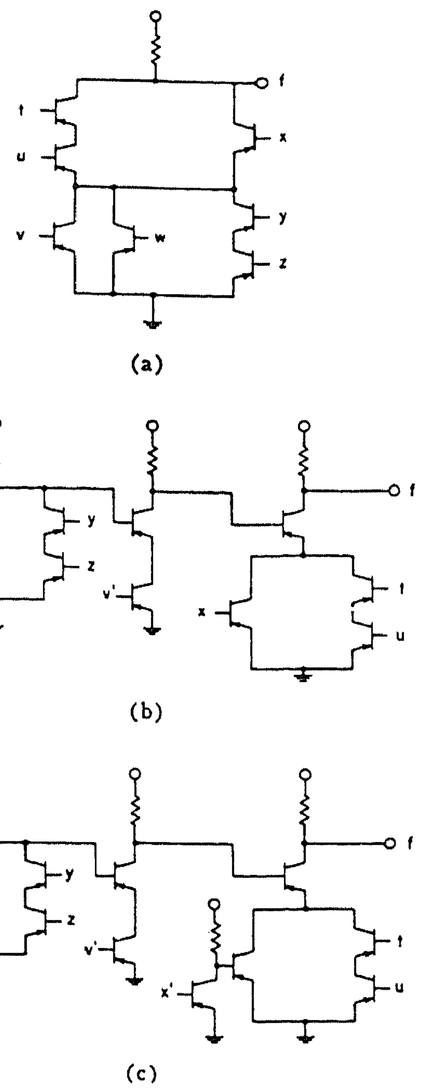


Fig. 6—(a) Infeasible network corresponding to expression (5) before cascading. (b) Infeasible network corresponding to expression (6). (c) Feasible network corresponding to expression (7).

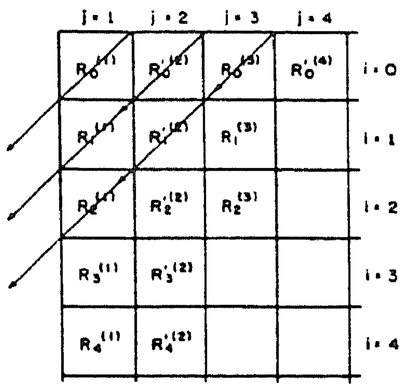


Fig. 7—Sequence of generation of sets $R_i(j)$.

The procedure is continued until the original network is found to belong to some set $R_i(1)$, or until it is evident that no feasible circuit will be found at any finite cost i. A sufficient condition for terminating the procedure is for stage 1 to be *complete*. Stage j is said to be complete if:

1) feasible circuits have been found for all subnetworks whose inputs are permitted to enter at stage j in one polarity or the other,

or

2) if feasible circuits have not been found for all these subnetworks, stage $j+1$ is complete, and each node of the tree for each remaining subnetwork is terminated because of failure of the sufficiency test.

If there are no delay restrictions, a solution will in general be found since the number of stages is then unlimited, and subnetworks can be cascaded at will. If delay restrictions exist, a solution may or may not be possible.

The method described offers a systematic, mechanized means for obtaining a feasible circuit, subject to restrictions of dimension, polarity, and delay. Although the circuit that is obtained is not necessarily the most economical implementation of an arbitrary Boolean function, it is the most economical circuit that can be obtained from a given network by cascading operations.

Bibliography

[1] T. C. Bartee, "The automatic design of logical networks," *Proc. WJCC*, pp. 103–107; March, 1959.
[2] J. P. Roth, and E. G. Wagner, "Algebraic topological methods for the synthesis of switching functions—III. Minimization of non-singular Boolean trees," *IBM J. Res. & Dev.*, vol. 4, pp. 326–344; October, 1959.
[3] A. E. Randlev, J. P. Roth, and E. G. Wagner, "Algorithms for logical design," presented at the AIEE Fall General Mtg., Chicago, Ill., October 11–16, 1959; Conf. Paper 59-1191; October, 1959.
[4] J. Lukasiewicz, "Aristotle's Syllogistic from the Standpoint of Modern Formal Logic," Oxford University Press, Oxford, Eng.; 1951.
[5] H. Whitney, "Nonseparable and planar graphs," *Trans. Amer. Math. Soc.*, vol. 34, pp. 339–363; April, 1932.
[6] P. Rosenbloom, "The Elements of Mathematical Logic," Dover Publications, New York, N. Y., ch. 4; 1950.
[7] B. W. Arden, "On the Construction of Algorithm Translators," *Preprints of the 14th Natl. ACM Meeting*, Cambridge, Mass.; September 1–3, 1959; pp. 231–4.
[8] A. W. Burks, D. W. Warren, and J. B. Wright, "An Analysis of a Logical Machine," *M.T.A.C.*, vol. 8, pp. 53–57; April, 1954.
[9] W. L. Eastman, "Linear Programming with Pattern Constraints," Doctoral dissertation, Harvard University, Cambridge, Mass., ch. 3; July, 1958.

Reprinted from MANAGEMENT SCIENCE
Vol. 9, No. 4, July, 1963
Printed in U.S.A.

THE QUADRATIC ASSIGNMENT PROBLEM*†

EUGENE L. LAWLER

University of Michigan, Ann Arbor

This paper presents a formulation of the quadratic assignment problem, of which the Koopmans-Beckmann formulation is a special case. Various applications for the formulation are discussed. The equivalence of the problem to a linear assignment problem with certain additional constraints is demonstrated. A method for calculating a lower bound on the cost function is presented, and this forms the basis for an algorithm to determine optimal solutions. Further generalizations to cubic, quartic, N-adic problems are considered.

1. Introduction

The *linear assignment problem* is a special case of the wellknown transportation problem, and can be stated as follows: Given an $n \times n$ cost matrix $C = \| x_{ij} \|$, determine an $n \times n$ solution matrix $X = \| x_{ij} \|$ so as to

$$\text{(1)} \qquad \text{minimize} \quad \sum_{ij} c_{ij}x_{ij}$$

$$\text{(2)} \qquad \text{subject to} \quad \sum_{j} x_{ij} = 1 \qquad (i = 1, 2, \cdots, n),$$

$$\text{(3)} \qquad \sum_{i} x_{ij} = 1 \qquad (j = 1, 2, \cdots, n),$$

$$\text{(4)} \qquad \text{and} \quad x_{ij} = 0 \text{ or } 1 \qquad (1, j = 1, 2, \cdots, n).$$

There are several efficient methods for solving this problem, e.g., the method of Ford and Fulkerson [2]. The fact that the solution must be discrete, i.e., a permutation matrix, causes no difficulty. Any fractional solution, i.e., a doubly stochastic matrix, is a convex combination of permutation matrices, each having the same cost.

The *quadratic assignment problem* is defined as follows: Given n^4 cost coefficients c_{ijpq} $(i, j, p, q = 1\ 2, \cdots, n)$, determine an $n \times x$ solution matrix $X = \| x_{ij} \|$ so as to

$$\text{(5)} \qquad \text{minimize} \quad \sum_{ij} \sum_{pq} c_{ijpq}x_{ij}x_{pq}$$

subject to constraints (2), (3), and (4). In this case, the discrete nature of the problem does cause difficulty. The only exact methods of solution known to the author are those presented in this paper.

The next section describes the original Koopmans-Beckmann version of the

* Received August 1962.

† This paper constitutes, with minor revisions, one chapter of the author's Ph.D. thesis in applied mathematics at Harvard University. The writing of the thesis was supported in part by a contractural arrangement with the Bell Telephone Laboratories. The author wishes to acknowledge the advice and encouragement of Dr. Willard Eastman during the writing of the thesis, and that of Dr. Kenneth Iverson during the early stages of investigation.

quadratic assignment problem. The sections that follow describe the formulation of a problem in sequencing, the transformation to an equivalent integer linear program and a method for computing minimal solutions. A final section discusses extensions to cubic, quartic, $\cdots$ n-adic assignment problems.

For notational convenience, the *dot product* of two matrices will sometimes be referred to. If A and B are compatible,

$$A \cdot B = \sum_{ij} a_{ij} b_{ij} .$$

2. The Koopmans-Beckmann Formulation

The first published statement of the quadratic assignment problem was by Koopmans and Beckmann [6], in the context of an analysis of the location of economic activity. Their formulation of the problem, which is somewhat less general than that given above, is as follows.

Let it be required to assign n plants to n locations in such a way that the total cost of interplant transportation is minimized. Two $n \times n$ matrices, $D = \| d_{jq} \|$ and $T = \| t_{ip} \|$ are given, where

d_{jq} = cost of transporting one unit of commodity from location j to location q.

t_{ip} = number of units of commodity to be transported from plant i to plant p.

Each assignment of plants to locations is represented by an $n \times n$ permutation matrix $X = \| x_{ij} \|$ where

$$\begin{aligned} x_{ij} &= 1 \text{ if plant } i \text{ is assigned to location } j, \\ &= 0 \text{ otherwise.} \end{aligned}$$

The Koopmans-Beckmann problem is to minimize the dot product of D and T with respect to a symmetric permutation of the rows and columns of one of the matrices. That is, to

$$\text{minimize} \quad T \cdot (XDX^t). \tag{6}$$

It is seen that (6) is a special case of the quadratic form (5) with $c_{ijpq} = t_{ip} d_{jq}$. The coefficient c_{ijpq} represents "the cost of transportation from plant i at location j to plant p at location q." Additional linear costs of the form $B \cdot X$ can be incorporated into the quadratic form (5) by setting $c_{ijij} = t_{ii} d_{jj} + b_{ij}$.

Although not so generalized by Koopmans and Beckmann, a multicommodity problem can also be stated. Let $D^{(1)}, T^{(1)}; D^{(2)}, T^{(2)}; \cdots ; D^{(m)}, T^{(m)}$ be m pairs of matrices for m different commodities. Then it may be required to

$$\text{minimize} \quad \sum_k T^{(k)} \cdot (XD^{(k)}X^t),$$

which is equivalent to a quadratic form with

$$c_{ijpq} = \sum_k t_{ip}^{(k)} d_{jq}^{(k)} .$$

3. The Candidates' Problem

It is noteworthy that the traveling-salesman problem is a special case of the Koopmans-Beckmann formulation, in which D is a distance matrix and T is a

cyclic permutation matrix of the form

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

It is therefore not surprising that a variety of problems involving permutations and sequences can be formulated as quadratic assignment problems. For example, consider the following "candidates' problem":

The Presidential and Vice-Presidential candidates of one of the major parties are planning their campaign itinerary. They decide that there are $2n$ cities that should be subjected to a full day's visit by one or the other of them during a particular n-day period. It does not matter which candidate visits any given city, or on what day within the n-day period he visits it. Each night during the period the two candidates will confer for a fixed length of time by long distance telephone. The day before his tour begins, the Presidential candidate is scheduled to be in city P_0, and the day after in P_{n+1}. Similar engagements for the Vice-Presidential candidate are V_0 and V_{n+1}.

The problem is to plan the tours for the two candidates in such a way that the total of the transportation costs and the telephone costs is minimized. Travel costs for the Presidential candidate and his entourage are p cents per airline mile, and for the Vice-Presidential candidate v cents. Telephone charges are t cents per airline mile.

Let a $2n \times 2n$ matrix D represent the distance between the $2n$ cities. Also define a $2n \times 2n$ matrix T of the form

$$T = \left[\begin{array}{ccccc:ccccc} 0 & p & 0 & 0 & 0 & t & 0 & 0 & 0 & 0 \\ 0 & 0 & p & 0 & 0 & 0 & t & 0 & 0 & 0 \\ 0 & 0 & 0 & p & 0 & 0 & 0 & t & 0 & 0 \\ 0 & 0 & 0 & 0 & p & 0 & 0 & 0 & t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & v & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & v & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

The three nonzero quadrants are related to the Presidential candidate's tour, the telephone conferences, and the Vice-Presidential candidate's tour, respectively.

The solution is represented by a $2n \times 2n$ permutation matrix $X = \| x_{ij} \|$, whose elements are defined as follows:

For $i \leqq n$:

$x_{ij} = 1$ if the Presidential candidate visits city j on the i^{th} day,
$= 0$ otherwise.

For $i \geqq n + 1$:

$x_{ij} = 1$ if the Vice-Presidential candidate visits city j on the $(i - n)^{\text{th}}$ day,
$= 0$ otherwise.

The quantity

$$T \cdot (XDX^t)$$

represents all costs except the travel costs to and from the cities P_0, P_{n+1}, V_0 and V_{n+1}. For this purpose, define a $2n \times 2n$ matrix B, where

$$b_{1j} = p \text{ times the distance from city } P_0 \text{ to city } j,$$

$$b_{n+1,j} = v \text{ times the distance from city } V_0 \text{ to city } j,$$

$$b_{nj} = p \text{ times the distance from city } j \text{ to city } P_{n+1},$$

$$b_{2n,j} = v \text{ times the distance from city } j \text{ to city } V_{n+1},$$

and the remaining b_{ij} are zero. Then the objective of the problem is to minimize

$$T \cdot (XDX^t) + B \cdot X,$$

which is a single commodity Koopmans-Beckmann problem with additional linear costs.

4. Other Problems

Among the other problems that can be formulated are the minimization of "latency" in magnetic drum computers [8] (for which a multicommodity Koopmans-Beckmann problem is an excellent model), the placement of electronic assemblies so as to reduce total wire length [3, 5, 9], and various problems in the synthesis of sequential switching circuits.

5. An Equivalent Linear Program

A quadratic assignment problem with n^2 variables x_{ij} can be linearized by defining n^4 variables y_{ijpq}, where, effectively,

$$y_{ijpq} = x_{ij}x_{pq}.$$

Consider the following integer linear program in the $n^4 + n^2$ variables y_{ijpq} and x_{ij}:

$$\text{(7)} \qquad \text{minimize} \quad \sum_{ijpq} c_{ijpq}y_{ijpq}$$

$$\text{(8)} \qquad \text{subject to} \quad \sum_j x_{ij} = 1 \qquad (i = 1, 2, \cdots, n),$$

$$\text{(9)} \qquad \sum_i = 1 \qquad (j = 1, 2, \cdots, n),$$

$$\text{(10)} \qquad \sum_{ijpq} y_{ijpq} = n^2$$

$$\text{(11)} \qquad x_{ij} + x_{pq} - 2y_{ijpq} \geqq 0 \quad (i, j, p, q = 1, 2, \cdots, n),$$

and

$$\text{(12)} \qquad x_{ij} = 0 \text{ or } 1 \qquad (i, j = 1, 2, \cdots, n),$$

$$\text{(13)} \qquad y_{ijpq} = 0 \text{ or } 1 \qquad (i, j, p, q = 1, 2, \cdots, n).$$

Let the quadratic assignment problem defined by conditions (2) through (5) be designated problem Q, and the integer linear programming problem defined by conditions (7) through (13) be designated problem L. The following theorem asserts the equivalence of Q and L for any given set of cost coefficients.

Theorem: The feasible solutions of problems Q and L can be placed in one-to-one correspondence with equal values of the cost functions. A feasible solution $X^{(Q)}$ of Q corresponds to a feasible solution $(X^{(L)}, Y)$ of L if and only if $X^{(Q)} = X^{(L)}$.

Proof: It is sufficient to show that the constraints of problem L are such that for any given permutation matrix $X^{(L)}$, Y is determined uniquely by the relation

$$y_{ijpq} = x_{ij}x_{pq} .$$

Since all of the variables are restricted to the values 0 and 1, this relation is equivalent to

$$y_{ijpq} = 1 \Leftrightarrow x_{ij} = x_{pq} = 1.$$

It follows immediately from (11) that

$$y_{ijpq} = 1 \Rightarrow x_{ij} = x_{pq} = 1.$$

In order to prove the converse, let $x_{ij_i} = 1$ and $x_{ij} = 0$ for $j \neq j_i$, and similarly let $x_{pq_p} = 1$ and $x_{pq} = 0$ for $q \neq q_p$. Then, from (11), $y_{ijpq} = 0$ unless $j = j_i$ and $q = q_p$. It follows that

$$\sum_{ip} y_{ijpq} \leqq 1$$

and this inequality is strict unless $y_{ij_ipq_p} = 1$. Now summing over all i and p, we have

$$\sum_{ip} \left(\sum_{jq} y_{ijpq}\right) < n^2$$

unless $y_{ijpq} = 1$ whenever $x_{ij} = x_{pq} = 1$. Q.E.D.

6. Cost Bounds

The quadratic assignment problem is a finite problem. In principle, each of the $n!$ feasible solutions could be evaluated to find the minimum. Alternatively, existing integer programming techniques could be applied to the equivalent linear problem L. However, neither of these alternatives is attractive. This section is devoted to a discussion of lower bounds on the quadratic cost function, as a basis for computational methods that are better suited to the special nature of the problem.

One way to calculate a lower bound on the cost function is to replace (4) by

$$x_{ij} \geqq 0 \qquad (i, j, = 1, 2, \cdots, n)$$

and to apply quadratic programming techniques to the resulting quadratic program. Since these techniques can be applied only to symmetric and positive semidefinite quadratic forms, the cost coefficients should be revised as follows.

For symmetry, replace c_{ijpq} by c'_{ijpq}, where

$$c'_{ijpq} = \tfrac{1}{2}(c_{ijpq} + c_{pqij}).$$

For positive semidefiniteness, replace all "diagonal" coefficients c_{ijij} by c'_{ijij}, where

$$c'_{ijij} = c_{ijij} + k \tag{14}$$

and k is some suitably large positive constant.

The effect of (14) is to add a positive constant nk to the cost function of the original quadratic assignment problem. However, for relatively small values of k, this constant may exceed the value of the minimum for the quadratic program, making the resulting bound negative. This is a very unsatisfactory result, particularly if all cost coefficients are nonnegative to begin with. Therefore, another type of bound will be developed.

Consider the equivalent integer program L described in the previous section. It is possible to arrange the n^4 variables y_{ijpq} into an $n^2 \times n^2$ matrix Y in such a way that for any feasible solution (X, Y), Y is a Kronecker second power of the $n \times n$ permutation matrix X. That is,

$$Y = X \times X = \begin{bmatrix} x_{11}X & x_{12}X & \cdots & x_{1n}X \\ x_{21}X & x_{22}X & \cdots & x_{2n}X \\ \vdots & & & \vdots \\ x_{n1}X & x_{n2}X & \cdots & x_{nn}X \end{bmatrix} \tag{15}$$

For example, if $X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$, then

$$Y = \left[\begin{array}{ccc:ccc:ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right]$$

In fact, an $n \times n$ quadratic assignment problem is exactly equivalent to an $n^2 \times n^2$ linear assignment problem, with the additional constraint:

(16) the $n^2 \times n^2$ solution matrix must be the Kronecker second power of an $n \times n$ permutation matrix.

There is an analogy with the traveling salesman problem. The n-city traveling

salesman problem is exactly equivalent to an $n \times n$ linear assignment problem, with the additional constraint [1]:

(17) the $n \times n$ solution matrix must be cyclic, i.e., represent a cyclic permutation of integers 1 to n.

The question in both cases is how to impose the additional constraint on the linear assignment problem. The measures that can be taken directly are limited in effect. In the case of the traveling salesman problem, subcycles of length one can be excluded by prohibiting nonzero elements on the main diagonal of the solution matrix. If the problem is symmetric, other measures can be imposed to exclude subcycles of length two [1].

In the case of the quadratic assignment problem, similar measures are possible. Suppose Y is partitioned into n^2 *minors* of n^2 elements each, as suggested by the form of the Kronecker product (15). If the matrix Y is to satisfy condition (16) each minor must contain either n 1's or no 1's at all. Each minor that contains n 1's must itself be a permutation matrix, but with one degree of freedom removed. This is due to the fact that the diagonal element y_{ijij} of minor (i, j) must be 1, if the minor contains any 1's at all. Such an element corresponds to a linear component of the cost function.

It is possible to impose the restrictions described above in the following way. For fixed i, j, define

$$C^{(ij)} = \| c_{ijpq} \|.$$

For each minor $C^{(ij)}$ solve an $(n-1) \times (n-1)$ linear assignment problem, or equivalently, an $n \times n$ linear assignment problem subject to the condition $x_{ij} = 1$. Denote the solution $X^{(ij)}$, and its cost

$$f_{ij} = C^{(ij)} \cdot X^{(ij)}.$$

Then solve an $n \times n$ linear assignment problem for the matrix $F = \| f_{ij} \|$. Denote the solution to this last problem $Z = \| z_{ij} \|$. Then if the $n^2 \times n^2$ permutation matrix

$$Y = \begin{bmatrix} z_{11} X^{(11)} & z_{12} X^{(12)} & \cdots & z_{1n} X^{(1n)} \\ z_{21} X^{(21)} & z_{22} X^{(22)} & \cdots & z_{2n} X^{(2n)} \\ \vdots & & & \vdots \\ z_{n1} X^{(n1)} & z_{n2} X^{(n2)} & \cdots & z_{nn} X^{(nn)} \end{bmatrix}$$

satisfies condition (16), a minimal solution to the quadratic assignment problem has been found. Condition (16) is satisfied if and only if

(18) $$\frac{1}{n} \sum_{ij} z_{ij} X^{(ij)}$$

is itself a permutation matrix.

In any case, a lower bound for the quadratic assignment problem is given by

$$\begin{aligned} C \cdot Y &= F \cdot Z \\ &= \textstyle\sum_{ij} z_{ij} C^{(ij)} \cdot X^{(ij)}. \end{aligned}$$

In the special case of a single-commodity Koopmans-Beckmann problem, a significant shortcut is possible.[1] In this case, each minor $C^{(ij)}$ is actually a Cartesian product of two n-vectors T^i and D^j (row i of T and row j of D). Finding a solution to the linear assignment problem for such a minor is equivalent to finding a permutation of the components of one of the vectors that minimizes the scalar product of T^i and D^j. It is not difficult to prove that the dot product of two vectors is minimized if and only if the k^{th} largest component of one vector is paired with the k^{th} smallest component of the other, for $k = 1, 2, \cdots, n$. Thus, the solution of the linear assignment problems for the n^2 minors is a trivial matter.

Example 1: Suppose it is desired to find a lower bound for the single commodity Koopmans-Beckmann problem, for which

$$D = \begin{bmatrix} 0 & 5 & 0 & 5 & 0 & 5 & 4 \\ 5 & 0 & 9 & 7 & 3 & 8 & 6 \\ 0 & 9 & 0 & 9 & 4 & 4 & 4 \\ 5 & 7 & 9 & 0 & 1 & 1 & 9 \\ 0 & 3 & 4 & 1 & 0 & 5 & 5 \\ 5 & 8 & 4 & 1 & 5 & 0 & 4 \\ 4 & 6 & 4 & 9 & 5 & 4 & 0 \end{bmatrix} \tag{19}$$

$$T = \begin{bmatrix} 0 & 0 & 6 & 1 & 1 & 8 & 4 \\ 0 & 0 & 1 & 0 & 3 & 1 & 3 \\ 6 & 1 & 0 & 8 & 8 & 4 & 2 \\ 1 & 0 & 8 & 0 & 7 & 6 & 4 \\ 1 & 3 & 8 & 7 & 0 & 0 & 6 \\ 8 & 1 & 4 & 6 & 0 & 0 & 9 \\ 4 & 3 & 2 & 4 & 6 & 9 & 0 \end{bmatrix} \tag{20}$$

In addition, let the problem have linear costs, represented by

$$B = \begin{bmatrix} 51 & 27 & 14 & 9 & 0 & 18 & 0 \\ 0 & 1 & 22 & 17 & 0 & 41 & 13 \\ 2 & 0 & 13 & 22 & 2 & 12 & 27 \\ 38 & 11 & 0 & 0 & 22 & 13 & 14 \\ 62 & 56 & 0 & 67 & 1 & 0 & 5 \\ 61 & 0 & 3 & 14 & 9 & 1 & 67 \\ 41 & 12 & 23 & 0 & 18 & 41 & 0 \end{bmatrix} \tag{21}$$

As noted in Section 2,

$$\begin{aligned} c_{ijpq} &= t_{iq}d_{jq} + b_{ij} \quad \text{if } i = p \text{ and } j = q, \\ &= t_{ip}d_{jq} \quad \text{otherwise.} \end{aligned}$$

Thus minor $C^{(ij)}$ is obtained by multiplying together elements the rows T^i and

[1] The independent work of Gilmore [6] on the single-commodity Koopmans-Beckmann problem is acknowledged. For this special case of the quadratic assignment problem, Gilmore's algorithm and the author's appear to be essentially the same.

D^j and adding b_{ij} to the "diagonal" coefficient. For example,

$$C^{(42)} = \begin{bmatrix} 5 & 0 & 9 & 7 & 3 & 8 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 40 & 0 & 72 & 56 & 24 & 64 & 48 \\ 0 & \textcircled{11} & 0 & 0 & 0 & 0 & 0 \\ 35 & 0 & 63 & 49 & 21 & 56 & 42 \\ 30 & 0 & 54 & 42 & 18 & 48 & 36 \\ 20 & 0 & 36 & 28 & 12 & 32 & 24 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 0 \\ 8 \\ 0 \\ 7 \\ 6 \\ 4 \end{bmatrix} = T^4, \tag{22}$$

$$D^2 = (\,5 \quad 0 \quad 9 \quad 7 \quad 3 \quad 8 \quad 6),$$

$$b_{42} = 11.$$

The diagonal coefficient, c_{4242}, is encircled.

The first step is to solve an $(n-1) \times (n-1)$ linear assignment problem for each of the n^2 minors. Thus, for $C^{(42)}$ there is a linear assignment problem with the cost matrix

$$\begin{bmatrix} 5 & 9 & 7 & 3 & 8 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 40 & 72 & 56 & 24 & 64 & 48 \\ 35 & 63 & 49 & 21 & 56 & 42 \\ 30 & 54 & 42 & 18 & 48 & 36 \\ 20 & 36 & 28 & 12 & 32 & 24 \end{bmatrix},$$

which is obtained from (22) by deleting the 4th row and 2nd column. As noted previously, this particular problem can be solved by permuting the elements of T^4 (with its 4th element removed) and D^2 (with its 2nd element removed) in such a way that their scalar product is minimized. Pairing the k^{th} largest element of one with the k^{th} smallest of the other gives:

$$131 = (8)(3) + (7)(5) + (6)(6) + (4)(7) + (1)(8) + (0)(9),$$

which is equivalent to the solution

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Thus,

$$X^{(42)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

and

$$f_{42} = C^{(42)} \cdot X^{(42)}$$
$$= 131 + c_{4242}$$
$$= 142.$$

The complete F matrix is

$$F = \begin{bmatrix} 77 & 120 & 67 & 59 & 27 & 76 & 83 \\ 9 & 38 & 42 & 35 & 10 & 65 & 46 \\ 61 & 153 & 112 & 123 & 59 & 114 & 156 \\ 87 & 142 & 68 & 82 & 68 & 98 & 124 \\ 106 & 180 & 73 & 143 & 43 & 80 & 110 \\ 110 & 139 & 84 & 98 & 56 & 91 & 185 \\ 102 & 136 & 124 & 108 & 77 & 141 & 132 \end{bmatrix}.$$

The next step is to solve the $n \times n$ linear assignment problem for which F is the cost matrix. A minimal solution Z is

$$Z = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

with a cost of 492.

Condition (16) is not satisfied by

$$Y = \| z_{ij} X^{(ij)} \|,$$

since

$$\frac{1}{n} \sum_{ij} z_{ij} X^{(ij)} = \frac{1}{7} \begin{bmatrix} 0 & 1 & 2 & 1 & 0 & 0 & 3 \\ 1 & 3 & 1 & 1 & 0 & 0 & 1 \\ 4 & 0 & 2 & 0 & 1 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 5 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 6 & 0 \\ 1 & 1 & 0 & 3 & 1 & 0 & 1 \end{bmatrix}$$

is not a permutation matrix. Hence a feasible solution has not been found. However, 492 is a valid lower bound on the cost function. It will be shown in Example 2 that the actual minimum is 559.

7. Method of Solution

The feasible solutions to the quadratic assignment problem can be partitioned into subsets. For given i, j, subproblems can be defined for the alternatives $x_{ij} = 0$

and $x_{ij} = 1$. It is also convenient to partition a problem into more than two subproblems. For instance, an $n \times n$ problem can be broken down into n problems, each of dimension $(n-1) \times (n-1)$, corresponding to $x_{i1} = 1$, $x_{i2} = 1, \cdots, x_{in} = 1$.

For each subproblem that is created as a result of partitioning, a lower bound can be calculated. If a feasible solution of still lower cost is known, it follows that no solution to the subproblem corresponds to a minimal solution for the original problem. Thus, the lower bound described in the previous section can be used to good effect to eliminate subsets of solutions from consideration. Partitioning is terminated when there is no subproblem remaining with a bound that is lower than the cost of the best known feasible solution[2].

The precise manner in which partitioning should be carried out is a topic for further investigation. It is probable that the type of partitioning performed at each step should be controlled by the matrix (18).

Regardless of the type of partitioning performed, it is reasonable to expect that the lower bounds calculated for the resulting subproblems will be somewhat sharper than the bound for the original problem. This is partly due to a reduction in degrees of freedom. For an $n \times n$ problem there are $n!\,[(n-1)!]^n$ permutation matrices Y upon which to base a bound, and only $n!$ of these represent feasible solutions. When the dimension of the problem is reduced to $(n-1) \times (n-1)$, the ratio between $(n-1)!\,[(n-2)!]^{n-1}$ and $(n-1)!$ is very much smaller than the ratio between $n!\,[(n-1)!]^n$ and $n!$.

Another reason that lower bounds tend to improve with partitioning is that quadratic costs are converted into linear costs. Consider an $n \times n$ problem with no linear costs, i.e., $c_{ijij} = 0$ for all i, j. Suppose n-way partitioning is performed by setting $x_{n1} = 1$, $x_{n2} = 1$, $\cdots$, $x_{nn} = 1$. Let $C[k]$ denote the cost matrix for the k^{th} of the $(n-1) \times (n-1)$ problems so obtained. This problem involves linear costs represented by the coefficients

$$c_{ijij}[k] = c_{ijnk} + c_{nkij} \quad (i = 1, 2, \cdots, n-1; j = 1, 2, \cdots, k-1, k+1, \cdots, n),$$

and quadratic costs represented by the coefficients

$$c_{ijpq}[k] = c_{ijpq} \quad (i, p = 1, 2, \cdots, n-1; j, q = 1, 2, \cdots, k-1, k+1, \cdots, n)$$

Next, suppose $(n-1)$-way partitioning is performed by setting $x_{n-1,1} = 1$, $x_{n-1,2} = 1$, $\cdots$, $x_{n-1,k+1} = 1$, $\cdots$, $x_{n-1,n} = 1$. Let $C[kl]$ denote the cost matrix for the l^{th} $(n-2) \times (n-2)$ problem so obtained. The cost function for the problem involves a constant,

$$c_{nk(n-1)l} + c_{(n-1)lnk},$$

[2] Similar methods of partitioning have been proposed by Eastman [1] for the solution of the traveling salesman problem and by Land and Doig [7] for the solution of integer linear programming problems.

It should be pointed out that partitioning methods can easily be modified to produce, with less computation, solutions that differ from the optimum by no more than a certain specified amount.

linear costs represented by the coefficients

$$c_{ijij}[kl] = c_{ijnk} + c_{nkij} + c_{ij(n-1)l} + c_{(n-1)lij} ,$$

and quadratic costs represented by the coefficients

$$c_{ijpq}[kl] = c_{ijpq} .$$

After p partitioning operations, the cost function of the resulting $(n - p) \times (n - p)$ problems involves a constant term that is the sum of $p^2 - p$ nonzero coefficients, linear costs represented by coefficients that are the sum of $2p$ of the original quadratic coefficients, and quadratic costs that are essentially unchanged (except in dimension) from the original problem.

Example 2: Consider the single-commodity Koopmans-Beckmann problem defined by (19), (20), (21) and for which a lower bound of 492 was calculated in Example 1. A minimal solution to this problem is found by performing n-way partitions as shown in Figure 1. An initial partition creates seven 6×6 subproblems corresponding to $x_{11} = 1$, $x_{21} = 1$, $\cdots$, $x_{71} = 1$, which have cost bounds of 583, 647, 529, 568, 570, 538, and 570, respectively. Four additional partitioning operations are required as indicated in the figure. The following is a tabulation of upper and lower bounds on the cost function after each successive partition. An upper bound is provided by a known feasible solution.

	Lower Bound	Upper Bound
Initial	492	$+\infty$
1	529	$+\infty$
2	532	$+\infty$
3	537	$+\infty$
4	538	559
5	559	559

The unique minimal solution to this problem is

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

with a cost of 559.

8. Cubic, Quartic, $\cdots$, N-adic Assignment Problems

It is not unreasonable to consider possible extensions to cubic, quartic, $\cdots$, even n-adic assignment problems. For example, the cubic assignment problem can be defined as follows: Given n^6 cost coefficients c_{ijpqrs} $(i, j, p, q, r, s =$

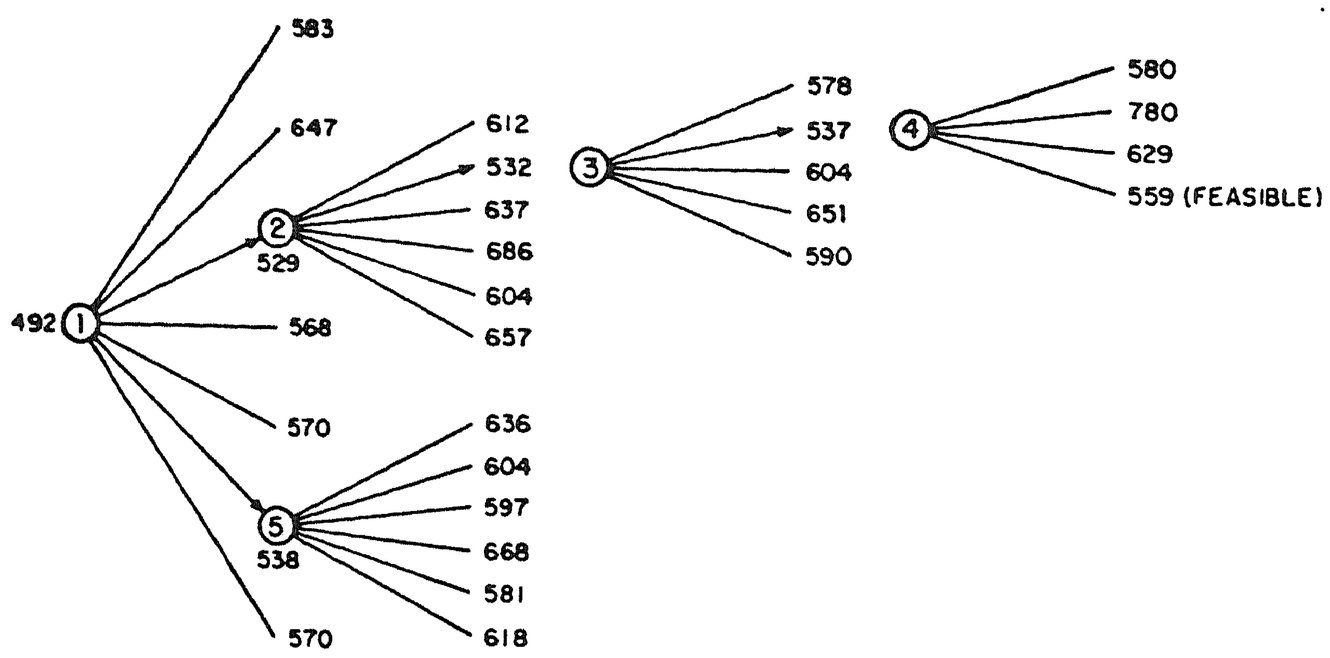


FIG. 1. Solution of example

$1, 2, \cdots, n$), determine an $n \times n$ solution matrix $X = \| x_{ij} \|$ so as to

$$\text{minimize} \quad \sum_{ij} \sum_{pq} \sum_{rs} c_{ijpqrs} x_{ij} x_{pq} x_{rs}$$

subject to constraints (2), (3), (4).

The cubic problem is equivalent to an $n^3 \times n^3$ linear assignment problem with the additional constraint that the $n^3 \times n^3$ solution matrix must be the Kronecker third power of an $n \times n$ permutation matrix. The method of calculating lower bounds, the test for feasible solutions (18) and the partitioning procedure are easily generalized from the quadratic case.

It should not be overlooked that a more straightforward enumerative approach may be effective for certain higher-order problems. Suppose an $n \times n$ assignment problem is defined, for which it is necessary to minimize $f(X) + g(X)$, where $f(X)$ is a low-order polynomial function and $g(X)$ is any well-defined function that is "less important" than $f(X)$. Suppose λ is a lower bound on $g(X)$ and μ is the value of some known feasible solution. One approach is to find the set S of solutions, where

$$S = \{X \mid f(X) + \lambda < \mu\}.$$

The set S can be enumerated in a straightforward way by means of partitioning, and $g(X)$ can be evaluated for each $X \in S$ in order to find the minimum.

References

1. EASTMAN, W. L., "Linear Programming with Pattern Constraints," Ph.D. thesis, *Bell Laboratories Report* BL-20, Harvard Computation Laboratory (1958).
2. FORD, L. R., JR. AND D. R. FULKERSON, "Solving the Transportation Problem," *Management Science*, Vol. 3 (1956), pp. 24–32.
3. GLASER, R. H., "A Quasi-simplex Method for Designing Sub-optimum Packages of Electronic Building Blocks (Burroughs 220)," General Electric Company, unpublished (1959).
4. GILMORE, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *J. Soc. Indust. Appl. Math.*, Vol. 10, (1962), pp. 305–313.

5. KODRES, U. R., "Geometrical Positioning of Circuit Elements in a Computer," Conference Paper 1172, AIEE Fall General Meeting (October 1959).
6. KOOPMANS, T. C., AND M. J. BECKMANN, "Assignment Problems and the Location of Economic Activities," *Econometrica*, Vol. 25 (1957), pp. 52–76.
7. LAND, A. H., AND A. G. DOIG, "An Automatic Method for Solving Discrete Programming Problems," *Econometrica*, Vol. 27 (1960), pp. 497–520.
8. LAWLER, E. L., "Notes on the Quadratic Assignment Problem," Harvard Computation Laboratory, unpublished (April 1960).
9. STEINBERG, L., "The Backboard Wiring Problem: A Placement Algorithm," *Soc. Indust. Appl. Math. Review*, Vol. 3 (1961), pp. 37–50.

An Approach to Multilevel Boolean Minimization

EUGENE L. LAWLER

The University of Michigan,† Ann Arbor, Michigan

Abstract. An approach to the problem of multilevel Boolean minimization is described. The conventional prime implicant is generalized to the multilevel case, and the properties of multilevel prime implicants are investigated. A systematic procedure for computing multilevel prime implicants is described, and several examples are worked out. It is shown how "absolutely minimal" forms can be obtained by carrying out multilevel minimization to a sufficiently large number of levels.

1. *Introduction*

Boolean forms can be classified by the number of levels of sums and products they contain; e.g. a sum of products or a product of sums has two levels, whereas a sum of products of sums has three. There are a number of well-known methods for finding minimal two-level forms for a given function. However, there has been comparatively little progress in the development of methods for finding minimal forms with more than two levels; some references to prior work are appended to this paper.

This paper describes an approach to the problem of finding minimal N-level forms and "absolutely minimal" forms. The techniques described are applicable to any and all Boolean functions, and are suitable for automatic computation. The approach generalizes two-level minimization procedures, and it is assumed that the reader is familiar with these methods.

2. *Definitions and Notation*

Attention is restricted to Boolean forms in which only individual letters are complemented.

Definition 1. (1) Expressions composed of single literals, e.g. "x," "$\bar{x}$," "y," and "$\bar{y}$," are the only *zero-level* forms. (2) If $\phi_1, \phi_2, \cdots, \phi_p$ $(p \geqq 1)$ are N-level forms, then $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_p$ is a *sum of N-level forms* and $(\phi_1) \wedge (\phi_2) \wedge \cdots \wedge (\phi_p)$ is a *product of N-level forms.* (3) Sums of N-level forms and products of N-level forms are the only $(N + 1)$*-level forms.*

For brevity, we refer to N-level forms as N*-forms*, and sums of N-level forms and products of N-levels forms as sN*-forms* and pN*-forms*. It is very important

This constitutes a major revision of the paper, "Minimal Boolean Expressions with More than Two Levels of Sums and Products," presented at the Third Annual Symposium on Switching Circuit Theory and Logical Design, AIEE Fall General Meeting, Chicago, October 1962. The revision was supported by Air Force Contract 33(657)-7811.

† Information Systems Laboratory, Department of Electrical Engineering.

Journal of the Association for Computing Machinery, Vol. 11, No. 3 (July, 1964), pp. 283–295

to note that, by definition, every N-form is an $(N+1)$-form, being both an sN-form and a pN-form.

Examples:

$p0$-forms: u, $u\bar{v}w$.

$s0$-forms: u, $u \vee w \vee \bar{z}$.

$p1$-forms: u, $u\bar{v}w$, $u \vee w \vee \bar{z}$, $u(v \vee \bar{w})$, $(u \vee \bar{v})(v \vee \bar{w})$.

$s2$-forms: u, $u\bar{v}w$, $u \vee w \vee \bar{z}$, $u(v \vee \bar{w})$, $(u \vee \bar{v})(v \vee \bar{w})$, $u \vee u\bar{v}w \vee (u \vee \bar{v})(v \vee \bar{w})$.

Boolean functions are denoted by the letters f, g, and h, and canonical terms of functions by their decimal equivalents. (The numbering of the canonical terms is such that $\bar{z}\bar{y}x$ is assigned the number 1.) The function having $i, j, \cdots, k$ as canonical terms is denoted by the letter f, g, or h with the superscripts $i, j, \cdots, k$, i.e. $f^{i,j,\cdots,k}$, $g^{i,j,\cdots,k}$, or $h^{i,j,\cdots,k}$.

Boolean functions are also defined by Boolean forms. For example, the function f defined by the form $c \vee \bar{y}$ is specified by writing

$$f(x, y) = x \vee \bar{y}.$$

To distinguish between forms and functions, the function defined by a given Boolean form ϕ is denoted by $|\phi|$. Thus,

$$f = |x \vee \bar{y}|.$$

(Occasionally we abuse this notation by omitting the absolute value sign if the meaning is obvious.)

An *incompletely specified* Boolean function, or *incomplete* function, is represented by an *interval* of functions, (f_0, f_1), where f_0 and f_1 are Boolean functions such that f_0 implies f_1, i.e. $f_0 \leqq f_1$. The function f_0 is obtained by setting all "don't cares" of the incomplete function to 0, and f_1 is obtained by setting all "don't cares" to 1. (An ordinary complete function f can be denoted by (f, f).) A function f satisfies the incomplete specification (f_0, f_1) if and only if it is contained in the interval, i.e. $f_0 \leqq f \leqq f_1$. Thus, a form ϕ satisfies the incomplete specification (f_0, f_1) if and only if $f_0 \leqq |\phi| \leqq f_1$.

An sN- or pN-form satisfying (f_0, f_1) is said to be a *minimal* sN- or pN-form if there is no other sN- or pN-form that satisfies (f_0, f_1) with fewer literals. It is said to be *absolutely minimal* if there is no other form with fewer literals, regardless of the number of levels of sums and products it may contain.

3. *Generalization of Prime Implicants*

A pN-form ϕ is called a *pN-implicant* of f if and only if $|\phi|$ implies f. By accepted definition, a $p0$-implicant that does not imply any other $p0$-implicant is a *prime implicant*. This is equivalent to defining a $p0$-form ϕ to be a prime implicant if and only if ϕ is a minimal $p0$-form for $(|\phi|, f)$. Hence, we more generally define the "primeness" of pN-implicants as follows.

Definition 2. A form ϕ is a *prime pN-implicant* of the function f if and only if ϕ is a minimal pN-form for $(|\phi|, f)$.

It is well known that every minimal "sum of products," i.e. every minimal $s1$-form, is a sum of prime implicants. This result is generalized in the following theorem. (The reader should convince himself that prime $p0$-implicants are indeed the same as conventional prime implicants.)

THEOREM 1. *Every minimal sN-form for (f_0, f_1) is a sum of prime $p(N-1)$-implicants of f_1.*

PROOF. Every minimal sN-form for (f_0, f_1) is of the form $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_p$, where each ϕ_i is a $p(N-1)$-form. Suppose there is a ϕ_i that is not a prime $p(N-1)$-implicant of f_1. Then there is a $p(N-1)$-form for $(|\phi_i|, f_1)$ that contains fewer literals than ϕ_i. Call this form ξ. Then

$$\phi' = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_{i-1} \vee \xi \vee \phi_{i+1} \vee \cdots \vee \phi_p$$

is a valid N-form for (f_0, f_1), and has fewer literals than ϕ. (Note that $|\phi| \leqq |\phi'| \leqq f_1$.) But this contradicts the assumption that ϕ is minimal. Hence each ϕ_i is a prime $p(N-1)$-implicant of f_1. Q.E.D.

It is shown below that it is not necessary to obtain all the prime $p(N-1)$-implicants of f_1 to be able to find some or all of the minimal sN-forms for (f_0, f_1).

Definition 3. Let P be a set of prime pN-implicants of f_1. (1) P is a *sufficient* set of prime pN-implicants for (f_0, f) if and only if for every $h \leqq f_0$, all minimal pN-forms for (h, f_1) are contained in P. (2) P is a *necessary* set of prime pN-implicants for (f_0, f_1) if and only if for every $h \leqq f_0$, at least one minimal pN-form for (h, f_1) is contained in P (if such a form exists).

The parenthetical qualification is necessitated by the fact that there are functions for which no $p0$-form exists.

THEOREM 2. *Let P be a sufficient set of prime $p(N-1)$-implicants for (f_0, f_1). Then every minimal sN-form for (f_0, f_1) is a sum of forms contained in P.*

PROOF. By Theorem 1, every minimal sN-form for (f_0, f_1) is of the form $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_p$, where each ϕ_i is a minimal $p(N-1)$-form for $(|\phi_i|, f_1)$. Suppose there is a ϕ_i that is not also a minimal $p(N-1)$-form for $(f_0 \wedge |\phi_i|, f_1)$. Then there is a $p(N-1)$-form for $(f_0 \wedge |\phi_i|, f_1)$ that contains fewer literals than ϕ_i. Call this form ξ. Then

$$\phi' = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_{i-1} \vee \xi \vee \phi_{i+1} \vee \cdots \vee \phi_p$$

is a valid sN-form for (f_0, f_1) and has fewer literals than ϕ. But this contradicts the assumption that ϕ is minimal. Hence, each ϕ_i is a minimal $p(N-1)$-form for $(f_0 \wedge |\phi_i|, f_1)$. But because $f_0 \wedge |\phi_i| \leqq f_0$, P contains all minimal $p(N-1)$-forms for $(f_0 \wedge |\phi_i|, f_1)$. Therefore each ϕ_i must be contained in P. Q.E.D.

THEOREM 3. *Let P be a necessary set of prime $p(N-1)$-implicants for (f_0, f_1). Then there exists at least one minimal sN-form for (f_0, f_1) that is a sum of forms contained in P.*

PROOF. By Theorem 2, every minimal sN-form for (f_0, f_1) is of the form $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_p$, where each ϕ_i is a minimal $p(N-1)$-form for $(f_0 \wedge |\phi_i|, f_1)$. Suppose there is a ϕ_i that is not contained in P. Then there is a minimal $p(N-1)$-form for $(f_0 \wedge |\phi_i|, f_1)$ that *is* contained in P. Call this form ξ. Then

$$\phi' = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_{i-1} \vee \xi \vee \phi_{i+1} \vee \cdots \vee \phi_p$$

is also a minimal sN-form for (f_0, f_1). Each ϕ_i that is not contained in P can be replaced in this way to obtain a minimal sN-form for (f_0, f_1) that is a sum of forms contained in P. Q.E.D.

The following are duals of the above definitions and theorems.

Definition 2′. A form ϕ is a *prime sN-implicant* of the function f if and only if ϕ is a minimal sN-form for $(f, \mid \phi \mid)$.

THEOREM 1′. *Every minimal pN-form for (f_0, f_1) is a product of prime $s(N-1)$-implicants of f_0.*

Definition 3′. Let P be a set of prime sN-implicants of f_0. (1) P is a *sufficient* set of prime sN-implicants for (f_0, f_1) if and only if for every $h \geqq f_1$, all minimal sN-forms for (f_0, h) are contained in P. (2) P is a *necessary* set of prime sN-implicants for (f_0, f_1) if and only if for every $h \geqq f_1$, at least one minimal sN-form for (f_0, h) is contained in P (if such a form exists).

THEOREM 2′. *Let P be a sufficient set of prime $s(N-1)$ implicants for (f_0, f_1). Then every minimal pN-form for (f_0, f_1) is a product of forms contained in P.*

THEOREM 3′. *Let P be a necessary set of prime $s(N-1)$-implicants for (f_0, f_1). Then there exists at least one minimal pN-form for (f_0, f_1) that is a product of forms contained in P.*

4. *Computing Necessary and Sufficient Sets of Prime Implicants*

The proposed approach to minimization is recursive: 2-level minimization is the basis for 3-level minimization, 3-level minimization is the basis for 4-level minimization, and so forth. It is apparent that this recursive approach will be impracticable if it is necessary to carry out pN-minimization for all possible (h, f_1) to compute either a necessary or a sufficient set of prime pN-implicants for (f_0, f_1). To avoid this difficulty, we shall make use of the following lemmas, which follow immediately from the definitions.

LEMMA 1. *If ϕ is a minimal sN- or pN-form for (h, f), then ϕ is a minimal sN- or pN-form for each and every (h^*, f), where $h \leqq h^* \leqq \mid \phi \mid$.*

LEMMA 2. *Let $\Phi = \{\phi_1, \phi_2, \cdots, \phi_k\}$ be the set of all minimal sN- or pN-forms for (h, f). Then Φ contains all minimal sN- or pN-forms for each and every (h^*, f), where $h \leqq h^* \leqq \mid \phi_i \mid$ for any $\phi_i \in \Phi$.*

It is not necessary to carry out pN-minimization for all intervals of the form (h, f_1). Lemma 1 enables us to skip many of these intervals when compiling necessary sets of prime implicants, and Lemma 2 makes it possible to do the same thing for sufficient sets.

Let the canonical terms of f_0 be $i_1, i_2, \cdots, i_m$. The first step in the computation of prime pN-implicants is to find minimal pN-forms for (h_1, f_1), (h_2, f_1), $\cdots$, (h_m, f_1), where $h_j = h^{i_j}$. To compute a sufficient set of prime implicants, we must find all minimal pN-forms for each (h_j, f_1); a necessary set requires only one minimal pN-form for each (h_j, f_1). In either case, let the forms obtained for (h_j, f_1) be denoted $\phi_{j1}, \phi_{j2}, \cdots$, etc. We then find the smallest functions

implying f_0 that are not contained in any of the intervals $(h^1, \mid \phi_{11} \mid)$, $(h^1, \mid \phi_{12} \mid)$, $\cdots$, $(h^2, \mid \phi_{21} \mid)$, $(h^2, \mid \phi_{22} \mid)$, $\cdots$, $(h^m, \mid \phi_{m1} \mid)$, $(h^m, \mid \phi_{m2} \mid)$, $\cdots$, etc. Call these smallest functions $h_{m+1}, h_{m+2}, \cdots, h_n$. The next step is to find minimal pN-forms for (h_{m+1}, f_1), (h_{m+2}, f_1), $\cdots$, (h_n, f_1). We then find the smallest functions implying f_0 that are not contained in any of the intervals $(h^1, \mid \phi_{11} \mid)$, $(h^1, \mid \phi_{12} \mid)$, $\cdots$, $(h^n, \mid \phi_{n1} \mid)$, $(h^n, \mid \phi_{n2} \mid)$, $\cdots$, etc. The process is repeated until there are no smallest functions remaining after a given step.

This approach would certainly be of doubtful value if the determination of smallest functions required an exhaustive examination of the 2^m functions implying f_0. The algorithm below avoids this difficulty (and is an improvement over the procedure described in the original version of this paper).

Algorithm

(0) Let $H = \{h\}$ contain the functions found to be smallest at the previous step of the prime implicant computational procedure.

(1) If $h \in H$ is contained in an interval $(h_j, \mid \phi_{jk} \mid)$, replace h by $h \bigvee h^{i_1}$ $h \bigvee h^{i_2}$, $\cdots$, $h \bigvee h^{i_p}$, where $i_1, i_2, \cdots, i_p$ are the canonical terms of $f_0 \bigwedge \mid \overline{\phi_{jk}} \mid$.

(2) If $h \leqq h'$, for an $h' \in H$, delete h from H.

(3) Repeat steps (1) and (2) until neither step can any longer be carried out. The functions remaining in H are the smallest functions implying f_0 that are not contained in any of the intervals $(h_j, \mid \phi_{jk} \mid)$.

Example 1. Find all minimal s2-forms (sums of products of sums) for the function

$$f^{0,1,2,7}(x, y, z) = \bar{x}\bar{y} \bigvee \bar{y}\bar{z} \bigvee xyz,$$

which is shown in its unique minimal sum-of-products form.

Solution. We must compute a sufficient set of prime $p1$-implicants for (f, f). The first step is to find all minimal $p1$-forms for (h^0, f), (h^1, f), (h^2, f), and (h^7, f). These forms are all conventional prime implicants and are listed below with the canonical terms that they cover indicated within parentheses. (Recall that canonical term $\bar{z}\bar{x}y$ is numbered 1.)

(h^0, f):	$\bar{x}\bar{z}$	(0, 2)
:	$\bar{y}\bar{z}$	(0, 1)
(h^1, f):	$\bar{y}\bar{z}$	(0, 1)
(h^2, f):	$\bar{x}\bar{z}$	(0, 2)
(h^7, f):	xyz	(7)

At this stage,

$$H = \{h^0, h^1, h^2, h^7\},$$

and the intervals $(h_j, \mid \phi_{jk} \mid)$ are

$$(h^0, h^{0,2}), \quad (h^0, h^{0,1}), \quad (h^1, h^{0,1}), \quad (h^2, h^{0,2}), \quad (h^7, h^7).$$

The algorithm for finding smallest functions proceeds as follows. The function h^0 is contained in $(h^0, h^{0,2})$; so replace h^0 by $h^{0,1}$, $h^{0,7}$. But $h^1 \leqq h^{0,1}$, and $h^7 \leqq h^{0,7}$; hence eliminate $h^{0,1}$ and $h^{0,7}$. At this point

$$H = \{h^1, h^2, h^7\}.$$

Next, h^1 is contained in $(h^1, h^{0,1})$; so replace h^1 by $h^{1,2}$ and $h^{1,7}$. But these two functions can be eliminated because $h^2 \leqq h^{1,2}$ and $h^7 \leqq h^{1,7}$, giving the result

$$H = \{h^2, h^7\}.$$

Next, replace h^2 and h^7 by $h^{1,2}$, $h^{2,7}$ and $h^{0,7}$, $h^{1,7}$, and $h^{2,7}$, respectively. After removing the duplicated function, obtain the result

$$H = \{h^{0,7}, h^{1,2}, h^{1,7}, h^{2,7}\}.$$

Next, obtain the prime $p1$-implicants:

$(h^{0,7}, f)$:	$(x \vee \bar{y})(\bar{y} \vee z)(y \vee \bar{z})$	(0, 1, 7)
	$(x \vee \bar{z})(\bar{y} \vee z)(y \vee \bar{z})$	(0, 1, 7)
	$(\bar{x} \vee y)(\bar{x} \vee z)(x \vee \bar{z})$	(0, 2, 7)
	$(\bar{x} \vee z)(x \vee \bar{z})(y \vee \bar{z})$	(0, 2, 7)
	$(x \vee \bar{y})(\bar{x} \vee z)(y \vee \bar{z})$	(0, 7)
	$(\bar{x} \vee y)(x \vee \bar{z})(\bar{y} \vee z)$	(0, 7)
$(h^{1,2}, f)$:	$\bar{z}(\bar{x} \vee \bar{y})$	(0, 1, 2)
$(h^{1,7}, f)$:	$x(\bar{y} \vee z)(y \vee \bar{z})$	(1, 7)
$(h^{2,7}, f)$:	$y(x \vee \bar{z})(\bar{x} \vee z)$	(2, 7)

The smallest remaining function is now $h^{1,2,7}$, and we obtain the prime $p1$-implicant,

$$(h^{1,2,7}, f): \quad (x \vee \bar{z})(y \vee \bar{z})(\bar{x} \vee \bar{y} \vee z) \qquad (0, 1, 2, 7)$$

This expression is the unique minimal product-of-sums form for f itself, and terminates the prime implicant computations.

Now that a sufficient set of prime $p1$-implicants has been obtained, we solve a covering problem to find all minimal $s2$-forms. First, set up the incidence matrix shown below, in which each row is identified with a prime implicant and each column with a canonical term of f.

	0	1	2	7
$\bar{x}\bar{z}$	1	0	1	0
$\bar{y}\bar{z}$	1	1	0	0
xyz	0	0	0	1
$(x \vee \bar{y})(\bar{y} \vee z)(y \vee \bar{z})$	1	1	0	1
$(x \vee \bar{z})(\bar{y} \vee z)(y \vee \bar{z})$	1	1	0	1
$(\bar{x} \vee y)(\bar{x} \vee z)(x \vee \bar{z})$	1	0	1	1
$(\bar{x} \vee z)(x \vee \bar{z})(y \vee \bar{z})$	1	0	1	1
$(x \vee \bar{y})(\bar{x} \vee z)(y \vee \bar{z})$	1	0	0	1
$(\bar{x} \vee y)(x \vee \bar{z})(\bar{y} \vee z)$	1	0	0	1
$\bar{z}(\bar{x} \vee \bar{y})$	1	1	1	0
$x(\bar{y} \vee z)(y \vee \bar{z})$	0	1	0	1
$y(x \vee \bar{z})(\bar{x} \vee z)$	0	0	1	1
$(x \vee \bar{z})(y \vee \bar{z})(\bar{x} \vee \bar{y} \vee z)$	1	1	1	1

The covering problem can be solved by any of the methods that are used for selecting a minimal sum of prime implicants in conventional two-level minimization. The unique minimal covering gives the form

$$f^{0,1,2,7} = xyz \vee \bar{z}(\bar{x} \vee \bar{y}),$$

which contains only 6 literals compared with 7 literals in the minimal sum-of-products and product-of-sums forms.

5. *Further Considerations*

The amount of computation required to obtain a set of prime implicants for (g, f_1) is likely to be small if g is small. We would like to find a function $g \leqq f_0$, where g is as small as possible, such that some minimal sN-form for (g, f_1) also satisfies (f_0, f_1). For in this case, Lemma 1 tells us that such a form is also a minimal sN-form for (f_0, f_1). And from Lemma 2 it follows that a sufficient set of prime $p(N-1)$-implicants for (g, f_1) is sufficient to find *all* minimal sN-forms for (f_0, f_1).

The following approach suggests itself. First carry out the minimization procedure for (g^i, f_1), where i is any canonical term of f_0. (The forms obtained are all conventional prime implicants of f_1.) If none of the minimal forms for (g^i, f_1) satisfies (f_0, f_1), find a subset of the canonical terms of f_0 that is not covered by any single form. (For computational efficiency, this subset should be as small as possible, preferably a single canonical term.) Suppose such a subset contains $j, k, \cdots, l$. Then the next step is to carry out the minimization procedure for $(g^{i,j,k,\cdots,l}, f_1)$. The process is repeated until some (g, f_1) yields a minimal form satisfying (f_0, f_1). Hopefully, this will occur for some $g < f_0$.

It is very important to note that the prime implicants required for each step of the procedure are a subset of the prime implicants required for the following step. This point is illustrated in the following examples.

Example 2. Rework Example 1 with the procedure just described.

Solution. With the exercise of foresight, begin the computation with (g^1, f), rather than (g^0, f). For (g^1, f), the unique minimal form is $\bar{y}\bar{z}$. Since $\bar{y}\bar{z}$ covers canonical terms 0 and 1, but not 2, the second step is to find all minimal $s2$-forms for $(g^{1,2}, f)$. For this purpose, the following table of prime $p1$-implicants is compiled.

$$\begin{array}{lll} (h^1, f): & \bar{y}\bar{z} & (0, 1) \\ (h^2, f): & \bar{x}\bar{z} & (0, 2) \\ (h^{1,2}, f): & \bar{z}(\bar{x} \vee \bar{y}) & (0, 1, 2). \end{array}$$

A trivial covering problem is solved to find the unique minimal $s2$-form for $(g^{1,2}, f)$, which is $\bar{z}(\bar{x} \vee \bar{y})$. This expression does not cover canonical term 7. Thus, for the third step, find all minimal $s2$-forms for $(g^{1,2,7}, f)$. This requires the compilation of the following table of prime implicants, the first three entries of which are carried over from above.

$$\begin{array}{lll} (h^1, f): & \bar{y}\bar{z} & (0, 1) \\ (h^2, f): & \bar{x}\bar{z} & (0, 2) \\ (h^{1,2}, f): & \bar{z}(\bar{x} \vee \bar{y}) & (0, 1, 2) \\ (h^7, f): & xyz & (7) \\ (h^{1,7}, f): & x(\bar{y} \vee z)\ (y \vee \bar{z}) & (1, 7) \\ (h^{2,7}, f): & y(\bar{x} \vee z)\ (x \vee \bar{z}) & (2, 7) \\ (h^{1,2,7}, f): & (x \vee \bar{z})\ (y \vee \bar{z})\ (\bar{x} \vee \bar{y} \vee z) & (0, 1, 2, 7) \end{array}$$

The minimal forms for $(g^{1,2,7}, f)$ are found by solving the following covering problem.

	1	2	7
$\bar{y}\bar{z}$	1	0	0
$\bar{x}\bar{z}$	0	1	0
xyz	0	0	1
$\bar{z}(\bar{x} \vee \bar{y})$	1	1	0
$x(\bar{y} \vee z)\ (y \vee \bar{z})$	1	0	1
$y(x \vee \bar{z})\ (\bar{x} \vee z)$	0	1	1
$(x \vee \bar{z})\ (y \vee \bar{z})\ (\bar{x} \vee \bar{y} \vee z)$	1	1	1

The unique minimal $s2$-form for $(g^{1,2,7}, f)$ is $xyz \vee \bar{z}(\bar{x} \vee \bar{y})$, which represents f and is therefore the expression desired.

Example 3. Find all minimal $s2$-forms for the function

$$f^{3,5,6,7,11,13,15}(w, x, y, z) = wx \vee wy \vee xy\bar{z} = (x \vee y)\ (w \vee x)\ (w \vee y)\ (w \vee \bar{z}),$$

which is shown in its unique minimal sum-of-products and product-of-sums forms.

Solution. The computation proceeds through the determination of minimal forms for (g^3, f), $(g^{3,5}, f)$, and $(g^{3,5,6}, f)$. In Table I, lines are drawn between sets of prime $p1$-implicants that are compiled for successive (g, f).

TABLE I. PRIME IMPLICANTS FOR EXAMPLE 3

(h^3, f):	wx	(3, 7, 11, 15)
(h^5, f):	wy	(5, 7, 13, 15)
$(h^{3,5}, f)$:	$w(x \vee y)$	(3, 5, 7, 11, 13, 15)
(h^6, f):	$xy\bar{z}$	(6, 7)
$(h^{3,6}, f)$:	$x\bar{z}(w \vee y)$	(3, 6, 7)
$(h^{5,6}, f)$:	$y\bar{z}(x \vee w)$	(5, 6, 7)
$(h^{3,5,6}, f)$:	$\bar{z}(w \vee x)(w \vee y)(x \vee y)$	(3, 5, 6, 7)

Solution of the final covering problem yields three minimal $s2$-forms for $(g^{3,5,6}, f)$

$$wx \vee y\bar{z}(w \vee x) \qquad wy \vee x\bar{z}(w \vee y) \qquad xy\bar{z} \vee w(x \vee y).$$

The first two expressions do not represent the function f, but the third does. Hence

$$f = xy\bar{z} \vee w(x \vee y)$$

is the unique minimal $s2$-form. It contains 6 literals, compared with 7 and 8 literals for the minimal sum-of-products and product-of-sums.

Example 4. Find a minimal $s2$-form for the 5-variable function

$$f^{0,1,2,4,8,9,10,12,24,25,28} = \bar{u}\bar{v}x \vee \bar{u}\bar{v}\bar{y} \vee \bar{u}\bar{w}\bar{y} \vee \bar{v}\bar{w}\bar{y} \vee \bar{v}\bar{w}x$$
$$= (\bar{u} \vee \bar{v})\ (\bar{u} \vee \bar{w})\ (\bar{v} \vee \bar{w})\ (\bar{v} \vee \bar{y})\ (x \vee \bar{y})$$

which is shown in its minimal sum-of-products and product-of-sums forms.

Solution. We must compute a necessary set of prime $p1$-implicants for each (g, f_1) that we specify. Table II displays the prime $p1$-implicants that are used to obtain the minimal $s2$-form $f = \bar{v}(\bar{w} \vee \bar{u})\ (x \vee \bar{y}) \vee \bar{u}\bar{w}\bar{y}$.

As in the previous example, lines are drawn between the sets of prime implicants that are computed for (g^1, f), $(g^{1,2}, f)$, $(g^{1,2,4}, f)$, $(g^{1,2,4,24}, f)$, and $(g^{1,2,4,24,28}, f)$.

TABLE II. PRIME IMPLICANTS FOR EXAMPLE 4

(h^1, f):	$\bar{v}\bar{w}\bar{y}$	(0, 1, 8, 9)
(h^2, f):	$\bar{u}\bar{w}\bar{y}$	(0, 2, 8, 10)
$(h^{1,2}, f)$:	$\bar{w}\bar{y}(\bar{u} \vee \bar{v})$	(0, 1, 2, 8, 9, 10)
(h^4, f):	$\bar{u}\bar{v}\bar{y}$	(0, 4, 8, 12)
$(h^{1,4}, f)$:	$\bar{v}\bar{y}(\bar{u} \vee \bar{w})$	(0, 1, 4, 8, 9, 12)
$(h^{2,4}, f)$:	$\bar{u}\bar{y}(\bar{v} \vee \bar{w})$	(0, 2, 4, 8, 10, 12)
$(h^{1,2,4}, f)$:	$\bar{x}\bar{y}(\bar{u} \vee \bar{v})(\bar{u} \vee \bar{w})(\bar{v} \vee \bar{w})$	(0, 1, 2, 4, 8)
(h^{24}, f):	$\bar{u}\bar{v}x$	(8, 12, 24, 28)
$(h^{1,24}, f)$:	$\bar{v}\bar{w}(x \vee \bar{y})$	(0, 1, 8, 9, 24, 25)
$(h^{2,24}, f)$:	$\bar{u}\bar{w}(\bar{v} \vee \bar{y})(v \vee x)$	(2, 8, 10, 24)
$(h^{1,2,24}, f)$:	$\bar{w}(\bar{u} \vee \bar{v})(\bar{v} \vee \bar{x})(x \vee \bar{y})$	(0, 1, 2, 8, 9, 10, 24, 25)
$(h^{4,24}, f)$:	$\bar{u}\bar{v}(x \vee \bar{y})$	(0, 4, 8, 12, 24, 28)
$(h^{1,4,24}, f)$:	$\bar{v}(\bar{u} \vee \bar{w})(x \vee \bar{y})$	(0, 1, 4, 8, 9, 12, 24, 25, 28)
$(h^{2,4,24}, f)$:	$\bar{u}(\bar{v} \vee \bar{w})(\bar{v} \vee \bar{y})(x \vee \bar{y})$	(0, 2, 4, 8, 10, 12, 24, 28)
$(h^{1,2,4,24}, f)$:	$(\bar{u} \vee \bar{v})(\bar{u} \vee \bar{w})(\bar{v} \vee \bar{w})(\bar{v} \vee \bar{y})(x \vee \bar{y})$	(0, 1, 2, 4, 8, 9, 10, 12, 24, 25, 28)
(h^{28}, f):	$\bar{u}\bar{v}x$	(8, 12, 24, 28)
$(h^{1,28}, f)$.	$\bar{v}(\bar{u} \vee \bar{w})(x \vee \bar{y})$	(0, 1, 4, 8, 9, 12, 24, 25, 28)
$(h^{2,28}, f)$:	$\bar{u}(\bar{v} \vee \bar{w})(\bar{v} \vee \bar{y})(x \vee \bar{y})$	(0, 2, 4, 8, 10, 12, 24, 28)
$(h^{1,2,28}, f)$:	$(\bar{u} \vee \bar{v})(\bar{u} \vee \bar{w})(\bar{v} \vee \bar{w})(\bar{v} \vee \bar{y})(x \vee \bar{y})$	(0, 1, 2, 4, 8, 9, 10, 12, 24, 25, 28)
$(h^{4,28}, f)$:	$\bar{u}\bar{v}(x \vee \bar{y})$	(0, 4, 8, 12, 24, 28)

Example 5. Find all minimal s3-forms for the incompletely specified 4-variable function (f_0, f_1), where $f_0 = f^{3,5,6}$, $f_1 = f^{0,3,5,6,7,8,9,10,12,13,15}$.

Solution. Compile a complete set of prime $p2$-implicants (Table III). Four nontrivial 3-level minimizations are required to compile Table III. The lengthiest 3-level minimization is that for $(h^{3,5,6}, f_1)$, which in turn requires 11 nontrivial 2-level minimizations.

TABLE III. PRIME IMPLICANTS FOR EXAMPLE 5

(h^3, f_1):	wx	(3, 7, 11, 15)
(h^5, f_1):	wy	(5, 7, 13, 15)
$(h^{3,5}, f_1)$:	$w(x \vee y)$	(3, 5, 7, 11, 13, 15)
(h^6, f_1):	$xy\bar{z}$	(6, 7)
$(h^{3,6}, f_1)$:	$x(w \vee y\bar{z})$	(3, 6, 7, 11, 15)
	$x\bar{z}(w \vee y)$	(3, 6, 7)
$(h^{5,6}, f_1)$:	$y(w \vee x\bar{z})$	(5, 6, 7, 13, 15)
	$y\bar{z}(w \vee x)$	(5, 6, 7)
$(h^{3,5,6}, f_1)$:	$(y \vee wx)(w \vee x\bar{z})$	(3, 5, 6, 7, 11, 13, 15)
	$(x \vee wy)(w \vee y\bar{z})$	(3, 5, 6, 7, 11, 13, 15)
	$(w \vee x)(wx \vee y\bar{z})$	(3, 5, 6, 7, 11, 15)
	$(w \vee y)(wy \vee x\bar{z})$	(3, 5, 6, 7, 13, 15).

The solution of the covering problem yields nine minimal $s3$-forms for (f_0, f_1):

$$xy\bar{z} \vee w(x \vee y) \qquad (y \vee wx)\ (w \vee x\bar{z})$$
$$wx \vee y\bar{z}(w \vee x) \qquad (x \vee wy)\ (w \vee y\bar{z})$$
$$wx \vee y(w \vee x\bar{z}) \qquad (w \vee x)\ (y\bar{z} \vee wx)$$
$$wy \vee x(w \vee y\bar{z}) \qquad (w \vee y)\ (x\bar{z} \vee wy).$$
$$wy \vee x\bar{z}(w \vee y)$$

6. *Absolutely Minimal Forms*

There are various conditions sufficient to insure that a given form is absolutely minimal. Two sets of such conditions are given in Theorems 4 and 5 below, for which the reader may supply proofs.

THEOREM 4. *Let ϕ be an s0-form in which no letter appears more than once, e.g. $\phi = w \vee \bar{y} \vee z$, but not $\phi = w \vee \bar{w} \vee x$, and let f be any function such that $f \leqq |\phi|$. Then ϕ is the unique absolutely minimal form for $(f, |\phi|)$.*

We may say that a form is a *proper* sN- or pN-form if and only if it is an sN- or pN-form, but is not itself an N-form. Every form other than a 0-form is either a proper sN-form or a proper pN-form, but not both.

LEMMA 3. *Every proper sN- or pN-form contains at least $N+2$ literals.*

THEOREM 5. *Let ϕ be a form that contains no more than $N+2$ literals. Then if ϕ is a minimal sN- or pN- or N-form for (f_0, f_1), it is an absolutely minimal form.*

Theorem 5 suggests one way in which absolutely minimal forms can, at least in principle, be obtained. One can first carry out 2-level minimization, then 3-level minimization, 4-level minimization, etc. Suppose, after any given step of the process, the most economical known form contains M literals. Then it follows that every minimal $(M-3)$-form is an absolute minimal form and vice-versa. Hence, the process need be carried no further than to $(M-3)$-level minimization.

There are other conditions that can be invoked to terminate the process described above. One such set of conditions is given in Theorem 6 below, in preparation for which we introduce some definitions and lemmas.

If ϕ is a proper sN-form, then $\phi = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_p$, where each ϕ_i is either a 0-form or a proper pM-form with $0 \leqq M \leqq N-1$. Likewise, if ϕ is a proper pN-form, then $\phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_p$, where each ϕ_i is either a 0-form or a proper sM-form with $0 \leqq M \leqq N-1$. In either case, each ϕ_i is said to be a *subform* of ϕ. Each subform of a subform of ϕ, and each subform of a subform of $\cdots$ a subform of ϕ is also a subform of ϕ.

We *eliminate* subforms from forms in the obvious way. Thus, if ϕ_{ij} is a subform of ϕ_i, and ϕ_i is a subform of ϕ, ϕ_{ij} is eliminated from ϕ by replacing ϕ by

$$\phi' = \phi_1 \vee \phi_2 \vee \cdots \vee \phi_{i-1} \vee \phi_i' \vee \phi_{i+1} \vee \cdots \vee \phi_p,$$

where

$$\phi_i' = \phi_{i1} \wedge \phi_{i2} \wedge \cdots \wedge \phi_{i(j-1)} \wedge \phi_{i(j+1)} \wedge \cdots \wedge \phi_{iq}.$$

We say that ϕ is a *redundant* form for (f_0, f_1) if there exists a subform of ϕ

that can be eliminated to yield a form ϕ' which also satisfies (f_0, f_1). A form that is not redundant is *irredundant*.

Clearly, every absolutely minimal form is an irredundant form. Hence, if we can be sure that every irredundant form for (f_0, f_1) is an N-level form, we can be sure that every absolutely minimal form for (f_0, f_1) is also an N-level form.

LEMMA 4. *Let m_0 be the number of canonical terms of f_0. Then every irredundant form for (f_0, f_1) is either*

(1) *a 0-form, or*

(2) *a proper sN-form, with $N \leqq 2m_0 - 3$, or*

(3) *a proper pN-form, with $N \leqq 2m_0 - 2$.*

PROOF. We shall construct functions S and P of the parameter m_0 such that $N \leqq S(m_0)$ and $N \leqq P(m_0)$ for parts (2) and (3) of the theorem.

For $m_0 = 1$, the only possible type of irredundant form is a single prime $p0$-implicant of f_1. Since a prime $p0$-implicant is either a 0-form or a proper $p0$-form, we may set

$$P(1) = 0.$$

For $m_0 = 2$, the only possible type of irredundant proper sN-form is a sum of exactly two prime $p0$-implicants of f_1. Thus,

$$S(2) = 1.$$

The only irredundant proper pN-forms are products of sN-forms, $\phi_1, \phi_2, \cdots, \phi_p$, which are themselves irredundant forms for $(f_0, |\phi_1|), (f_0, |\phi_2|), \cdots, (f_0, |\phi_p|)$. Thus, these forms are products of sN-forms, where

$$N \leqq S(2) = 1,$$

and we thereby set

$$P(2) = 1 + S(2) = 2.$$

Now, for $m_0 = k+1$, the only possible type of irredundant proper sN-form is a sum of no more than $k + 1$ proper pN-forms, each of which covers no more than k canonical terms of f_0. Thus, we set

$$S(k + 1) = 1 + \max_{1 \leqq j \leqq k} P(j).$$

If it is true that $P(j) = 2j - 2$, then it follows that

$$S(k + 1) = 1 + P(k) = 2k - 1.$$

The only irredundant proper pN-forms are products of sN-forms, $\phi_1, \phi_2, \cdots, \phi_p$, which are themselves irredundant forms for $(f_0, |\phi_1|), (f_0, |\phi_2|), \cdots, (f_0, |\phi_p|)$. Thus, these forms are products of sN-forms, where

$$N \leqq S(k + 1).$$

If it is true that $S(k + 1) = 2k - 1$, then we can set

$$P(k + 1) = 1 + S(k + 1) = 2k.$$

We have now constructed $S(m_0)$ and $P(m_0)$ recursively:

$$P(1) = 0, \qquad S(2) = 1, \qquad P(2) = 2,$$
$$P(k) = 2k - 2 \Rightarrow S(k+1) = 2k - 1 \Rightarrow P(k+1) = 2k.$$

Q.E.D.

LEMMA 5. *ϕ is an irredundant form for (f_0, f_1) if and only if its dual, ϕ^D, is an irredundant form for (f_1^D, f_0^D).*

THEOREM 6. *Let m_0, m_1 be the number of canonical terms of the n-variable functions f_0 and f_1. Then every irredundant form (and therefore every absolute minimal form) for (f_0, f_1) is either*

(1) *a 0-form,*
(2) *a proper sN-form, with $N \leqq \min [2m_0 - 3, 2^{n+1} - 2m_1 - 2]$,*
(3) *a proper pN-form, with $N \leqq \min [2m_0 - 2, 2^{n+1} - 2m_1 - 3]$.*

The proof follows directly from Lemmas 4 and 5.

Example 6. Find all absolutely minimal forms for

$$f^{3,5,6,7,11,13,15} = xy\bar{z} \vee w(x \vee y),$$

which is shown in the minimal sum-of-products-of-sums form obtained in Example 3.

Solution. First, find all absolutely minimal forms for (g^3, f). These are simply prime $p0$-implicants of f. Accordingly, find:

$$wx \qquad (3, 7, 11, 15).$$

Canonical term 5 is not covered by the above expression; so next find all absolute minimal forms for $(g^{3,5}, f)$. It follows from Theorem 6 that these are all minimal $p2$-forms. Accordingly, find:

$$w(x \vee y) \qquad (3, 5, 7, 11, 13, 15).$$

Canonical term 6 is not covered by the above expressions; so next find all absolute minimal expressions for $(g^{3,5,6}, f)$. It follows from Theorem 6 that these are minimal $p4$-forms. This latter requires the compilation of a table (not shown) of prime $s3$-implicants, involving 4 nontrivial 4-level minimizations and 5 nontrivial 3-level minimizations. The lengthiest 4-level minimization is equivalent to the problem solved in Example 5.

The following are the minimal $p4$-forms found for $(g^{3,5,6}, f)$:

$xy\bar{z} \vee w(x \vee y)$	*	$(y \vee wx)\ (w \vee x\bar{z})$	*
$wx \vee y\bar{z}(w \vee x)$		$(x \vee wy)\ (w \vee y\bar{z})$	*
$wy \vee x\bar{z}(w \vee y)$		$(w \vee y)\ (wx \vee y\bar{z})$	
$wx \vee y(w \vee x\bar{z})$	*	$(w \vee y)\ (x\bar{z} \vee wy)$	
$wy \vee x(w \vee y\bar{z})$	*	$(x \vee y)\ (w \vee xy\bar{z})$	*.

The six starred expressions represent f, and are all the absolutely minimal forms for f. The first of these forms was obtained by 3-level minimization, but was not previously known to be absolutely minimal. (However, Lemma 4 could have been used to show that it differed from an absolutely minimal form by at most one literal.)

Acknowledgements. The author wishes to thank Karl Menger, Jr., and Edward Sussenguth of Harvard University, Stewart Ogden of IBM Corporation, and

Richard Arnold of The University of Michigan, for their helpful suggestions and comments.

Received November, 1963

REFERENCES

1. Abhyankar, S. Minimal "sum of products of sums" expressions of Boolean functions. *IRE Trans. EC-7* (Dec. 1958), 268–276.
2. ——. Absolute minimal expressions of Boolean functions. *IRE Trans. EC-8* (March 1959), 3–8.
3. Karp, R. M., McFarlin, F. E., Roth, J. P., Wilts, J. R. A computer program for the synthesis of combinational switching circuits. Paper, AIEE Symp. Switching Circuit Theory, Detroit, Mich., October, 1961.
4. Meo, A. R. On the minimal third order expression of a Boolean function. Paper, AIEE Symp. Switching Circuit Theory and Logical Design, Chicago, Sept. 1962.
5. Mukhopadhyay, A. Detection of disjuncts of switching functions and multi-level circuit design. *J. Electr. Contr. 10*, 1 (Jan. 1961), 45–55.
6. Roth, J. P. Minimization over Boolean trees. *IBM J. Research Develop. 4*, 5 (Nov. 1960), 543–558.
7. —— and Wagner, E. G. Algebraic topological methods for the synthesis of switching systems, Part III: Minimization of nonsingular Boolean trees. *IBM J. Research Develop. 4* (Oct. 1959), 326–344.
8. Semon, W. L. Synthesis of series-parallel network functions. *Bell Systems Tech. J. 37*, 4 (1958).
9. Veitch, E. W. Logical equation minimization involving higher order solutions. *Proceedings of the International Conference on Information Processing, UNESCO*, 423–424 (Butterworths, London, 1959).
10. Wright, W. Synthesis of minimal series-parallel switching circuits. *Investigations of the Theory of Switching, Vol. BL-13*, Harvard Computation Laboratory, 1955.

Reprinted from OPERATIONS RESEARCH
Vol. 14, No. 4, July-August, 1966
Printed in U.S.A.

BRANCH-AND-BOUND METHODS: A SURVEY*

E. L. Lawler and D. E. Wood

The University of Michigan, Ann Arbor, Michigan

(Received February 11, 1966)

The essential features of the branch-and-bound approach to constrained optimization are described, and several specific applications are reviewed. These include integer linear programming (LAND-DOIG and BALAS methods), nonlinear programming (minimization of nonconvex objective functions), the traveling-salesman problem (EASTMAN AND LITTLE, ET. AL. methods), and the quadratic assignment problem (GILMORE AND LAWLER methods). Computational considerations, including trade-offs between length of computation and storage requirements, are discussed and a comparison with dynamic programming is made. Various applications outside the domain of mathematical programming are also mentioned.

AMONG the most general approaches to the solution of constrained optimization problems is that of 'branching-and-bounding' (or, in the words of BERTIER AND ROY,[10] "séparation et évaluation progressives"). Like dynamic programming, branching-and-bounding is an intelligently structured search of the space of all feasible solutions. Most commonly, the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets, and a lower bound (in the case of minimization) is calculated for the cost of the solutions within each subset. After each partitioning, those subsets with a bound that exceeds the cost of a known feasible solution are excluded from all further partitionings. The partitioning continues until a feasible solution is found such that its cost is no greater than the bound for any subset. These basic ideas of branching-and-bounding are formalized in the next section of this paper.

There has been some recent interest in branch-and-bound methods for the solution of integer linear programming and traveling-salesman problems. These particular applications, and others, are reviewed in the third section. The remaining sections discuss general computational considerations, relations with dynamic programming, and various applications outside the domain of mathematical programming.

ESSENTIAL FEATURES OF BRANCHING-AND-BOUNDING

THERE ARE many constrained optimization problems for which 'direct' methods of solution either do not exist, or are inefficient. Problems may

* This research was supported in part by NSF Grant GP-2778, and in part by Air Force contract AF 30(602)-3546.

be of this nature because the objective function or constraints are non-convex, or some or all of the variables are restricted to discrete values. Branching-and-bounding enables us to solve these 'difficult' problems by applying existing methods of solution to 'easy' problems.

Suppose we are confronted with a 'difficult' constrained optimization problem of the form

$$\left.\begin{array}{ll} \text{minimize} & c^{(0)}(x) \\ \text{subject to} & g_1^{(0)}(x) \geqq 0, \\ & g_2^{(0)}(x) \geqq 0, \\ & \cdots \\ & g_m^{(0)}(x) \geqq 0 \\ \text{and} & x \in X^{(0)}, \end{array}\right\} \qquad (0)$$

where the set $X^{(0)}$ denotes the permissible domain of optimization, e.g., the positive orthant of Euclidean n-space, and x denotes a vector $(x_1, x_2, \cdots, x_n)$. In line with accepted terminology, a solution vector x that satisfies the constraints and lies within the domain of optimization is said to be a *feasible* solution, and a feasible solution for which $c^{(0)}(x)$ is minimal is said to be an *optimal* solution.

One way to solve a difficult problem is to solve a related 'easy' problem, and hope that the solution to the easy problem can be shown to be a solution to the difficult problem. In the case at hand, suppose we replace problem (0) with an easy problem (1), which *bounds* problem (0), in the sense that the following 'bounding' property is satisfied.

(B) There exists at least one optimal solution $x^{(0)}$ of problem (0), such that $x^{(0)}$ is feasible for problem (1) and $c^{(1)}(x^{(0)}) \leqq c^{(0)}(x^{(0)})$.

[$c^{(1)}$ denotes the objective function of problem (1).]

Suppose we obtain an optimal solution $x^{(1)}$ to problem (1). It is not hard to show that if $x^{(1)}$ satisfies the following *optimality* conditions:

(O1) $x^{(1)}$ is a feasible solution to problem (0),

and

(O2) $$c^{(1)}(x^{(1)}) = c^{(0)}(x^{(1)}),$$

then $x^{(1)}$ is an optimal solution to problem (0) as well.

Of course, if the solution $x^{(1)}$ does not satisfy both (O1) and (O2) we must try something else. If condition (O1) is not satisfied, we might try to strengthen the constraint or to restrict the domain of optimization of problem (1), in order to make the solution $x^{(1)}$ infeasible for (1). If (O2) is not satisfied, we might attempt to modify the objective function $c^{(1)}$ so as to increase the cost of $x^{(1)}$, in order to either make $x^{(1)}$ nonoptimal, or

to cause the equality (O2) to be satisfied. Regardless of what the objective is, it generally proves to be easiest to replace problem (1) by a *set* $\mathrm{P} = \{(2), (3), \cdots\}$ of problems that bound problem (0) in the sense that they jointly satisfy the following generalized bounding property:

(B′) There exists at least one optimal solution $x^{(0)}$ of problem (0), such that $x^{(0)}$ is feasible for at least one problem $(j)\epsilon P$, and $c^{(j)}(x^{(0)}) \leqq c^{(0)}(x^{(0)})$.

Now suppose we obtain an optimal solution $x^{(j)}$ to each problem $(j)\epsilon P$. Let $x^{(k)}$ be such that

$$c^{(k)}(x^{(k)}) = \min_{(j)\epsilon P} c^{(j)}(x^{(j)}).$$

It is not hard to show that $x^{(k)}$ is an optimal soluton to problem (0) if conditions (O1) and (02) are satisfied (with $x^{(k)}$ replacing $x^{(1)}$ in their statement).

Now, of course, if the solution $x^{(k)}$ does not satisfy both the conditions (O1) and (O2), we must again try something else. Our approach is to replace one of the problems in the bounding set P by a set of new problems. For example, suppose we replace problem (k) by a set of problems $P^{(k)}$. In addition to requiring that the new set of bounding problems $(P-\{k\})\cup P^{(k)}$ satisfy the bounding condition (B'), we may impose the following weak 'convergence' condition,

(C) For each problem $(j)\epsilon P^{(k)}$, either $x^{(k)}$ is infeasible for (j) or $c^{(j)}(x^{(k)}) > c^{(k)}(x^{(k)})$,

or the stronger condition,

(C′) For each problem $(j)\epsilon P^{(k)}$ and each feasible solution x to problem (k), either x is infeasible for (j) or $c^{(j)}(x) > c^{(k)}(x)$.

These conditions are not sufficient to guarantee that repeated revision of the set P will yield an optimal solution to problem (0) with a finite amount of computation. However, they do represent minimal conditions we would like to impose in order that some progress toward a final solution can be made. (Actually, in practice the finiteness of branch-and-bound methods is rarely an issue.)

A tree like that shown in Fig. 1 helps to visualize matters. Each node of the tree is identified with a problem (j), which is of the form

$$\left.\begin{array}{ll} \text{minimize} & c^{(j)}(x), \\ \text{subject to} & g_1^{(j)}(x) \geqq 0, \\ & g_2^{(j)}(x) \geqq 0, \\ & \cdots \\ & g_m^{(j)}(x) \geqq 0, \\ \text{and} & x\epsilon X^{(j)}. \end{array}\right\} \qquad (j)$$

(The number of constraints may vary from problem to problem; for notational simplicity, this number is simply designated as m.) The problems that replace problem (j) in the bounding set are pointed to by the branches directed outward from node (j) in the tree. (Hence the term 'branching,' in 'branching-and-bounding.') Thus, at any intermediate point in the calculations, the set P of current bounding problems is identified with the set of nodes that are 'leaves' of the tree, as it has been developed at that point in the computation.

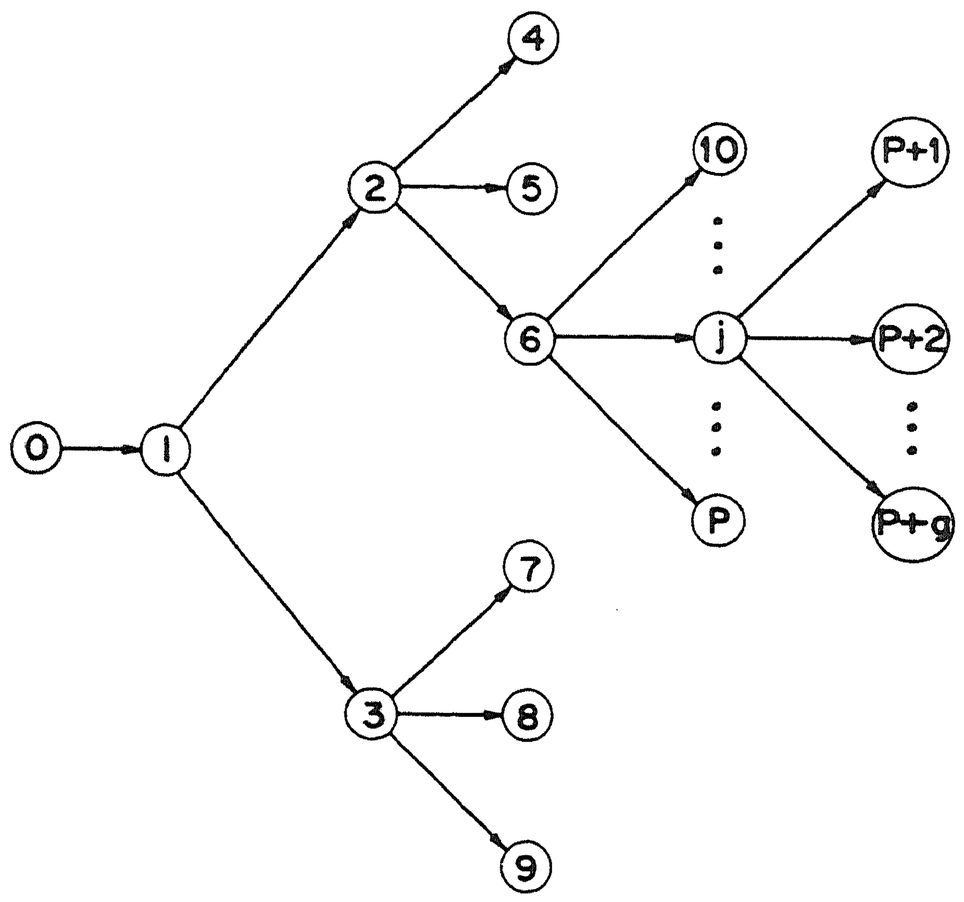


Fig. 1. Tree representation.

The total amount of computation is related to the number of distinct bounding problems created, and hence to the total number of nodes in the fully developed tree. The amount of temporary storage required is related to maximum cardinality of the bounding set P, and hence to the maximum number of leaves of the tree at any intermediate stage in its development. In a later section, we describe how various strategies may be employed to reduce the amount of computation at the expense of additional storage, and vice-versa. We now merely make a simple observation that indicates how the cardinality of P can be reduced somewhat.

During the course of the calculations, various feasible solutions to problem (0) are discovered. At any intermediate stage, let $\hat{x}$ denote the least costly (in terms of $c^{(0)}$) feasible solution that has been discovered. Sup-

pose (j) is a bounding problem with optimal solution $x^{(j)} \neq \hat{x}$. Then, if $c^{(0)}(\hat{x}) \leqq c^{(j)}(x^{(j)}$, it follows that problem (j) can be removed from the set P without affecting the bounding condition (B').

We associate with each node (j) of the tree a *bound* $c^{(j)}(x^{(j)})$. We say that any leaf node of the tree whose bound is strictly less than $c^{(0)}(\hat{x})$ is *active*; otherwise it is designated as *terminated*, and need not be considered in any further computation. Roughly speaking, what we seek to do in branching-and-bounding is to develop the tree until every leaf can be terminated.

The complete description of a branch-and-bound algorithm requires a specification of the rule that determines which of the currently active bounding problems is to be chosen for branching, as well as the method for deriving new bounding problems. In the fourth section, various rules for choosing active problems are discussed; this discussion will be more meaningful in the context of the applications presented in the following section.

APPLICATIONS

FOUR AREAS of application in mathematical programming will serve to illustrate branching-and-bounding. These are (1) integer programming, (2) nonlinear programming, (3) the traveling-salesman problem, and (4) the quadratic assignment problem. Branch-and-bound methods can, of course, be applied to a variety of other types of problems in scheduling, combinatorics, decision processes, etc.

(1) Integer Programming

It is well-known that a wide variety of nonlinear and combinatorial problems can be formulated as integer linear programming problems. These problems differ from ordinary linear programming problems in that the variables are restricted to nonnegative integer values:

$$\begin{aligned} &\text{minimize} && c\dot{x}, \\ &\text{subject to} && Ax \geqq b, \\ & && x_j \text{ nonnegative integer.} \qquad (j=1, 2, \cdots, n) \end{aligned}$$

The Gomory cutting-plane and all-integer algorithms are often used to solve these problems. Another approach has been suggested by LAND AND DOIG.[28] This is roughly as follows: Apply the simplex method to the problem. If the optimal solution thereby obtained is integral (and there are many linear programs for which this will be the case), the problem is solved. If the optimal solution is not integral, then create two new problems as follows: Choose some variable that has a noninteger value (say $x_3 = 4.4$), and restrict that variable to the next lower integer value for one

problem ($x_3 \leqq 4$) and to the next higher integer value for the other ($x_3 \geqq 5$). The process is then repeated on each of the new problems.

Under the Land-Doig approach all bounding problems are conventional linear programming problems, and hence differ from problem (0) essentially only in their domain of optimization. Each feasible solution x of problem (0) is a feasible solution of at least one bounding problem (j) and it is always true that $c^{(0)}(x) = c^{(j)}(x)$, hence condition (B') is easily seen to be satisfied. Branching forces fractional solutions to become infeasible, and condition (C') is seen to hold.

The Land-Doig approach is particularly appropriate for 'mixed,' problems (only a proper subset of the variables restricted to integers). Various elaborations and refinements of the Land-Doig approach have been proposed and tested by BEALE AND SMALL,[6] DAKIN,[11] and DRIEBECK.[12] Reportedly, computational results have been favorable, i.e., competitive with cutting-plane and all-integer methods. In particular, Dakin[11] reports the solution of a problem with 17 constraints and 93 variables, 80 of which were constrained to be integers. The solution of this problem took 7 minutes and 28 seconds on a KDF9 computer, while the GOMORY[22] mixed-integer algorithm failed to find the solution in over 2000 iterations. Several smaller problems were reported solved in less than a minute on the KDF9.

Whereas the Land-Doig approach uses the simplex method to obtain a lower bound on the integer program, a totally different approach, not employing the simplex method, is proposed by Balas.[2,3,4] Balas characterizes his algorithm as 'additive,' in that no multiplications or divisions are required.

For convenience, Balas deals only with variables that are restricted to 0, 1 values. (An integer variable x_j that is bounded above by $2^K - 1$ can be expressed as the sum of K(0, 1)-variables: $x_{j1}, x_{j2}, \cdots, x_{jK}$ by the relation $x_j = x_{j1} + 2x_{j2} + \cdots + 2^{K-1}x_{jK}$. For each problem that is created by branching, Balas partitions the variables into three classes:

(1) those which are set to 0,
(2) those which are set to 1,
(3) those which are free to assume either the value 0 or 1.

Assume all cost coefficients c_j are positive (it is a simple matter to put the problem into this form). If the solution obtained by setting all variables in class (3) to 0 is feasible, it is clearly optimal. If this solution is not feasible the sum of the c_j corresponding to the class (2) variables plus the smallest c_j corresponding to a class (3) variable is an easily-obtained lower bound on the cost of an optimal solution. Branching is effected by forming one problem in which a specific class (3) variable is transferred to class (1) and a second problem in which the same variable is transferred to class (2).

An essential feature of the Balas algorithm is the testing of the bounding problems for infeasibility. Each linear constraint is examined to see if it can be satisfied by setting each class (3) variable equal to its 'more favorable' value. Inability to satisfy a constraint implies the nonexistence of feasible solutions; however, since the constraints are tested independently, the converse is not true. An infeasible bounding problem is, of course, terminated. (In effect, its bound is $+\infty$.)

Although some persons view the Balas algorithm as involving 'branching,' but not 'bounding,' the present authors do not share this view. The concept of bounding is certainly an essential aspect of the algorithm. The only departure from the formulation of the preceding section is that the bounds determined are not the actual costs of optimal solutions to the bounding problems, but are lower bounds on the actual costs.

R. J. FREEMAN[16] reports favorable results with the Balas algorithm as modified by GEOFFRION[18] at Rand. Freeman reports the solution of the ten fixed-charge problems given by HALDI[23] in times ranging from 2 seconds to 7 minutes on an IBM 7090. The sizes of these problems ranged from 4 inequalities in 14 binary variables to 6 inequalities in 20 binary variables. Also closely related to the method of Balas are the proposals of BENDERS, ET AL.,[9] GILMORE AND GOMORY,[20] and LAWLER AND BELL.[32]

(2) *Nonlinear Programming*

Branch-and-bound methods are possibly the best way to obtain globally optimal solutions to nonlinear programming problems in which a nonconvex function is to be minimized.

A limited amount of computation experience has been reported by BEALE[5] for a particular problem of this type. However, this experience is not sufficient to base any general conclusions concerning the expected amount of computation or general applicability of the approach. The following example is intended to be suggestive of the possibilities in this area.

Suppose we seek to minimize a separable cost function

$$f(x) = \sum_{j=1}^{j=n} f_j(x_j),$$

where the f_j (and therefore f) are not necessarily convex functions. Suppose, for example, that f_1 is as shown in Fig. 2.

Suppose we can bound x_j from above by x_j'. We define the *convex envelope* of f_j to be the largest function f_j^c such that

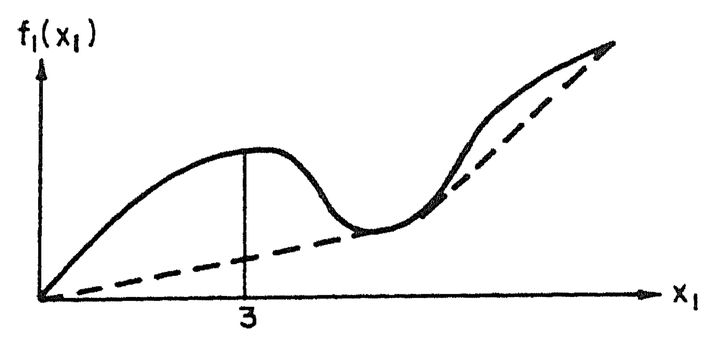
(1) $$f_j^c(x_j) \leqq f_j(x_j), \qquad (0 \leqq x_j \leqq x_j')$$

(2) $$f^c \text{ is convex.}$$

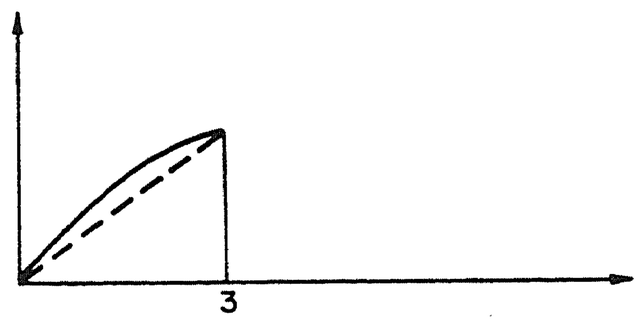
The convex envelope of f_j is indicated by dashed lines in Fig. 2.

Having replaced each f_j by its convex envelope, we proceed to apply an appropriate convex programming algorithm to the minimization of the convex function

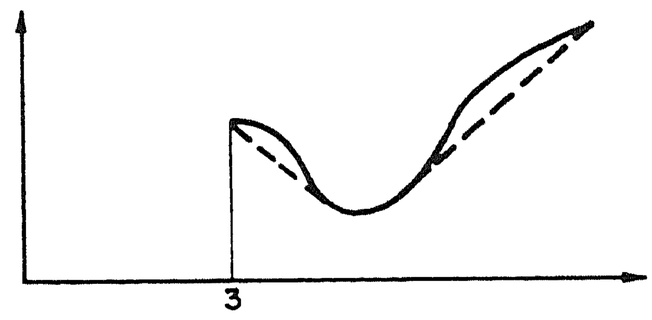
$$f^c(x) = \sum_{j=1}^{j=n} f_j{}^c(x_j),$$



(a) Nonconvex Function and Convex Envelope.



(b) Convex Envelope for Restriction of Function to $x_1 \leq 3$.



(c) Convex Envelope for Restriction of Function $x_1 \geq 3$

Fig. 2. Example of nonconvex function.

subject to the constraints of the original problem. If the optimal solution x^0 thereby obtained has the property that

$$f^c(x^0) = f(x^0),$$

then a global optimum has been obtained and the problem is solved. If this is not the case, then two new problems are created as follows. Choose some variable x_j such that

$$f_j^c(x_j^0) < f_j(x_j^0).$$

In one of the new problems, f_j^c is replaced by a convex envelope that is valid for the interval $0 \leqq x_j \leqq x_j^0$ and in the other for the interval $x_j^0 \leqq x_j \leqq x_j'$. This is illustrated in Fig. 2 for the case $x_1^0 = 3$.

It is believed that, properly managed, the process will yield a globally optimal solution in a finite number of steps, provided each f_j is piecewise convex. In any case, computations can be carried out until a solution is obtained that differs from the optimum by no more than a predetermined amount (*see* the fifth section).

Note that this proposal uses branching to overcome the obstacle of a nonconvex cost function, but could also be used to overcome nonconvex constraints. Problems involving nonconvex (but monotone) cost functions and nonconvex constraints can also be solved by the method of Lawler and Bell.[32]

(3) *The Traveling-Salesman Problem*

The well-known traveling-salesman problem requires for its solution the shortest possible circuit through n cities, where the distances between the cities are given. Any feasible solution to this problem can be expressed in terms of n^2 variables $x_{ij} (i = 1, 2, \cdots, n; j = 1, 2, \cdots, n)$, where

$x_{ij} = 1$ if the solution requires the salesman to travel from city i directly to city j,
$= 0$ otherwise.

The salesman is to enter each city exactly once, so we have,

$$\sum_{i=1}^{i=n} x_{ij} = 1, \qquad (j = 1, 2, \cdots, n)$$

and he is to leave it exactly once, so

$$\sum_{j=1}^{j=n} x_{ij} = 1. \qquad (i = 1, 2, \cdots, n)$$

We are to minimize

$$\sum_{i=1}^{i=n} \sum_{j=1}^{j=n} d_{ij} x_{ij},$$

where d_{ij} represents the distance from city i to city j.

It is evident that the traveling-salesman problem is very much like the conventional assignment problem. Indeed, *the n-city traveling-salesman problem is an $n \times n$ assignment problem with the added constraint that the solution must be cyclic*, i.e., representable by a cyclic permutation of the integers 1 to n.

The approach of EASTMAN[13,14] is to solve the assignment problem corresponding to the traveling-salesman problem. If the optimal solution

thereby obtained is cyclic, then the problem is solved. On the other hand, if the solution is not cyclic, a constraint can be added as follows.

Suppose the optimal solution to the assignment problem contains the subcycle (1, 3, 5), i.e., $x_{13}=1$, $x_{35}=1$, $x_{51}=1$. At least one of the links in this subcycle must be missing from any feasible solution to the traveling-salesman problem. Hence, a possible constraint is

$$\begin{aligned} \text{either} \quad & x_{13}=0, \\ \text{or} \quad & x_{35}=0, \\ \text{or} \quad & x_{51}=0. \end{aligned}$$

This suggests three new problems. A feasible solution to the traveling-salesman problem is a feasible solution to *at least* one of the new problems.

Consider the 10-city (unsymmetric) problem as defined by the following matrix:

	1	2	3	4	5	6	7	8	9	10
1	∞	24	18	22	31	19	33	25	30	26
2	15	∞	19	27	26	32	25	31	28	18
3	22	23	∞	23	16	29	27	18	16	27
4	24	31	18	∞	19	13	28	9	19	27
5	23	18	34	20	∞	31	24	15	25	8
6	24	12	17	15	10	∞	11	16	21	31
7	28	15	27	35	19	18	∞	21	21	19
8	13	24	18	13	13	22	25	∞	29	24
9	17	21	18	24	27	24	34	31	∞	18
10	18	19	29	16	23	17	18	31	23	∞

The initial assignment problem, which has this matrix as its cost matrix, we designate problem (1). An optimal solution to this assignment problem is the permutation (1, 6, 7, 2) (3, 9) (4, 8, 5, 10), with a cost of 140.

Two new assignment problems are created: problem (2) with $x_{39}=0$ and problem (3) with $x_{93}=0$. These two problems have optimal solutions with costs of 147 and 143 respectively. Since there is a lesser lower bound on the solutions to problem (3), we proceed to branch from it next. An optimal solution to problem (3) is the permutation (1, 6, 7, 2, 3, 9) (4, 8, 5, 10), and we create problems (4), (5), (6), and (7), for which $x_{48}=0$, $x_{85}=0$, $x_{5,10}=0$, $x_{10,4}=0$, respectively.

The complete solution of the example is indicated in Fig. 3a. The nodes of the tree represent assignment problems and the branches are given labels which denote excluded links. The number, l, within the node is an index specifying the point in the computation where problem (l) is derived, while

the bound for problem (l) is the number adjacent to the node. It can be seen that the problem requires the solution of 11 assignment problems. (There are various ways in which this number could be reduced.) At the conclusion, when a cyclic optimal solution is found to problem (11), with a cost of 146, every other assignment problem has a (noncyclic) optimal solution whose cost is at least as great.

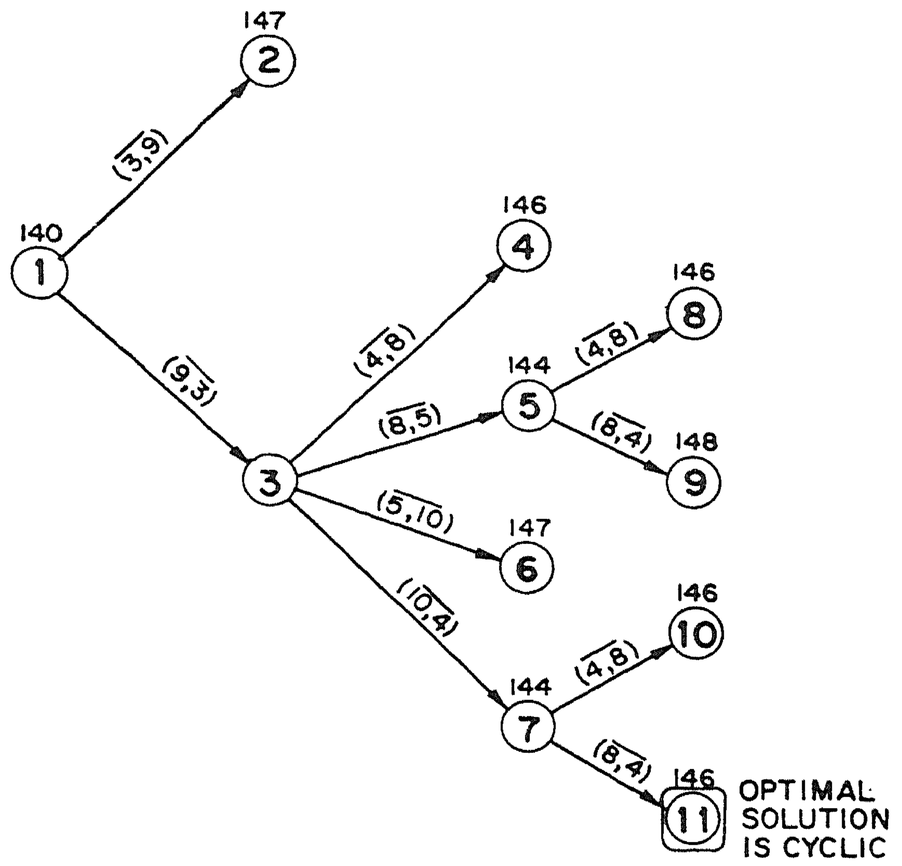


Fig. 3a. Solution of example of traveling-salesman problem using the Eastman[13] approach.

LITTLE'S ET AL.[34] approach to the traveling-salesman problem differs mainly in that

(1) two problems are created at each branching step, corresponding to $x_{ij}=0$ and $x_{ij}=1$.
(2) an optimal solution to the assignment problem is not obtained. Instead, the smallest element is subtracted from each row of the distance matrix, and then the smallest element from each column of the result. The sum of these elements is a valid (and fairly good) lower bound on the optimal solution.

Little employs what appears to be a very worthwhile heuristic. At each branching step, the pair (i, j) is chosen in such a way that the problem corresponding to $x_{ij}=0$ will yield as large a bound as possible. The bound is deemed to be less important for the other problem (corresponding

to $x_{ij}=1$), since the dimension of this problem is effectively diminished by one.

Little's method is similar to Balas' method for integer programming in that instead of seeking optimal solutions to bounding problems, only lower bounds on the costs of optimal solutions are calculated. This enables

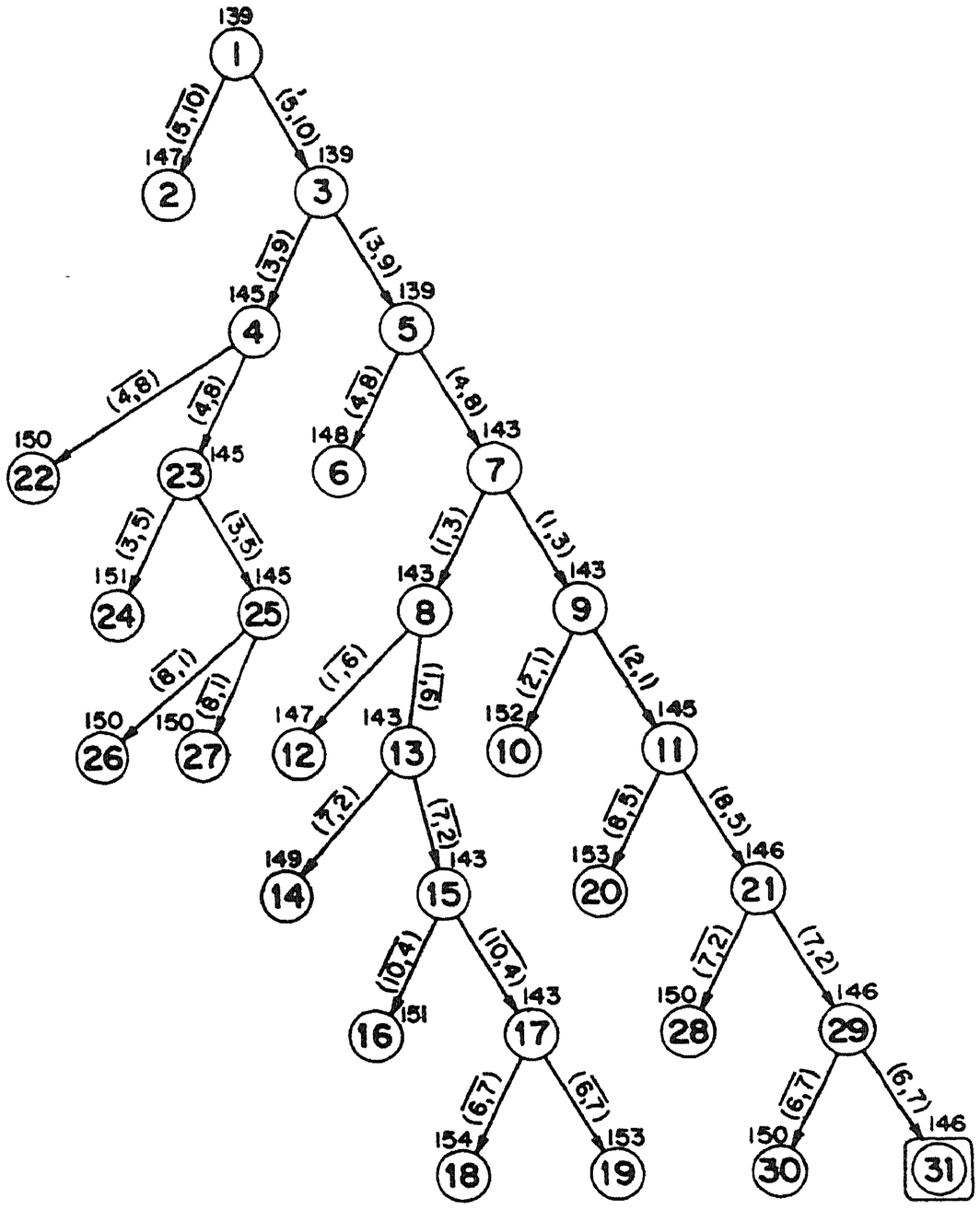


Fig. 3b. Solution of example of traveling-salesman problem using the Little, et. al.[34] approach.

Little to take advantage of one of the trade-offs possible in branch-and bound methods. Namely, a poorer, but more easily calculated, bound results in more branching operations, but less computation at each node. The solution of the example problem by the Little, et al., algorithm is indicated in Fig. 3b with the same notational conventions for the nodes as in Fig. 3a. A branch label of the form (j, k) indicates that link (j, k)

is included in the solution. On the other hand, $(\overline{j, k})$ indicates that link (j, k) is excluded. The trade-off between the amount of computation for each bounding problem and the number of problems solved is dramatically seen by a comparison of Figs. 3a and 3b.

Little's, et al., computational results have been favorable. The mean computation time on an IBM 7090 for 100 thirty-city problems was 58.5 seconds and for 5 forty-city problems was 8.37 minutes.

Eastman's method is not known to have been programmed.

Another type of bound which could be used for the traveling-salesman problem, but has not been tested, is based on the construction of minimal spanning trees. One city can be duplicated to produce an $(n+1)$-city network. Then a minimal tree sapnning for all $n+1$ cities can be calculated by one of the simple algorithms available for that purpose. The total length of the arcs in the tree is a valid lower bound (sometimes better and sometimes worse) than the bound obtained from an optimal solution to the assignment problem.

(4) *The Quadratic Assignment Problem*

The quadratic assignment problem differs from the ordinary assignment problem only in that an arbitrary quadratic cost function is to be minimized:

$$\begin{aligned} &\text{minimize} && \textstyle\sum_{i,j} \sum_{p,q} c_{ijpq}\, x_{ij}\, x_{pq}, && \\ &\text{subject to} && \textstyle\sum_i x_{ij}=1, && (j=1, 2, \cdots, n) \\ & && \textstyle\sum_j x_{ij}=1, && (i=1, 2, \cdots, n) \\ & && x_{ij}=0 \text{ or } 1. && (i, j=1, 2, \cdots, n) \end{aligned}$$

GILMORE[19] (for a somewhat more specialized version of the problem) and LAWLER[30] have proposed essentially equivalent branch-and-bound methods. The bound is obtained by solving n^2 (usually very simple) $(n-1)\times(n-1)$ linear assignment problems and then one $n\times n$ assignment problem for which the initial computations determine the cost coefficients.

Gilmore reports some fairly favorable computational experience with problems of moderate dimension.

(5) *Miscellaneous Examples*

Efroymson and Ray[15] describe a branch-and-bound algorithm for the plant location problem. Simply stated, the plant location problem is a transportation problem in which there are no constraints on the sources (plants), but there is a setup cost incurred in opening each plant. The algorithm is extended to handle the cases where

(1) there is a 'variable plant cost' as well as the fixed cost;
(2) no fixed cost, but concave plant cost—two linear segments;
(3) a fixed cost and many linear segments.

A number of 50-plant, 200-customer problems have been solved on an IBM 7094 in about 15 minutes each. The problems were of the types indicated in (1) and (2) above.

The techniques of branching and bounding have also been applied to job shop and machine scheduling problems. LOMNICKI[35] describes a branch-and-bound algorithm for the solution of machine-scheduling problems with three machines. While there is no computer processing times given, the amount of computation is comparable with that reported by INGALL AND SCHRAGE.[25] Ingall and Schrage report the solution of three-machine scheduling problems with up to 10 jobs to be scheduled in less than 3 minutes on a CDC 1604. The mean time for 8 problems with 6 jobs was 1.8 seconds of CDC 1604 time.

The application of branch-and-bound techniques to sequencing problems and assembly-line balancing is given by JAESCHKE.[26] Jaeschke further considers the traveling-salesman problem in a manner similar to Little, et al.

STRATEGIES OF BRANCHING

ANY BOUNDING problem with a bound less than the cost of the least costly feasible solution discovered to date is a possible candidate for branching. A policy is needed to choose among the candidates, which may be quite large in number. Two such policies are

(1) branch from lowest bound,
and (2) branch from newest active bounding problem.

(1) Branch from Lowest Bound

As the name suggests, this policy is to branch from that bounding problem, which currently has the lowest bound. Suppose that for any given problem the set of new bounding problems is uniquely determined. Then this policy has the advantage that the total amount of computation is minimized, in the sense that any branching operation performed under this policy must also be performed under any other policy. However, the volume of intermediate data that must be stored in the computer may be quite large.

Land-Doig and Eastman proposed using the branch-from-lowest bound policy.

(2) Branch from Newest Active Bounding Problem

The idea here is always to branch from the latest problem created by branching. This strategy can be implemented with a 'pushdown' stack.

Each time branching is carried out, the new problems are placed on the top of the stack, and those below are 'pushed down.' Each time branching is to be performed, the problem currently on the top of the stack is examined. If it is active, branching is carried out upon it. If it is not active, it is discarded and the next problem on the top of the stack is examined. This policy has the distinct advantage that, for most problems, a minimum of computer storage is required. (The maximum size of the stack is proportional to the length of the longest possible path directed away from the root of the branching tree.) However, many more branching operations may have to be performed than under the branch-from-lowest-bound policy.

Little, et al., argued in favor of the branch-from-newest-bounding-problem policy. However, the branch-from-lowest-bound policy was used in reported computations.

It should be noted that in most situations the set of derived bounding problems at any point in the computation is not uniquely determined. For example, consider the Balas algorithm for integer programs, with the variables restricted to be either 0 or 1. In this case we may choose any one of the available 'free' variables [i.e., class (3) variables] x_j to create two new problems corresponding to $x_j=0$ and $x_j=1$. The choice of which x_j to choose in general must be made by some heuristic rule, such as that used by Little, et al.

SUBOPTIMIZATION

A VERY USEFUL feature of branching-and-bounding is the opportunity to compute solutions that differ from the optimum by no more than a prescribed amount. Suppose, at the outset, that it is decided that a feasible solution whose cost exceeds the cost of an optimal solution by no more than 10 per cent would be acceptable. Then, if a feasible solution is found with a cost of 150, we can terminate all problems with bounds of 137 or more ($1.10\times137=150.7>150$). Both Gilmore and Lawler have pointed out that this technique would substantially reduce the amount of computation required for the quadratic assignment problem.

In order to find the best solution possible in a bounded length of time, various 'adaptive' computational schemes are possible. A scheme to find the best solution in a length of time T might proceed as follows:

(i) Search for an optimal solution for a length of time equal to $T/2$. If an optimal solution is not found proceed to (ii).

(ii) Search for a suboptimal solution that differs from the optimal by no more than 5 per cent. If such a solution is not found in length $T/4$ proceed to (iii).

(iii) Search for a suboptimal solution differing from the optimum by no more than 10 per cent for $T/8$ units of time, etc.

RELATION TO DYNAMIC PROGRAMMING

CERTAIN branch-and-bound algorithms are closely related to dynamic programming. A further discussion of the traveling-salesman problem may help to illustrate this relation.

There are many ways to branch in the traveling-salesman problem. We may initially create $n-1$ problems, corresponding to the alternatives

$$\begin{aligned}
&(1) && x_{12}=1,\ x_{1j}=0, && (j\neq 2)\\
&(2) && x_{13}=1,\ x_{1j}=0, && (j\neq 3)\\
& && \cdots\\
&(n-1) && x_{1n}=1,\ x_{1j}=0. && (j\neq n)
\end{aligned}$$

Each of these $n-1$ problems has dimension $n-1$. The appropriate $(n-1)$-dimensional assignment problem for case (k) is obtained by removing row 1 and column k from the distance matrix.

Now suppose we carry out further branching upon each of these $(n-1)$-dimensional problems. For problem (k) we have these cases:

$$\begin{aligned}
&x_{k2}=1,\ x_{kj}=0, && (j\neq 2)\\
&x_{k3}=1,\ x_{jk}=0, && (j\neq 3)\\
& \cdots\\
&x_{kn}=1,\ x_{kj}=0. && (j\neq n)
\end{aligned}$$

The appropriate $(n-2)\times(n-2)$ assignment problem for the case $x_{kl}=1$ is obtained by removing row k and column l from the matrix of distances.

When the process is carried one step further, many of the assignment problems are seen to be identical. For example, the $(n-3)\times(n-3)$ assignment problem for the case $x_{12}=x_{23}=x_{34}=1$ is the same as the problem for the case $x_{13}=x_{32}=x_{24}=1$. In both cases, the first, second, and third rows and second, third, and fourth columns of the original distance matrix have been removed. All that differs between the two problems are the 'constant' costs $d_{12}+d_{23}+d_{34}$ and $d_{13}+d_{32}+d_{24}$.

If we represent the branching process by a tree, as before, and we identify those nodes that correspond to equivalent assignment problems, we notice that the tree 'collapses,' as shown in Fig. 4.

This result should be compared with BELLMAN's dynamic programming treatment[7] of the same problem, which is as follows. With no loss in generality, fix the origin of the circuit at some city, say 1. Suppose that at some stage in an optimal circuit starting at city 1 one has reached city i and there remain k cities $j_1, j_2, \cdots, j_k$ to be visited before returning to 1. Then it is clear that if the tour is to be optimal, the path from i through $j_1, j_2, \cdots, j_k$ in some order and then to 1 must be of minimum length.

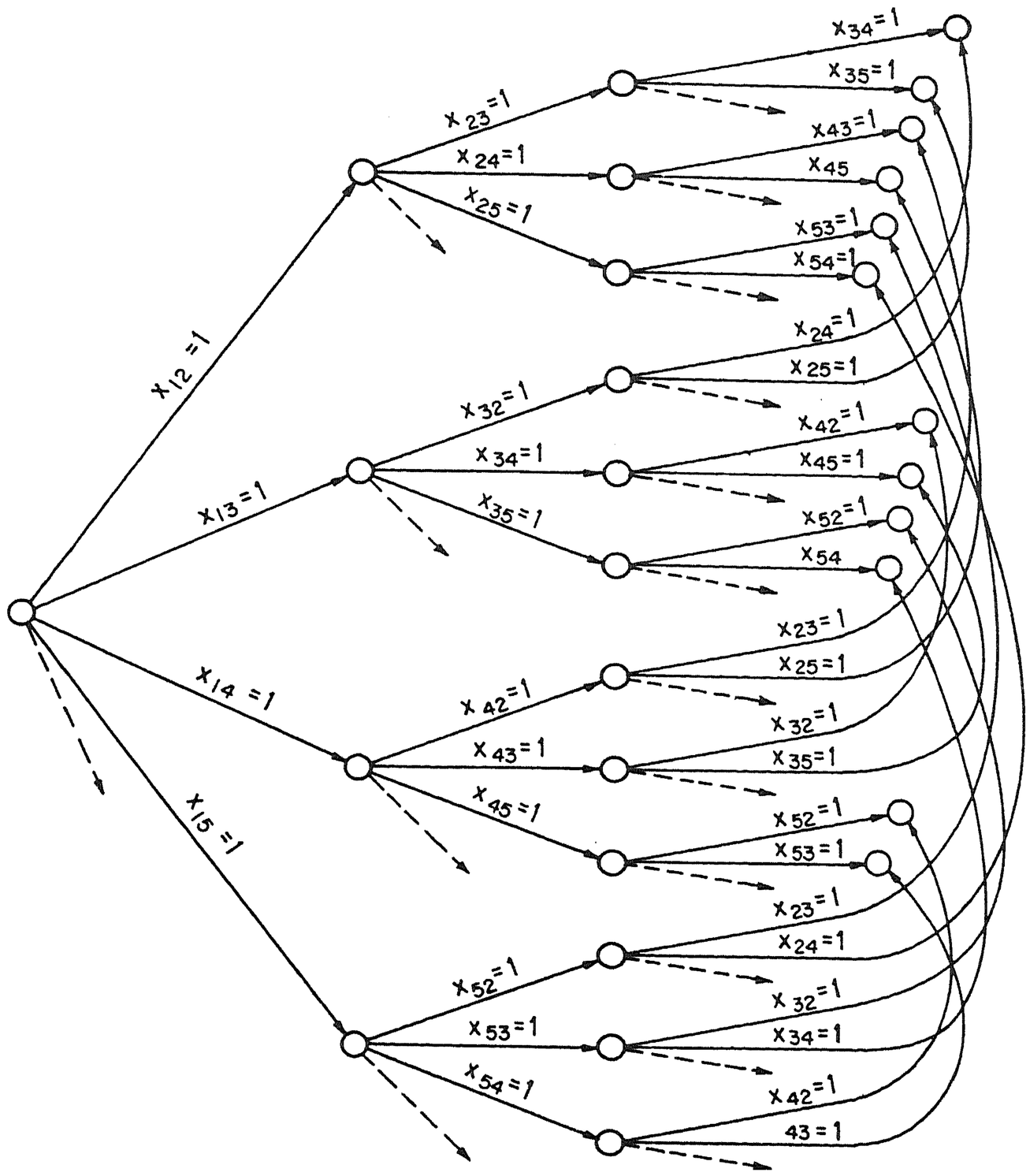


Fig. 4. Collapsing of tree for traveling-salesman problem.

Define

$C(i; j_1, j_2, \cdots, j_k)$ = the length of a path of minimum length from i to 1, which passes once and only once through each of the remaining k unvisited cities $j_1, j_2, \cdots, j_k$.

Thus, $C(1; 2, 3, \cdots, n)$, and the path that has this as its length, constitutes a solution to the problem.

It is easy to show that

$$C(i; j_1, j_2, \cdots, j_k) = \min_{1 \leq m \leq k}\{d_{ij_m} + C(j_m; j_1, j_2, \cdots, j_{m-1}, j_{m+1}, \cdots, j_k)\}$$

and $$C(i; j) = d_{ij} + d_{j1},$$

which enables us to define a sequence of computations.

The computational scheme for dynamic programming is essentially to develop the *complete* collapsed tree shown in Fig. 4, to associate with the directed arc $x_{ij}=1$ the distance d_{ij}, and then to find the shortest path between the two ends of the tree. The traveling-salesman problem is thus converted to a shortest-route problem.

The branch-and-bound approach uses an assignment problem to calculate a lower bound on the length of the shortest path from any given node in the collapsed tree to the right-hand end. Thus, various routes are eliminated from consideration, even though the length of the shortest path is not known with precision.

NONMATHEMATICAL PROGRAMMING APPLICATIONS

COMPUTATIONAL techniques that are essentially the same as, or very similar to, branch-and-bound methods, have been applied in areas outside of mathematical programming. These include

(1) switching circuit minimization,
(2) 'artificial intelligence,' e.g., game-playing and theorem proving,
(3) 'pure' combinatorics.

(1) *Switching Circuit Minimization*

The synthesis problem for switching circuits is roughly as follows. Given a supply of 'elementary' switching circuits, use as few of these components as possible to construct a switching circuit having a prescribed input-output behavior. Algorithms proposed and programmed by J. Paul Roth and his associates at IBM definitely fall under the heading of branch-and-bound methods.[27,31,43,44,45]

It should also be mentioned that one of the fundamental problems in switching theory is the 'covering' problem, i.e., the special type of integer program with an A matrix composed entirely of 0's and 1's and a b-vector composed entirely of 1's. Switching theorists, such as Roth, and McCLUSKEY,[36] have proposed branch-and-bound methods for solving these problems. See also LAWLER.[33]

(2) *Artificial Intelligence*

It has long been observed that the analysis of chess and similar games can be carried out by means of a tree, where the branches of the tree correspond to possible moves. Game-playing programs, such as the checker-playing program of SAMUEL[46] use 'scoring functions' or other estimates to

assign values to the branches in an effort to reduce the extent of the tree that must be explored in order to make an intelligent move. The effect is a branch-and-bound method with (necessarily) inexact bounds.

An essentially similar approach is used for some types of theorem proving and heuristic programming.

(3) *Pure Combinatorics*

Branching methods are commonly used to enumerate or to prove or disprove the existence of combinatorial designs of various types. These methods should possibly be classified as 'branch-and-exclude' rather than 'branch-and-bound' in character. PROFESSOR D. H. LEHMER of the University of California at Berkeley has named this method *backtrack*. WALKER[49] and GOLOMB AND BAUMERT [21] have applied backtracking to various combinatorial problems.

REFERENCES

1. ABADIE, J., "État actuel de la programmation linéaire," presented at *Compte-Rendu du Congres de Paris*, August 1961.
2. BALAS, E., "Un algorithme additif pour la résolution des programmes linéaires en variables bivalentes," *C. R. Acad. Sc. Paris* **258,** 3817–3820 (1964).
3. ———, "Extension de l'algorithme additif à la programmation en nombres entiers et à la programmation non linéaire," *C. R. Acad. Sci. Paris* **258,** 5136–5139 (1964).
4. ———, "An Additive Algorithm for Solving Linear Programs with 0-1 Variables," *Opns. Res.* **13,** 517–549 (1965).
5. BEALE, E. M. L., "Two Transportation Problems," *Actes de la 3eme Conference Internationale de Recherche Operationelle*, Oslo (1963), 780–788, Dunod (1964).
6. ———, AND SMALL, R. E., "Mixed Integer Programming by a Branch-and-Bound Technique," presented at *IFIP Congress* **65,** New York, May 1965.
7. BELLMAN, R. E., "Dynamic Programming Treatment of the Traveling-Salesman Problem," *J. Assoc. for Comp. Mach.* **9,** 61–63 (1962).
8. ———, AND DREYFUS, S. E., *Applied Dynamic Programming*, Princeton University Press (1962).
9. BENDERS, J. F., CATCHPOLE, A. R., AND KUIKEN, L. C., "Discrete-Variable Optimization Problems," *Rand Symposium on Mathematical Programming*, Santa Monica, California, March 1959.
10. BERTIER, P., AND ROY, B., "Procédure de Résolution pour une Classe de Problèmes pouvant avoid un Caractère Combinatoire," *Cahiers du Centre D'Études de Recherche Opérationnelle* **6,** 202–208 (1964).
11. DAKIN, R. J., "A Tree-search Algorithm for Mixed Integer Programming Problems," *The Computer Journal* **8,** 250–255 (1965).
12. DRIEBECK, N. J., "An Algorithm for the Solution of Mixed Integer Programming Problems," Management Sci. **12,** 576–587 (1966).
13. EASTMAN, W. L., *Linear Programming with Pattern Constraints*, Ph.D. Thesis,

Report No. BL. 20, The Computation Laboratory, Harvard University (1958).

14. ———, "A Solution to the Traveling-Salesman Problem," presented at the *American Summer Meeting of the Econometric Society*, Cambridge, Massachusetts, August 1958.
15. Efroymson, M. A., and Ray, T. L., "A Branch-Bound Algorithm for Plant Location," Esso Research and Engineering Company (mimeographed) (1965).
16. Freeman, R. J., "Computational Experience with the Balas Integer Programming Algorithm," The Rand Corporation, P-3241, October 1965.
17. Gavett, J. W., "Three Heuristic Rules for Sequencing Jobs to a Single Production Facility," *Management Sci.* **11**, B-166–B-176 (1965).
18. Geoffrion, A., "Integer Programming by Implicit Enumeration and Balas' Method," The Rand Corporation, RM-4783-PR, February, 1966.
19. Gilmore, P. C., "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *J. Soc. Indust. Appl. Math.* **10**, 305–313 (1962).
20. ———, and Gomory, R. E., "A Linear Programming Approach to the Cutting-Stock Problem, Part II," *Opns. Res.* **11**, 863–888 (1963).
21. Golomb, S. W., and Baumert, L. D., "Backtrack Programming," *J. Assoc. for Comp. Mach.* **12**, 516–524 (1965).
22. Gomory, R. E., "An Algorithm for the Mixed Integer Problem," The Rand Corporation, RM-2597 (1960).
23. Haldi, J., "Twenty-Five Integer Programming Test Problems," Working Paper no. 43, Graduate School of Business, Stanford University, December 1964.
24. Hall, M., Jr., and Knuth, D. E., "Combinatorial Analysis and Computers," *Am. Math. Monthly* **72**, 21–28 (1965).
25. Ignall, E., and Schrage, L., "Application of the Branch-and-Bound Technique to Some Flow-Shop Scheduling Problems," *Opns. Res.* **13**, 400–412 (1965).
26. Jaeschke, G., "Eine allgemeine Methode zur Lösung kombinatorischer Probleme," *Ablanf- und Planungforschung* **5**, 133–155 (1964).
27. Karp, R. M., McFarlin, F. E., Roth, J. P., and Wilts, J. R., "A Computer Program for the Synthesis of Combinatorial Switching Circuits," *Proc. Second Annual Symp. on Switching Circuit Theory and Logical Design*, AIEE Publication S-134 (1961).
28. Land, A. H., and Doig, A., "An Automatic Method of Solving Discrete Programming Problems," *Econometrica* **28**, 497–520 (1960).
29. ———, "A Problem of Assignment with Interrelated Costs," *Opnal. Res. Quart.* **14**, 185–199 (1963).
30. Lawler, E. L., "The Quadratic Assignment Problem," *Management Sci.* **9**, 586–599 (1963).
31. ———, "An Analysis of Roth's Methods of Synthesis," presented at the *Seventh Midwest Symposium on Circuit Theory*, Ann Arbor, Michigan, May 1964.
32. ———, and Bell, M. D., "A Method for Solving Discrete Optimization Problems," te appear in *Opns. Res.*

33. Lawler, E. L., "Covering Problems: Duality Relations and a New Method of Solution," to appear in *Journal of SIAM*.
34. Little, J. D. C., Murty, K. G., Sweeney, D. W., and Karel, C., "An Algorithm for the Traveling-Salesman Problem," *Opns. Res.* **11**, 972–989 (1963).
35. Lomnicki, Z. A., "A 'Branch-and-Bound' Algorithm for the Exact Solution of the Three-machine Scheduling Problem," *Opnal. Res. Quart.* **16**, 89–100 (1965).
36. McCluskey, E. J., "Minimization of Boolean Functions," *Bell System Tech. J.* **35**, 1417–1444 (1956).
37. Pandit, S. N. N., "The Loading Problem," *Opns. Res.* **10**, 639–646 (1962).
38. ———, "Some Observations on the Longest Path Problem," *Opns. Res.* **12**, 361–364 (1964).
39. ———, "An Enumerational Approach to Integer Programming Problems," (to appear).
40. Rado, F., "Linear Programming with Logic Conditions," *Communicarile Academiei Republicil Populare Romine* **13**, 1039–1041 (1963).
41. Rossman, M. J., and Twery, R. J., "Combinatorial Programming," presented at the *Sixth Annual ORSA Meeting*, May 1958 (mimeographed).
42. ———, and Twery, R. J., "A Solution to the Traveling-Salesman Problem by Combinatorial Programming," Peat, Marwick, Mitchell and Co., Chicago (mimeographed).
43. Roth, A. P., and Wagner, E. G., "Algebraic Topological Methods for the Synthesis of Switching Systems. Part III: Minimization of Nonsingular Boolean Trees," *IBM J. of Res. Dev.* **3**, 326–344 (1959).
44. ———, "Minimization over Boolean Trees," *IBM J. of Res. Dev.* **4**, 543–558 (1960).
45. ———, and Karp, R. M., "Minimization over Boolean Graphs," *IBM J. of Res. Dev.* **6**, 227–238 (1962).
46. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. of Res. Dev.* **3**, 210–229 (1959).
47. Stanley, E. D., Honig, D. P., and Gainen, L., "Linear Programming in Bid Evaluation," *Naval Res. Log. Quart.* **1**, 48–54 (1954).
48. Sweeney, D. W., "The Exploration of a New Algorithm for Solving the Traveling-Salesman Problem," M.S. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (1963).
49. Walker, R. S., "An Enumerative Technique for a Class of Combinatorial Problems," *Am. Math. Soc. Symposium on Applied Math. Proc.* **10**, 91–94 (1960).

Reprinted from OPERATIONS RESEARCH
Vol. 14, No. 6, November-December, 1966
Printed in U.S.A.

A METHOD FOR SOLVING DISCRETE OPTIMIZATION PROBLEMS†

E. L. Lawler and M. D. Bell

The University of Michigan, Ann Arbor, Michigan

(Received February 28, 1966)

This paper describes a simple, easily-programmed method for solving discrete optimization problems with monotone objective functions and completely arbitrary (possibly nonconvex) constraints. The method is essentially one of partial enumeration, and is closely related to the 'lexicographic' algorithm of GILMORE AND GOMORY for the 'knapsack' problem and to the 'additive' algorithm of BALAS for the general integer linear programming problem. The results of a number of sample computations are reported. These indicate that the method is computationally feasible for problems in which the number of variables is fairly small.

IN THIS paper we describe a simple, easily programmed method for solving discrete optimization problems with monotone objective functions and arbitrary (possibly nonconvex) constraints. More specifically, the method is applicable to any problem that can be put into the form:

$$\left.\begin{array}{lc} & g_0(x), \\ \text{Minimize} & \\ \text{subject to} & g_{11}(x)-g_{12}(x)\geqq 0, \\ & g_{21}(x)-g_{22}(x)\geqq 0, \\ & \cdots \\ & g_{m1}(x)-g_{m2}(x)\geqq 0, \\ \text{where} & x=(x_1, x_2, \cdots, x_n), \\ \text{and} & x_j=0 \quad \text{or} \quad 1 \qquad (j-1, 2, \cdots, n), \end{array}\right\} \quad (1)$$

and where we apply the restriction that each of the functions $g_0, g_{11}, \cdots, g_{m2}$ *is monotone nondecreasing in each of the variables* $x_1, x_2, \cdots, x_n$.

Nonnegative integer variables can be transformed into binary variables by the method described in the section, "Problem Formulation." If necessary, an arbitrary objective function of the form

$$\text{Minimize} \quad g_0(x)$$

† The research reported in this paper was supported by NSF Grant GP-2778. The authors gratefully acknowledge the assistance of V. M. POWERS and D. E. WOOD, who provided programming support.

can be replaced by a monotone nonincreasing objective function by the simple trick:

$$\text{Minimize} \quad z$$

subject to

$$z - g_0(x) \geqq 0.$$

An arbitrary inequality constraint of the form $g_i'(x) \geqq 0$ can be replaced by an inequality constraint $g_i(x) \geqq 0$ involving a polynomial of degree $2^n - 1$. This polynomial can be separated into two monotone parts, $g_{i1}(x)$ and $g_{i2}(x)$ of the type required in (1). Thus, a problem can always be transformed into the form (1), provided its variables can be made to assume a finite number of discrete values.

The method is essentially one of partial enumeration, and is closely related to the 'lexicographic' method of GILMORE AND GOMORY[6] for the 'knapsack' problem and the 'additive' algorithm of BALAS,[1-3] for the general integer linear programming problem. In the second through fourth sections the reasoning behind the method is explained. In the fifth section is described in some detail how various problems can be transformed into the form (1). The results of a number of test computations are presented in the section, "Test Problems." Finally, in the last section, a comparison is made between the present method and the algorithms of Gilmore-Gomory and Balas.

SEQUENCES OF BINARY VECTORS

IN THIS section we discuss some preliminaries that are essential for the description of the algorithm.

We consider vectors x of the form $x = (x_1, x_2, \cdots, x_n)$, which are 'binary' in the sense that each x_j is either 0 or 1. We say that $x \leqq y$ if and only if $x_j \leqq y_j$, for $j = 1, 2, \cdots, n$. E.g., $x \leqq y$, where $x = (0, 1, 0)$ and $y = (0, 1, 1)$. This is the *vector partial ordering*.

There is also the lexicographic or *numerical ordering* of these vectors that is obtained by identifying with each vector x the integer value

$$n(x) = x_1 2^{n-1} + x_2 2^{n-2} + \cdots + x_n 2^0.$$

Note that the numerical ordering is a refinement of the vector partial ordering. I.e.,

$$x \leqq y \quad \text{implies} \quad n(x) \leqq n(y),$$

but

$$n(x) \leqq n(y) \quad \text{does not imply} \quad x \leqq y.$$

Suppose all binary n-vectors are listed in numerical order, i.e.,

$$(0, \cdots, 0, 0, 0),$$
$$(0, \cdots, 0, 0, 1),$$

$$(0, \cdots, 0, 1, 0),$$
$$(0, \cdots, 0, 1, 1),$$
$$(0, \cdots, 1, 0, 0),$$
$$\text{etc.}$$

Immediately following an arbitrary vector x, there may (or may not) be a number of vectors x' with the property that $x \leqq x'$. Roughly speaking, these are vectors that differ from x only in that they have 1's in place of one or more of the 'right-most' 0's of x. For example, immediately following $x = (0, 1, 0, 0)$ are $(0, 1, 0, 1)$, $(0, 1, 1, 0)$, and $(0, 1, 1, 1)$, each of which is greater than x in the vector partial ordering.

We let x^* denote the first vector following x in the numerical ordering that has the property that $x \leqq x^*$. For any given x, the vector x^* is very easily calculated on a computer as follows:

Treat x as a binary number:

(1) Subtract 1 from x,

(2) Logically 'or' x and $x-1$ to obtain x^*-1,

(3) Add 1 to obtain x^*.

Some examples:

Let $x=0101100$,
(1) $x-1=0101011$,
(2) $x^*-1=0101111$,
(3) $x^*=0110000$.

Let $x=0101011$,
(1) $x-1=0101010$,
(2) $x^*-1=0101011$,
(3) $x^*=0101100$.

Let $x=0101000$,
(1) $x-1=0100111$,
(2) $x^*-1=0101111$,
(3) $x^*=0110000$.

Note that x^*-1 is greater than each of $x, x+1, \cdots, x^*-2$, in the vector partial ordering.

A SIMPLE OPTIMIZATION PROBLEM

CONSIDER THE following simple optimization problem:
Minimize

$$g_0(x),$$

subject to

$$-g_1(x) \geqq 0,$$
$$g_2(x) \geqq 0,$$

where

$$x = (x_1, x_2, \cdots, x_n),$$

and $$x_j=0 \quad \text{or} \quad 1. \qquad (j=1, 2, \cdots, n)$$

Each of the functions g_0, g_1, and g_2 is assumed to be monotone nondecreasing in each of the variables $x_1, x_2, \cdots, x_n$.

We can solve this problem by examining each of the 2^n possible solution vectors in numerical order, beginning with $x=(0, 0, \cdots, 0)$, and ending with $(1, 1, \cdots, 1)$. However, this process can be considerably shortened by invoking certain rules, which are stated below.

As we proceed through the list of vectors, we keep a record of the least costly solution found to date. Let $\hat{x}$ denote this solution, which has cost $g_0(\hat{x})$. Let x denote the vector that is currently being examined. The following rules indicate conditions under which certain vectors in the numerical ordering can be skipped over.

Rule 1:

If $g_0(x) \geqq g_0(\hat{x})$, skip to x^*.

Justification:

Because g_0 is monotone nondecreasing, none of the vectors $x+1, x+2, \cdots, x^*-1$ can be less costly than $\hat{x}$.

Rule 2:

If x is a feasible solution, i.e., $-g_1(x) \geqq 0$ and $g_2(x) \geqq 0$, skip to x^*.

Justification:

Because g_0 is monotone nondecreasing, none of the vectors $x+1, x+2, \cdots, x^*-1$ can be less costly than x.

Of course, if Rule 2 applies and $g_0(x) < g_0(\hat{x})$, then x is substituted for x.

Rule 3:

If either $-g_1(x) \ngeqq 0$ or $g_2(x^*-1) \ngeqq 0$, skip to x^*.

Justification:

The function g_1 is monotone nondecreasing and $-g_1$ is monotone nonincreasing. Hence it follows that none of the vectors $x+1, x+2, \cdots, x^*-1$ can satisfy the constraint $-g_1(x) \geqq 0$ if x does not. Likewise, since g_2 is monotone nondecreasing, none of the vectors $x, x+1, \cdots, x^*-2$ can satisfy the constraint $g_2(x) \geqq 0$ if x^*-1 does not.

As an example, let

$$\begin{aligned} g_0(x_1, x_2, x_3, x_4) &= 3x_1+2x_2+x_3+x_4, \\ -g_1(x_1, x_2, x_3, x_4) &= -x_2x_3-x_4+1, \\ g_2(x_1, x_2, x_3, x_4) &= 2x_1+x_2x_3+x_4-3. \end{aligned}$$

A straightforward enumeration of all $2^4 = 16$ possible solution vectors yields the following results:

x_1	x_2	x_3	x_4	
0	0	0	0	Infeasible: $g_2(x) \ngeq 0$.
0	0	0	1	Ditto.
0	0	1	0	Ditto.
0	0	1	1	Ditto.
0	1	0	0	Ditto.
0	1	0	1	Ditto.
0	1	1	0	Ditto.
0	1	1	1	Ditto.
1	0	0	0	Ditto.
1	0	0	1	Feasible, $g_0(x) = 4$.
1	0	1	0	Infeasible: $g_2(x) \ngeq 0$.
1	0	1	1	Feasible, $g_0(x) = 5$.
1	1	0	0	Infeasible: $g_2(x) \ngeq 0$.
1	1	0	1	Feasible, $g_0(x) = 6$.
1	1	1	0	Feasible, $g_0(x) = 6$.
1	1	1	1	Infeasible: $-g_1(x) \ngeq 0$.

The application of the rules yields the following results:

x_1	x_2	x_3	x_4	
0	0	0	0	Skip to $x^* = (0001)$, by Rule 3.
0	0	0	1	Skip to $x^* = (0010)$, by Rule 3.
0	0	1	0	Skip to $x^* = (0100)$, by Rule 3.
0	1	0	0	Skip to $x^* = (1000)$, by Rule 3.
1	0	0	0	Infeasible: $g_2(x) \ngeq 0$.
1	0	0	1	Feasible, $g_0(x) = 4$. Skip to $x^* = (1010)$, by Rule 2.
1	0	1	0	Skip to $x^* = (1100)$, by Rule 1.
1	1	0	0	Skip to $x^* = 1(0000)$ [overflow], by Rule 1.

The rules allow only 8 possible solutions to be examined instead of 16. This is a rather modest reduction in computation, but one suspects that much greater savings will be made in problems with a larger number of variables. This suspicion will be shown to be borne out in practice.

A MORE GENERAL OPTIMIZATION PROBLEM

CONSIDER THE more general type of optimization problem mentioned in the first section, namely:

Minimize $g_0(x)$,

subject to
$$g_{11}(x)-g_{12}(x)\geqq 0,$$
$$g_{21}(x)-g_{22}(x)\geqq 0,$$
$$\cdots$$
$$g_{m1}(x)-g_{m2}(x)\geqq 0,$$

where $x=(x_1, x_2, \cdots, x_n)$,

and $x_j=0$ or 1, $(j=1, 2, \cdots, n)$,

subject to the restriction that each of the functions $g_0, g_{11}, g_{12}, \cdots, g_{m2}$ is monotone nondecreasing in each of the variables $x_1, x_2, \cdots, x_n$.

Rules 1 and 2 carry over exactly as they were stated in the previous section. Rule 3 is generalized very simply as follows:

Rule 3:

If, for any $i(i=1, 2, \cdots, m)$, $g_{i1}(x^*-1)-g_{i2}(x)\ngeqq 0$, skip to x^*.

Justification:

With respect to vectors in the interval $[x, x^*-1]$, x^*-1 maximizes g_{i1} and x maximizes $-g_{i2}$. Hence if it is the case that $g_{i1}(x^*-1)-g_{i2}(x)\ngeqq 0$, there can be no vector x' in the interval such that $g_{i1}(x')-g_{i2}(x')\geqq 0$.

If g_{i1} and g_{i2} are actually functions of disjoint sets of variables, it follows that

$$g_{i1}(x^*-1)-g_{i2}(x)=g_{i1}(x')-g_{i2}(x'),$$

for some vector x' in the interval $[x, x^*-1]$. As we shall see, this condition occurs when the optimization problem is derived from an integer linear programming problem.

The actual embodiment of these rules in a computer routine is shown in the flow chart in the section, "Test Problems."

PROBLEM FORMULATION

Problems involving nonnegative integer variables can be transformed into problems involving binary variables by means of the substitution

$$x_j=x_{j1}+2x_{j2}+2^2x_{j3}+\cdots+2^{K-1}x_{jK},$$

where $x_{ji}=0$ or $1(i=1, 2, \cdots, k)$. K is chosen to be sufficiently large for 2^K-1 to be an upper bound on the value of x_j.

Linear objective functions are made monotone by appropriate substitutions of variables, and the constraints of integer linear programming problems are put into the form of differences of monotone functions by appropriate grouping of positive and negative coefficients.

As an example, consider the problem

$$\text{Minimize} \qquad 6x_1 - x_2,$$

$$\text{subject to} \qquad 3x_1 - x_2 \geqq 4,$$
$$2x_1 + x_2 \geqq 3,$$
$$-x_1 - x_2 \geqq -3,$$

$$\text{and} \qquad x_1, x_2 \quad \text{nonnegative integers.}$$

The third constraint requires both x_1 and x_2 to be no greater than 3. Hence we apply the substitutions:

$$x_1 = x_{11} + 2x_{12},$$
$$x_2 = x_{21} + 2x_{22},$$

and obtain:

$$\text{Minimize} \qquad 6x_{11} + 12x_{12} - x_{21} - 2x_{22},$$

$$\text{subject to} \qquad 3x_{11} + 6x_{12} - x_{21} - 2x_{22} \geqq 4,$$
$$2x_{11} + 4x_{12} + x_{21} + 2x_{22} \geqq 3,$$
$$-x_{11} - 2x_{12} - x_{21} - 2x_{22} \geqq -3,$$

$$\text{and} \qquad x_{ij} = 0 \quad \text{or} \quad 1. \qquad (i = 1, 2; j = 1, 2)$$

We now apply to the substitutions

$$x_{21} = 1 - x'_{21},$$
$$x_{22} = 1 - x'_{22},$$

in order to make the objective function monotone nondecreasing, and obtain

$$\text{Minimize} \qquad 6x_{11} + 12x_{12} + x'_{21} + 2x'_{22} - 3,$$

$$\text{subject to} \qquad 3x_{11} + 6x_{12} + x'_{21} + 2x'_{22} - 7 \geqq 0,$$
$$2x_{11} + 4x_{12} - x'_{21} - 2x'_{22} \geqq 0,$$
$$-x_{11} - 2x_{12} + x'_{21} + 2x'_{22} \geqq 0,$$

$$\text{and} \qquad x_{11}, x_{12}, x'_{21}, x'_{22} = 0 \quad \text{or} \quad 1.$$

This is now an optimization problem in the form (1), with

$$g_0(x) = 6x_{11} + 12x_{12} + x'_{21} + 2x'_{22} - 3,$$

$$g_{11}(x) = 3x_{11}+6x_{12}+x'_{21}+2x'_{22}-7,$$
$$-g_{12}(x) = 0,$$
$$g_{21}(x) = 2x_{11}+4x_{12},$$
$$-g_{22}(x) = -x'_{21}-2x'_{22},$$
$$g_{31}(x) = x'_{21}+2x'_{22},$$
$$-g_{32}(x) = -x_{11}-2x_{12}.$$

TEST PROBLEMS

THE METHOD was tested on two series of test problems. The first series consisted of 11 problems with variously contrived constraints and objective functions, mostly nonlinear. The second series consisted of 33 randomly generated covering problems of various dimensions. For the first series of problems, a statistical tabulation was made of the total number of solutions that were examined, the number of times each of the Rules was applied, as well as running times on the IBM 7090 computer. For the second series of problems, only computer running times were recorded.

A complete flow chart of the computation is shown in Fig. 1. This flow chart gives a precise definition of the statistics that were tabulated. As can be seen by inspection, c_1 is a count of the total number of solutions that were examined. Count c_2 recorded the number of these solutions to which Rule 1 did not apply. Count c_3 recorded the number of the remaining solutions to which Rule 3 did not apply. Finally, c_4 recorded the number of times Rule 2 did apply to the residue.

The 11 problems in the first series were as follows:

Problem 1:

Minimize $$x_1^2+x_2^2+x_3^3+x_4^2+x_5^2,$$

subject to

$$x_1+2x_2+x_4 \geqq 4,$$
$$x_2+2x_3 \geqq 3,$$
$$x_1+2x_5 \geqq 5,$$
$$x_1+x_2+2x_3 \leqq 6,$$
$$2x_1+x_3 \leqq 4,$$
$$x_1+4x_5 \leqq 13,$$
$$x_j \leqq 3, \qquad (j=1, 2, \cdots, 5)$$

and $$x_j \geqq 0 \quad \text{and integer} \qquad (j=1, 2, \cdots, 5).$$

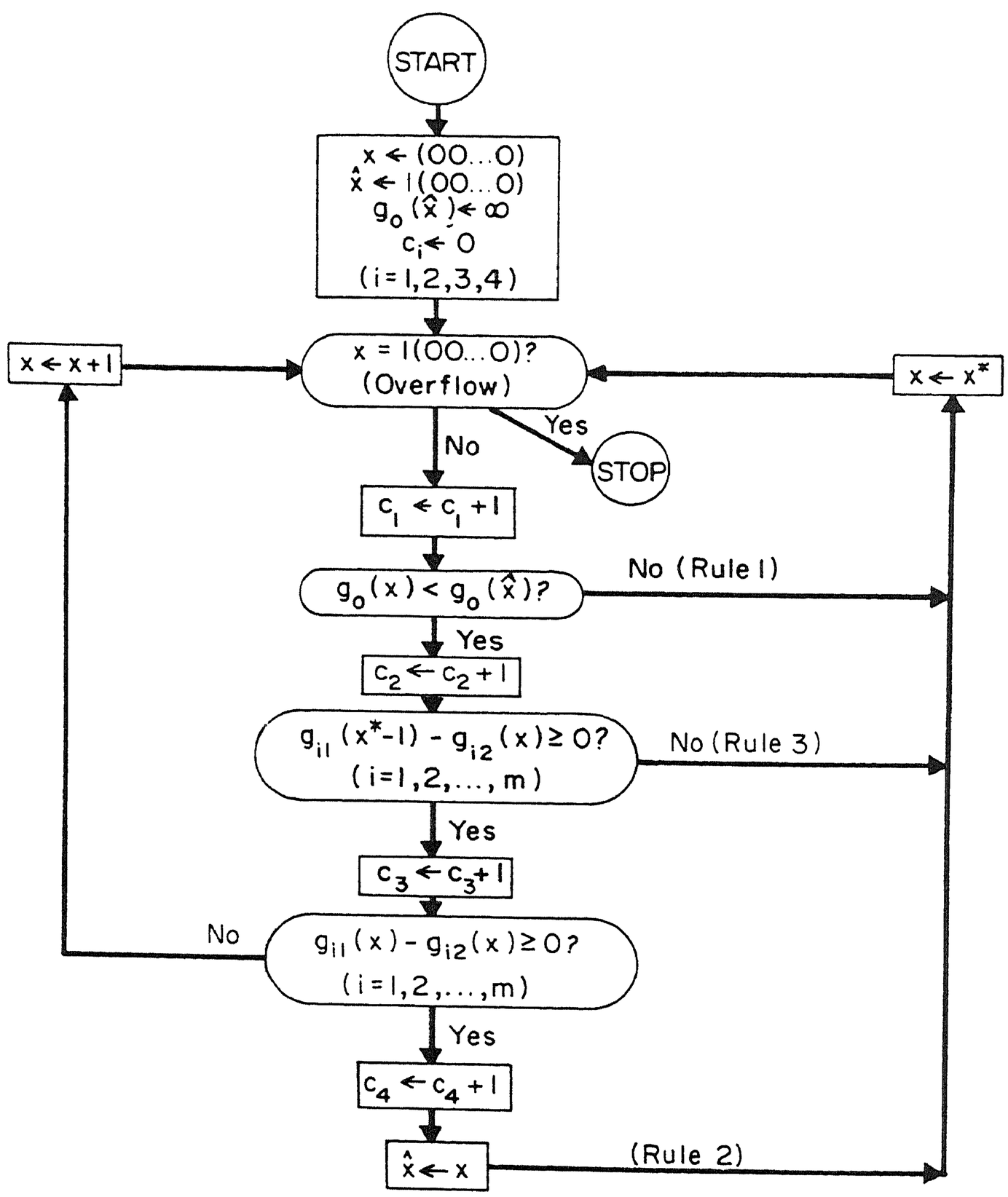


Fig. 1. Flow chart of computation.

Problem 2:

Minimize $$x_1x_7+3x_2x_6+x_3x_5+7x_4,$$

subject to

$$x_1+x_2+x_3 \geqq 6,$$

$$x_4+x_5+6x_6 \geqq 8,$$

$$x_1x_6+x_2+3x_5\geqq 7,$$
$$4x_2x_7+3x_4x_5\geqq 25,$$
$$3x_1+2x_3+x_5\geqq 7,$$
$$3x_1x_3+6x_4+4x_5\leqq 20,$$
$$4x_1+2x_3+x_6x_7\leqq 15,$$
$$x_4\leqq 15,$$
$$x_5\leqq 15,$$
$$x_j\leqq 7, \qquad (j=1, 2, 3, 6)$$

and $x_j\geqq 0$ and integer. $(j=1, 2, \cdots, 7)$

Problem 3:

Minimize $\sum_{j=1}^{j=7} x_j,$

subject to the same constraints as Problem 2.

Problem 4:

Minimize $\sum_{i=1}^{i=7} \sum_{j=1}^{j=7} x_ix_j-\sum_{j=1}^{j=7} x_j^2,$

subject to the same constraints as Problem 2.

Problem 5:

Minimize $\sum_{j=1}^{j=7} x_j^2,$

subject to the same constraints as Problem 2.

Problem 6:

Minimize $\sum_{i=1}^{i=7} \sum_{j=1}^{j=7} \sum_{k=1}^{k=7} x_ix_jx_k,$

subject to the same constraints as Problem 2.

Problem 7:

Minimize $3x_1x_4+x_2+x_2x_5+x_2x_7+6x_3x_8+x_5x_7+11x_6,$

subject to

$$2x_1+2x_4+8x_8\geqq 12,$$
$$11x_1+7x_4+13x_6\geqq 41,$$
$$6x_2+9x_4x_6+5x_7\geqq 60,$$
$$3x_2+5x_5+7x_8\geqq 42,$$
$$6x_2x_7+9x_3+5x_5\geqq 53,$$

$$4x_3x_7+x_5 \geqq 13,$$
$$2x_1+4x_2+7x_4+3x_5+x_7 \leqq 69,$$
$$9x_1x_8+6x_3x_5+4x_3x_7 \leqq 47,$$
$$12x_2+8x_2x_4+2x_3x_6 \leqq 73,$$
$$x_3+4x_5+2x_6+9x_8 \leqq 31,$$
$$x_2 \leqq 15,$$
$$x_5 \leqq 15,$$
$$x_7 \leqq 15,$$
$$x_j \leqq 7, \qquad (j=1, 3, 4, 6, 8)$$
$$x_j \geqq 0 \quad \text{and integer.} \qquad (j=1, 2, \cdots, 8)$$

Problem 8:

Minimize $\sum_{j=1}^{j=8} x_j^2$,

subject to the same constraints as Problem 7.

Problem 9:

Minimize $\sum_{j=1}^{j=8} x_j$,

subject to the same constraints as Problem 7.

Problem 10:

Minimize $x_1x_2x_3+x_1x_4x_5+x_2x_4x_6+x_2x_5x_7+x_6x_7x_8$,

subject to the same constraints as Problem 7.

Problem 11:

A 'covering' problem of the form

Minimize $x_1+x_2+x_3+\cdots+x_{15}$,

subject to $Ax \geqq 1$,

and $x_j=0 \quad \text{or} \quad 1. \qquad (j=1, 2, \cdots, 15)$

The matrix A is of the form

$$A = \begin{Vmatrix} Z & E & O \\ O & Z & E \\ E & O & Z \\ I & I & I \end{Vmatrix},$$

where

$$Z = \begin{Vmatrix} 00110 \\ 00011 \\ 10001 \\ 11000 \\ 01100 \\ 01001 \\ 10100 \\ 01010 \\ 00101 \\ 10010 \end{Vmatrix}, \qquad E = \begin{Vmatrix} 10000 \\ 01000 \\ 00100 \\ 00010 \\ 00001 \\ 10000 \\ 01000 \\ 00100 \\ 00010 \\ 00001 \end{Vmatrix},$$

and I is the 5×5 identity matrix. (This problem was suggested by FULKERSON AND RYSER.[5])

Each of these problems was transformed into the standard form with binary variables. Intuition indicated that the algorithm would be most efficient if those variables that were—roughly speaking—'least significant' were assigned positions at the 'right-hand' end of the solution vector. In order to verify this heuristic judgment, two different orderings of the variables were tested. Each nonnegative integer variable was transformed as follows:

$$x_j = x_{j1} + 2x_{j2} + \cdots + 2^{K-1}x_{jK}.$$

Two orderings were as follows:

Ordering 1:

$$x = (x_{1K}, x_{2K}, \cdots, x_{nK}, x_{1(K-1)}, \cdots, x_{n2}, x_{11}, x_{21}, \cdots, x_{n1}).$$

Ordering 2:

$$x = (x_{1K}, x_{1(K-1)}, \cdots, x_{11}, x_{21}, \cdots, x_{(n-1)1}, x_{nK}, \cdots, x_{n1}).$$

The results of the computations for the first series of test problems are summarized in Table I. The following conclusions can be drawn:

1. The nature of the objective function seems to be relatively unimportant. Problems 2–6, which shared the same set of constraints and differed only in objective function, all required about the same amount of computation; this was also true of Problems 7–10.

2. The proper ordering of the variables appears to be of great importance. Ordering 1 gave consistently better results than Ordering 2.

3. The count c_4 was consistently very small, indicating that the first feasible solution found by the method was likely either to be optimal or near-optimal.

The second series of test problems consisted of 33 covering problems of

TABLE I

COMPUTATIONAL RESULTS FOR FIRST SERIES OF PROBLEMS

Problem	Number (o. 1) variables	Ordering 1					Ordering 2					Optimal solution
		c_1	c_2	c_3	c_4	7090 Comp. time (sec)	c_1	c_2	c_3	c_4	7090 Comp. time (sec)	
1	10	123	118	37	1	0.5	206	189	107	3	0.65	(1, 1, 1, 1, 2)
2	23	3,496	2,782	818	5	9.9	10,477	5,990	1,860	5	24.0	(1, 4, 1, 0, 2, 1, 2)
3	23	1,954	1,605	458	1	5.4	6,233	2,934	1,053	2	12.9	(0, 6, 0, 1, 1, 1, 1)
4	23	1,837	1,383	404	2	5.65	4,219	1,822	594	3	10.6	(0, 7, 0, 0, 0, 2, 1)
5	23	1,710	1,322	425	5	4.7	5,127	3,268	1,111	6	12.6	(1, 4, 1, 1, 1, 1, 2)
6	23	1,733	1,414	398	3	6.7	3,550	1,536	438	2	11.4	(0, 7, 0, 0, 0, 2, 1)
7	27	11,497	10,142	1,963	2	41.7	194,126	147,782	37,270	2	634.4	(2, 4, 1, 4, 6, 1, 2, 0)
8	27	7,330	6,354	1,225	1	26.3	92,440	74,103	18,890	1	304.1	(3, 4, 1, 3, 6, 1, 2, 0)
9	27	10,795	9,816	1,907	1	38.3	150,769	112,082	29,144	1	457.7	(3, 4, 1, 3, 6, 1, 2, 0)
10	27	15,443	13,703	2,843	2	55.9	200,810	177,186	39,640	3	716.1	(5, 4, 1, 1, 6, 3, 2, 0)
11	15	7,654	5,666	2,961	1	29.4	—	—	—	—	—	315 optimal solutions exist

the form:

$$\text{Minimize} \qquad \sum c_j x_j,$$

$$\text{subject to} \qquad Ax \geqq 1,$$

$$\text{and} \qquad x_j = 0 \quad \text{or} \quad 1. \qquad (j = 1, 2, \cdots, n)$$

The (0–1)-elements of A were randomly generated for various specified densities of 1's. Each element c_j of the objective function was set equal to the number of 1's in the jth column of A. According to BALINSKI,[4]

TABLE II

COMPUTATIONAL RESULTS FOR SECOND SERIES OF PROBLEMS

Dimension	Density	No. problems	Range in 7090 comp. time ($\frac{1}{60}$ sec)
10 × 10	0.20	2	6–17
	0.25	6	8–30
	0.30	3	11–21
	0.40	3	8–9
	0.45	1	8
	0.50	5	1–12
15 × 15	0.25	3	98–323
20 × 20	0.25	3	535–3540
	0.50	1	139
25 × 25	0.25	3	3888–9139
30 × 30	0.25	2	39,744 – >79,416[a]
35 × 35	0.25	1	>33,160[a]

[a] Computation not completed.

this type of randomly generated problem has proved to be particularly difficult for cutting-plane methods of solution.

The results of the computations for this second series of problems are summarized in Table II. This table shows that, for example, running times for 15 × 15 problems ranged from 98/60 to 323/60 sec. Although the length of the computations does not seem to grow quite exponentially with the dimension of the problem, the growth is nevertheless quite rapid. The solution of covering problems larger than 20 × 20 by the direct application of this method does not seem to be warranted.

In conclusion, it would seem that this method could be useful for solving nonlinear optimization problems with unusual nonlinear objective functions and constraints, and for which no other satisfactory method of solution exists. Although the number of constraints does not seem to be particularly important, it is necessary that the number of (0,1) variables be reasonably small.

COMPARISON WITH OTHER METHODS

THE METHOD of partial enumeration presented in this paper is quite similar to the method of solution proposed by Gilmore and Gomory[6] for the so-called 'knapsack' problem. The only essential differences are:

1. Gilmore and Gomory choose to work with integer rather than binary variables. Moreover, because their problem is formulated as one involving the maximization of a monotone increasing linear function, they process vectors in reverse numerical order.

2. Because of the special nature of the problem—linear objective function and single linear constraint—they are able to use a stronger rule than Rule 1. In effect, their rule involves the calculation of a sharper bound on the cost of an optimal solution in the interval $[x, x^*-1]$ than that given by $g_0(x)$.

The present method also has much in common with various methods of the 'branch-and-bound' variety. In fact, when the method is applied to integer linear programming problems, the computations are essentially equivalent to the 'additive' algorithm proposed by Balas, [1–3], with the exception that the order in which branching is carried out on the variables is fixed instead of variable. In effect, x_1 is assigned a definite value, then x_2, then x_3, $\cdots$, and finally x_n. This is definitely a disadvantage, in the sense that a larger number of solution vectors must be examined than if the order of variables is allowed to be variable. However, it is believed that this disadvantage may be compensated for by the extreme ease of programming, the almost complete absence of housekeeping operations (no lists, no pushdown storage routines), and the very small amount of storage that is required.

REFERENCES

1. E. BALAS, "Un algorithme additif pour la résolution des programmes lineaires en variables bivalentes," *C. R. Acad. Sc. Paris* **258,** 3817–20 (1964).
2. ———, "Extension de l'algorithme additif à la programmation en nombres entiers et à la programmation non linéaire," *C. R. Acad. Sc. Paris,* **258,** 5136–5139 (1964).
3. ———, "An additive Algorithm for Solving Linear Programs with 0–1 Variables," *Op. Res.* **13,** 517–549 (1965).
4. M. L. BALINSKI, private communication (January, 1965).
5. D. R. FULKERSON AND H. J. RYSER, "Width Sequences for Special Classes of (0, 1)-Matrices," Rand Memorandum RM-2898-PR (March 1962).
6. P. C. GILMORE AND R. E. GOMORY, "A Linear Programming Approach to the Cutting Stock Problem—Part II," *Opns. Res.* **11,** 863–888 (1963).
7. E. L. LAWLER AND D. E. WOOD, "Branch-and-Bound Methods, A Survey," *Opns. Res.* **14,** 699–719 (1966).

Theory of Graphs, International Symposium, edited by P. Rosenstiehl.
Gordon and Breach, New York, 1967, pp. 209–213. Reprinted by permission.

OPTIMAL CYCLES IN DOUBLY WEIGHTED DIRECTED LINEAR GRAPHS

E. L. LAWLER*

RÉSUMÉ.

Cycles optimaux de graphes linéaires orientés doublement pondérés.

Le problème suivant se pose dans des questions d'ordonnancement et de contrôle optimal. Soit G un graphe orienté où deux nombres réels quelconques (positifs ou négatifs) sont associés à tout arc $u(i, j)$: les poids a_{ij} et b_{ij} ; on doit trouver un cycle C pour lequel le quotient des sommes des poids

$$q(C) = \frac{\sum_{(i,j) \in C} a_{ij}}{\sum_{(i,j) \in C} b_{ij}} \text{ est minimal .}$$

Dans cette communication on décrit les méthodes de calcul pour le cas général et pour les cas particuliers suivants :

(1) les b_{ij} sont tous égaux à 1 ;

(2) les b_{ij} sont des entiers non négatifs. Pour chaque cas, on détermine une borne algébrique de la longueur du calcul.

* * *

1. Introduction.

A problem which arises in a variety of contexts is the following. Given a directed graph G in which each directed edge (i, j) is assigned two arbitrary (positive or negative) real number *weights*, a_{ij} and b_{ij}, find a cycle C for which the objective function,

$$q(C) = \frac{\sum_{(i,j) \in C} a_{ij}}{\sum_{(i,j) \in C} b_{ij}}$$

is minimal (or maximal, as desired).

Each of the following problems can easily be put into the form of the optimal cycle problem :

(1) optimal control of a simple type of deterministic, finite-state system, such as that considered by Dantzig, et al., [1],

(*) Systems Engineering Laboratory, Department of Electrical Engineering. The University of Michigan, Ann Arbor, Michigan, U. S. A.

(2) an industrial scheduling problem, formulated by Cunninghame-Green [2],

(3) the determination of a maximum-computation-rate periodic schedule for Karp-Miller computation graphs, as described by Reiter [3].

Of course, special restrictions apply to each of these problems. For example, in the scheduling problem of Dantzig, in which each weight a_{ij} represents « cost », « distance », or « profit », and each b_{ij} represents « transit time », it is natural to assume that the b_{ij} are all strictly positive.

Furthermore, the still more special case in which all b_{ij}'s are unity is also of particular interest. In fact, in this case one may be interested only in knowing if there exists a cycle C for which $q(C)$ is negative. This « negative cycle » problem is intimately related to the classical assignment and transportation problems, and to the directed case of the Chinese Postman's problem, solved by Edmonds [4].

2. Special Case : $b_{ij} = 1$.

We begin with the special case in which all $b_{ij} = 1$. This case can be solved simply and directly by adaptation of a shortest-route algorithm.

If we interpret the weights a_{ij} as distances, we can let

$u_{ij}(m)$ = the length of the shortest chain from i to j which contains *exactly* m edges.

Then we have

$$u_{ij}(m) = \min_k \{ a_{ik} + u_{kj}(m-1) \}$$
$$u_{ij}(1) = a_{ij}, \text{ if there is an edge from } i \text{ to } j,$$
$$= +\infty \text{ otherwise.}$$

(We assume no self-loops.)

Since we wish to find a cycle for which the mean edge weight is minimal we simply determine the smallest quotient of the form $u_{ii}(m)/m$. Its value necessarily indicates the minimum mean edge weight of any cycle.

Let m^* denote the smallest value of m for which $u_{ii}(m)/m$ is minimal. Then there must be a simple cycle containing exactly m^* edges for which $q(C)$ is minimal. If this were not so, there would be a cycle of m^* edges which could be decomposed into two or more simple cycles, each containing fewer than m^* edges, and at least one of which would have a value of $q(C)$ at least as small as that for the original cycle. But this would be contrary to assumption.

In order to actually determine a simple cycle for which $q(C) = u_{ii}(m^*)/m^*$, we simply find an edge (i, k) such that

$$u_{ii}(m^*) = a_{ik} + u_{ki}(m^* - 1) ,$$

and then an edge (k, l), such that

$$u_{kj}(m^* - 1) = a_{kl} + u_{li}(m^* - 2) ,$$

etc., until the complete cycle has been determined.

Consider now the length of computation as a function of n, the number of nodes in the graph. The $u_{ij}(m)$ must be computed for i, j, and $m = 1, 2, 3, \ldots, n$, for a total of n^3 values. Moreover, each $u_{ij}(m)$ is determined by a minimization over $n - 1$ alternatives. Hence, the length of the computation is of the order of n^4.

3. Determination of Negative Cycles.

The problem of determining a negative cycle, i. e., a cycle C for which $\sum_{(i,j)\in C} a_{ij} < 0$, is quite similar : one simply looks for terms of the form $u_{ii}(m)$ which are negative. The procedure can be shortened, however, to take advantage of the fact that one need not know the exact number of edges used to obtain a given value u_{ij}. For example, one may use a method such as that of T. C. Hu [5]. Let

$u'_{ij}(m)$ = the length of the shortest chain from i to j, which *does not* pass through nodes $m + 1, m + 2, \ldots, n$.

Then we have a recursion of the form :

$$u'_{ij}(m) = \min \{ u'_{ij}(m-1), u'_{im}(m-1) + u'_{mj}(m-1) \},$$
$$u'_{ij}(0) = a_{ij}, \text{ if there is an edge from } i \text{ to } j,$$
$$= +\infty \text{ otherwise.}$$

A negative cycle exists if and only if one of the values $u'_{ii}(m)$ is negative. There are n^3 such values $u'_{ij}(m)$ to be determined, and each value requires minimization over exactly two alternatives. Hence the length of the negative cycle computation is of order n^3.

4. Special Case : b_{ij} nonnegative integers.

The case in which the coefficients b_{ij} are nonnegative integers is closely related to the cases treated above.

Although we allow some of the b_{ij} to be zero, we assume that the sum of the b_{ij}'s about any cycle is nonzero. This condition is easy enough to check.

The recursion equations ave now formulated as follows :

$u_{ij}(m)$ = the length of the shortest chain from i to j for which the sum of the b_{ij}'s is *exactly* m.

Let $u_{ij}^{(p)}(m)$ be the p^{th} approximation in a succession of approximations of $u_{ij}(m)$, where

$$u_{ij}^{(0)}(m) \geqslant u_{ij}^{(1)}(m) \geqslant \cdots \geqslant u_{ij}^{(n)}(m) = u_{ij}(m) .$$

Then

$$u_{ij}^{(0)}(m) = a_{ij}, \text{ if there is an edge from } i \text{ to } j \text{ and if } b_{ij} = m \text{ for that edge.}$$
$$= +\infty \text{ otherwise}$$

and

$$u_{ij}^{(p)}(m) = \min_k \{ a_{ik} + u_{kj}^{(p-1)}(m - b_{ik}) \} .$$

Note that this system of successive approximations is necessitated by the implicit functional relations that would otherwise arise. These equations must be solved for i, j and $p = 1, 2, \ldots, n$, and for $m = 1, 2, \ldots, M$, where M is an upper bound on the sum of the b_{ij}'s around any cycle. Each term of the recursion involves minimization over $n - 1$ alternatives. Hence the overall length of the computation is of the order of $n^4 M$.

5. General Case.

We now describe a computational method for the general case in which the coefficients b_{ij} are arbitrary (positive, negative, or zero) real numbers, but subject to the restriction that $\sum_{(i,j)\in C} b_{ij} > 0$ for all cycles C.

Let us choose a trial value q^* for the objective function. We then carry out a negative cycle computation on the same graph but with weights a_{ij}^*, where

$$a_{ij}^* = a_{ij} - q^* b_{ij} .$$

Three possible cases result :

(1) There exists a negative cycle, i. e. a cycle C for which

$$\sum_{(i,j)\in C} a_{ij}^* < 0 .$$

(2) There exists no negative cycle, and the length of the shortest cycle is strictly positive, i. e. $\sum_{(i,j)\in C} a_{ij}^* > 0$, for all cycles C.

(3) There exists no negative cycle, and the length of the shortest cycle is exactly zero.

In the case of (1), we know that the optimum value of q must be strictly less than q^*. This is because

$$\sum_{(i,j)\in C} a_{ij}^* = \sum_{(i,j)\in C} a_{ij} - q^* \sum_{(i,j)\in C} b_{ij} < 0 ,$$

which holds if and only if

$$\frac{\sum_{(i,j)\in C} a_{ij}}{\sum_{(i,j)\in C} b_{ij}} < q^* .$$

(Note that the denominator is assumed to be strictly positive.)

By similar reasoning, we know that in case (2), the optimum value of q is strictly greater than q^*. And in case (3), the optimum value of q is exactly q^*.

We thus have the basis of a search procedure : First try an arbitrary value q^*. If it is too large, try a smaller value. If it is too small, try a larger value. Then, when upper and lower bounds on the optimum have been established, continue with a binary search procedure which halves the interval of uncertainty at each iteration. The reader should have no difficulty in discovering how to determine a cycle C for which $q(C)$ is within the interval.

Each iteration requires a negative cycle computation whose length is of order n^3. The number of iterations is proportional to the number of significant digits of accuracy required, or to $|\log \varepsilon|$, where ε is the desired interval of uncertainty. Hence the overall computation is of order $n^3 |\log \varepsilon|$.

6. Conclusion.

The methods of solution described in this paper are alternatives to the linear programming method proposed by Dantzig [1]. It is interesting that whereas no good bound on the length of computation is known for the linear programming method, strict algebraic bounds can be obtained for each of the methods described in this paper.

Acknowledgement.

This research was supported in part by National Science Foundation Grant GP-2778, and in part by Air Force Contract AF-30 (602)-3546. This version of the paper incorporates various comments and suggestions made by participants of the conference.

REFERENCES

[1] DANTZIG, G. B., BLATTNER, W. and RAO, M. R., « Finding a Cycle with Minimum Cost to Time Ratio with Application to a Ship Routing Problem », paper given at this conference.

[2] CUNNINGHAME-GREEN, R. A., « Describing Industrial Processes with Interference and Approximating their Steady-State Behaviour », *Operational Research Quarterly*, vol. 13, n° 1, pp. 95-100.

[3] REITER, R., « Initiation Timing in a Model for Parallel Computation », Systems Engineering Laboratory Tech. Report SEL-66-3, The University of Michigan, Ann Arbor, March 1966.

[4] BUSAKER, R. G. and SAATY, T. L., *Finite Graphs and Networks*, McGraw Hill, 1965, p. 179.

[5] HU, T. C., « Revised Matrix Algorithms for Shortest Paths », Research Paper RC-1478, IBM Watson Research Center, Yorktown Heights, N. Y., September 28, 1965.

MANAGEMENT SCIENCE
Vol. 16, No. 1, September, 1969
Printed in U.S.A.

A FUNCTIONAL EQUATION AND ITS APPLICATION TO RESOURCE ALLOCATION AND SEQUENCING PROBLEMS*,**

E. L. LAWLER[1] AND J. M. MOORE[2]

A functional equation, similar to that formulated for the knapsack problem, is applied to the solution of a problem of resource allocation in critical path scheduling, and to a variety of single-machine sequencing problems with deadlines and loss functions. Among these are: minimization of the weighted number of tardy jobs, maximization of weighted earliness, minimization of tardiness with respect to common relative and absolute deadlines, and minimization of weighted tardiness with respect to a common deadline. Extensions to multiple-machine sequencing problems are discussed, and the sequencing of two machines in series is treated in detail.

1. Introduction

Suppose n jobs are to be performed one at a time in the fixed order $1, 2, \cdots, n$. Each job can be performed in either one of two different modes. The processing of job j requires a_j units of time in one mode and b_j units in the other. In the first mode, a loss of $\alpha_j(t)$ units is incurred upon the completion of the job at time t, and $\beta_j(t)$ units in the other. What assignment of modes to jobs and what timing of the jobs will minimize the total loss?

Let $f(j, t)$ = the minimum total loss for the first j jobs, subject to the constraint that job j is completed no later than time t.

By the usual dynamic programming argumentation:

$$f(0, t) = 0, \qquad (t \geqq 0),$$

$$f(j, t) = +\infty, \qquad (j = 0, 1, \cdots, n; t < 0),$$

$$f(j, t) = \min \begin{Bmatrix} f(j, t-1), \\ \alpha_j(t) + f(j-1, t-a_j), \\ \beta_j(t) + f(j-1, t-b_j) \end{Bmatrix}, \qquad (j = 1, 2, \cdots, n; t \geqq 0). \tag{1}$$

The problem is solved by the calculation of $f(n, T)$, where T is a sufficiently large number. For example, if all of the α_j's and β_j's are monotone nondecreasing, we may choose

$$T = \sum_{j=1}^{n} \max \{a_j, b_j\}.$$

The overall computation requires on the order of nT computational steps.

We shall show that the functional equation (1) can be used to solve a variety of sequencing and scheduling problems, including several which do not at all involve a fixed order for the processing of jobs.

The computations for these sequencing problems are vastly more efficient than, for example, the computational methods of Held and Karp [1], Schild and Fredman [9],

* Received January 1968 and revised January 1969.

** Some of the material in this paper was presented at the 34th National ORSA Meeting, Philadelphia, November 1968, under the title, "Optimal Sequencing of Jobs Subject to Deadlines."

[1] The University of Michigan, currently visiting the University of California at Berkeley.
[2] Esso Mathematics and Systems, Incorporated.

or Lawler [4]. The reason for this is that those methods carry out an implicit search over n ! possibilities, whereas ours is a search over only 2^n.

2. Sequencing with Deadlines and Precedence Constraints

In order to illustrate the initial application of Equation (1), we must first solve the following problem. Suppose n jobs are to be processed by a single machine. Job j requires a_j units of processing time and it has a deadline d_j for its completion. There are certain precedence constraints which must be satisfied, and these are expressed in the form of a partial ordering relation "ρ" on the jobs; if $i\rho j$, then job i must precede j. Does there exist a sequence for the jobs, consistent with the given precedence constraints, such that each job is completed by its dealine?

Assume, without loss of generality, that the jobs are numbered in such a way that $i\rho j$ implies $i \leqq j$. (There are a number of possible schemes for obtaining such a numbering.) Now, for $j = 1, 2, \cdots, n$, let

$$\bar{d}_j = \min \{d_k \mid j\rho k\} + j\epsilon,$$

where ϵ is a small number and $j\epsilon$ is used to break ties. It is important to note that $i\rho j$ implies $\bar{d}_i < \bar{d}_j$.

Theorem: Each job can be completed on time, consistent with the given precedence constraints, if and only if each job is completed on time in the sequence obtained by ordering jobs according to increasing values of $\bar{d}_j$.

Proof: The "if" part is obvious, so we consider the situation in which there exists a hypothetical sequence in which all jobs are completed on time, but in which the jobs are not ordered according to increasing values of $\bar{d}_j$. Let i, j be a consecutive pair of jobs in this hypothetical sequence, with $\bar{d}_i > \bar{d}_j$. It cannot be the case that $i\rho j$, because that would imply $\bar{d}_i < \bar{d}_j$, contrary to assumption. Hence, if we interchange the positions of i and j, we do not violate the precedence constraints.

A little further analysis reveals that either j, or some successor of j, has a dealine at least as early as that of i. Since j, or its successor, is on time in the given sequence, placing i in the position of j will maintain the on-time status of i. And, of course, if i and j are interchanged, j will remain on time since it will be earlier in the sequence.

Thus, i and j can be interchanged in the sequence. And, by a finite number of such interchanges, the hypothetical sequence can be transformed into one in which the jobs are placed in order of increasing values of $\bar{d}_j$, with all jobs on time, and with the precedence constraints satisfied. *Q.E.D.*

This theorem generalizes the result of Jackson [2], reported by Smith [10], to the effect that an unrestricted set of jobs can all be completed on time if and only if they are completed on time when sequenced according to deadlines, earliest first. Operationally, the generalized rule can be stated as follows: When operating a machine, always take next from among the jobs which are currently available (*i.e.*, jobs none of whose predecessors remain unprocessed), a job which has a successor with the earliest possible deadline (considering a job to be one of its own successors).

It is particularly important to note that the sequence specified by the theorem is completely independent of processing times. This fact makes possible the solution method described in the next section.

3. Resource Allocation in Critical Path Scheduling

Consider a project which is composed of a large number of tasks related by a complex set of precedence constraints. A critical path analysis is employed to determine a

deadline for each individual task. Suppose a certain subset of tasks can be performed only one at a time (perhaps because of the limited availability of specialized personnel or equipment). For each task j in the subset, there is a choice between a long processing time a_j at low cost α_j or a short processing time b_j at high cost β_j. How should the tasks be processed with least cost so that all deadlines are met?

The answer is to order the jobs according to the generalized dealine rule given in the theorem, and then to employ Equation (1) in a straightforward way. This requires on the order of nd_n computational steps.

4. Application to Single-Machine Sequencing Problems

In the next several sections, we describe the application of Equation (1) to a number of single-machine sequencing problems involving deadlines and loss functions. The essential problem formulation is as follows. A set of n jobs are to be performed in sequence—one immediately following the other—by a single machine. Job j requires a'_j units of processing time, and a loss of $c_j(t)$ is incurred for its completion at time t. The problem is to find a permutation π (where $\pi(j) = k$ if job j is the k^{th} to be performed) such that

$$\sum_{j=1}^{n} c_j(t_j)$$

is minimized, where

$$t_j = \sum_{k=1}^{\pi(j)} a'_{\pi^{-1}(k)}$$

is the time of completion of job j.

This is essentially the general problem which Held and Karp [1] solved with on the order of $n2^n$ computational steps. By contrast, we shall deal with a number of special instances that can be solved with algebraic, rather than exponential, growth in the number of computational steps.

The trick in all of these applications is to notice that, for certain types of loss functions $c_j(t)$, the jobs can be partitioned into two classes. One class of jobs will be performed in a predetermined order, and the other in an arbitrary order, either preceding or following the first class. The actual assignment of jobs to classes is determined by the solution of Equation (1) with "modes" corresponding to classes.

5. Minimization of Weighted Number of Tardy Jobs

Suppose the loss functions $c_j(t)$ are of the form

$$c_j(t) = 0, \qquad (t \leqq d_j),$$

$$= pj, \qquad (t > d_j).$$

That is, a penalty p_j is exacted if job j is completed later than its assigned deadline d_j. How should the jobs be sequenced so as to minimize the total loss, *i.e.*, the weighted number of tardy jobs?

Following the reasoning of Moore [6] (who dealt with the special case $p_j = 1$) and Rothkopf [8], we note that the problem effectively requires us to partition the jobs into two classes: those which are to be on time, and those which are to be tardy. We can assume that the on-time jobs will be sequenced in order of their deadlines with the tardy jobs following them in arbitrary order.

The problem is solved by ordering the jobs by deadlines, earliest deadline first, and

then applying Equation (1) with

$$a_j = a_j', \qquad \alpha_j(t) = 0$$
$$b_j = 0, \qquad \beta_j(t) = p_j.$$

Whichever jobs are given zero processing times in the solution of (1) will be deemed to be tardy jobs.

6. Relation to Knapsack Problem

The reader may already have been struck by the resemblance between Equation (1) and functional equations which are used to solve the knapsack problem.

Using consistent notation, the knapsack problem is:

$$\text{Maximize } \sum_{j=1}^{n} p_j x_j$$
$$\text{Subject to } a'_1 x_1 + a'_2 x_2 + \cdots + a'_n x_n \leqq d,$$
$$x_j = 0 \quad \text{or} \quad 1 \quad (j = 1, 2, \cdots, n).$$

It can be solved by the equation

$$f(j, t) = \max \begin{Bmatrix} f(j, t-1), \\ f(j-1, t), \\ p_j + f(j-1, t-a'_j) \end{Bmatrix}, \qquad (t \leqq d), \tag{2}$$
$$= f(j, d), \qquad (t > d).$$

A little manipulation shows that the problem in the previous section is equivalent to:

$$\text{Maximize } \sum_{j=1}^{n} p_j x_j$$
$$\text{Subject to } a'_1 x_1 \leqq d_1$$
$$a'_1 x_1 + a'_2 x_2 \leqq d_2$$
$$a'_1 x_1 + a'_2 x_2 + a'_3 x_3 \leqq d_3$$
$$\vdots$$
$$a'_1 x_1 + a'_2 x_2 + \cdots + a'_n x_n \leqq d_n$$

where

$x_j = 0$ if job j is to be tardy,

$= 1$ if job j is to be on time.

For this problem, Equation (1) becomes equivalent to

$$f(j, t) = \max \begin{Bmatrix} f(j, t-1), \\ f(j-1, t), \\ p_j + f(j-1, t-a'_j) \end{Bmatrix}, \qquad (t \leqq d_j) \tag{3}$$
$$= f(j, d_j). \qquad (t > d_j)$$

This problem formulation and method of solution were previously noted by Rothkopf [8].

Note: In both (2) and (3) above, the term $f(j, t-1)$ is dominated by the other two alternatives and can be eliminated; this term has been included so as to point up the correspondence to (1).

7. Linear Loss Functions, Common Deadline

Again suppose n jobs are to be processed on a single machine, with loss functions of the form

$$\begin{aligned} c_j(t) &= p_j t, \qquad (t \leqq d), \\ &= q_j, \qquad (t > d), \end{aligned}$$

where d is a deadline common to all jobs. How should the jobs be sequenced so as to minimize the total loss?

The problem effectively requires us to partition the jobs into two classes: those which are completed on or before the deadline, and those which are tardy. The on-time jobs will be sequenced according to the ratios p_j/a'_j (see Smith [10] and McNaughton [5]), followed by the tardy jobs in arbitrary order.

The problem is solved by ordering the jobs by the ratios p_j/a_j', the job with the largest ratio first, and applying Equation (1) with

$$\begin{aligned} a_j &= a'_j, \qquad \alpha_j(t) = p_j t \\ b_j &= 0, \qquad \beta_j(t) = q_j . \end{aligned}$$

The calculation of $f(n, d)$ yields the desired solution with on the order of nd computational steps.

8. Maximization of Weighted Earliness

Let

$$\begin{aligned} c_j(t) &= p_j t, \qquad (t \leqq d), \\ &= q_j, \qquad (t > d), \end{aligned}$$

as in the previous section, and suppose that $p_j\, d \geqq q_j$ for all j. Then a sequence which is minimal with respect to the functions $c_j(t)$ is also minimal with respect to the functions

$$c'_j(t) = \min \{p_j t, q_j\}.$$

But a sequence which is minimal with respect to the functions $c'_j(t)$ is also maximal with respect to the functions

$$c''_j(t) = \max \{p_j(d_j - t), 0\},$$

where $d_j = q_j/p_j$.

We note that $c''_j(t)$ represents weighted earliness of the job j with respect to an individual dealine d_j. Thus, the methods of Section 7 can be used to maximize weighted earliness with respect to individual deadlines, or, for that matter, to maximize weighted tardiness. However, the problem of *minimizing* weighted tardiness is a much more difficult matter, as we shall see below.

9. Minimization of Tardiness, Common Relative and Absolute Deadlines

Consider the same problem, but this time let

$$\begin{aligned} c_j(t) &= 0, \qquad (t \leqq d), \\ &= t - d, \qquad (d \leqq t \leqq d'), \\ &= q_j, \qquad (t > d'), \end{aligned}$$

where d and d' are two deadlines common to all jobs. We may view d as a "relative" and d' as an "absolute" deadline.

The problem requires us to partition the jobs into two classes: those which are on time with respect to d', and those which are tardy. The on-time jobs can be sequenced according to the order of their processing times, shortest processing time first. (See Smith [10].)

The problem is solved by ordering the jobs by the shortest processing time rule, and then applying Equation (1) with

$$a_j = a'_j, \qquad \alpha_j(t) = \max\{0, t - d\},$$
$$b_j = 0, \qquad \beta_j(t) = q_j.$$

The calculation of $f(n, d)$ yields the desired solution.

10. Minimization of Weighted Tardiness, Common Deadline

Consider the same problem, but let

$$c_j(t) = \max\{0, p_j(t - d)\}.$$

In other words, $c_j(t)$ represents weighted tardiness with respect to a common deadline d.

This problem requires a partitioning of the jobs into three classes: (A) those which are completed on time, which can be sequenced in arbitrary order, (B) those which are late, and which are sequenced in order of the ratios p_j/a'_j, largest ratio first and (C) a single job which begins before the deadline d and is completed on or after it.

There appears to be no way to know, a priori, the identity of the job in Class (C), except that it must possess a p_j no greater than that of any job in Class (A). Hence, it seems just as well to resolve the problem for each possible job in Class (C) and choose the best result.

The procedure is as follows:

(1) Order the jobs according to increasing ratios p_j/a'_j, *i.e.*, $p_1/a'_1 \leqq p_2/a'_2 \leqq \cdots \leqq p_n/a'_n$.

(2) Solve Equation (1) for each subset of $n - 1$ jobs, with

$$a_j = a'_j, \qquad \alpha_j(t) = 0,$$
$$b_j = 0, \qquad \beta_j(t) = p_j t.$$

Let $f^{(k)}(n, t)$ denote the solution for the subset of jobs $\{1, 2, \cdots, k-1, k+1, \cdots, n\}$.

(3) Find the values of k and t for which

$$f^{(k)}(n, t) + p_k(t + a'_k - d), \qquad (d - a'_k \leqq t < d),$$

is minimum.

These values indicate the identity of the job in Class (C) and its time of completion, while $f^{(k)}(n, t)$ can be used to determine the jobs in Classes (A) and (B). The overall computation requires on the order of $n^2 d$ steps.

In order to justify the expression for $\beta_j(t)$, the reader should consider the partition of the first j jobs into the two classes (A) and (B). Let

$$A_j = \sum_{i=1}^{j} a'_i.$$

Then if the total processing time of the jobs in Class (A) is t, the total processing time of the jobs in Class (B) is $A_j - t$. It follows that if job j is assigned to Class (B) its completion time will be $A_n - (A_j - t)$, and the loss incurred will be

$$p_j(A_n - (A_j - t)) = p_j t + \text{constant}.$$

11. Machines in Parallel

Each of the problem formulations and solution methods we have discussed can be extended in a very natural way to the situation in which there are m machines which can process jobs simultaneously. For machine i, job j can be processed either in time $a_j^{(i)}$ or $b_j^{(i)}$, with losses $\alpha_j^{(i)}(t)$ and $\beta_j^{(i)}(t)$.

It is important to note that the jobs which are assigned to any given machine are processed in an order which is consistent with the ordering that would be used in the related single machine problem. This enables us to define a function

$f(j, t_1, t_2, \cdots, t_m)$ = the minimum total loss for the first j jobs, subject to the constraint that no job is completed later than time t_i on machine i,

and to obtain the appropriate functional equation.

In these cases, the number of computational steps grows as mnT^m, where T is an appropriately large number. Some reduction of computational complexity is possible because of symmetry in the case that the machines are identical (processing times and loss functions the same). One can then assume that $t_1 \leqq t_2 \leqq \cdots \leqq t_n$. Nevertheless, the computation still grows essentially exponentially. The reader is referred to Rothkopf [7, 8] for further details.

12. Two Machines in Series

Suppose n jobs are to be performed by each of two machines in sequence. That is, job j is to be performed by the first machine with processing time $a_j^{(1)}$ and then by the second with processing time $a_j^{(2)}$. The jobs are subject to a common deadline d. If job j is completed by the second machine at time t, a loss of

$$c_j(t) = 0, \qquad (t \leqq d),$$
$$= p_j, \qquad (t > d),$$

is incurred. How should the jobs be sequenced so as to minimize the total loss?

Again the problem is to partition the jobs into two classes: those which are to be on time, and those which are to be late. The on-time jobs can be assumed to be sequenced according to the rule of S. Johnson [3], and these jobs can be followed by the tardy jobs, in arbitrary order.

Assume that the jobs are numbered according to Johnson's rule:

$$\min\{a_j^{(1)}, a_{j+1}^{(2)}\} \leqq \min\{a_{j+1}^{(1)}, a_j^{(2)}\}$$

then let

$f(j, t_1, t_2)$ = the minimum total loss for the first j jobs, subject to the constraint that no job is completed later than time t_1 by the first machine and time t_2 by the second.

By arguments similar to those used above,

$$f(0, t_1, t_2) = 0, \qquad (t_1, t_2 \geqq 0),$$

$$f(j, t_1, t_2) = +\infty, \qquad (j = 0, 1, \cdots, n;\ t_1 < 0 \text{ or } t_2 < 0),$$

$$f(j, t_1, t_2) = \min\begin{Bmatrix} f(j, t_1, t_2 - 1), \\ f(j, t_1 - 1, t_2), \\ p_j + f(j-1, t_1, t_2), \\ f(j-1, t_1 - \delta, t_2 - a_j^{(2)}) \end{Bmatrix}, \quad (j = 1, 2, \cdots, n;\ t_1, t_2 \geqq 0),$$

where $\delta = \min\{t_1 - a_j^{(1)}, t_2 - a_j^{(1)} - a_j^{(2)}\}$.

An optimal solution is obtained by computing $f(n, d, d)$, and this requires on the order of $n\,d^2$ computational steps.

Acknowledgement

Support for the first author was provided under Air Force Contract AF 30(602)-3546 at the Systems Engineering Laboratory, The University of Michigan, Ann Arbor. Research was performed while the second author was affiliated with the Department of Industrial Engineering, The University of Michigan, Ann Arbor.

References

1. Held, Michael and Richard M. Karp, "A Dynamic Programming Approach to Sequencing Problems," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, No. 1, pp. 196–210, (March 1962).
2. Jackson, J. R., "Scheduling a Production Line to Minimize Maximum Tardiness," Management Science Research Project, Report No. 43, University of California, Los Angeles, (July 1955).
3. Johnson, S. M., "Optimal Two- and Three- Stage Production Schedules with Set-Up Times Included," *Naval Research Logistics Quarterly*, Vol. 1, No. 1, pp. 61–68, (1954).
4. Lawler, Eugene L., "On Scheduling Problems with Deferral Costs," *Management Science*, Vol. 11, No. 2, pp. 280–288, (November 1964).
5. McNaughton, Robert, "Scheduling with Deadlines and Loss Functions," *Management Science*, Vol. 6, No. 1, pp. 1–12, (October 1959).
6. Moore, J. M., "An *n* Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," *Management Science*, Vol. 15, No. 1, pp. 102–109, (September 1968).
7. Rothkopf, Michael H., "Scheduling Independent Tasks on Parallel Processors," *Management Science*, Vol. 12, No. 6, pp. 437–447, (January 1966).
8. Rothkopf, Michael H., "Scheduling Independent Tasks on One or More Processors," Ph.D. Thesis, M.I.T. School of Industrial Management, (January 1964). Also available as Interim Technical Report No. 2, Operations Research Center, Massachusetts Institute of Technology.
9. Schild, A. and I. J. Fredman, "Scheduling Tasks with Deadlines and Nonlinear Loss Functions," *Management Science*, Vol. 9, No. 1, pp. 73–81, (October 1962).
10. Smith, Wayne E., "Various Optimizers for Single-Stage Production," *Naval Research Logistics Quarterly*, Vol. 3, Nos. 1–2, pp. 59–66, (March–June 1956).

Mathematical Programming 1 (1971) 267–269. North-Holland Publishing Company
Reprinted by permission

NOTE

A SOLVABLE CASE OF THE TRAVELING SALESMAN PROBLEM

Eugene L. LAWLER
University of California, Berkeley, California, U.S.A.

Received 18 November 1970
Revised manuscript received 27 February 1971

Let $D = \|d_{ij}\|$ be the distance matrix defining a traveling salesman problem. If D is upper triangular, i.e. $d_{ij} = 0$, for $i \geq j$, then the traveling salesman problem can be solved with an amount of computation approximately equal to that required for an assignment problem of the same dimension.

Let $D = \|d_{ij}\|$ be the $n \times n$ distance matrix defining an n-city traveling salesman problem, i.e.

d_{ij} = the distance from city i to city j.

Suppose D is upper triangular, i.e. $d_{ij} = 0$, for $i \geq j$, whereas d_{ij} is completely arbitrary (positive, negative or zero), for $i < j$. Then the traveling salesman problem can be solved as follows.

First, solve the assignment problem for the $(n-1) \times (n-1)$ matrix D' obtained by deleting column one and row n of the matrix D. The solution of this assignment problem is in the form of a directed path P from city 1 to city n, plus various circuits involving the cities not in the path P. (If D is such that none of these circuits are negative in length, then P is a shortest path from city 1 to city n cf. [1].)

Call an arc (i, j) a *backward arc* if $i \geq j$. The next step is to remove from the solution to the assignment problem all backward arcs not contained in the path P. This converts the circuits in the solutions to paths, which may be assumed to be from i_1 to j_1, i_2 to j_2, ..., i_m to j_m, respectively, where $i_1 < i_2 < \ldots < i_m$ and $i_1 \leq j_1$, $i_2 \leq j_2$, ..., $i_m \leq j_m$.

Finally, introduce new backward arcs (n, i_m), (j_m, i_{m-1}), ..., (j_2, i_1), $(j_1, 1)$, to obtain a Hamiltonian circuit. Since only backward arcs have been removed from and added to the solution to the assignment problem, this Hamiltonian circuit has the same total length as the solution to the assignment problem.

Thus, a solution to the assignment problem for D' yields a Hamiltonian circuit with the same total length. The converse is also true. Any Hamiltonian circuit contains a path P from nodes 1 to n. Remove all backward arcs not contained in P and introduce new backward arcs creating circuits in addition to P. This results in a solution to the assignment problem for D'.

We have established that an optimal solution to the assignment problem for D' can be converted into an optimal solution to the traveling salesman problem for D. By similar techniques it is also possible to show that an optimal solution to the assignment problem for D (as opposed to D') can be converted into a minimum-length Hamiltonian path (as opposed to a Hamiltonian circuit).

Also note that if D is such that

$$d_{ij} = a_i + b_j, \quad i > j$$

where $a_2, a_3, \ldots, a_n$, and $b_1, b_2, \ldots, b_{n-1}$ are $2n - 2$ given parameters, then such a distance matrix can be converted to one of upper triangular form by simply subtracting a_i from row i and b_j from column j and setting $d_{ii} = 0$, for all i. This changes the length of all Hamiltonian circuits by the same constant value.

As a simple example, let

$$D = \begin{bmatrix} 0 & -1 & 7 & -20 & 3 & -2 & 5 \\ 0 & 0 & 12 & 8 & 16 & 9 & 8 \\ 0 & 0 & 0 & 3 & 7 & 6 & 2 \\ 0 & 0 & 0 & 0 & 4 & 4 & 9 \\ 0 & 0 & 0 & 0 & 0 & -18 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

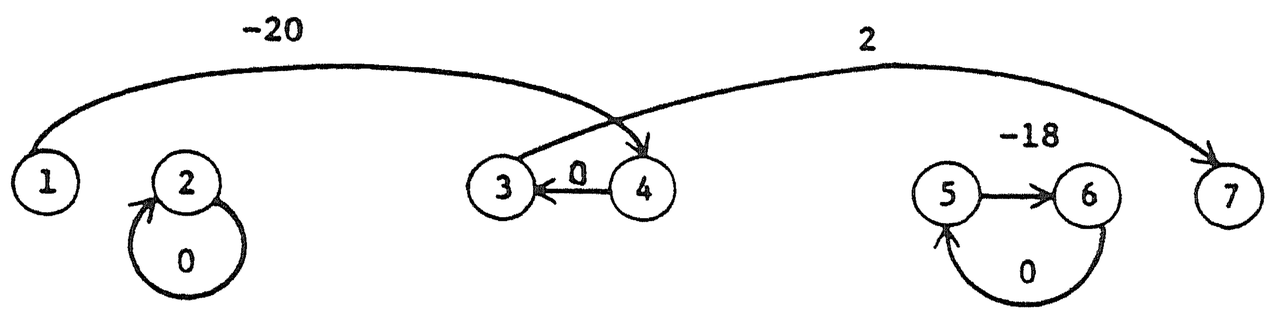


Fig. 1. (a) Optimal solution to assignment problem.

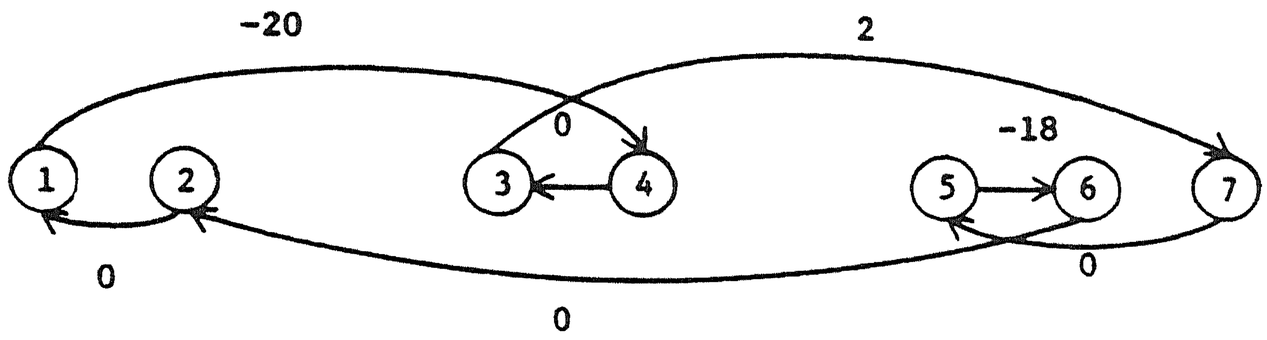


Fig. 1. (b) Optimal solution to traveling salesman problem.

Then

$$D' = \begin{bmatrix} -1 & 7 & \textcircled{-20} & 3 & -2 & 5 \\ \textcircled{0} & 12 & 8 & 16 & 9 & 8 \\ 0 & 0 & 3 & 7 & 6 & \textcircled{2} \\ 0 & \textcircled{0} & 0 & 4 & 4 & 9 \\ 0 & 0 & 0 & 0 & \textcircled{-18} & -1 \\ 0 & 0 & 0 & \textcircled{0} & 0 & 3 \end{bmatrix}$$

and an optimal solution to the assignment problem is indicated by the encircled entries. This is converted to an optimal solution to the traveling salesman problem as shown in fig. 1.

This special case of the traveling salesman problem can be compared and contrasted with that of Gilmore and Gomory [2]. Neither one is a special case of the other.

References

[1] A.J. Hoffman and H.M. Markovitz, "A note on shortest path, assignment, and transportation problems," *Naval Research Logistics Quarterly* 10 (1963) 375–380.

[2] P.C. Gilmore and R.E. Gomory, "Sequencing a one state-variable machine: A solvable case of the traveling salesman problem," *Operations Research* 12 (1964) 655–679.

MANAGEMENT SCIENCE
Vol. 19, No. 5, January, 1973
Printed in U.S.A.

OPTIMAL SEQUENCING OF A SINGLE MACHINE SUBJECT TO PRECEDENCE CONSTRAINTS*†

E. L. LAWLER

University of California, Berkeley

Suppose n jobs are each to be processed by a single machine, subject to arbitrary given precedence constraints. Associated with each job j is a known processing time a_j and a monotone nondecreasing cost function $c_j(t)$, giving the cost that is incurred by the completion of that job at time t. The problem is to find a sequence which will minimize the maximum of the incurred costs. An efficient computational procedure is given for this problem, generalizing and simplifying previous results of the present author and J. M. Moore.

1. Problem Formulation

Suppose n jobs are each to be processed by a single machine (one at a time, with no interruptions), subject to arbitrary given precedence constraints. Associated with each job j is a known processing time a_j and a monotone nondecreasing cost function $c_j(t)$, giving the cost that is incurred by the completion of that job at time t. The problem is to find a sequence which will minimize the maximum of the incurred costs.

2. Sequencing Theorem

THEOREM. *Let S denote the subset of jobs which may be performed last, i.e. those jobs which are not required to precede any others. Let T denote the sum of the processing times of all the jobs. Let k be a job in S such that*

$$c_k(T) = \min_{j \in S} \{c_j(T)\}.$$

Then there exists a minmax optimal sequence, i.e. a sequence which minimizes the maximum of the incurred costs, in which job k is last.

PROOF. Consider any sequence π' with job $k' \neq k$ last. Such a sequence can be represented schematically as follows, where A and B represent the portions of the sequence occupied by the $n - 2$ jobs other than k and k':

π':	A	k	B	k'

Suppose we modify this sequence to obtain:

π:	A	B	k'	k

If the given precedence constraints are observed by sequence π', then they are also observed by sequence π, since neither k nor k' is required to precede any other job.

No job is completed later in π than in π', except job k. Because the cost functions are assumed to be monotone, it follows that no incurred cost can be greater in π than in π', except possibly that of k. But job k was chosen to be such that $c_k(T) \leqq c_{k'}(T)$. Hence it follows that the maximum of the incurred costs is no greater for sequence π than for π'.

* Received September 1971.
† Research sponsored by the U.S. Naval Electronic Systems Command, Contract N00039-71-0255.

3. Sequencing Algorithm

An efficient algorithm for finding a minmax optimal sequence follows immediately from the theorem above. One simply finds a job k which can be placed last. Then, having removed k from the problem, one finds a job which can be placed last among the remaining $n - 1$ jobs and second-to-last in the complete sequence, and so on.

We will illustrate the procedure with a simple example.

Suppose there are four jobs, with precedence constraints as indicated in Figure 1. These jobs have processing times $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 1$, and cost functions as indicated in Figure 2.

Jobs 2 and 4 are eligible to be processed last in an optimal sequence, since they have no successors, as seen in Figure 1. Whatever job is last in the sequence will be completed at time $t = 6$. We compare the values $c_2(6)$ and $c_4(6)$, marked by "x's" in Figure 2, and find that $c_4(6) < c_2(6)$. Accordingly, we place job 4 in the last position of the optimal sequence.

If job 4 is last in the sequence, jobs 2 and 3 are both eligible to be processed in the next-to-last position. Whatever job is processed in this position in the sequence will be completed at time $t = 5$. We compare the values of $c_2(5)$ and $c_3(5)$ and find that $c_3(5) < c_2(5)$. Accordingly, we place job 3 in the next-to-last position of the optimal sequence.

There is only one way to order the remaining jobs 1 and 2. Hence a minmax optimal sequence is 1, 2, 3, 4. The maximal cost is incurred by the completion of job 3 at $t = 5$.

4. Computational Efficiency

In [2], J. M. Moore suggests a procedure for finding a minmax optimal sequence in the case that there are no precedence constraints. The present procedure applies to the

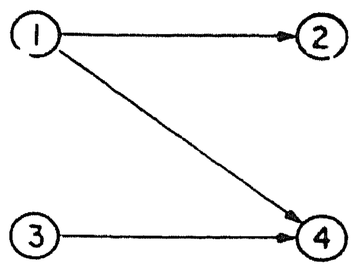


FIGURE 1. Precedence constraints for example.

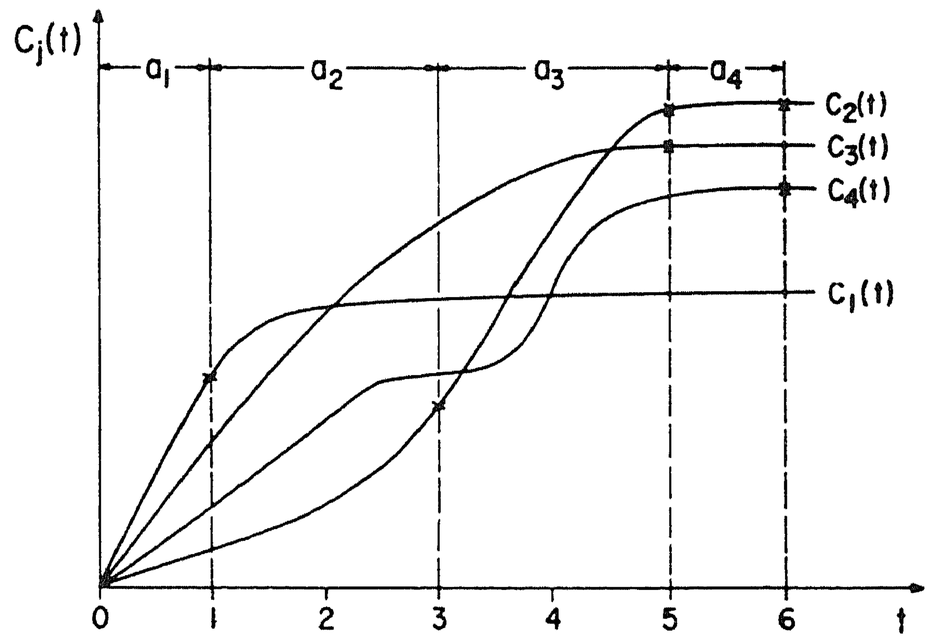


FIGURE 2. Plot of cost functions for example.

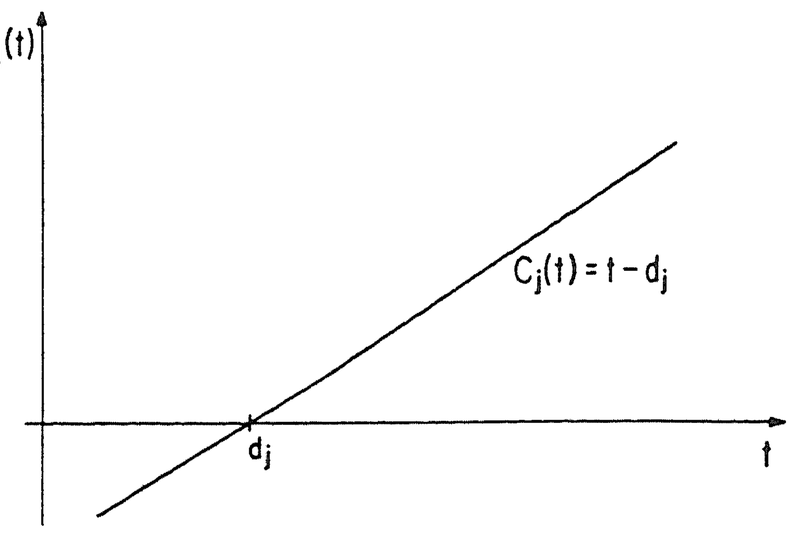


FIGURE 3. Cost function for deadline problem.

more general problem with precedence constraints and is, in addition, more efficient computationally. It can be shown that the present procedure requires a number of computational steps which grows as n^2, where n is the number of jobs.

5. Sequencing with Deadlines and Precedence Constraints

Suppose, instead of a cost function $c_j(t)$, there is simply specified for each job a deadline d_j. The problem is to find a sequence, subject to the given precedence constraints, which will cause each job to be completed by its deadline, if such a sequence exists.

This problem was solved in [1] by a method in which each job j was assigned a new deadline $\bar{d}_j \leqq d_j$, and the jobs were sequenced according to increasing values of $\bar{d}_j$. It was noted that the solution to this problem was independent of the processing times of the jobs.

The computation also suggested the following "first-to-last" rule:

Sequence the jobs from first to last, always choosing next from among the jobs which are currently available (i.e. jobs none of whose predecessors remain unchosen) a job which has a successor with the earliest possible deadline (considering a job to be one of its own successors).

This problem can also be analyzed as a minmax problem as follows. Assign to each job j a cost function $c_j(t)$ as shown in Figure 3. By following the algorithm of §3, we discover that a minmax optimal sequence is obtained by the following "last-to-first" rule, which is a generalization of the well-known "deadline rule" for optimal sequencing without precedence constraints.

Sequence the jobs from last to first, always choosing next, from among the jobs which are currently available (i.e. jobs none of whose successors remain unchosen) a job with the latest possible deadline.

As before, we note that this sequence is independent of the actual processing times. The processing times are used to check whether the sequence does in fact avoid tardiness of all of the jobs. If the sequence is not successful in this regard, no other sequence is.

Finally, we note that the sequence obtained by either of these two rules serves to minimize the maximum tardiness (or lateness) of the jobs, if it is not possible to complete all the jobs on time.

References

1. LAWLER, E. L. AND MOORE, J. M., "A Functional Equation and Its Application to Resource Allocation and Sequencing Problems," *Management Science*, Vol. 16 (1969), pp. 77–84.
2. MOORE, J. M., "An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," *Management Science*, Vol. 15 (1968), pp. 102–109.

Mathematical Programming 9 (1975) 31–56.
North-Holland Publishing Company
Reprinted by permission

MATROID INTERSECTION ALGORITHMS*

Eugene L. LAWLER
University of California, Berkeley, California, U.S.A.

Received 24 May 1972
Revised manuscript received 27 June 1974

Let $M_1 = (E, \mathcal{I}_1)$, $M_2 = (E, \mathcal{I}_2)$ be two matroids over the same set of elements E, and with families of independent sets $\mathcal{I}_1$, $\mathcal{I}_2$. A set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ is said to be an *intersection* of the matroids M_1, M_2. An important problem of combinatorial optimization is that of finding an optimal intersection of M_1, M_2. In this paper three matroid intersection algorithms are presented. One algorithm computes an intersection containing a maximum number of elements. The other two algorithms compute intersections which are of maximum total weight, for a given weighting of the elements in E. One of these algorithms is "primal-dual", being based on duality considerations of linear programming, and the other is "primal". All three algorithms are based on the computation of an "augmenting sequence" of elements, a generalization of the notion of an augmenting path from network flow theory and matching theory. The running time of each algorithm is polynomial in m, the number of elements in E, and in the running times of subroutines for independence testing in M_1, M_2. The algorithms provide constructive proofs of various important theorems of matroid theory, such as the Matroid Intersection Duality Theorem and Edmonds' Matroid Polyhedral Intersection Theorem.

1. Introduction

Let $M_1 = (E, \mathcal{I}_1)$, $M_2 = (E, \mathcal{I}_2)$ be two matroids over the same set of elements E, and with families of independent sets $\mathcal{I}_1$, $\mathcal{I}_2$. A set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ is said to be an *intersection* of the matroids M_1, M_2. In this paper, algorithms are presented for the computation of optimal intersections. We distinguish two types of problems: the *cardinality intersection problem*, in which we seek an intersection containing a maximum number of elements, and the *weighted intersection problem*, in which we seek an intersection of maximum total weight, with respect to a given weighting of the elements.

One algorithm is presented for the cardinality intersection problem and two algorithms for the weighted intersection problem. One of the algorithms for the weighted intersection problem is "primal-dual", being based on duality considerations of linear programming, and the other is

* Research sponsored by the Air Force Office of Scientific Research Grant 71-2076.

"primal". All three algorithms are based on the computation of an "augmenting sequence" of elements, a generalization of the notion of an augmenting path from network flow theory and matching theory.

The running time of each algorithm is polynomial in m, the number of elements in E, and in the running times of subroutines for independence testing in M_1, M_2.

The algorithms provide constructive proofs of various important theorems of matroid theory. Among these are the Matroid Intersection Duality Theorem (maximum cardinality of an intersection equals minimum rank of a covering) and Edmonds' Matroid Polyhedral Intersection Theorem (the intersection of two matroid polyhedra creates a polyhedron whose vertices are vertices of both of the original polyhedra).

2. Matroid definitions

We first review some basic definitions of matroid theory. A *matroid* $M = (E, \mathcal{I})$ is a structure in which E is a finite set of elements and $\mathcal{I}$ is a family of subsets of E, called *independent sets*, such that

(2.1) $\emptyset \in \mathcal{I}$ and all proper subsets of a set I in $\mathcal{I}$ are also in $\mathcal{I}$.

(2.2) If I_p, I_{p+1} are sets in $\mathcal{I}$ containing $p, p+1$ elements respectively, then there exists an element $e \in I_{p+1} - I_p$ such that $I_p \cup \{e\} \in I$.

(Hereafter we shall use the notation "$I + e$" for "$I \cup \{e\}$" and "$I - e$" for "$I - \{e\}$".)

There are a great variety of matroids which have been discussed in the literature: matric, graphic, transversal, binary, etc. We mention here only one type, which will be used as an example to motivate the discussion which follows. Let $G = (N, A)$ be a graph with node set N and arc set A. The *graphic* matroid $M = (A, \mathcal{I})$ of G has the arcs of G as its elements and as its independent sets all acyclic subsets $F \subseteq A$, i.e., all "forests" in G.

Some further definitions which are needed are the following. For a given matroid $M = (E, \mathcal{I})$, the *rank* $r(A)$ of a subset $A \subseteq E$ is the cardinality of a maximal independent subset of A. (All maximal independent subsets of A must have equal cardinality.) A subset of E which is not independent is *dependent*. A minimal dependent set is called a *circuit*. (Circuits need not have equal cardinality.) The *span* of a set $A \subseteq E$, denoted $\mathrm{sp}(A)$, is the maximal superset of A having the same rank as A. A set A which is equal to its own span, i.e., $A = \mathrm{sp}(A)$, is said to be a *closed set*.

Many basic theorems of matroid theory are assumed in the discussion which follows. For example we assume the reader knows that if I is independent and $I + e$ is dependent, then $I + e$ contains a unique circuit C. And if e' is any element in C, then $I + e - e'$ is independent and $\mathrm{sp}(I) = \mathrm{sp}(I + e) = \mathrm{sp}(I + e - e')$. We refer the reader to the literature for such results.

3. Augmenting sequences

Bipartite matching algorithms solve intersection problems involving two "partition" matroids. These algorithms can be generalized to solve intersection problems involving arbitrary pairs of matroids. Our first task in generalizing the bipartite matching algorithms is to find an appropriate generalization of the idea of an augmenting path.

Let I be any intersection of two matroids, M_1 and M_2. We can construct an "augmenting sequence" with respect to I as follows. The first element e_1 of such a sequence is such that $I + e_1$ is independent in M_1. If $I + e_1$ is independent in M_2 as well, the sequence is completed. Otherwise $I + e_1$ contains a unique circuit in M_2 and we choose e_2 to be any element other than e_1 in that circuit. $I + e_1 - e_2$ is clearly independent in both M_1 and M_2. Now we try to find an element e_3 such that $I + e_1 - e_2 + e_3$ is independent in M_1 (whereas $I + e_3$ is not); such an element e_3 is in $\mathrm{sp}_1(I) - \mathrm{sp}_1(I - e_2)$, where "$\mathrm{sp}_1$" denotes span in M_1. If $I + e_1 - e_2 + e_3$ contains a unique circuit in M_2, we choose e_4 to be an element in that circuit, and so on.

In other words, the addition to I of the 1st, 3rd, 5th, ... elements preserves independence in M_1 but creates dependence in M_2, whereas the removal of the 2nd, 4th, 6th, ... elements restores independence in M_2. This manner of playing off independence in M_1 against independence in M_2 is quite analogous to the way that one plays off incidence of arcs to nodes in one part of a bipartite graph against incidence of the same arcs to nodes in the other part in the construction of an augmenting path in the matching problem.

Let I be an intersection of two matroids $M_1 = (E, \mathcal{I}_1)$, $M_2 = (E, \mathcal{I}_2)$. Let $S = (e_1, e_2, \ldots, e_s)$ be a sequence of elements of E. Let $S_i = (e_1, e_2, \ldots, e_i)$, for $i \leqslant s$. We let "$I \oplus S_i$" denote the symmetric difference of I and S_i, where S_i is treated as an ordinary (unordered) set. We say that S is an *alternating sequence* with respect to I if

(3.1) $I + e_1 \in \mathcal{I}_1$.

(3.2) For all even i, $e_i \in I$ and $\mathrm{sp}_2(I \oplus S_i) = \mathrm{sp}_2(I)$. Hence $I \oplus S_i \in \mathcal{I}_2$.

(3.3) For all odd $i \geqslant 3$, $e_i \notin I$ and $\mathrm{sp}_1(I \oplus S_i) = \mathrm{sp}_1(I + e_1)$. Hence $I \oplus S_i \in \mathcal{I}_1$.

If, in addition,

(3.4) $|S| = s$ is odd and $I \oplus S \in \mathcal{I}_2$,

we say that S is an *augmenting sequence* with respect to I.

Theorem 3.1. *Let I_p, I_{p+1} be intersections of $M_1 = (E, \mathcal{I}_1)$, $M_2 = (E, \mathcal{I}_2)$ with p, $p+1$ elements respectively. Then there exists an augmenting sequence $S \subseteq I_p \oplus I_{p+1}$ with respect to I_p.*

This theorem follows from Theorem 4.1 below and from the constructive proof of the Matroid Intersection Duality Theorem (Theorem 5.2). This line of reasoning is analogous to proving the augmenting path theorem for network flows by invoking the Max-Flow Min-Cut Theorem.

Corollary 3.2. *An intersection is of maximum cardinality if and only if it admits no augmenting sequence.*

Corollary 3.3. *For any intersection I there exists a maximum cardinality intersection I^* such that $\mathrm{sp}_1(I) \subseteq \mathrm{sp}_1(I^*)$ and $\mathrm{sp}_2(I) \subseteq \mathrm{sp}_2(I^*)$.*

4. Cardinality intersection algorithm

The cardinality intersection algorithm constructs sequences of a restricted type. These sequences satisfy certain conditions which are sufficient to imply conditions (3.1)–(3.3).

For a given intersection I, let $e_i \in E - I$. If $I + e_i \notin \mathcal{I}_1$, let $C_i^{(1)}$ denote the unique M_1-circuit contained in $I + e_i$. If $I + e_1 \notin$ $_2$, let $C_i^{(2)}$ denote the unique M_2-circuit contained in $I + e_i$.

Now suppose $S = (e_1, e_2, \ldots, e_s)$ is a sequence of distinct elements, where $e_i \in E - I$ for i odd, and $e_i \in I$, for i even. Further, suppose

(4.1) $I + e_1 \in \mathcal{I}_1$.

(4.2) For all even i, $e_i \in C_{i-1}^{(2)}$.

Moreover, $e_i \notin C_{k-1}^{(2)}$ for any even $k < i$.

(4.3) For all odd $i \geqslant 3$, $e_{i-1} \in C_i^{(1)}$.

Moreover, $e_{k-1} \notin C_i^{(1)}$ for any odd $k < i$.

Theorem 4.1. *If a sequence S satisfies conditions* (4.1)–(4.3), *then S is an alternating sequence.*

Proof. Conditions (3.1) and (4.1) are, of course, identical.

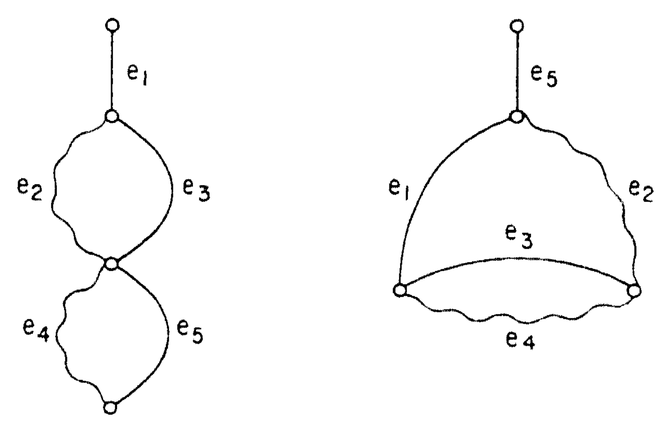
Consider condition (4.2). By inductive hypothesis, $\mathrm{sp}_2(I \oplus S_{i-2}) = \mathrm{sp}_2(I)$. Because $I + e_{i-1}$ is dependent in M_2, so is $I \oplus S_{i-1}$, and $I \oplus S_{i-1}$ contains a unique M_2-circuit which we denote $C_{i-1}^{[i]}$. It is necessary to show that $e_i \in C_{i-1}^{[i]}$.

Let $C_{i-1}^{[k]}$, for even k, denote the unique M_2-circuit contained in $(I \oplus S_{k-2}) + e_{i-1}$. Clearly $C_{i-1}^{[2]} = C_{i-1}$ (dropping the superscript (2), as we do throughout the argument). By inductive hypothesis, $e_i \in C_{i-1}^{[k-2]}$ and $e_k \in C_{k-1}^{[k]}$, where $C_{k-1}^{[k]} \subseteq I \oplus S_{k-1}$. Moreover, by another inductive argument, $e_i \notin C_{k-1}^{[k]}$. If $e_k \notin C_{i-1}^{[k-2]}$, then clearly $C_{i-1}^{[k]} = C_{i-1}^{[k-2]}$. If $e_k \in C_{i-1}^{[k-2]}$, then, by the property of circuits there exists a circuit $C_{i-1}^{[k]} \subseteq (C_{k-1}^{[k]} \cup C_{i-1}^{[k-2]}) - e_k$, such that $e_i \in C_{i-1}^{[k]}$. It follows that $e_i \in C_{i-1}^{[i]}$, $I \oplus S_i \in \mathcal{I}_2$, and $\mathrm{sp}_2(I \oplus S_i) = \mathrm{sp}_2(I)$. Thus condition (3.2) is satisfied.

The argument that (3.3) follows from (4.3) is much simpler. By inductive hypothesis, $\mathrm{sp}_1(I \oplus S_{i-2}) = \mathrm{sp}_1(I + e_1)$. From (4.3) it follows immediately that $C_i^{(1)} \subseteq (I \oplus S_{i-2}) + e_i$. Hence removing e_{i-1} from $(I \oplus S_{i-2}) + e_i$ yields independence in M_1, as desired.

It is important to note that not all augmenting sequences satisfy conditions (4.2) and (4.3). For example, let M_1, M_2 be the graphic matroids of the multi-graphs G_1, G_2 in Fig. 1, with $I = \{e_2, e_4\}$. The sequence $S = (e_1, e_2, e_3, e_4, e_5)$ satisfies condtions (3.1)–(3.4), but it does not satisfy condition (4.2) because $e_4 \in C_1^{(2)} = \{e_1, e_2, e_4\} \subseteq I + e_1$. (Note that the augmenting sequence $S' = (e_1, e_4, e_5)$ does satisfy (4.1)–(4.3).) It is sufficient to consider sequences of the restricted type satisfying conditions (4.1)–(4.3) for the purpose of the cardinality intersection problem, but not for the weighted intersection problem.

We now present a labeling procedure for constructing augmenting sequences. This procedure places labels on the individual matroid elements. Accordingly, when the procedure is applied to the special case of the bipartite matching problem, labels are given to arcs rather than to nodes.



Other than that difference, the cardinality intersection algorithm can be considered to be a fairly straightforward generalization of a bipartite matching algorithm.

At the start, no elements of E are labeled. Then each element in $E - \mathrm{sp}_1(I)$ is given the label ϕ^+. (If there are no such elements, the existing intersection I is clearly of maximum cardinality.) The symbol "ϕ" indicates that the element in question is the first element of whatever augmenting sequence it may be found to be a member of. The "+" indicates that the element in question is to be added to I; a "$-$"-sign indicates that the element is to be removed.

Additional elements are labeled by "scanning" existing labels. A "+" label on e_i is scanned by first determining if $I + e_i$ is independent in M_2. In this case an augmenting sequence has been discovered, with e_i as the final element. If $I + e_i$ is dependent, the unique circuit $C_i^{(2)} \subseteq I + e_i$ is found, and the label i^- is given to each unlabeled element in $C_i^{(2)}$. A "$-$" label on e_i is scanned by giving the label "i^+" to each unlabeled element e_j such that $e_i \in C_j^{(1)}$.

The labeling procedure terminates when no further elements can be labeled or when an augmenting sequence is discovered, as described above. The complete augmenting sequence is obtained by "backtracing". For example, if the label of the last element e_i is "j^+" the second-to-last element in the sequence is e_j. If the label of e_j is "k^-", the third-to-last element is e_k, etc. The initial element of the sequence, of course, has the label ϕ^+.

The reader can verify that the rules of labeling construct augmenting sequences satisfying conditions (4.1)–(4.3).

Cardinality Intersection Algorithm

Step 0 (Start). Let I be any intersection of M_1, M_2, possibly the empty set. No elements are labeled.

Step 1 (Labeling).

1.0. For each element $e_i \in E - I$, find $C_i^{(1)}$, $C_i^{(2)}$, if these circuits exist. Apply the label "ϕ^+" to each element $e_i \in E - \mathrm{sp}_1(I)$.

1.1. If all labels have been scanned, go to step 3. Otherwise, find an element e_i with an unscanned label. If the label is a "+" label, go to step 1.2; if it is a "$-$" label, go to step 1.3.

1.2. Scan the "+" label on e_i as follows. If $I + e_i \in \mathcal{I}_2$, go to step 2. Otherwise, give the label "i^-" to each unlabeled element in $C_i^{(2)}$. Return to step 1.1.

1.3. Scan the "−" label on e_i by giving the label "i^+" to each unlabeled element e_j such that $e_i \in C_j^{(1)}$. Return to step 1.1.

Step 2 (Augmentation). An augmenting sequence S has been discovered, of which e_i (found in step 1.2) is the final element. Identify the elements in S by backtracing from the label on e_i. Augment I by adding to I all elements in the sequence with "+" elements and removing from I all elements with "−" labels. Remove all labels from elements and return to step 1.0.

Step 3 (Hungarian Labeling). No augmenting sequence exists and I is of maximum cardinality. The labeling is "Hungarian", and can be used to construct a minimum-rank covering dual to I. Halt.

Let us now estimate the complexity of the algorithm. Suppose the ranks of the matroids M_1, M_2 are R_1, R_2, respectively, and let $R = \min\{R_1, R_2\}$. Thus, no intersection can contain more than R elements and there can be no more than R augmentations.

Assume there are subroutines available for independence testing in M_1, M_2. Suppose the running times of these subroutines are $C_1(m)$, $C_2(m)$, respectively, where $m = |E|$. Let $C(m) = \max\{C_1(m), C_2(m)\}$. For each augmentation, and each subsequent application of the labeling procedure, there is a computation of $C_i^{(1)}$, $C_i^{(2)}$ for each $e_i \in E - I$. The running time for this task is no greater than $O(mRC(m))$.

The labeling procedure, exclusive of circuit computation, is $O(mR)$ and backtracing, if an augmenting sequence is found, is $O(m)$. Since there are $O(R)$ applications of the labeling procedure, the overall running time of the algorithm is no greater than $O(mR^2 + mR^2C(m))$.

We might comment at this point that the cardinality intersection problem can also be solved by an appropriate application of Edmonds' matroid partitioning algorithm [1], but with what is believed to be a somewhat greater degree of complexity. If one examines Edmonds' algorithm closely, one finds that it relies on the equivalent of augmenting sequences, with backtracing operations to discover them.

5. Duality theory

The cardinality intersection computation provides a constructive proof of a duality theorem for matroid intersections. This theorem is of the min−max variety, similar to the max-flow min-cut theorem of network flows and the König−Egervary theorem, of which it represents a proper generalization.

We say that a pair of subsets E_1, E_2 of E is a *covering* of E if $E_1 \cup E_2 = E$. With respect to a given pair of matroids M_1, M_2, we define the *rank* of a covering $\mathcal{C} = (E_1, E_2)$ to be $r(\mathcal{C}) = r_1(E_1) + r_2(E_2)$.

Lemma 5.1. *For any covering $\mathcal{C}$ and any intersection I, $r(\mathcal{C}) \geqslant |I|$.*

Proof. Let $I_1 = I \cap E_1$, and $I_2 = I \cap (E_2 - E_1)$. Clearly $|I_1| \leqslant r_1(E_1)$ and $|I_2| \leqslant r_2(E_2)$ which implies $|I| = |I_1| + |I_2| \leqslant r(\mathcal{C})$.

Theorem 5.2 (Matroid Intersection Duality [3]). *For any two matroids M_1, M_2, the maximum cardinality of an intersection is equal to the minimum rank of a covering.*

Proof. By the lemma, the rank of a covering cannot be less than the cardinality of an intersection. The intersection algorithm enables us to construct a covering whose rank is equal to the cardinality of an intersection, as follows.

At the conclusion of the algorithm (when the labeling has become "Hungarian"), let the set I_L contain the elements of I that are labeled and I_U contain those which are not. Then $E_1 = \mathrm{sp}_1(I_U)$, $E_2 = \mathrm{sp}_2(I_L)$ are the sets of the desired covering. $\mathcal{C} = (E_1, E_2)$ is a covering, for suppose that there existed an element e that was not a member of either $\mathrm{sp}_1(I_U)$ or $\mathrm{sp}_2(I_L)$. If $e \notin \mathrm{sp}_2(I_L)$, it cannot be labeled, because all such elements are in $\mathrm{sp}_2(I_L)$, by the construction in step 1.2. But if e is not labeled, $e \in \mathrm{sp}_1(I - e')$, for all $e' \in I_L$, by the construction in steps 1.0, 1.3. But this implies that $e \in \mathrm{sp}_1(I_U)$, contrary to assumption.

A duality theorem for the max–min intersection problem follows from Theorem 5.2 in exactly the same way that the duality theorem for the max–min bipartite matching problem is derived from the König–Egervary theorem.

Theorem 5.3. *Let $M_1 = (E, \mathcal{I}_1)$, $M_2 = (E, \mathcal{I}_2)$ be any two matroids and $w(e)$ be any weighting of the elements. Then, for any k,*

$$\max_I \min\{w(e): e \in I,\ I \in \mathcal{I}_1 \cap \mathcal{I}_2,\ |I| = k\} =$$
$$= \min_{A_1 A_2} \max\{w(e): e \in E - (A_1 \cup A_2),\ r_1(A_1) + r_2(A_2) = k - 1\}.$$

Proof. Let I^*, $|I^*| = k$, be an intersection which is max–min optimal with respect to all intersections containing k elements. Let e^* be such that

$$w(e^*) = \min\{w(e): e \in I^*\} ,$$

and let

$$A^* = \{e \in E: w(e) > w(e^*)\} .$$

Clearly a maximum-cardinality intersection contained within A^* has at most $k - 1$ elements, for otherwise I^* would not be optimal. It follows from Theorem 5.2 that A^* can be partitioned into two sets, A_1^*, A_2^* such that $r_1(A_1^*) + r_2(A_2^*) \leqslant k - 1$. (Apply the theorem to the two matroids after deleting all elements not in A^*.) But e^* is the element with largest weight not in $A_1^* \cup A_2^*$. Hence we have established that

$$\max_I \min\{w(e)\} \geqslant \min_{A_1, A_2} \max\{w(e)\} .$$

Conversely, let A_1^*, A_2^*, $r_1(A_1^*) + r_2(A_2^*) = k - 1$, be a min–max optimal solution to the dual covering problem. Let e^* be such that

$$w(e^*) = \max\{w(e): e \in E - (A_1^* \cup A_2^*)\} .$$

It follows from Theorem 5.2 that a maximum-cardinality intersection contained within A^* has at most $k - 1$ elements. Thus, any intersection with k elements must contain at least one element not in A^*. At best this is e^*. Hence we have established that

$$\max_I \min\{w(e)\} \leqslant \min_{A_1, A_2} \max\{w(e)\} .$$

This establishes inequality in both directions and the proof of the theorem is complete.

6. Matroid polyhedra

In order to formulate the weighted intersection problem as a linear programming problem, we first formulate a system of linear inequalities which are satisfied by an independent set of a single matroid $M = (E, \mathcal{I})$. Clearly, if I is an independent set, then

$$|I \cap S| \leqslant r(S) \tag{6.1}$$

for any subset $S \subseteq E$, and in particular for any closed set S.

Equivalently, let A be an incidence matrix of closed sets and elements of E. In other words, each row i of A corresponds to a closed set of the matroid (the indexing of these sets being arbitrary) and each column j corresponds to an element e_j. We set

$$a_{ij}\begin{cases}=1 & \text{if } e_j \text{ belongs to closed set } i,\\ =0 & \text{otherwise .}\end{cases}$$

Let $r = (r_1, r_2, \ldots, r_m)$ be a vector, where r_i is the rank of closed set i. We shall show that the vertices of the convex polyhedron defined by the inequalities

$$Ax \leqslant r, \qquad x \geqslant 0$$

are in one–one correspondence with the independent sets of M. That is to say, if x is a vertex, then each component x_j is either 0 or 1, where $x_j = 1$ if element e_j is a member of the independent set identified with the vertex, and $x_j = 0$, if it is not.

It is not difficult to show that the only feasible solutions to the system

$$Ax \leqslant r, \qquad x_j = 0 \text{ or } 1 \tag{6.2}$$

are those which correspond to independent sets, and vice versa. What is more surprising is that when these constraints are used to define a linear programming problem, the (0, 1) restriction of the variables can be relaxed to nonnegativity.

Theorem 6.1 (Edmonds [3]). *For any matroid M, all vertices of the convex polyhedron defined by the system of inequalities*

$$Ax \leqslant r, \qquad x \geqslant 0$$

have integer components. Moreover, the vertices and the independent sets of the matroid are in one–one correspondence.

Let M_1, M_2 be two matroids over the same set of elements E and let A and B be the closed set incidence matrices of M_1 and M_2 respectively. Let r and s be the rank vectors associated with these two matrices. We propose to solve the weighted intersection problem by solving the linear programming problem

$$\begin{aligned}&\text{maximize} && wx,\\ &\text{subject to} && Ax \leqslant r, \qquad Bx \leqslant s, \qquad x \geqslant 0.\end{aligned}$$

If this linear programming problem has an integer optimal solution, then this is a valid approach. At this point it is by no means clear that the integrality property holds. However, the weighted intersection algorithm provides a constructive proof of the integrality property, just as the so-called "greedy" algorithm provides a constructive proof of Theorem 6.1 [4].

Theorem 6.2 (Matroid Polyhedral Intersection Theorem – Edmonds). *For any two matroids M_1 and M_2 all vertices of the convex polyhedron defined by the system of linear inequalities*

$$Ax \leqslant r, \qquad Bx \leqslant s, \qquad x \geqslant 0$$

have integer components. Moreover, the vertices and the intersections of the two matroids are in one–one correspondence.

An equivalent statement of the theorem is as follows.

Theorem 6.3. *The intersection of two matroid polyhedra is a polyhedron whose vertices are vertices of each of the two intersected polyhedra.*

Note, however, that the intersection polyhedron is not necessarily a matroid polyhedron, so the theorem, unhappily, does not extend to three or more matroids.

7. Explanation of primal-dual method

The primal-dual algorithm described below provides a constructive proof of the polyhedral intersection theorem of Edmonds. That is, it is shown that, regardless of what element weights $w = (w_1, w_2, \ldots, w_n)$ are chosen, the linear programming problem

$$\begin{aligned} &\text{maximize} \quad wx, \\ &\text{subject to} \quad Ax \leqslant r, \qquad Bx \leqslant s, \qquad x \geqslant 0 \end{aligned} \tag{7.1}$$

has an optimal solution in zeros and ones.

The primal problem is as indicated in (7.1). The dual problem is

$$\begin{aligned} &\text{minimize} \quad ru + sv, \\ &\text{subject to} \quad A^{\mathrm{T}}u + B^{\mathrm{T}}v \geqslant w, \\ &\qquad\qquad\qquad\quad u, v \geqslant 0\,, \end{aligned} \tag{7.2}$$

where each dual variable u_i is identified with a closed set of M_1 and v_k with a closed set of M_2.

Orthogonality conditions necessary and sufficient for optimality of a pair of feasible primal and dual solutions are

$$x_j > 0 \Rightarrow (A^T u + B^T v)_j = w_j , \tag{7.3}$$

$$u_i > 0 \Rightarrow (Ax)_i = r_i , \tag{7.4}$$

$$v_k > 0 \Rightarrow (Bx)_k = s_k . \tag{7.5}$$

The algorithm begins with the feasible primal solution $x_j = 0$, for $j = 1, 2, ..., n$ (i.e. $I = \emptyset$), and with the feasible dual solution in which each dual variable u_i or v_k is zero, except u_E, the dual variable identified with the closed set E. We set $u_E = \max\{w_j\}$. Thus, at the beginning of the computation the only orthogonality condition which is violated is

$$u_E > 0 \Rightarrow |I| = r_1(E) . \tag{7.6}$$

The algorithm proceeds in stages. At each stage either the primal solution is revised by augmenting the existing intersection, or the values of the dual variables are revised. At all times, both primal and dual feasibility are maintained. Moreover, at each stage the only orthogonality condition which is not satisfied is (7.6). After a finite number of stages (in fact, a number bounded by a polynomial function in the number of elements in E), the condition (7.6) is also satisfied, and the primal and dual solutions existing at that point are optimal.

For a given pair of primal and dual solutions, the labeling routine of the cardinality intersection algorithm is applied, in an attempt to augment the primal solution. Clearly, the use of any augmenting sequence will result in a new feasible primal solution. However, the labeling routine must be modified in such a way that the only augmenting sequences which can be discovered are those for which all the orthogonality conditions except (7.6) continue to be satisfied.

If the application of the labeling routine, as restricted, does not result in the discovery of an augmenting sequence, then the dual solution is modified. The change in the dual solution must be such as to maintain dual feasibility, maintain satisfaction of all orthogonality conditions except (7.6), and also provide some progress toward the termination of the algorithm with optimal primal and dual solutions.

As a consequence of the fact that (7.6) is the only unsatisfied orthogonality condition, *the intersection existing at any intermediate stage of*

the computation is of maximum weight, relative to all intersections containing $|I|$ *or fewer elements.* For suppose there were an additional constraint of the form

$$\sum_j x_j \leqslant k\ ,$$

and we were to incorporate this constraint into the objective function via a Lagrange multiplier λ. Then an intermediate solution is easily shown to be optimal for $\lambda = u_E$ and therefore for a value of k equal to $|I|$.

At each stage of the computation, no more than $2m$ dual variables are permitted to be nonzero. These nonzero variables, except u_E, are identical with spans of subsets of I in two different families $\mathcal{U}$ and $\mathcal{V}$. Notationally, we let

$$\mathcal{U} = \{U_0, U_1, \ldots, U_p\}, \qquad \mathcal{V} = \{V_0, V_1, \ldots, V_q\}\ ,$$

where

$$U_0 = \emptyset, \qquad U_i \subset U_{i+1}, \qquad U_p = I;$$

$$V_0 = \emptyset, \qquad V_k \subset V_{k+1}, \qquad V_q = I\ .$$

Associated with subsets U_i and V_k are dual variables u_i and v_k, where u_i is identified with the closed set $\mathrm{sp}_1(U_i)$ and v_k with $\mathrm{sp}_2(V_k)$.

The labeling procedure is modified in two ways. First, no element is given a label, unless it belongs to the set

$$E^* = \{e_j\colon (A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j = w_j\}\ .$$

This ensures that any augmenting sequence discovered by the labeling procedure will maintain satisfaction of the orthogonality conditions (7.3).

Suppose the primal solution I is augmented by the application of an augmenting sequence $S = (e_1, e_2, \ldots, e_s)$. We propose to revise the families $\mathcal{U}$ and $\mathcal{V}$ as follows. For $j = 3, 5, \ldots, s$ replace e_{j-1} by e_j in each of the subsets U_i in which e_{j-1} is contained. For $j = 1, 3, \ldots, s-2$, replace e_{j+1} by e_j in each of the subsets V_k in which e_{j+1} is contained. If $u_p = 0$, set $U_p = I$. Otherwise (if $u_p > 0$), set $p = p + 1$ and then set $U_p = I$. Finally, if $v_q = 0$, set $V_q = I$. And if $v_q > 0$, set $q = q + 1$ and set $V_q = I$.

Of course, this revision of the families $\mathcal{U}$, $\mathcal{V}$ does not affect the dual solution, in the sense that the values of no dual variables u_i, v_k are affected. However, unless the augmenting sequence S is of a restricted type, a proper relation will not be maintained between the sets U_i, V_k and the dual variables u_i, v_k. Specifically, it is necessary that the spans $\mathrm{sp}_1(U_i)$, $\mathrm{sp}_2(V_k)$ be unaffected by the changes in membership of U_i and V_k. This objective is insured by a modification of the labeling procedure.

Modify the rules for scanning and labeling as follows:

When a "−" label on $e_j \in I$ is scanned, find the smallest set U_i in which e_j is contained. Denote the index of this set by $u(j)$. Then apply the label "j^+" to each unlabeled element e_i such that $C_i^{(1)} - e_i \subseteq U_{u(j)}$.

When a "+" label on $e_j \in E - I$ is scanned, determine if $I + e_j$ is independent in M_2. If so, an augmenting sequence has been found. Otherwise, find the smallest set V_k such that $C_j^{(2)} - e_j \subseteq V_k$. Then apply the label "$j^-$" to each unlabeled element in $C_j^{(2)} - V_{k-1}$.

We assert that this system of labeling, in conjunction with the previously mentioned system of replacement of elements in $\mathcal{U}$, $\mathcal{V}$, does indeed maintain invariance of spans after augmentation. The reasoning is similar to the proof that the sequences constructed are valid augmenting sequences, and we postpone proof of this latter fact until the next section.

Sequences constructed by this modified labeling procedure are valid alternating sequences, but they do not necessarily satisfy conditions (4.2) and (4.3). For example, suppose we seek a maximum weight intersection of the graphic matroids M_1, M_2 of the multigraphs G_1, G_2 shown in Fig. 1, with element weights $w_1 = 3, w_2 = 5, w_3 = 6, w_4 = 10, w_5 = 8$. Assume that the computation has progressed to the point that $I = \{e_2, e_4\}$, with the following dual solution :

$$
\begin{aligned}
&u_E = 2, \\
&U_1 = \{e_4\}, && u_1 = 4, \\
&U_2 = \{e_2, e_4\}, && u_2 = 2, \\
&V_1 = \{e_4\}, && v_1 = 1, \\
&V_2 = \{e_2, e_4\}, && v_2 = 1 .
\end{aligned}
$$

The reader can easily verify that the dual solution is feasible and that $E^* = E = \{e_1, e_2, \ldots, e_5\}$. Moreover, it satisfies all orthogonality conditions except (7.6).

Now if the unmodified labeling procedure of the cardinality intersection algorithm is applied, the only augmenting sequence constructed is $S = (e_1, e_4, e_5)$, yielding the three-element intersection $I \oplus S = \{e_1, e_2, e_5\}$, with weight 16. However, when the modified labeling procedure is applied, the only augmenting sequence constructed is $S' = (e_1, e_2, e_3, e_4, e_5)$, which yields the three-element intersection $I \oplus S' = \{e_1, e_3, e_5\}$, with weight 17. Note that S' fails to satisfy condition (4.2), because $e_4 \in \mathrm{sp}_1(I) - \mathrm{sp}_1(I - e_1)$. However, in the modified procedure, e_4 can-

not be given the label "1−", because only e_2 is contained in $C_1^{(2)} - V_1$, where $C_1^{(2)} = \{e_1, e_2, e_4\}$ is the unique M_2-circuit contained in $I + e_1$.

The reader should be able to verify that the existing dual solution fails to satisfy orthogonality condition (7.5) for $I \oplus S$, but it does satisfy all orthogonality conditions, other than (7.6), for $I \oplus S'$. Hence $I \oplus S'$ is a maximum-weight intersection of three elements.

If the labeling procedure, as modified above, terminates without the discovery of an augmenting sequence, then the dual solution is revised. This is done as follows.

First we create additional sets in the families $\mathcal{U}$, $\mathcal{V}$, in such a way that each set $U_i - U_{i-1}$ or $V_k - V_{k-1}$ will contain only labeled elements. Let I be partitioned into subsets I_L, I_U of labeled elements and unlabeled elements. For each set U_i, such that $U_i - U_{i-1}$ contains both labeled and unlabeled elements, add one to the indexes of the sets $U_i, U_{i+1}, \ldots, U_p$, and then create a new set

$$U_i = U_{i-1} \cup (U_{i+1} \cap I_U),$$

and set $u_i = 0$. (In the expression for the new set U_i, U_{i+1} is the old U_i, with incremented index.) For each set V_k such that $V_k - V_{k-1}$ contains both labeled and unlabeled elements, add one to the indexes of the sets $V_k, V_{k+1}, \ldots, V_q$, and then create a new set

$$V_k = V_{k-1} \cup (V_{k+1} \cap I_L),$$

and set $v_k = 0$.

Let δ be a positive number yet to be determined. The dual variables are changed as follows. Variable u_E is decreased by δ. If the elements of $U_p - U_{p-1}$ are unlabeled, u_p is increased by δ. If, for $i = 1, 2, \ldots, p-1$, the elements of $U_i - U_{i-1}$ are labeled (unlabeled) and those of $U_{i+1} - U_i$ are unlabeled (labeled), then u_i is decreased (increased) by δ. If the elements of $V_q - V_{q-1}$ are labeled, v_q is increased by δ. If for $k = 1, 2, \ldots, t-1$, the elements of $V_k - V_{k-1}$ are labeled (unlabeled) and those of $V_{k+1} - V_k$ are unlabeled (labeled), then v_k is increased (decreased) by δ. No other dual variables are changed in value.

The reader should convince himself that if the elements in $U_i - U_{i-1}$ are labeled, then the effect of the changes in the dual variables is to decrease $(A^T u)_j$ by δ, for each $e_j \in \mathrm{sp}_1(U_i) - \mathrm{sp}_1(U_{i-1})$. However, if the elements in $U_i - U_{i-1}$ are unlabeled, there is no change in $(A^T u)_j$. Similarly, if the elements in $V_k - V_{k-1}$ are labeled, the effect of the changes in the dual variables is to increase $(B^T v)_j$ by δ, for each $e_j \in \mathrm{sp}_2(V_k) - \mathrm{sp}_2(V_{k-1})$, and if the elements in $V_k - V_{k-1}$ are unlabeled, there is no change in $(B^T v)_j$.

Quite clearly then, for each element $e_j \in I$ (for which $x_j = 1$), there is no change in $(A^T u + B^T v)_j$ caused by the revision of the dual solution. Hence conditions (7.3) continue to be satisfied.

The only dual variables u_i and v_k whose values are increased are associated with sets U_i and V_k for which it is the case that $(Ax)_i = r_i$ and $(Bx)_k = s_k$. Hence conditions (7.4) and (7.5) continue to be satisfied by the revision of the dual solution.

We next need to show that there exists a strictly positive value of δ, such that dual feasibility is maintained. First we confirm that the only dual variables u_i, v_k which are decreased by δ are those which are associated with sets U_i, V_k existing before the creation of new sets. The dual variables identified with these sets had strictly positive values before the revision of the dual solution. Hence there exists a $\delta > 0$ such that the nonnegativity of u, v is maintained.

Now let us consider inequalities of the form $(A^T u + B^T v) \geqslant w$. We have already disposed of the case that $e_j \in I$. Suppose $e_j \in E - \mathrm{sp}_1(I)$. There is no set U_i such that $e_j \in \mathrm{sp}_1(U_i)$, hence the only change in $(A^T u)_j$ is that occasioned by the change in u_E by $-\delta$. The element e_j is labeled if and only if $(A^T u + B^T v)_j = w$. If e_j is labeled, then there is some smallest set V_k such that $e_j \in \mathrm{sp}_2(V_k)$, and all the elements in $V_k - V_{k-1}$ are labeled. (There must be such a set V_k, else $S = (e_j)$ would be an augmenting sequence.) In this case the net effect on $B^T v$ is $+\delta$, and the net change in $(A^T u + B^T v)_j$ is zero. If e_j is unlabeled, the net change in $(A^T u + B^T v)_j$ may be either zero or $-\delta$. (The reader should verify that the change is zero if and only if the labeling and scanning of e_j would not result in the labeling of any previously unlabeled elements.) In any case, if e_j is unlabeled, then $(A^T u + B^T v)_j > w_j$, so there is some strictly positive value of δ which will not cause the dual inequality for e_j to be violated.

Now suppose $e_j \notin I$, but $e_j \in \mathrm{sp}_1(I)$, from which it follows that there is a smallest set U_i such that $e_j \in \mathrm{sp}_1(U_i)$. If e_j is labeled, the labeling resulted from the scanning of a labeled element $e_{j-1} \in I$ contained in U_i but not in U_{i-1}. Hence the elements in $U_i - U_{i-1}$ are labeled and the net change in $(A^T u)_j$ is $-\delta$. There must be a smallest V_k such that $e_j \in \mathrm{sp}_2(V_k)$, else e_j would be the final element of an augmenting sequence. The elements in $V_k - V_{k-1}$ are labeled, hence the net change in $(B^T v)_j$ is $+\delta$ and that in $(A^T u + B^T v)_j$ is zero. This is appropriate since the fact that e_j is labeled implies $(A^T u + B^T v)_j = w_j$.

Finally, suppose $e_j \notin I$, $e_j \in \mathrm{sp}_1(I)$, and e_j is not labeled. Let U_i be the

smallest set in $\mathcal{U}$ such that $e_j \in \mathrm{sp}_1(U_i)$. It is the case that either the elements in $U_i - U_{i-1}$ are unlabeled or the elements in $U_i - U_{i-1}$ are labeled and $(A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j > w_j$. In the first case, $(A^{\mathrm{T}}u)_j$ is unchanged, and in the second $(A^{\mathrm{T}}u)_j$ is decreased by δ. Without analyzing the effects of the changes in the dual variables on $(B^{\mathrm{T}}v)_j$, we observe that $(A^{\mathrm{T}}u - B^{\mathrm{T}}v)_j$ is decreased only if $(A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j > w_j$.

We now conclude that there does exist a strictly positive value of δ which can be chosen, such that dual feasibility is maintained. Let I^- denote the indexes of dual variables u_i, other than u_E, which are to be decreased by δ, K^- the indexes of dual variables u_k which are to be decreased by δ, and J^- the indexes of elements e_j for which $(A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j$ is to be decreased by δ. Then we may choose

$$\delta = \min\{u_E, \delta_u, \delta_v, \delta_w\} > 0 ,$$

where

$$\delta_u = \min\{u_i : i \in I^-\}, \qquad \delta_v = \min\{v_k : k \in K^-\},$$

$$\delta_w = \min\{(A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j - w_j : j \in J^-\} .$$

If $\delta = u_E$, then condition (7.6) is satisfied, the primal solution and the new dual solution are orthogonal and optimal, and the computation is completed. If $\delta < u_E$, but $\delta = \delta_u$ or δ_v, one or more of the dual variables u_i, v_k are reduced to zero and the corresponding sets U_i, V_k (except U_p, U_q) are removed from the families $\mathcal{U}$, $\mathcal{V}$ before the labeling procedure is resumed. This may enable additional elements to be labeled. If $\delta = \delta_w$, then at least one more element e_j enters E^* and is eligible for labeling.

If all the element weights e_j are integers, all arithmetic is integer, and each revision of the dual solution reduces u_E by an integer amount. This observation is sufficient to establish finite termination for the algorithm. However, a more sophisticated argument is given in the next section.

8. Primal-dual weighted intersection algorithm

We now specify the steps of the primal-dual algorithm in detail. In the statement of the algorithm we let $\bar{w}_j = (A^{\mathrm{T}}u + B^{\mathrm{T}}v)_j - w_j$.

Primal-Dual Algorithm for Weighted Matroid Intersections

Step 0 (Start). Set

$$I = \emptyset, \qquad u_E = \max_j \{w_j\},$$
$$\mathcal{U} = \{U_0\} = \{\emptyset\}, \qquad \mathcal{V} = \{V_0\} = \{\emptyset\},$$
$$u_0 = v_0 = 0, \qquad p = q = 0,$$
$$\bar{w}_j = u_E - w_j, \qquad j = 1, 2, \ldots, m,$$
$$E^* = \{e_j : \bar{w}_j = 0\}.$$

Step 1 (Labeling).

1.0. Give each element in $E^* - \mathrm{sp}_1(I)$ the label ϕ^+.

1.1. If there are no unscanned labels, go to step 3. Otherwise, find an element e_j with an unscanned label. If the label is a "+" label go to step 1.2; if it is a "−" label go to step 1.3.

1.2. Scan the "+" label on e_j as follows. If $I + e_j$ is independent in M_2, go to step 2. Otherwise, find the smallest set V_k such that $C_j^{(2)} - e_j \subseteq V_k$ and give each unlabeled element in $C_j^{(2)} - V_{k-1}$ the label "j^-". Return to step 1.1.

1.3. Scan the "−" label on e_j as follows. Find the smallest set $U_{u(j)}$ in which e_j is contained. Apply the label "j^+" to each unlabeled element $e_i \in E^*$ such that $C_i^{(1)} - e_i \subseteq U_{u(j)}$. Return to step 1.1.

Step 2 (Augmentation of Primal Solution). An augmenting sequence S has been discovered, of which e_j (found in step 1.2) is the last element. The elements in S are identified by backtracing. Augment I by adding to I all elements in the sequence with "+" labels and removing from I all elements with "−" labels.

Compute $C_j^{(1)}$, $C_j^{(2)}$, for all $e_j \in E - I$.

Suppose, without loss of generality, the augmenting sequence $S = (e_1, e_2, \ldots, e_s)$. Revise the families $\mathcal{U}$, $\mathcal{V}$ as follows. For $j = 3, 5, \ldots, s$, replace e_{j-1} by e_j in each of the subsets U_i in which e_{j-1} is contained. For $j = 1, 3, \ldots, s-2$, replace e_{j+1} by e_j in each of the subsets V_k in which e_{j+1} is contained. If $u_p = 0$, set $U_p = I$. Otherwise (if $u_p > 0$), set $p = p + 1$ and then set $U_p = I$. If $v_q = 0$, set $V_q = I$. Otherwise, set $q = q + 1$ and set $V_q = I$. Remove all labels from elements and return to step 1.0.

Step 3 (Revision of Dual Solution). Let I_L, I_U denote the subsets of labeled and unlabeled elements of I. For each set $U_i \in \mathcal{U}$, such that $U_i - U_{i-1}$ contains both labeled and unlabeled elements, add one to the indexes of the sets $U_i, U_{i+1}, \ldots, U_p$, and then create a new set

$$U_i = U_{i-1} \cup (U_{i+1} \cap I_U),$$

and set $u_i = 0$. For each set V_k, such that $V_k - V_{k-1}$ contains both labeled and unlabeled elements, add one to the indexes of the sets $V_k, V_{k+1}, \ldots, V_q$, and then create a new set

$$V_k = V_{k-1} \cup (V_{k+1} \cap I_L),$$

and set $v_k = 0$.

Form sets I^+, I^-, K^+, K^- as follows.

$$I^+ = \{i: i = p,\ U_p - U_{p-1} \subseteq I_U, \text{ or } i < p,\ U_{i+1} - U_i \subseteq I_L,\ U_i - U_{i-1} \subseteq I_U\},$$

$$I^- = \{i: i < q,\ U_{i+1} - U_i \subseteq I_U,\ U_i - U_{i-1} \subseteq I_L\},$$

$$K^+ = \{k: k = q,\ V_q - V_{q-1} \subseteq I_L \text{ or } k < q,\ V_{k+1} - V_k \subseteq I_U,\ V_k - V_{k-1} \subseteq I_L\},$$

$$K^- = \{k: k < q,\ V_{k+1} - V_k \subseteq I_L,\ V_k - V_{k-1} \subseteq I_U\}.$$

Form sets J^+, J^- as follows. For each element e_j, let $U_{u(j)}$, $V_{v(j)}$ denote the smallest sets, if any, in $\mathcal{U}$, $\mathcal{V}$ such that $e_j \in \mathrm{sp}_1(U_{u(j)})$, $e_j \in \mathrm{sp}_2(V_{v(j)})$. If these sets do not exist (because $e_j \in E - \mathrm{sp}_1(I)$, $e_j \in E - \mathrm{sp}_2(I)$, respectively), let $U_{u(j)} - U_{u(j)-1} = I_L$, $V_{v(k)} - V_{v(k)-1} = I_U$. Set

$$J^+ = \{j: U_{u(j)} - U_{u(j)-1} \subseteq I_U,\ V_{v(j)} - V_{v(j)-1} \subseteq I_L\},$$

$$J^- = \{j: U_{u(j)} - U_{u(j)-1} \subseteq I_L,\ V_{v(j)} - V_{v(j)-1} \subseteq I_U\}.$$

Set

$$\delta = \min\{u_E, \delta_u, \delta_v, \delta_w\},$$

where

$$\delta_u = \min\{u_i: i \in I^-\}, \qquad \delta_v = \min\{v_k: k \in K^-\},$$

$$\delta_w = \min\{\bar{w}_j: j \in J^-\}.$$

Set $u_E = u_E - \delta$.

Set

$$u_i = u_i + \delta \quad \text{for } i \in I^+, \qquad u_i = u_i - \delta \quad \text{for } i \in I^-,$$

$$v_k = v_k + \delta \quad \text{for } k \in K^+, \qquad v_k = v_k - \delta \quad \text{for } k \in K^-,$$

$$\bar{w}_j = \bar{w}_j + \delta \quad \text{for } j \in J^+, \qquad \bar{w}_j = \bar{w}_j - \delta \quad \text{for } j \in J^-.$$

If $u_E = 0$, stop; the primal and dual solutions are optimal. Otherwise, remove from $\mathcal{U}$, $\mathcal{V}$ all sets U_i, V_k, other than U_p, V_q, for which $u_i = 0$, $v_k = 0$, and renumber the sets in $\mathcal{U}$, $\mathcal{V}$ accordingly. Set $E^* = \{e_j : w_j = 0\}$. Remove all labels and return to step 1.0.

Let us now estimate the complexity of the algorithm. We make the same assumptions about R and $C(m)$ as before. For each augmentation, computation of circuits $C_i^{(1)}$, $C_i^{(2)}$ requires $O(mRC(m))$ running time, as before.

There may be many revisions of the dual variables, each revision requiring $O(m)$ steps for the revision itself, plus a reapplication of the labeling procedure, which is $O(mR)$. If all element weights w_i are integer, then the maximum number of revisions of the dual variables is $W = \max\{w_i\}$, where W is the value of u_E. There is also an application of the labeling procedure for each of the R possible augmentations. Thus, we conclude that the overall running time is no greater than

$$O(mRW + mR^2 + mR^2 C(m)) .$$

Even if the element weights are not integral, or even rational, a bound that is polynomial in m and $C(m)$ can be obtained. Each time a revision is made in the dual solution, at least one of the dual variables U_i, V_k or one of the $\bar{w}_j$ is reduced to zero. With this observation and a careful analysis of the algorithm, we can conclude that at most $O(m^2)$ revisions of the dual solution are possible between successive augmentations. This yields a bound of $O(m^3R^2 + mR^2 C(m))$.

In the following two sections we develop a "primal" algorithm, for which the complexity is $O(mR^3 + mR^2C(m))$.

9. Weighted augmenting sequences

It is possible to construct a "primal" algorithm for the weighted intersection problem which is analogous to certain algorithms for computing minimum-cost network flows. The matroid algorithm proceeds by computing maximum-weight intersections containing successively larger numbers of elements. That is, having obtained I_k, a maximum-weight intersection with k elements, I_{k+1} is obtained from I_k by constructing a "maximum-weight augmenting sequence", in exactly the same way that the corresponding network flow algorithm proceeds from a minimum-cost flow of value v to one of value $v + \epsilon$ by means of a minimum-cost flow augmenting path.

The algorithm is characterized as "primal" because it does not involve

dual variables or the calculation of a dual solution. It is believed that the algorithm is conceptually simpler and also more efficient than the primal-dual algorithm.

For any subset $A \subseteq E$ we let $w(A)$ denote the sum of the weights of the elements in A. I.e.,

$$w(A) = \sum_{e_j \in A} w_j .$$

Given an augmenting sequence S with respect to an intersection I, we define the *weight of* S to be

$$\Delta(S) = w(S - I) - w(S \cap I) .$$

Clearly,

$$w(I \oplus S) = w(I) + \Delta(S) .$$

Theorem 9.1. *Let I_k be a maximum-weight intersection with k elements and let S be a maximum-weight augmenting sequence with respect to I_k. Then $I_k \oplus S$ is a maximum-weight intersection with $k + 1$ elements.*

Proof. At any given point in the primal-dual computation, the intersection I_k existing at that point is of maximum weight with respect to all intersections containing $|I_k| = k$ elements. I_k is augmented by means of an augmenting sequence S to obtain a maximum-weight intersection with $k + 1$ elements. Such a sequence S must be a maximum-weight augmenting sequence, or else $I_k \oplus S$ would not be optimal. It follows that any maximum-weight sequence yields an optimal intersection with $k + 1$ elements.

In the next section we shall show that maximum-weight augmenting sequences can be computed by a procedure that is essentially a shortest path algorithm. Thus, it is clearly possible to start with the empty set and find maximum-weight augmenting sequences to obtain I_1, I_2, I_3, ... maximum-weight intersections with 1, 2, 3, ... elements respectively, stopping when no further augmentation is possible. One can then compare the weights of these various intersections so as to determine an intersection which has maximum weight without restriction on the number of elements.

However, "the maximum weight of intersections is concave in k", just as "the minimum cost of flows is convex in the value of the flow". This

means that if one seeks to compute a maximum-weight intersection without restriction on the number of elements, such a set is given by I_k, where k is the smallest number of elements such that $w(I_k) \geqslant w(I_{k+1})$.

Theorem 9.2. *Let $I_1, I_2, I_3, \ldots$ be maximum-weight intersections with 1, 2, 3, ... elements respectively. Then*

$$w(I_{k+1}) - w(I_k) \leqslant w(I_k) - w(I_{k-1}) \quad \text{for } k = 1, 2, 3 .$$

Proof. Maximal-weight intersections for I_{k-1} and I_{k+1} are feasible solutions of the linear programming problem formulated in Section 7. Any convex combination of these two solutions is also a feasible solution, and is dominated by an optimal solution at an extreme point of the polyhedron corresponding to an intersection with k elements.

10. Primal weighted intersection algorithm

Maximum-weight augmenting sequences with respect to I_k can be found by a procedure that is essentially a shortest path computation. Let

$\Delta(e_j)$ = the weight of a maximum-weight alternating sequence, with e_j as the last element.

We propose to compute $\Delta(e_j)$ by successive approximations. In effect, at successive iterations, we compute $\Delta^{(1)}(e_j)$, $\Delta^{(2)}(e_j)$, ..., where

$\Delta^{(p)}(e_j)$ = the weight of a maximum-weight alternating sequence containing no more than p elements, with e_j as the last element.

Since no alternating sequence contains more than $2|I|+1$ elements, $|I| \subseteq R$, it is clear that a maximum-weight augmenting sequence has weight $\Delta(S)$, where

$$\Delta(S) = \max_{e_j \notin I} \{\Delta^{(2R+1)}(e_j): I_k + e_j \in \mathcal{I}_2\} .$$

A labeling procedure for computing these successive approximations to $\Delta(e_j)$ can be implemented as follows. (Superscripts on $\Delta(e_j)$ are eliminated for conciseness.)

Initially, apply the label "ϕ^+" to each element $e_j \in E - \mathrm{sp}_1(I_k)$ and set $\Delta(e_j) = w_j$. For all other elements e_j, set $\Delta(e_j) = -\infty$.

Thereafter, find an element e_i with an unscanned label and scan it as follows. If the label is a "+" label and $I + e_i$ is dependent in M_2, apply

the (unscanned) label "i^-" to each element $e_j \in C_i^{(2)}$ for which $\Delta(e_i) - w_j > \Delta(e_j)$, and set $\Delta(e_j) = \Delta(e_i) - w_j$. If the label is a "$-$" label, apply the (unscanned) label "i^+" to each element e_j such that $e_i \in C_j^{(1)}$ and $\Delta(e_i) + w_j > \Delta(e_j)$, and set $\Delta(e_j) = \Delta(e_i) + w_j$.

Continue scanning and labeling until all labels are scanned. We assert that at that point $\Delta(e_j)$ has attained the correct value for all e_j. (Labels may be scanned in any order. However, in order to achieve a bound of $O(mR^2)$ on the labeling procedure, it is necessary that labels be scanned in the order in which they are applied.)

Let us work out the same example given in Section 7. Let M_1, M_2, be the graphic matroids of the multigraphs G_1, G_2. By inspection, we know that $I = \{e_2, e_4\}$ is a maximum-weight intersection of two elements, for element weights $w_1 = 3$, $w_2 = 5$, $w_3 = 6$, $w_4 = 10$, $w_5 = 8$.

Initially, e_1 is given the label "ϕ^+" and $\Delta(e_1) = 3$, $\Delta(e_i) = -\infty$, for $i = 2, 3, 4, 5$.

Scanning of the label on e_1 results in the application of labels "1^-" on e_2 and e_4, with $\Delta(e_2) = -2$ and $\Delta(e_4) = -7$. Scanning of the label on e_2 results in the application of the label "2^+" on e_3, with $\Delta(e_3) = 4$. Scanning of the label on e_4 results in the application of the label "4^+" on e_5, with $\Delta(e_5) = +1$. Scanning of the label on e_3 results in a new label "3^-" on e_4, with $\Delta(e_4) = -6$. Scanning of the label on e_5 results in the application of no new labels, since $I + e_5 \in \mathcal{I}_2$. At this point, backtracing from e_5 yields the desired maximum-weight augmenting sequence, but $\Delta(e_5)$ is incorrect. Scanning of the label on e_4 results in the application of the label "4^+" (the same as before) to e_5, with the correct value $\Delta(e_5) = +2$. A final scanning of e_5, with the application of no new labels, completes the labeling procedure.

The only labeled element e_j such that $I + e_j \in \mathcal{I}_2$ is e_5. Backtracing from e_5 yields the augmenting sequence $S = (e_1, e_2, e_3, e_4, e_5)$, the same as in the primal-dual method. Moreover, $w(I \oplus S) = w(I) + \Delta(S) = w(I) + \Delta(e_5)$, as predicted.

If the set I_k to which the labeling procedure is applied is of maximum weight with respect to all intersections containing $|I| = k$ elements, then the sequences produced are valid alternating sequences. However, they do not necessarily satisfy either conditions (4.2) or (4.3). [1]

We now summarize the primal algorithm.

Primal Weighted Intersection Algorithm

Step 0 (Start). Set $I = \emptyset$, $\Delta(S) = -\infty$, and $(e_j) = -\infty$, for all e_j. No elements are labeled.

[1] See note (1) added in proof.

Step 1 (Labeling).

1.0. Apply the label "ϕ^+" to each element $e_j \in E - \mathrm{sp}_1(I)$ and set $\Delta(e_j) = w_j$.

1.1. If there are no unscanned labels and $\Delta(S) > -\infty$, go to step 2. If there are no unscanned labels and $\Delta(S) = -\infty$, go to step 3. Otherwise, from among the elements whose labels are unscanned, find that element e_i whose label was first to be applied. If the label is a "+" label, go to step 1.2; if it is a "−" label, go to step 1.3.

1.2. Scan the "+" label on e_i as follows. If $I + e_i$ is independent in M_2 and $\Delta(e_i) > \Delta(S)$, set $\Delta(S) = \Delta(e_i)$ and $s = i$. Otherwise apply the (unscanned) label "i^-" (replacing any existing label) to each element $e_j \in C_i^{(2)}$ for which $\Delta(e_j) < \Delta(e_i) - w_j$ and set $\Delta(e_j) = \Delta(e_i) - w_j$. Return to step 1.1.

1.3. Scan the "−" label on e_i as follows. Apply the (unscanned) label "i^+" (replacing any existing label) to each element e_j such that $e_i \in C_j^{(1)}$ and $\Delta(e_i) + w_j > (e_j)$, and set $\Delta(e_j) = \Delta(e_i) + w_j$. Return to step 1.1.

Step 2 (Augmentation). A maximum-weight augmenting sequence S can be identified by backtracing from e_s. If $\Delta(S) \leqslant 0$, stop; the existing intersection I is of maximum weight. Otherwise, augment I, compute $C_j^{(1)}$, $C_j^{(2)}$ for all $e_j \notin I$, set $\Delta(e_j) = -\infty$ for all e_j, set $\Delta(S) = -\infty$, remove all labels from elements, and return to step 1.0.

Step 3 (Hungarian Labeling). No augmenting sequence exists. I is not only of maximum weight but of maximum cardinality. The labeling is "Hungarian" and can be used to construct a minimum-rank covering dual to I. Halt.

It is quite easy to estimate the complexity of the primal algorithm. Consider the running time for each of R possible applications of the labeling procedure. The computation of $C_i^{(1)}$, $C_i^{(2)}$ for all $e_i \in E - I$ requires $O(mRC(m))$ running time, as before. Each of the m elements may receive $O(R)$ labels (corresponding to $\Delta^{(1)}(e_j)$, $\Delta^{(2)}(e_j)$, ...). Hence the labeling procedure consumes $O(mR^2)$ running time per augmentation. Backtracing and other operations are dominated by those already mentioned. It follows that the overall running time is $O(mR^3 + mR^2C(m))$.

Acknowledgment

The author must acknowledge the profound influence of Jack Edmonds, from whom he first learned of matroid theory and the matroid polyhedral intersection theorem nearly ten years ago. It was with the objective of finding a proof of that theorem that the author first began to develop the algorithms in this paper. The author is grateful for the con-

tributions several other individuals have made to the research in this paper, particularly S. Kundu and R.M. Karp.

Notes added in proof

(1) The conditions satisfied by augmenting sequences found by both the primal-dual and the primal algorithms are more complicated than (4.2) and (4.3). Moreover, these conditions are rather hard to state in terms of the formalism we have introduced here. A satisfactory characterization of these conditions, together with a "direct" proof of the validity of the primal algorithm (that is, without requiring the primal-dual algorithm for the proof of Theorem 9.1) has been given by Krogdahl [10].

(2) It appears that an algorithm for the weighted matroid intersection problem has also been developed by Iri and Tomizawa [9].

(3) It has been pointed out that direct proofs of Theorem 5.2 exist, and that Theorem 5.3 is an example of the duality of "clutters", as treated in the work of J. Edmonds and D.R. Fulkerson on bottleneck extrema.

(4) It has very recently been brought to the author's attention that several of the results on the cardinality intersection problem, including a version of Theorem 3.1, are contained in [8].

References

[1] J. Edmonds, "Minimum partition of a matroid into independent subsets", *Journal of Research of the National Bureau of Standards* 69B (1965) 67–72.
[2] J. Edmonds and D.R. Fulkerson, "Transversals and matroid partition", *Journal of Research of the National Bureau of Standards* 69B (1965) 147–153.
[3] J. Edmonds, "Submodular functions, matroids and certain polyhedra", *Combinatorial structures and their applications, proceedings of the Calgary international conference* (Gordon and Breach, New York, 1970) pp. 67–87.
[4] J. Edmonds, "Matroids and the greedy algorithm", *Mathematical Programming* 1 (1971) 127–136.
[5] S. Kundu and E.L. Lawler, "A matroid generalization of a theorem of Mendelsohn and Dulmage", *Discrete Mathematics* 4 (1973) 159–163.
[6] W.T. Tutte, *Introduction to the theory of matroids* (American Elsevier, New York, 1971).
[7] D.J.A. Welsh, "On matroid theorems of Edmonds and Rado", *Journal of the London Mathematical Society* 45 (1970) 251–256.

Additional references

[8] M. Aigner and T.A. Dowling, "Matching theory for combinatorial geometries", *Transactions of the American Mathematical Society* 158 (1971) 231–245.
[9] M. Iri and N. Tomizawa, "An algorithm for finding an optimal 'independent assignment' ", University of Tokyo, unpublished.
[10] Stein Krogdahl, "A combinatorial proof for Lawler's Matroid Intersection Algorithm", to appear.

Annals of Discrete Mathematics 1 (1977) 331–342

A "PSEUDOPOLYNOMIAL" ALGORITHM FOR SEQUENCING JOBS TO MINIMIZE TOTAL TARDINESS*

Eugene L. LAWLER
Computer Science Division, University of California, Berkeley, CA

Suppose n jobs are to be processed by a single machine. Associated with each job j are a fixed integer processing time p_j, a due date d_j, and a positive weight w_j. The weighted tardiness of job j in a given sequence is $w_j \max(0, C_j - d_j)$, where C_j is the completion time of job j. Assume that the weighting of jobs is "agreeable", in the sense that $p_i < p_j$ implies $w_i \geq w_j$. Under these conditions, it is shown that a sequence minimizing total weighted tardiness can be found by a dynamic programming algorithm with worst-case running time of $O(n^4P)$ or $O(n^5 p_{\max})$, where $P = \sum p_j$ and $p_{\max} = \max\{p_j\}$. The algorithm is "pseudopolynomial", since a true polynomial-bounded algorithm should be polynomial in $\sum \log_2 p_j$.

1. Introduction

Suppose n jobs are to be processed by a single machine. Associated with each job j are a fixed integer processing time p_j, a due date d_j, and a positive weight w_j. The tardiness of job j in a sequence is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j. The problem is to find a sequence which minimizes total weighted tardiness, $\sum w_j T_j$, where the processing of the first job is to begin at time $t = 0$.

Let us assume that the weighting of jobs is *agreeable*, in the sense that $p_i < p_j$ implies $w_i \geq w_j$. Under these conditions, it is shown in this paper that an optimal sequence can be found by a dynamic programming algorithm with worst-case running time of $O(n^4P)$ or $O(n^5 p_{\max})$, where $P = \sum p_j$, and $p_{\max} = \max\{p_j\}$.

The proposed algorithm is distinguished from previous algorithms [5, 7, 15] for this problem in that its running time is bounded by a function that is polynomial, rather than exponential, in n. However, the present algorithm does not qualify as a polynomial algorithm in the accepted sense of the term. This is because the running time is not bounded by a polynomial in the number of bits required to specify an instance of the problem in binary encoding. To be polynomial in this sense, the running time should be polynomial in $\sum \log_2 p_{ij}$, rather than P or $p_{\max}$.

Although the proposed algorithm is not polynomial with respect to binary encoding of data, it is polynomial with respect to an encoding in which the p_j values are expressed in unary notation. For this reason, we say that the algorithm is *pseudopolynomial.*

* Research supported by National Science Foundation Grant GJ–43227X.

If the weights of jobs are unrestricted ("disagreeable"), then the weighted tardiness problem is NP-complete, even if all data are encoded in unary notation. (See proof in appendix.) This means that the existence of a pseudopolynomial algorithm is very unlikely. Or more precisely, such an algorithm exists if and only if there are similar algorithms for the traveling salesman problem, the three dimensional assignment problem, the chromatic number problem, and other well-known "hard" problems [6].

It should be mentioned that there is as yet no proof that the agreeably weighted tardiness problem is NP-complete with respect to binary encoding. Hence one may still hope to find a polynomial algorithm. Some unsuccessful attempts are described in the final section of this paper.

There are many closely related types of sequencing problems in which the distinctions between agreeable weighting and unrestricted weighting and between binary encoding and unary encoding are significant. For example, suppose all jobs have the same due date. Then the unrestricted weighted tardiness problem can be solved by a pseudopolynomial algorithm with $O(n^2P)$ complexity [10], whereas the agreeably weighted case yields to an $O(n \log n)$ procedure (SPT order). Or suppose we seek to minimize the weighted *number* of tardy jobs (with respect to arbitrary due dates). The unrestricted problem is NP-complete with respect to binary encoding, but can be solved in $O(nP)$ time [10]. The agreeably weighted case can be solved in $O(n \log n)$ time [9, 11].

2. Theoretical development

Theorem 1. *Let the jobs have arbitrary weights. Let π be any sequence which is optimal with respect to the given due dates $d_1, d_2, \ldots, d_n$, and let C_j be the completion time of job j for this sequence. Let d'_j be chosen such that*

$$\min(d_j, C_j) \leqslant d'_j \leqslant \max(d_j, C_j).$$

Then any sequence π' which is optimal with respect to the due dates $d'_1, d'_2, \ldots, d'_n$ is also optimal with respect to $d_1, d_2, \ldots, d_n$ (but not conversely).

Proof. Let T denote total weighted tardiness with respect to $d_1, d_2, \ldots, d_n$ and T' denote total weighted tardiness with respect to $d'_1, d'_2, \ldots, d'_n$. Let π' be any sequence which is optimal with respect to $d'_1, d'_2, \ldots, d'_n$, and let C'_j be the completion time of job j for this sequence. We have

$$T(\pi) = T'(\pi) + \sum_j A_j, \tag{1.1}$$

$$T(\pi') = T'(\pi') + \sum_j B_j \tag{1.2}$$

where, if $C_j \leqslant d_j$,

$$A_j = 0$$

$$B_j = -w_j \max(0, \min(C'_j, d_j) - d'_j),$$

and, if $C_j \geqslant d_j$,

$$A_j = w_j(d'_j - d_j)$$

$$B_j = w_j \max(0, \min(C'_j, d'_j) - d_j).$$

Clearly $A_j \geqslant B_j$ and $\Sigma_j A_j \geqslant \Sigma_j B_j$. Moreover, $T'(\pi) \geqslant T'(\pi')$, because π' is assumed to minimize T'. Therefore the right hand side of (1.1) dominates the right hand side of (1.2). It follows that $T(\pi) \geqslant T(\pi')$ and π' is optimal with respect to $d_1, d_2, \ldots, d_n$. □

Theorem 2. *Suppose the jobs are agreeably weighted. Then there exists an optimal sequence π in which job i precedes job j if $d_i \leqslant d_j$ and $p_i < p_j$, and in which all on time jobs are in nondecreasing deadline order.*

Proof. Let π be an optimal sequence. Suppose i follows j in π, where $d_i \leqslant d_j$ and $p_i < p_j$. Then a simple interchange of i and j yields a sequence for which the total weighted tardiness is no greater. (Cf. [13, proof of Theorem 1].) If i follows j, where $d_i \leqslant d_j$ and i and j are both on time, then moving j to the position immediately following i yields a sequence for which the total weighted tardiness is no greater. Repeated applications of these two rules yields an optimal sequence satisfying the conditions of the theorem. □

In order to simplify exposition somewhat, let us assume for the purposes of the following theorem that all processing times are distinct. If processing times are not distinct, they may be perturbed infinitesimally without upsetting the assumption of agreeable weighting or otherwise changing the problem significantly. Hence there is no loss of generality.

Theorem 3. *Suppose the jobs are agreeably weighted and numbered in nondecreasing due date order, i.e. $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$. Let job k be such that $p_k = \max_j \{p_j\}$. Then there is some integer δ, $0 \leqslant \delta \leqslant n - k$, such that there exists an optimal sequence π in which k is preceded by all jobs j such that $j \leqslant k + \delta$, and followed by all jobs j such that $j > k + \delta$.*

Proof. Let C'_k be the latest possible completion time of job k in any sequence which is optimal with respect to due dates $d_1, d_2, \ldots, d_n$. Let π be a sequence which is optimal with respect to the due dates $d_1, d_2, \ldots, d_{k-1}, d'_k = \max(C'_k, d_k), d_{k+1}, \ldots, d_n$, and which satisfies the conditions of Theorem 2 with respect to these due dates. Let C_k be the completion time of job k for π. By Theorem 1, π is optimal with respect to the original due dates. Hence, by

assumption, $C_k \leq d'_k$. Job k cannot be preceded in π by any job j such that $d_j > d'$, else job j would also be on time, in violation of the conditions of Theorem 2. And job k must be preceded by all jobs j such that $d_j \leq d'_k$. Let δ be chosen to be the largest integer such that $d_{k+\delta} \leq d'_k$ and the theorem is proved. □

3. Dynamic programming solution

Assume the jobs are agreeably weighted and numbered in nondecreasing deadline order. Suppose we wish to find an optimal sequence of jobs $1, 2, \ldots, n$, with processing of the first job to begin at time t. Let k be the job with largest processing time. It follows from Theorem 3 that, for some δ, $0 \leq \delta \leq n - k$, there exists an optimal sequence in the form of:

(i) jobs $1, 2, \ldots, k-1, k+1, \ldots, k+\delta$, in some sequence, starting at time t, followed by

(ii) job k, with completion time $C_k(\delta) = t + \sum_{j \leq k+\delta} p_j$, followed by,

(iii) jobs $k+\delta+1, k+\delta+2, \ldots, n$, in some sequence, starting at time $C_k(\delta)$.

By the well known principle of optimality it follows that the overall sequence is optimal only if the sequences for the subsets of jobs in (i) and (iii) are optimal, for starting times t and $C_k(\delta)$, respectively. This observation suggests a dynamic programming method of solution. For any given subset S of jobs and starting time t, there is a well-defined sequencing problem. An optimal solution for problem S, t can be found recursively from optimal solutions to problems of the form S', t', where S' is a proper subset of S and $t' \geq t$.

The subset S which enter into the recursion are of a very restricted type. Each subset consists of jobs in an interval $i, i+1, \ldots, j$, with processing times strictly less than some value p_k. Accordingly, denote such a set by

$$S(i, j, k) = \{j' \mid i \leq j' \leq j, p_{j'} < p_k\},$$

and let

$$T(S(i, j, k), t) = \text{the total weighted tardiness for an optimal sequence of the jobs in } S(i, j, k), \text{ starting at time } t.$$

By the application of Theorem 3 and the principle of optimality, we have:

$$T(S(i, j, k), t) = \min_{\delta} \{T(S(i, k+\delta, k'), t) + w_{k'} \max(0, C_{k'}(\delta) - d_{k'}) + T(S(k'+\delta+1, j, k'), C_{k'}(\delta)\} \tag{3.1}$$

where k' is such that

$$p_{k'} = \max\{p_{j'} \mid j' \in S(i, j, k)\},$$

and

$$C_{k'}(\delta) = t + \sum p_{j'},$$

where the summation is taken over all jobs $j' \in S(i, k + \delta, k')$.

The initial conditions for the equations (3.1) are

$$T(\phi, t) = 0$$

$$T(\{j\}, t) = w_j \max(0, t + p_j - d_j).$$

It is easy to establish an upper bound on the worst-case running time required to compute an optimal sequence for the complete set of n jobs. There are no more than $O(n^3)$ subsets $S(i, j, k)$. (There are no more than n values for each of the indices, i, j, k. Moreover, several distinct choices of the indices may specify the same subset of jobs.) There are surely no more than $P = \sum p_j \leqslant np_{\max}$ possible values of t. Hence there are no more than $O(n^3P)$ or $O(n^4P_{\max})$ equations (3.1) to be solved. Each equation requires minimization over at most n alternatives and $O(n)$ running time. Therefore the overall running time is bounded by $O(n^4P)$ or $O(n^5p_{\max})$.

At this point we have accomplished the primary objective of this paper, which is to present an algorithm which is polynomial in n. The remaining sections are devoted to a discussion of various computational refinements.

4. Refinements of the algorithm

There are several possible refinements of the basic algorithm that may serve to reduce the running time significantly. However, none of these refinements is sufficient to reduce the theoretical worst-case complexity; some may actually worsen it.

Representation of subsets

It should be noted that $S(i, j, k)$ may denote precisely the same subset of jobs as $S(i', j', k')$ even though $i \neq i'$, $j \neq j'$, $k \neq k'$. The notation used in (3.1) is employed only for convenience in specifying subsets. Obviously, the computation should not be allowed to be redundant.

State generation

Only a very small fraction of the possible subproblems S, t are of significance in a typical calculation. Any practical scheme for implementing the recursion should have two phases. In the first, *subproblem generation* phase, one starts with the problem $S = \{1, 2, \ldots, n\}$, $t = 0$ and successively breaks it down into only those subproblems S, t for which equations (3.1) need to be solved. In the second, *recursion* phase, one solves each of the subproblems generated in the first phase, working in the order opposite to that in which they were generated.

Restriction of δ

It is often not necessary for δ to range over all possible integer values in (3.1). The range of δ can sometimes be considerably restricted by the technique described in the next section, thereby reducing the number of subproblems that need be generated and solved.

Shortcut solutions .

There are some "shortcut" methods of solution for the sequencing problem. Whenever one of these shortcut methods is applicable to a subproblem S, t generated in the first phase of the algorithm, it is unnecessary to solve that problem by recursion of the form (3.1) and no further subproblems need be generated from it. A discussion of shortcut solution methods is given in Section 6.

Branch-and-bound

At least in the case of problems of moderate size, there appears to be relatively little duplication of the subproblems produced in the subproblem generation phase of the algorithm. In other words, the recursion tends to be carried out over a set of subproblems related by a tree structure, or something close to it. It follows that there may be some advantage to a branch-and-bound method, based on the structure of equations (3.1). Such a branch-and-bound method might have a very poor theoretical worst-case running time bound, depending on the nature of the bounding calculation and other details of implementation. However, if a depth-first exploration of the search tree is implemented, storage requirements could be very drastically reduced.

It is apparent that the form of recursion (3.1) furnishes a point of departure for the development of many variations of the basic computation.

5. Restriction of δ

The number of distinct values of δ over which minimization must be carried out in equation (3.1) can sometimes be reduced by appropriately invoking Theorems 1 and 2. If this is done in the state generation phase of the algorithm, there may be a considerable reduction in the number of subproblems which must be solved.

Consider a subproblem S, t. Let k be such that

$$p_k = \max_{j \in S} \{p_j\},$$

and assume that $p_k > p_j$, for all $j \in S - \{k\}$. We also assume that the jobs are numbered so that $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$. The following algorithm determines distinct values δ_i, $i = 1, 2, \ldots \leqslant n - k$, over which it is sufficient to carry out minimization in equation (3.1).

(0) Set $i = 1$.
(1) Set $d'_k = t + \sum_{j \in S'} p_j$, where $S' = \{j \mid d_j \leqslant d_k, j \in S\}$.

Comment. If job k has due date d_k, then by Theorem 2 there exists an optimal sequence in which the completion time of job k is at least as large as d'_k.

(2) If $d'_k > d_k$ set $d_k = d'_k$ and return to Step 1.

Comment. By Theorem 1, there exists a sequence which is optimal with respect to d'_k which is optimal with respect to d_k.

Let j be the largest index in S such that $d_j \leqslant d_k$. Set $\delta_i = j - k$.
Let $S'' = \{j \mid d_j > d_k, j \in S\}$. If S'' is empty, stop. Otherwise, let j' be such that

$$d_{j'} = \min_{j \in S''} \{d_j\},$$

and set $d_k = d_{j'}$. Set $i = i + 1$ and return to Step 1.

As an example of the application of the above procedure, consider the first test problem given in Appendix A of [1]. All $w_j = 1$. The p_j and d_j values are as follows:

j	1	2	3	4	5	6	7	8
p_j	121	79	147	83	130	102	96	88
d_j	260	266	269	336	337	400	683	719

Note that $k = 3$. Equation (3.1) yields:

$$T(\{1,2,\ldots,8\},0) = \min \left\{ \begin{array}{l} T(S(1,3,3),0) + 78 + T(S(4,8,3),347), \\ T(S(1,4,3),0) + 161 + T(S(5,8,3),430), \\ T(S(1,5,3),0) + 291 + T(S(6,8,3),560), \\ T(S(1,6,3),0) + 393 + T(S(7,8,3),662), \\ T(S(1,7,3),0) + 489 + T(S(8,8,3),758), \\ T(S(1,8,3),0) + 577 + T(\phi,846) \end{array} \right\}$$

Applying the procedure above, we obtain $\delta_1 = 3$, $\delta_2 = 5$ and the simplified equation:

$$T(\{1,2,\ldots,8\},0) = \min \left\{ \begin{array}{l} T(S(1,6,3),0) + 393 + T(S(7,8,3),662, \\ \\ T(S(1,8,3),0) + 577 + T(\phi,846) \end{array} \right\} \quad (5.1)$$

6. Shortcut solutions

"Shortcut" solutions are sometimes provided by generalizations of two well-known theorems for the unweighted tardiness problem [3].

Theorem 4. *Let the jobs be given arbitrary weights. Let π be a sequence in which jobs are ordered in nonincreasing order of the ratios w_j/p_j. If all jobs are tardy, then π is optimal.*

Proof. Note that

$$\sum w_j T_j = \sum w_j C_j + \sum w_j \max(0, d_j - C_j) - \sum w_j d_j.$$

It is well-known [14] that π minimizes $\sum w_j C_j$. If all jobs are tardy, then each term in the second summation is zero and that sum is also minimized. □

Note that if jobs are agreeably weighted and processing times are distinct, then w_j/p_j-ratio order is equivalent to shortest processing time order.

Theorem 5. *Let the jobs be given arbitrary weights. Let π be a sequence for which*

$$\max_j \{w_j T_j\}$$

is minimum. If at most one job is tardy, then π is optimal.

Proof. Obvious. □

Note that in the unweighted case, nondecreasing due date order minimizes maximum tardiness. In the case of arbitrary weightings, a minmax optimal order can be constructed by the $O(n^2)$ algorithm given in [8].

The application of these two theorems can be strengthened considerably by applying them to an earlier or a later set of due dates induced by Theorems 1 and 2.

For a problem S, t let the jobs in S be numbered so that $p_1 > p_2 > \cdots > p_n$. New (earlier) deadlines d'_j for the application of Theorem 4 can be induced by the following algorithm.

(0) Set $k = n + 1$.
(1) If $k = 1$, stop. Otherwise, set $k = k - 1$.
(2) Set $d'_k = d_k$.
(3) Let $S^{(k)} = \{j \mid j \in S, d_j \geq d'_k, p_j > p_k\}$. Set $C_k = t + \sum_{j \in S - S^{(k)}} p_j$.

Comment. $S^{(k)}$ contains all those jobs which can be assumed to follow k by Theorem 2.

(4) If $C_k < d'_k$, set $d'_k = C_k$ and return to Step 3. Otherwise, return to Step 1.

New (later) due dates d'_k can be induced by the following algorithm.

(0) Set $k = 1$.
(1) If $k = n$, stop. Otherwise, set $k = k + 1$.
(2) Set $d'_k = d_k$.
(3) Let $S^{(k)} = \{j \mid j \in S, d_j \leq d'_k, p_j < p_k\}$. Set $C_k = t + p_k + \sum_{j \in S^{(k)}} p_j$.
(4) If $C_k > d'_k$, set $d'_k = C_k$ and return to Step 3. Otherwise, return to Step 1.

By Theorem 1, an optimal solution to the sequencing problem with respect to induced due dates d'_j, $j = 1, 2, \ldots, n$, is optimal with respect to the due dates d_j. Hence Theorems 4 and 5 can be applied with respect to the induced due dates.

As an application of Theorems 4 and 5, let us solve equation (5.1). Consider first the application of Theorem 5 to $S(1, 8, 3)$, $t = 0$. If the jobs in $S(1, 8, 3)$ are sequenced in increasing d_j-order, i.e. 1, 2, 4, 5, 6, 7, 8, then jobs 5 and 6 are tardy so Theorem 5 does not apply. However, if induced due dates are computed, it is found that $d'_5 = 515$, with $d'_j = d_j$, for $j \neq 5$. When the jobs are sequenced in increasing d'_j-order, i.e. 1, 2, 4, 6, 5, 7, 8, no jobs are tardy with respect to d'_j. By Theorem 1, the sequence is optimal with respect to the original due dates and $T(S(1, 8, 3), 0) = 178$. Also by Theorem 5, $T(S(1, 6, 3), 0) = 178$. And by Theorem 4, $T(S(7, 8, 3), 662) = 194$. Hence (5.1) becomes:

$$T(\{1, 2, \ldots, 8\}, 0) = \min \begin{Bmatrix} 178 + 393 + 194, \\ 178 + 577 + 0 \end{Bmatrix}$$

$$= 755,$$

as indicated by Baker [1]. An optimal sequence is: 1, 2, 4, 6, 5, 7, 8, 3. Most of the test problems on the same list can be resolved with similar simplicity.

It should be mentioned that even in the case that Theorems 4 and 5 do not yield shortcut solutions, it may be possible to reduce the size of a subproblem with the following observation.

Theorem 6. *Let k be such that $d'_k = \max\{d'_j \mid j \in S\}$, where the d'_j are induced deadlines obtained as above. Let P be the sum of the processing times of jobs in S. If $P + t \leqslant d'_k$, then*

$$T(S, t) = T(S - \{k\}, t) + w_k \max\{0, P + t - d_k\}.$$

Proof. Cf. [2]. □

7. Possibilities for a polynomial algorithm

As we have commented, the status of the agreeably weighted tardiness problem is unclear. The proposed algorithm is only "pseudopolynomial". However, no problem reduction has been devised to show that the problem is NP-complete, and one may still reasonably suppose that a polynomial algorithm does exist.

There are some possibilities that *do not* seem rewarding in searching for a polynomial algorithm. For a given set S, $T(S, t)$ is a piecewise linear function of t. If $T(S, t)$ were also convex, and all $w_j = 1$, then $T(S, t)$ could be characterized by at most $n + 1$ linear segments, with successive slopes $0, 1, 2, \ldots, n$. The function $T(S, t)$

could then be computed in polynomial time, using equation (3.1). Unfortunately, $T(S, t)$ is *not* convex, as can be shown by simple counterexamples.

If the values of δ for which the minimum is obtained in (3.1) were monotonically nondecreasing with t, then this would also suggest a polynomial bounded algorithm. Unfortunately, there are simple counterexamples for this property, as well.

Appendix

The following proof of the unary NP-completeness of the weighted tardiness problem was communicated to the author by M.R. Garey and D.S. Johnson. An alternative proof has been developed by J.K. Lenstra. [12].

The so-called 3-partition problem was shown to be unary NP-complete in [4]. This problem is as follows. Given a set of $3n$ integers $a_1, a_2, \ldots, a_{3n}$ between 1 and $B-1$ such that $\sum a_i = nB$, we wish to determine whether there is a partition of the a_i's into n groups of 3, each summing exactly to B.

The corresponding scheduling problem:

"X"-jobs: $X_i, 1 \leqslant i \leqslant n.$

"A"-jobs: $A_i, 1 \leqslant i \leqslant 3n.$

Processing times: $p(X_i) = L = (16B^2)\dfrac{n(n+1)}{2} + 1, 1 \leqslant i \leqslant n,$

$p(A_i) = B + a_i, 1 \leqslant i \leqslant 3n.$

Weights: $w(X_i) = W = (L + 4B)(4B)\dfrac{n(n+1)}{2} + 1, 1 \leqslant i \leqslant n,$

$w(A_i) = p(A_i) = B + a_i, 1 \leqslant i \leqslant 3n.$

Due dates: $d(X_i) = iL + (i-1)4B, 1 \leqslant i \leqslant n,$

$d(A_i) = 0, 1 \leqslant i \leqslant 3n.$

Question: Is there a schedule π with total weighted tardiness $T(\pi) \leqslant W - 1$?

Suppose the desired partition exists. We may assume without loss of generality that the groups are $\langle a_{3j-2}, a_{3j-1}, a_{3j}\rangle$, $1 \leqslant j \leqslant n$. Consider the following ordering of the jobs:

$$\pi = \langle X_1, A_1, A_2, A_3, X_2, A_4, A_5, A_6, X_3, \ldots, X_i,$$
$$A_{3i-2}, A_{3i-1}, A_{3i}, \ldots, X_n, A_{3n-2}, A_{3n-1}, A_{3n}\rangle.$$

By assumption $\sum_{i=-2}^{0} p(A_{3j-i}) = 4B$ for $1 \leqslant j \leqslant n$. Thus X_i will finish at time $iL + (i-1)4B = d(X_i)$, $1 \leqslant i \leqslant n$, and so none of the X-jobs are tardy. On the other hand, *all* the A jobs are tardy, with tardinesses equal to their completion

times. For each j, $1 \leqslant j \leqslant n$, the three jobs A_{3j-2}, A_{3j-1}, and A_{3j} all finish by $j(L+4B)$, and their total weight is $4B$. Hence their collective weighted tardiness is at most $j(L+4B)4B$. Hence

$$T(\pi) \leqslant \sum_{j=1}^{n} j(4B)(L+4B) = \frac{n(n+1)}{2}(4B)(L+4B) = W-1,$$

and π is the desired schedule.

Conversely, suppose that π is such that $T(\pi) \leqslant W-1$. Clearly no X-job can be tardy, for even a tardiness of 1 would yield $T(\pi) \geqslant W$. Now define W_i to be the total weight of the A-jobs which follow i X-jobs, with $W_{n+1} = 0$ by convention. Then

$$T(\pi) \geqslant \sum_{i=1}^{n} (W_i - W_{i+1})(iL) = L\sum_{i=1}^{n} W_i.$$

Since all X-jobs meet their due date, we must have $W_i \geqslant (n-i+1)4B$, $1 \leqslant i \leqslant n$. Suppose some $W_i \geqslant (n-i+1)4B+1$. Then

$$\sum W_i \geqslant 1 + \sum_{i=1}^{n} (n-i+1)4B = \frac{n(n+1)}{2}(4B)+1.$$

This would imply that

$$T(\pi) \geqslant L(4B)\left(\frac{n(n+1)}{2}\right) + 16B^2\frac{n(n+1)}{2} + 1 = W,$$

a contradiction.

Thus $W_i = (n-i+1)4B$, $1 \leqslant i \leqslant n$. From this we conclude that the set of A-jobs between X_i and X_{i+1} in π has total weight $4B$, $1 \leqslant i \leqslant n-1$, and similarly for the set of A-jobs following X_n. Since all A-jobs have $B+1 \leqslant w(A) \leqslant 2B-1$, each such set must contain exactly 3 jobs. These n groups of 3 jobs correspond to the desired partition. □

Acknowledgement

The author wishes to acknowledge the helpful comments of Dr. Alexander Rinnooy Kan.

References

[1] K.R. Baker, Introduction to Sequencing and Scheduling, (Wiley, New York, 1974).
[2] S. Elmahgraby, The one-machine sequencing problem with delay costs, J. Industrial Eng. 19 (1974) 187–199.
[3] H. Emmons, One-machine sequencing to minimize certain functions of job tardiness, Operations Res. 17 (1969) 701–715.

[4] M.R. Garey and D.S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comp. 4 (1975) 397–411.
[5] M. Held and R.M. Karp, A dynamic programming approach to sequencing problems, J. Soc. Industr. Appl. Math. 10 (1962) 196–210.
[6] R.M. Karp, On the computational complexity of combinatorial problems, Networks 5 (1975) 45–68.
[7] E.L. Lawler, Sequencing problems with deferral costs, Management Sc. 11 (1964) 280–288.
[8] E.L. Lawler, Optimal sequencing of a single processor subject to precedence constraints, Management Sc. 19 (1973) 544–546.
[9] E.L. Lawler, Sequencing to minimize the weighted number of tardy jobs, to appear in Rev. Française Automat. Informat. Recherche Opérationnelle, Suppl. to 10 (1976) 27–33.
[10] E.L. Lawler and J.M. Moore, A functional equation and its application to resource allocation and sequencing problems, Management Sc. 16 (1969) 77–84.
[11] J.M. Moore, An n job, one machine scheduling algorithm for minimizing the number of late jobs, Management Sci. 1 (1968) 102–109.
[12] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, Ann. Discrete Math. 1 (1977) 343–362.
[13] A.H.G. Rinnooy Kan, B.J. Lageweg, J.K. Lenstra, Minimizing total costs in one-machine scheduling, Operations Res. 23 (1975) 908–927.
[14] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logistics Quarterly 3 (1956) 59–66.
[16] V. Srinivasan, A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness, Naval Res. Logistics Quarterly 18 (1971) 317–327.

On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming

E. L. LAWLER

University of California at Berkeley, Berkeley, California

AND

J. LABETOULLE

IRIA, Rocquencourt, France

ABSTRACT. It is shown that certain problems of optimal preemptive scheduling of unrelated parallel processors can be formulated and solved as linear programming problems. As a by-product of the linear programming formulations of these problems, upper bounds are obtained on the number of preemptions required for optimal schedules. In particular it is shown that no more than $O(m^2)$ preemptions are necessary, in order to schedule n jobs on m unrelated processors so as to minimize makespan.

KEY WORDS AND PHRASES: preemptive scheduling, parallel processors, linear programming

CR CATEGORIES: 4.32, 5.39

1. *Introduction*

The general problem we wish to deal with in this paper is that of finding optimal preemptive schedules for independent jobs on unrelated parallel processors. We show that certain specific scheduling problems of this type, e.g. minimization of makespan, can be formulated and solved as linear programming problems. We also show that the linear programming formulations provide a means for establishing upper bounds on the number of preemptions required for an optimal schedule.

As part of the general problem formulation, we assume that there are *m processors*, indexed $i = 1, 2, \ldots, m$, and *n jobs*, indexed $j = 1, 2, \ldots, n$. A processor can work on only one job at a time, and a job can be worked on by only one processor at a time. The processing of a job may be interrupted at any time and resumed at a later time, by the same processor or a different processor. There is no cost and no time loss associated with such an interruption or "preemption."

We assume that the input data for a problem instance include mn positive numbers p_{ij}, where p_{ij} represents the total processing time required to complete job j, if the job is worked on exclusively by processor i. More generally, if processor i works on job j for a total time t_{ij}, then it is necessary that

This is a revised version of the paper "Scheduling of Parallel Machines with Preemptions," IRIA Rapport de Recherche No. 203, 1976, prepared while the first author was a visitor at IRIA. The revision was supported in part by the National Science Foundation under Grant MCS 76-17605.

Authors' addresses: E.L. Lawler, Computer Science Division, University of California at Berkeley, Berkeley, CA 94720; J. Labetoulle, IRIA, Rocquencourt 78150, France.

Journal of the Association for Computing Machinery, Vol. 25, No. 4, October 1978, pp. 612-619

$$\sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1,$$

in order for the job to be completed.

We assume no particular relation between the p_{ij} values. That is, the processors are *unrelated*. This is in contrast to two more specialized cases, as follows. If, for all i, j, k, $p_{ij} = p_{kj}$, then the processors are *identical*. If each p_{ij} can be expressed in the form $p_{ij} = q_i p_j$, where q_i (a "slowness factor") and p_j are parameters associated with machine i and job j, then the machines are said to be *uniform*.

For a given feasible schedule, the last point in time at which job j is processed is its *completion time* C_j. The first and most important problem we wish to consider is that of finding a feasible schedule for which "makespan" or maximum completion time,

$$C_{\max} = \max_j \{C_j\},$$

is minimized. We shall demonstrate that there is a polynomial bounded reduction of this problem to a linear programming problem. More specifically, we shall formulate a linear programming problem in $mn + 1$ variables and $2n + m$ (equality and inequality) constraints. We shall then show that one can obtain an optimal schedule (via "open shop" theory) from an optimal solution to the linear programming problem. As a by-product of this analysis, we prove that there exists a $C_{\max}$-optimal schedule with no more than $O(m^2)$ preemptions.

These results are in contrast to the situation for identical and uniform processors. For identical processors, a very simple $O(n)$ algorithm, due to McNaughton [4], yields a $C_{\max}$-optimal schedule with no more than $m - 1$ preemptions. Gonzalez and Sahni [2] have obtained a more complex $O(n + m \log m)$ algorithm for the case of uniform processors, and show that no more than $2(m - 1)$ preemptions are required for an optimal solution.

There is no known polynomial-bounded algorithm for the general linear programming problem, nor has the problem been shown to be NP-complete. It follows that our solution to the $C_{\max}$ problem for unrelated processors does not resolve the question of whether the problem is either NP-complete or polynomial bounded. However, in a later paper we shall show that *for any fixed number of processors m* there is a polynomial-bounded algorithm. (Note: For the case $m = 2$, there is a particularly efficient algorithm [3].)

Following our discussion of the $C_{\max}$ problem, we consider extensions of the linear programming method of solution to objective functions more general than $C_{\max}$. In particular we consider the problem of minimizing

$$L_{\max} = \max_j \{L_j\}, \tag{1.1}$$

where

$$L_j = C_j - d_j \tag{1.2}$$

denotes the *lateness* of job j with respect to a given *due date* d_j. Upper bounds on the number of preemptions required for an optimal schedule are obtained for this, and for a much more general class of objective functions.

2. *Linear Programming Formulation of C_{max} Problem*

We suppose all jobs are available for processing at time $t = 0$. Consider any feasible schedule of n jobs on m unrelated processors, where with respect to this schedule,

t_{ij} = the total amount of time that processor i works on job j.

It is evident that the values of $C_{\max}$ and t_{ij} for the schedule constitute a feasible solution to the following linear programming problem:

$$\text{minimize} \quad C_{\max}$$

subject to

$$\sum_{i=1}^{m} \frac{t_{ij}}{p_{ij}} = 1, \qquad j = 1, 2, \ldots, n; \tag{2.1}$$

$$\sum_{i=1}^{m} t_{ij} \le C_{\max}, \quad j = 1, 2, \ldots, n; \tag{2.2}$$

$$\sum_{j=1}^{n} t_{ij} \le C_{\max}, \quad i = 1, 2, \ldots, m; \tag{2.3}$$

$$t_{ij} \ge 0.$$

We assert that the converse is also true. That is, for any feasible solution to the linear programming problem, there is a feasible schedule with the same values of t_{ij} and $C_{\max}$. In order to prove this assertion, we solve what Gonzalez and Sahni [1] call the preemptive "open shop" scheduling problem. In Section 3 we indicate a solution to this problem, rather than merely referring to [1], in order to have the tools more readily at hand for obtaining a bound on the number of preemptions required for an optimal solution.

3. *Construction of a Feasible Solution*

Suppose we are given an $m \times n$ nonnegative matrix $T = (t_{ij})$ and a value $C_{\max}$, where

$$C_{\max} = \max\left\{\max_i\left\{\sum_j t_{ij}\right\}, \max_j\left\{\sum_i t_{ij}\right\}\right\}. \tag{3.1}$$

We wish to show that it is possible to construct a feasible schedule with the given value of $C_{\max}$.

The pertinent assumptions are as follows. Processor i is to work on job j for a total amount of time t_{ij}. A processor can work on only one job at a time and a job can be worked on by only one processor at a time. There is no restriction on the order in which a given job can be worked on by the different processors, or on the order in which a given processor can work on jobs. (Hence the term "open shop.") There is no loss of time occasioned by the interruption or preemption of jobs. All jobs are available for processing at time $t = 0$.

Let us call row i (column j) of matrix T *tight* if $\sum_j t_{ij} = C_{\max}$ ($\sum_i t_{ij} = C_{\max}$), and *slack* otherwise. Suppose we are able to find a subset of strictly positive elements of T, with exactly one element of the subset in each tight row and in each tight column and no more than one element in any slack row or column. We shall call such a subset of elements a *decrementing set*, and use it to construct a *partial schedule* of length δ, for some suitably chosen $\delta > 0$. In this partial schedule processor i works on job j for $\min\{t_{ij}, \delta\}$ units of time, for each element t_{ij} in the decrementing set. We then replace t_{ij} by $\max\{0, t_{ij} - \delta\}$, for each element in the decrementing set, thereby obtaining a new matrix T', for which $C'_{\max} = C_{\max} - \delta$ satisfies condition (3.1).

For example, suppose $C_{\max} = 11$ and

$$T = \begin{pmatrix} 3 & \textcircled{4} & 0 & 4 & 11 \\ \textcircled{4} & 0 & 6 & 0 & 10 \\ 4 & 0 & 0 & \textcircled{6} & 10 \\ 11 & 4 & 6 & 10 & \end{pmatrix},$$

with row and column sums as indicated on the margins of the matrix. One possible decrementing set is indicated by the encircled elements. Choosing $\delta = 4$, we obtain $C'_{\max} = 7$ and T' as shown below, with the partial schedule indicated to the right:

$$T' = \begin{pmatrix} \textcircled{3} & 0 & 0 & 4 & 7 & \boxed{2} \\ 0 & 0 & \textcircled{6} & 0 & 6 & \boxed{1} \\ 4 & 0 & 0 & \textcircled{2} & 6 & \boxed{4} \\ 7 & 0 & 6 & 6 & & 4 \end{pmatrix}.$$

A decrementing set of T' is indicated by the encircled elements.

There are various constraints that must be satisfied by δ, in order for $C'_{\max} = C_{\max} - \delta$ to satisfy condition (3.1) with respect to T'. First, if t_{ij} is an element of the decrementing set in a tight row or column, then clearly it is necessary that $\delta \leq t_{ij}$, else there will be a row or column sum of T' which is strictly greater than $C_{\max} - \delta$. Similarly, if t_{ij} is an element of the decrementing set in a slack row (slack column), then it is necessary that

$$\delta \leq t_{ij} + C_{\max} - \sum_k t_{ik} \qquad (\delta \leq t_{ij} + C_{\max} - \sum_k t_{kj}).$$

And if row i (column j) contains no element of the decrementing set (and is therefore necessarily slack), it is necessary that

$$\delta \leq C_{\max} - \sum_j t_{ij} \qquad (\delta \leq C_{\max} - \sum_i t_{ij}).$$

Thus for the example above we have

$$\delta \leq t_{12} = 4, \qquad \delta \leq t_{21} = 4,$$

$$\delta \leq t_{34} + C_{\max} - \sum_k t_{3k} = 7, \qquad \delta \leq t_{34} + C_{\max} - \sum_k t_{k4} = 7,$$

$$\delta \leq C_{\max} - \sum_k t_{k3} = 5.$$

Suppose δ is chosen to be maximum, subject to conditions indicated above. Then either T' will contain at least one less strictly positive element than T or else T' will contain at least one more tight column or tight row (with respect to $C'_{\max}$) than T. It is thus apparent that no more than $r + m + n$ iterations, where r is the number of strictly positive elements in T, are necessary to construct a feasible schedule of length $C_{\max}$.

To illustrate this point, we continue with the example. Choosing $\delta = 3$, we obtain from T' the matrix T'', with the augmented partial schedule shown to the right:

$$T'' = \begin{pmatrix} 0 & 0 & 0 & \textcircled{4} \\ 0 & 0 & \textcircled{3} & 0 \\ \textcircled{4} & 0 & 0 & 0 \end{pmatrix} \begin{matrix} 4 \\ 3 \\ 4 \end{matrix}$$

$$\begin{matrix} 4 & 0 & 3 & 4 \end{matrix}$$

2	1		
1	3		
4	4	∅	
4	6	7	

(The symbol "∅" indicates idle time.) The final decrementing set yields the following complete schedule:

2	1		4	
1	3		3	∅
4	4	∅	1	
4	6	7	10	11

To complete our proof, we need the following lemma.

LEMMA 1. *For any nonnegative matrix T and C_{max} satisfying condition* (3.1), *there exists a decrementing set.*

PROOF. From the $m \times n$ matrix T construct an $(m + n) \times (m + n)$ matrix U, as indicated below:

$$U = \left(\begin{array}{c|c} T & D_m \\ \hline D_n & T^t \end{array}\right).$$

Here T^t denotes the transpose of T. D_m and D_n are $m \times m$ and $n \times n$ diagonal matrices of nonnegative "slacks," determined in such a way that each row sum and column sum of U is equal to $C_{\max}$. It follows that $(1/C_{\max})U$ is a doubly stochastic matrix. The well-known Birkhoff–von Neumann theorem states that a doubly stochastic matrix is a convex combination of permutation matrices. It is easily verified that any one of the permutation matrices in such a convex combination is identified with a decrementing set of T. Q.E.D.

There are several possible ways to construct a decrementing set. For our purposes, it is sufficient to note that one can construct the matrix U from T and then solve an assignment

problem over U, which can be done in polynomial time. This observation, together with the observation that no more than a polynomial number of such assignment problems need be solved, is sufficient to establish a polynomial bound for the schedule construction procedure. Gonzalez and Sahni [1] have obtained time bounds of $O(r^2)$ and $O(r(\min\{r, m^2\} + m \log n))$, where r is the number of strictly positive elements in T.

4. *Bounding the Number of Preemptions*

We now seek to establish an upper bound on the number of preemptions required for a C_{max}-optimal schedule on unrelated parallel processors.

To be precise, we say that a job is *preempted* at time t if execution of the job is suspended on some processor at time t before its completion. If a processor begins or resumes execution of a job at time t' and its processing is continuous until time t, when the job is either completed or preempted, then $[t', t]$ is called an *active period* for the job. The total number of preemptions in a schedule is thus equal to the total number of active periods in excess of n.

Now consider the linear programming problem formulated in Section 2. This problem has n equality constraints (2.1), m inequality constraints (2.2), and m inequality constraints (2.3). It follows from elementary linear programming theory that there exists an optimal basic solution with no more than $n + r_2 + r_3$ strictly positive variables, where r_2 and r_3 denote the number of inequality constraints (2.2) and (2.3) which are satisfied with strict equality. Clearly $0 \le r_3 \le m$. If $n > m$, $0 \le r_2 \le m - 1$. And if $n \le m$, at most $m - 1$ constraints (2.2) are nonredundant. It follows that there is an optimal solution with at most $n + 2m - 1$ strictly positive variables, one of which is C_{max}.

We may thus assume there exists an optimal solution to the linear programming problem with no more than $n + 2(m - 1)$ strictly positive t_{ij} values. If we could construct a schedule (with the given value of C_{max}) with exactly one active period for each positive t_{ij} value, then we should have an upper bound of $2(m - 1)$ on the number of preemptions required for a C_{max}-optimal schedule. However, the schedule construction procedure generally introduces additional preemptions. We must now establish an upper bound on this number.

We shall propose a variation of the schedule construction procedure, with the objective of reducing the number of preemptions in the resulting schedule. (This variation also happens to admit a better polynomial bound on its running time, but this is not our principal concern here.) What we shall do is replace all of the jobs which are represented by a single positive t_{ij} value by m dummy jobs. We shall then apply the schedule construction procedure to find a feasible schedule with these dummy jobs. Finally, we shall create a schedule for the original set of jobs by reassigning the active periods for the dummy jobs to the jobs which they replaced.

Consider the example from the previous section where $C_{max} = 11$ and

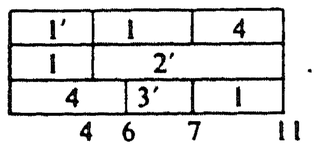
$$T = \begin{pmatrix} 3 & 4 & 0 & 4 & 11 \\ 4 & 0 & 6 & 0 & 10 \\ 4 & 0 & 0 & 6 & 10 \\ 11 & 4 & 6 & 10 & \end{pmatrix}.$$

(This is actually not a *basic* feasible solution to the linear program, but this is of no consequence.) Let us remove the columns containing exactly one strictly positive t_{ij} value and add dummy columns, to obtain the matrix T':

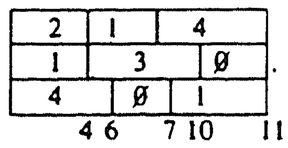
$$T' = \begin{pmatrix} 3 & 4 & 4 & 0 & 0 & 11 \\ 4 & 0 & 0 & 7 & 0 & 11 \\ 4 & 6 & 0 & 0 & 1 & 11 \\ 11 & 10 & 4 & 7 & 1 & \end{pmatrix}.$$

The indices of the jobs identified with the first two columns of T' are 1 and 4. Let us assign indices 1′, 2′, 3′ to the dummy jobs. Note that we have given the dummy jobs t_{ij} values so that all rows of T' are tight.

The schedule construction procedure applied to T' yields as a schedule:



We now fill in the active periods for the dummy jobs with active periods for the jobs which they replaced, plus idle time, obtaining the same schedule as we happened to obtain by the original procedure:



In the case of this example, a schedule was constructed for the matrix T' in which there were no preemptions of the dummy jobs. Hence it was possible to create a schedule for the original set of jobs in which there were no preemptions of any of the jobs which the dummy jobs replaced. In general the number of preemptions required for the original set of jobs is bounded by the number of preemptions in the schedule constructed for the matrix T'.

The matrix T' has at most $m + r_2 + r_3 - 1$ columns and at most $m + 2(r_2 + r_3 - 1)$ strictly positive elements. Each iteration of the schedule construction procedure either reduces an element of the T' matrix to zero, or causes an additional column to become tight. Exactly m elements become zero at the last iteration. Hence there are at most $2(r_2 + r_3) - 1$ iterations of the first kind. There are $m - r_2$ iterations of the second kind, and hence no more than $m + r_2 + 2r_3 - 1$ iterations in all. Each iteration introduces at most m active periods into the resulting schedule, so there are at most $m(m + r_2 + 2r_3 - 1)$ active periods in all. The number of active periods in excess of $m + r_2 + r_3 - 1$, and hence the number of preemptions, is thus bounded by $m(m + r_2 + 2r_3 - 1) - (m + r_2 + r_3 - 1)$. Taking $r_2 = m - 1$, $r_3 = m$, we have the following theorem.

THEOREM 1. *An upper bound on the number of preemptions required for a C_{max}-optimal schedule on unrelated processors is $4m^2 - 5m + 2$.*

The bound indicated by the theorem is certainly not tight, inasmuch as it is known that no more than 2 preemptions are required for the case $m = 2$ [3]. Moreover, we have not been able to establish that $O(m^2)$ preemptions may be required for an optimal schedule, or even that more than $2(m - 1)$ may be necessary.

5. *The L_{max} Problem*

We now formulate a linear programming problem to minimize $L_{\max}$, as defined by (1.1) and (1.2).

Assume the jobs are numbered in nondecreasing due date order, i.e. $d_1 \leq d_2 \leq \cdots \leq d_n$. Let

$t_{ij}^{(1)}$ = the total amount of time that processor i works on job j in the time period $[0, d_1 + L_{\max}]$,

and, for $k = 2, 3, \ldots, n$, let

$t_{ij}^{(k)}$ = the total amount of time that processor i works on job j in the time period $[d_{k-1} + L_{\max}, d_k + L_{\max}]$.

Then we have the linear programming problem

minimize $L_{\max}$

subject to

$$\sum_{i=1}^{m} \sum_{k=1}^{j} \frac{t_{ij}^{(k)}}{p_{ij}} = 1, \qquad j = 1, 2, \ldots, n;$$

$$\sum_{i=1}^{m} t_{ij}^{(1)} \leq d_1 + L_{\max}, \quad j = 1, 2, \ldots, n;$$

$$\sum_{i=1}^{m} t_{ij}^{(k)} \leq d_k - d_{k-1}, \quad j = k, k+1, \ldots, n, \ k = 2, 3, \ldots, n;$$

$$\sum_{j=1}^{n} t_{ij}^{(1)} \leq d_1 + L_{\max}, \quad i = 1, 2, \ldots, m;$$

$$\sum_{j=k}^{n} t_{ij}^{(k)} \leq d_k - d_{k-1}, \quad i = 1, 2, \ldots, m, \qquad k = 2, 3, \ldots, n;$$

$$t_{ij}^{(k)} \geq 0.$$

Given an optimal solution to this linear programming problem, an $L_{\max}$-optimal schedule can be obtained by applying the schedule construction procedure of Section 3 to each matrix $T^{(k)} = (t_{ij}^{(k)})$, $k = 1, 2, \ldots, n$. Let $p(m, k) \leq O(m^2)$ be an upper bound on the number of preemptions required for the subschedule constructed from $T^{(k)}$. Then an upper bound on the total number of preemptions required for an $L_{\max}$-optimal schedule can be seen to be

$$2(m-1)n + \sum_{k=1}^{n} p(m, k) \leq O(m^2 n).$$

It is not difficult to construct examples for which $O(n)$ preemptions are necessary for an $L_{\max}$-optimal schedule.

6. *Costs of Processing*

The linear programming formulations we have obtained suggest that we might include a "cost of processing" in the objective functions for these problems. Let c_{ij} = the cost of processing job j on processor i for one unit of time. Then, for example, rather than only minimizing $L_{\max}$, we may choose to minimize

$$L_{\max} + \sum_i \sum_j \sum_k c_{ij} t_{ij}^{(k)}.$$

It is a well-known fact that the convex combination of any two feasible solutions to a linear programming problem is also a feasible solution. Thus, if $L_{\max}$, T and $L'_{\max}$, T' are feasible solutions, then so is $\lambda L_{\max} + (1-\lambda)L'_{\max}$, $\lambda T + (1-\lambda)T'$, for any λ, $0 \leq \lambda \leq 1$.

Let us say that (L, C) is a *feasible* pair of values if there exists a feasible schedule for which

$$\sum_i \sum_j \sum_k c_{ij} t_{ij}^{(k)} \leq C, \qquad L_{\max} \leq L.$$

The preceding remarks about convexity indicate that the feasible points (L, C) form a convex region in the plane, as indicated in Figure 1. Or, to put it another way, if $L(C)$ denotes the minimum attainable value of $L_{\max}$, over all schedules with cost of processing not exceeding C, then L is a convex function of C.

7. *A General Bound on the Number of Preemptions*

We shall now obtain an upper bound on the number of preemptions required for an optimal schedule, with respect to a very broad class of optimization criteria. Specifically, we suppose that we wish to find a schedule which minimizes

$$f(C_1, C_2, \ldots, C_n) + \sum_i \sum_j c_{ij} t_{ij},$$

where f is a monotone nondecreasing, but otherwise arbitrary, function of the completion times of the jobs, and c_{ij} is defined as in the previous section.

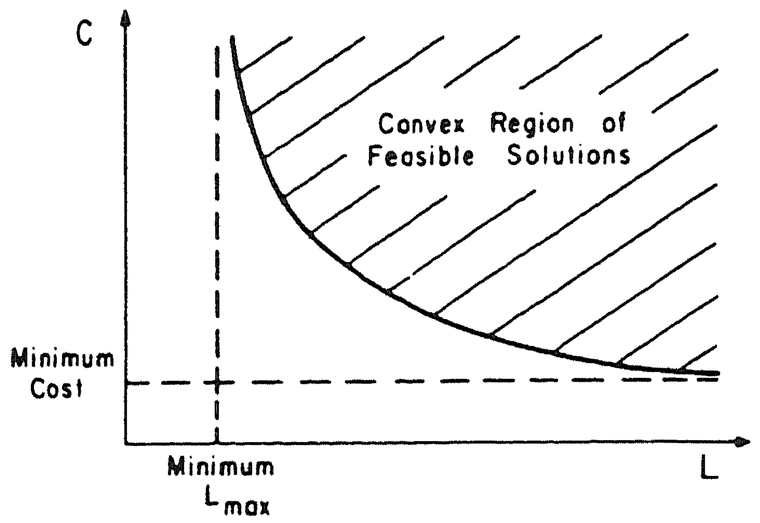


FIG. 1

Suppose there is an optimal schedule with completion times $C_1^*, C_2^*, \ldots, C_n^*$. We can let these completion times assume the roles of deadlines and solve a linear programming problem of the form described in Section 5. The schedule construction procedure can then be applied to obtain an optimal schedule for which we can bound the number of preemptions as follows. From our previous observations, we have the following theorem:

THEOREM 2. *For any monotone nondecreasing function f and coefficients c_{ij}, there exists an optimal schedule in which the number of preemptions is bounded by*

$$2(m-1)n + \sum_{k=1}^{n} p(m,k) \leq O(m^2 n),$$

where $p(m, k)$ is an upper bound on the number of preemptions required for a C_{max}-optimal schedule for k jobs on m unrelated processors.

Theorem 2 can easily be generalized to the case in which we seek to minimize

$$f(S_1, S_2, \ldots, S_n, C_1, C_2, \ldots, C_n) + \sum_i \sum_j c_{ij} t_{ij},$$

where f is monotone nonincreasing in the starting times $S_1, S_2, \ldots, S_n$ and monotone nondecreasing in the completion times $C_1, C_2, \ldots, C_n$. We leave details to the reader.

REFERENCES

(Note. Reference [5] is not cited in the text.)

1. GONZALEZ, T., AND SAHNI, S. Open shop scheduling to minimize finish time. *J. ACM 23*, 4 (Oct. 1976), 665–679.
2. GONZALEZ, T., AND SAHNI, S. Preemptive scheduling of uniform processor systems. *J. ACM 25*, 1 (Jan. 1978), 92–101.
3. SAHNI, S., AND GONZALEZ, T. Preemptive scheduling of two unrelated machines. Tech. Rep. 76-16, Comptr. Sci. Dept., U. of Minnesota, Minneapolis, Minn., Nov. 1976.
4. MCNAUGHTON, R. Sequencing with deadlines and loss functions. *Manage. Sci. 6* (1959), 1–12.
5. STERN, H.I. Minimizing makespan for independent jobs on nonidentical parallel machines—An optimal procedure. Tech. Rep., Dept. of Industrial Eng. and Mgt., Ben-Gurion U. of the Negev, Beer-Sheva, Israel, March 1976.

RECEIVED AUGUST 1976; REVISED DECEMBER 1977

Annals of Discrete Mathematics 2 (1978) 75–90.

SEQUENCING JOBS TO MINIMIZE TOTAL WEIGHTED COMPLETION TIME SUBJECT TO PRECEDENCE CONSTRAINTS

E.L. LAWLER
Computer Science Division, University of California, Berkeley, CA 94720, USA

Suppose n jobs are to be sequenced for processing by a single machine, with the object of minimizing total weighted completion time. It is shown that the problem is NP-complete if there are arbitrary precedence constraints. However, if precedence constraints are "series parallel", the problem can be solved in $O(n \log n)$ time. This result generalizes previous results for the more special case of rooted trees. It is also shown how a decomposition procedure suggested by Sidney can be implemented in polynomial-bounded time.

Equivalence of the sequencing problem with the optimal linear ordering problem for directed graphs is discussed.

1. Introduction

There are n jobs to be sequenced for processing by a single machine. The sequencing of the jobs is to be consistent with precedence constraints imposed by a given acyclic digraph $G = (N, A)$. Each node $j \in N$ is identified with a job. Job i is to precede job j if there is a directed path from i to j in G.

Each job j is assigned a positive processing time p_j and a weight w_j. Since the processing times of the jobs are known and fixed, the completion time C_j of job j is well determined for any given sequence (assuming that the processing of the first job begins at time $t = 0$ and there is no idle time between consecutive jobs). The objective is to find a feasible sequence for which the weighted sum of the completion times, $\sum w_j C_j$, is minimized.

This problem has a history of slow progress from one special case to another. In 1956, Smith [14] showed that, in the absence of precedence constraints, an optimal solution could be found by sequencing the jobs in nondecreasing order of the ratios w_j / p_j. This can, of course, be done in $O(n \log n)$ running time.

Conway, Maxwell and Miller [5], in 1964, proposed a procedure for solving the problem in the case that G is in the form of "parallel chains", and all $w_j = 1$.

In 1971, 1972, Baker [3] and Horn [7] proposed algorithms for the case in which G is a rooted tree, and in 1973 Adolphson and Hu [1] showed that such an algorithm can be implemented in $O(n \log n)$ running time.

Finally, Sidney [13] contributed a number of theorems which apply to the problem with arbitrary precedence constraints. In particular, he showed that "job modules" can be identified and dealt with independently of the remainder of the

problem. He also showed that structures we call "ρ-maximal initial sets" can be used for the purpose of decomposition. However, he indicated no efficient method for finding these ρ-maximal initial sets.

There are three principal contributions in this paper. First, we show that the problem is NP-complete [7] for arbitrary precedence constraints, even if all $w_j = 1$ or all $p_j = 1$. This means that there is very little likelihood of ever finding a polynomial-bounded algorithm for the general problem.

Second, we present an $O(n \log n)$ algorithm for the special case in which G is a "series parallel" digraph. A rooted tree is a special type of series parallel digraph. Hence this result provides a proper generalization of the class of problems solved by Baker, Horn and Adolphson and Hu.

Third, we present a polynomial-bounded algorithm for finding ρ-maximal initial sets. This algorithm can be used as part of a decomposition procedure for problems that are not series parallel. The running time of the algorithm is essentially $O(m^2 n \log n)$, where m is the number of arcs and n the number of nodes of G.

We also establish the equivalence of the sequencing problem with the so-called optimal linear ordering problem for directed graphs. Because of this equivalence, all of the results of this paper, including NP-completeness, apply with equal validity to that problem.

2. NP-Completeness

The following problem, variously called "linear arrangement" or "(undirected) linear ordering", has been shown to be NP-complete by Garey, Johnson and Stockmeyer [6]. Given an arbitrary (undirected) graph $G = (N, A)$, assign the nodes of G to integer points $1, 2, \ldots, n$ on the real line, in such a way that the sum of arc lengths is minimized.

We shall exhibit a reduction, in steps, from the linear arrangement problem to various versions of the sequencing problem defined in the previous section.

Given the undirected graph $G = (N, A)$, create a sequencing problem with a "node" job j for each $j \in N$, and two "arc" jobs, $(i, j)^-$, $(i, j)^+$ for each arc $(i, j) \in A$.

Let

$$\begin{aligned} &p_j = 1, && w_j = 0, && j \in N, \\ &p_{(i,j)^-} = 0, && w_{(i,j)^-} = -1, && (i,j) \in A, \\ &p_{(i,j)^+} = 0, && w_{(i,j)^+} = +1, && (i,j) \in A. \end{aligned}$$

Impose precedence constraints for the sequencing problem which force arc job $(i, j)^-$ to precede each of the node jobs i and j, and force arc job $(i, j)^+$ to follow each of the node jobs i and j.

It should be quite evident, without further explanation, that the sequencing problem is equivalent to the original linear arrangement problem.

We now wish to obtain a problem in which all processing times are strictly positive. It should be intuitively clear that if we let the processing time of each arc job be unity, and the processing time of each node job be "sufficiently large", then the sequencing problem remains essentially unchanged. In particular, let

$$p_{(i,j)^-} = p_{(i,j)^+} = 1, \qquad (i,j) \in A$$
$$p_j = n^4, \qquad j \in N.$$

Suppose, for a given feasible sequence, $K_{(i,j)^-}$, $K_{(i,j)^+}$ denote the number of node jobs preceding arc jobs $(i,j)^-$, $(i,j)^+$, respectively. Then:

$$n^4 K_{(i,j)^-} + 1 \leqslant C_{(i,j)^-} \leqslant n^4 K_{(i,j)^-} + 2m - 1,$$
$$n^4 K_{(i,j)^+} + 2 \leqslant C_{(i,j)^+} \leqslant n^4 K_{(i,j)^+} + 2m,$$

where $m \leqslant (n(n-1)/2)$ is the number of arcs in the original graph G.

The total weighted completion time for the sequence is bounded above and below as follows:

$$n^4\left[\sum K_{(i,j)^+} - \sum K_{(i,j)^-}\right] - 2m^2 + 3m \leqslant \sum C_{(i,j)^+} - \sum C_{(i,j)^-}$$
$$\leqslant n^4\left[\sum K_{(i,j)^+} - \sum K_{(i,j)^-}\right] + 2m^2 - m.$$

Since the term $[\sum K_{(i,j)^+} - \sum K_{(i,j)^-}]$ is an integer, and $4m^2 < n^4$, it is clear that optimality depends only on the number of node jobs preceding the various arc jobs. Hence the new problem is equivalent to the previous problem.

We now wish to make each of the processing times unity. This is arranged quite simply by replacing each node job by a chain of n^4 jobs, each with unit processing time. The reader should have little difficulty in verifying that this does not change the problem.

Once we have obtained unit processing times for all jobs, we are free to add a constant to the weight of each job, without changing the problem. (This has the effect of adding a constant to the total weighted completion time, for any possible sequence.)

We have thus shown that the sequencing problem is NP-complete, *even if all* $p_j = 1$ *and* $w_j \in \{k, k+1, k+2\}$, *for any integer* k.

We now wish to show that the sequencing problem is NP-complete, even if all $w_j = 1$. To do this, we simply take the special case in which $p_j = 1$, $w_j \in \{1,2,3\}$, and do the following. Replace each job j with $w_j = 1$ with a job with $p_j = 3$, $w_j = 1$. Replace each job j with $w_j = 2$ by a chain consisting of a job j_1 with $p_{j_1} = 2$, $w_{j_1} = 1$, followed by a job j_2 with $p_{j_2} = 1$, $w_{j_2} = 1$. Replace each job with $w_j = 3$ by a chain of three jobs j_1, j_2, j_3, with $p_{j_1} = p_{j_2} = p_{j_3} = 1$, $w_{j_1} = w_{j_2} = w_{j_3} = 1$. The reader should be able to verify that this problem is equivalent to the previous problem.

We have thus shown that the sequencing problem is NP-complete, *even if all* $w_j = 1$ *and* $p_j \in \{1,2,3\}$.

The reader may be interested in showing that the case $w_i = 1$, $p_i \in \{0, 1\}$ is NP-complete as well.

3. Series parallel digraphs

The class of *transitive series parallel* digraphs is defined recursively as follows:

(3.1) A digraph consisting of a single node, e.g. $G = (\{i\}, \emptyset)$, is transitive series parallel.

(3.2) If $G_1 = (N_1, A_1)$ and $G_2 = (N_2, A_2)$, where $N_1 \cap N_2 = \emptyset$, are transitive series parallel, then

$$G = G_1 \times G_2 = (N_1 \cup N_2, A_1 \cup A_2 \cup N_1 \times N_2)$$

is also transitive series parallel. G is said to be formed by the *series composition* of G_1 and G_2.

(3.3) If $G_1 = (N_1, A_1)$ and $G_2 = (N_2, A_2)$, where $N_1 \cap N_2 = \emptyset$, are transitive series parallel, then

$$G = G_1 \cup G_2 = (N_1 \cup N_2, A_1 \cup A_2)$$

is also transitive series parallel. G is said to be formed by the *parallel composition* of G_1 and G_2.

(3.4) Only those digraphs which can be obtained by a finite number of applications of rules (3.1)–(3.3) are transitive series parallel.

A digraph G is said to be *series parallel* if and only if its transitive closure is transitive series parallel. Some examples of series parallel digraphs are shown in Fig. 1. Note that every series parallel digraph is acyclic. The simplest acyclic digraph which is not series parallel is shown in Fig. 2.

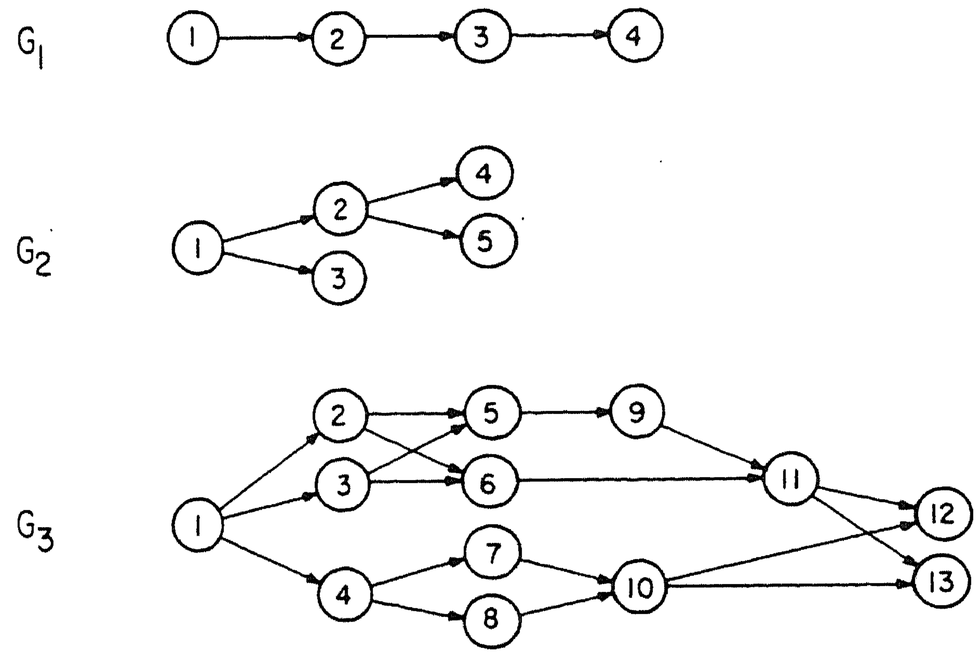


Fig. 1. Series parallel digraphs.

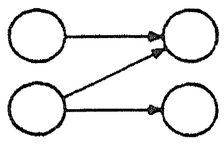


Fig. 2. A Nonseries parallel digraph.

Given a series parallel digraph G, it is possible to repeatedly decompose G into series and parallel components, so as to show how the transitive closure of G is obtained by rules (3.1)–(3.3). The result is a rooted binary tree we call a *decomposition tree*. Each leaf of the decomposition tree is identified with a node of G. Each internal node marked "S" indicates the series composition of the subgraphs identified with its sons, with the convention that the left son precedes the right son. Each internal node marked "P" indicates the parallel composition of the subgraphs identified with its sons. (Here the left-right ordering of sons is unimportant). Decomposition trees T_1, T_2, T_3 for the digraphs G_1, G_2, G_3 shown in Fig. 1 are given in Fig. 3. The "S's" and "P's" in the internal nodes of T_3 are given subscripts to facilitate reference in the next section.

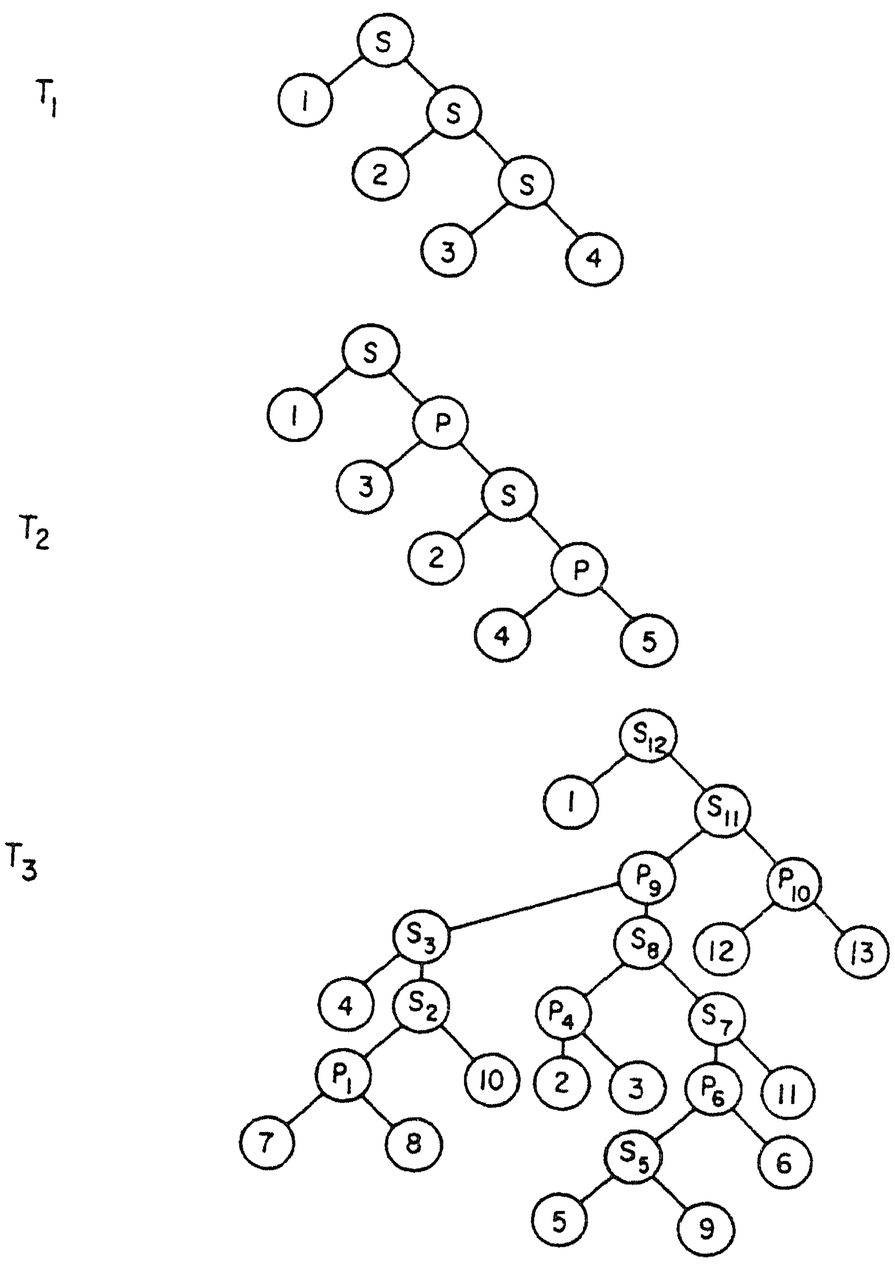


Fig. 3. Decomposition trees.

In [9] it is shown how to test a given digraph to determine if it is series parallel and, if it is, to obtain a decomposition tree. This task can be accomplished on $O(m + n)$ running time. In the sequel, we shall assume that a decomposition tree is already known for any given precedence constraints. Hence, when the claim of $O(n \log n)$ running time is made for the algorithm presented in this paper, this claim must be qualified to apply to a problem for which a decomposition tree is given.

4. Sidney's theory

Sidney's paper [13] contains a total of 25 lemmas and theorems. His most important results, and all that we shall need, are summarized in the three definitions and two theorems below. All definitions are with reference to a given digraph $G = (N, A)$, in which the nodes are identified with jobs.

Definition 1. A nonempty subset $M \subseteq N$ is a *(job) module* if, for each job $j \in N - M$, exactly one of the following three conditions holds:

(4.1) j must precede every job in M,

(4.2) j must follow every job in M,

(4.3) j is not constrained with respect to any job in M.

Every singleton subset of N is a job module, as is N itself. Thus there are $n + 1$ trivial modules for every sequencing problem with $n \geq 2$ jobs. If G contains no arcs (all jobs are independent), then all $2^n - 1$ nonempty subsets of N are job modules.

The family of job modules of a given digraph G seems to exhibit no interesting algebraic structure except that of a semilattice. Given two modules M_1 and M_2, there is a unique smallest module $M \supseteq M_1 \cup M_2$, and M is not hard to determine.

What is important for our purposes is the module structure of a series parallel digraph. If G is series parallel, the subtree rooted to any node of its decomposition tree is identified with a module.

For example, consider the subtree below the node S_8 in tree T_3 of Fig. 3. The leaves of this tree correspond to jobs 2, 3, 5, 6, 9, 11 and $M = \{2, 3, 5, 6, 9, 11\}$ is a module. To verify that this is true, choose any job $j \in N - M$, and find the least common ancestor of j and S_8. For example, the least common ancestor of $j = 8$ and S_8 is node P_9, which shows that 8 is unconstrained with respect to M. The least common ancestor of S_8 and $j = 13$ is S_{11}, which shows that 13 must follow each job in M.

Theorem 1. *Let M be a module of $G = (N, A)$ and σ be an optimal sequence for M. Then there exists an optimal sequence for N consistent with σ (i.e. in which the jobs in M appear in the same order as in σ).*

Proof. See Sidney [13], Lemma 23. □

A very important consequence of Theorem 1 is the following. When an optimal sequence σ for M has been determined, we can proceed as though the digraph G were augmented by arcs forming a chain corresponding to σ. This means that any subsequence of consecutive elements in σ can be dealt with as though they constituted a module of G. (As indeed they do with respect to the augmented graph.)

The reader can probably already anticipate the outlines of a recursive algorithm for solving the series parallel sequencing problem. One starts at the bottom of the decomposition tree and works upward. To obtain an optimal sequence for the series composition of two modules, M_1, M_2, one simply joins together the chains σ_1, σ_2 already determined for M_1, M_2. To obtain an optimal sequence for the parallel composition of M_1, M_2, one somehow "merges" the chains σ_1, σ_2 to obtain a single chain.

Definition 2. Let M be a module. A subset $I \subseteq M$ is an *initial set* of M, if for each $j \in I$, all predecessors of j in M are also in I.

For any subset $I \subseteq N$, let

$$\rho(I) = \frac{\sum_{i \in I} w_i}{\sum_{i \in I} p_i}$$

If each $p_i > 0$, then $\rho(I)$ is well-defined, for all $I \subseteq N$.

Definition 3. Let M be a module. An initial set I^* of M is said to be ρ-maximal if $\rho(I^*) \geqslant \rho(I)$, where I is any initial set of M.

Every module M admits at least one ρ-maximal initial set, possibly M itself.

Theorem 2. *Let M be a module of $G = (N, A)$ and I be a ρ-maximal initial set of M. Then there exists an optimal sequence for N in which the jobs in I are a consecutive subsequence, preceding all other jobs in M.*

Proof. See Sidney [13], Lemmas 3 and 5. These lemmas are stated in terms of a *minimal* ρ-maximal initial set of N, rather than an arbitrary ρ-maximal initial set of an arbitrary module M. However, these differences do not affect the proof significantly. □

One consequence of Theorem 2 is that we can now state a rule for merging two chains σ_1, σ_2 to obtain a single chain for the parallel composition of M_1, M_2. Roughly speaking the procedure is to build up a single sequence σ by adding to it ρ-maximal prefixes of the remaining portions of σ_1, σ_2 (If M is a module consisting of two parallel chains, then there is a ρ-maximal initial set I of M which is an initial subsequence of just one of the chains. After I is added to σ, $M - I$ is also a module

consisting of two parallel chains). This is, in fact, Sidney's "parallel chain" algorithm, which is a generalization of the procedure of Conway, Maxwell and Miller [5]. However, we shall not use this approach, at least not explicitly, because we do not know how it can be implemented to yield the desired running time of $O(n \log n)$.

A more important consequence of Theorem 2, for our purposes, is that whenever a ρ-maximal initial set I is identified in the course of our computations, the jobs in I can be replaced by a single *composite* job. The weight and the processing time of the composite job are set equal to the sum of the weights and the sum of the processing times of the jobs in I. Thereafter, the composite job is treated as though it were a single job.

5. The series parallel algorithm

Recall that we are going to proceed from the bottom of the decomposition tree upward, finding an optimal sequence for a module M from previously determined optimal sequences for its sons, M_1 and M_2. We already know one way in which this can be done: by joining chains of jobs for series composition, and merging chains (e.g. by Sidney's parallel chain algorithm) for parallel composition. However, as we have commented, we know of no way to do this within the time bound we have set for ourselves.

Instead, we propose to represent an optimal sequence for a module simply as a set of jobs, some of which may be composite, as described in the previous section. For any such set of jobs, an optimal sequence can be found by simply placing them in nonincreasing order of the ratios, $\rho(j) = w_j / p_j$. (The precedence constraints will have been dealt with by the formation of composite jobs, so that such a sequence is necessarily feasible). Whereas in the previous paragraph we suggested a very simple rule for series composition and a more difficult one for parallel composition now just the opposite is the case.

For parallel composition of M_1 and M_2, all that is necessary is to form the union of the two sets M_1 and M_2. Nonincreasing ratio order is feasible and optimal for $M = M_1 \cup M_2$, assuming this is true for M_1, M_2 individually.

For series composition of M_1 and M_2, we first find a minimum-ratio job i in M_1 and a maximum-ratio job j in M_2. If $\rho(i) > \rho(j)$, all that is necessary is to form the union of the sets M_1 and M_2. Nonincreasing ratio order is feasible and optimal for $M = M_1 \cup M_2$, assuming this is true for M_1, M_2 individually.

Now suppose $\rho(i) \leqslant \rho(j)$. In this case, $\{i, j\}$ can be treated as a module, consistent with our remarks following Theorem 1. (Job i is the last job in an optimal sequence for M_1 and j is the first job in an optimal sequence for M_2. If we imagine G to be augmented by a chain of arcs corresponding to these optimal sequences, then $\{i, j\}$ is a module with respect to the augmented digraph.) But since $\rho(i) \leqslant \rho(j)$, it follows that $\{i, j\}$ is a ρ-maximal initial set of $\{i, j\}$. Hence, by Theorem 2, there exists an

optimal sequence for N in which i and j are consecutive. What we do is to remove i from M_1, j from M_2 and form a composite job $k = (i, j)$. (Note: either i and j, or both, may themselves be composite jobs. The job k represents a sequence formed by joining together the two sequences represented by i and j.)

Now let us find the next minimal element i in M_1. If $\rho(i) \leqslant \rho(k)$, we remove i from M_1 and form a new composite job $k = (i, k)$. We continue in this way until either M_1 is empty or $\rho(i) > \rho(k)$. Then we find the next maximal element in M_2. If $\rho(k) > \rho(j)$, we can safely let $M = M_1 \cup M_2 \cup \{k\}$. But if $\rho(k) \leqslant \rho(j)$, we remove j from M_2 and form a new composite job $k = (k, j)$. At this point we start all over again with M_1, at the top of this paragraph.

The procedure for series composition is outlined below. In order to avoid tests for emptiness, we assume that M_1 contains a dummy element with ratio $+\infty$ and M_2 a dummy element with ratio $-\infty$.

Step 1. Find a minimal element i in M_1 and a maximal element j in M_2. If $\rho(i) > \rho(j)$, let $M = M_1 \cup M_2$ and halt. Otherwise, remove i from M_1, j from M_2 and form the composite job $k = (i, j)$.

Step 2.

2.1. Find a minimal element i in M_1. If $\rho(i) > \rho(k)$, go to Step 3.1.

2.2. Remove i from M_1 and form the composite job $k = (i, k)$. Return to Step 2.1.

Step 3.

3.1. Find a maximal element j in M_2. If $\rho(k) > \rho(j)$, let $M = M_1 \cup M_2 \cup \{k\}$ and halt.

3.2. Remove j from M_2 and form the composite job $k = (k, j)$. Go to Step 2.1.

It is apparent that this procedure requires various operations:

(1) form the union of two sets,

(2) find a minimal element in a set,

(3) find a maximal element in a set,

(4) remove an element from a set.

None of these operations is performed more than $O(n)$ times overall in determining an optimal sequence for N. For example, note that each time a maximal element is found, either the series composition of two sets is completed or else that element is made part of a composite set, thereby reducing the number of elements by one. Neither of these events can occur more than $O(n)$ times. It follows that if we can insure that each of these operations requires no more than $O(\log n)$ time, we should be able to meet the target of $O(n \log n)$ running time.

There are a variety of data structures which enable us to meet the desired time bound, e.g. "mergeable heaps" [2] and "binomial trees" [15]. Our requirements are slightly novel, in that we require both min and max operations, whereas these data structures are intended to provide only one. Hence there may be some duplication required, e.g. one heap for "min" and a second for "max". However, this is quite feasible to do.

The record-keeping required to keep track of composite jobs is not difficult, and the reader may care to consider various possibilities. At the end of the computation, an optimal sequence is found by listing the jobs in the set N in nonincreasing ratio order. At that point it is necessary to produce the sequence of jobs that has been built up for each composite job in N.

6. Example

As an example, consider the problem with precedence constraints given by digraph G_3 in Fig. 1 and with weights and processing times as shown in Table 1. Optimal solutions for the various subproblems, defined by the decomposition tree T_3 in Fig. 3, are as follows. We denote composite jobs by sequences, e.g. M_8 contains the elementary job 3 and composite jobs (2, 5) and (6, 9, 11). The reader should trace through the subalgorithm for series composition to verify the formation of composite jobs.

Table 1

j	1	2	3	4	5	6	7	8	9	10	11	12	13
w_j	1	1	3	2	7	2	10	3	1	3	4	9	1
p_j	1	4	1	3	2	8	1	5	10	7	8	2	6

$P_1 : M_1 = \{7, 8\}$,
$S_2 : M_2 = \{7, 8, 10\}$,
$S_3 : M_3 = \{(4, 7), 8, 10\}$,
$P_4 : M_4 = \{3, 2\}$,
$S_5 : M_5 = \{5, 9\}$,
$P_6 : M_6 = \{5, 6, 9\}$,
$S_7 : M_7 = \{5, (6, 9, 11)\}$,
$S_8 : M_8 = \{3, (2, 5), (6, 9, 11)\}$,
$P_9 : M_9 = \{(4, 7), 3, (2, 5), 8, 10, (6, 9, 11)\}$,
$P_{10} : M_{10} = \{12, 13\}$,
$S_{11} : M_{11} = \{(4, 7), 3, (2, 5), (8, 10, 6, 9, 11, 12), 13\}$,
$S_{12} : M_{12} = \{(1, 4, 7, 3), (2, 5), (8, 10, 6, 9, 11, 12), 13\}$.

7. Finding ρ-maximal initial sets

Sidney [13] has suggested a decomposition procedure for dealing with precedence constraints imposed by an arbitrary acyclic digraph $G = (N, A)$. First one finds a minimal ρ-maximal initial set I_1 of N. There exists an optimal sequence for N in which the jobs in I_1, optimally ordered, are followed by the jobs in $N - I_1$, also

optimally ordered. Hence set aside the jobs in I_1 and continue, finding a minimal ρ-maximal initial set I_2 for $N - I_1$ in the subgraph induced on $N - I_1$, and then $I_3, I_4, \ldots$, until the jobs in N are exhausted.

This decomposition procedure, of course, does not tell one how to find an optimal sequence for any one of the ρ-maximal initial sets I_i. However, if these sets are proper subsets of N, at least one has succeeded in decomposing the problem into disjoint smaller problems, each of which can be solved by dynamic programming, branch-and-bound or some other method. The difficulty is that no such decomposition may be possible. This occurs when the only ρ-maximal initial set of N is N itself.

We shall now develop a polynomial-bounded algorithm for finding a ρ-maximal initial set. The set I which is found is not necessarily (setwise) minimal. However, if minimality is desired, it should not be difficult for the reader to see how this can be effected, at the expense of a factor of n in the complexity of the algorithm.

The procedure can be used to find a ρ-maximal initial set of an arbitrary module M, by simply applying it to the subgraph of G induced on the nodes of M. For convenience, we shall assume that we wish to find a ρ-maximal initial set of N, where $G = (N, A)$ has m arcs and n nodes.

We shall need as a subroutine a procedure for finding an initial set of maximum total weight, where the weights of the individual jobs are permitted to be either positive or negative. This problem is a slightly more general version of a problem solved by Rhys [12] (see also [4, 11 Chap. 4]), who considered only the case in which G is bipartite. The method of solution is essentially the same as for the Rhys problem.

Let us form a flow network G^* from G, by adding a source s and a sink t, with an arc (s, j) from s and an arc (j, t) to t for each node j of G. These arcs are assigned capacities $c_{sj} = \max(0, -w_j)$, $c_{jt} = \max(0, +w_j)$. Each arc (i, j), $i \neq s$, $j \neq t$, is assigned capacity $c_{ij} = +\infty$. Note that G^* has $n + 2$ nodes and $m + 2n$ arcs.

We recall from network flow theory that a partition of the nodes of G^* into two sets S, T, $s \in S$, $t \in T$, determines an (s, t) cutset. The capacity of such a cutset is:

$$c(S, T) = \sum_{i \in S} \sum_{j \in T} c_{ij},$$

i.e. the sum of the capacities of all arcs in the cutset directed from S to T.

There is a one-to-one correspondence between initial sets of G and finite-capacity cutsets of G^*. For suppose (S, T) is such a cutset. Then $I = T - \{t\}$ is an initial set, because if $i \in S$, $j \in T$, where (i, j) is an arc of G, then (i, j) would be in the cutset (S, T), where $c_{ij} = +\infty$, contrary to the assumption that $c(S, T)$ is finite. The converse is equally easy to prove.

Let (S, T) be a cutset with $c(S, T)$ finite, and let $I = T - \{t\}$. Then

$$c(S, T) = \sum_{j \in S} c_{jt} + \sum_{j \in S} c_{sj}$$

$$= \sum_{j \in S} \max(0, +w_j) + \sum_{j \in T} \max(0, -w_j)$$

$$= \sum_{j \in S \cup T} \max(0, +w_j) - \sum_{j \in T} w_j$$

$$= \text{constant} - \sum_{j \in I} w_j.$$

It follows that a minimum capacity (s, t) cutset yields a maximum weight initial set I. A minimal capacity cutset can be found by standard network flow techniques in $O(m^2 n)$ running time [11, Chap. 4].

We are now prepared to find a ρ-maximal initial set. We shall do this by testing various trial values of the ratio ρ, until we find a maximum feasible value.

In order to test a trial value ρ, the weight w_j of each node is replaced by

$$\overline{w_j} = w_j - \rho p_j.$$

A maximum weight initial set I is then found, with respect to the weights $\overline{w_j}$. There are three possible outcomes:

Case 1. The weight of I is strictly positive. Then

$$\sum_{j \in I} \overline{w_j} = \sum_{j \in I} w_j - \rho \sum_{j \in I} p_j > 0.$$

Since $\sum p_j > 0$, it follows immediately that the trial value ρ is too small; I itself yields a larger ratio.

Case 2. The weight of I is strictly negative. It follows that the trial value ρ is too large.

Case 3. The weight of I is zero. It follows that the trial value ρ is optimal and I is a ρ-maximal initial set.

We propose to carry out a binary search on the optimal value of ρ, very similar to the search the author has proposed for the minimal cost-to-time ratio cycle problem in graphs [10, 11, Chap. 3]. Our convergence argument is essentially the same as that employed for the ratio cycle problem.

Suppose each w_j, p_j is an integer, with

$$-w^* \leqslant w_j \leqslant w^*,$$

$$1 \leqslant p_j \leqslant p^*.$$

Then surely the optimal value of ρ is contained in the interval $[-w^*, w^*]$. Each trial value of ρ halves the remaining interval in which the optimal value of ρ can be contained. To reduce the remaining interval to length ε, no more than

$$\log_2 w^* - \log_2 \varepsilon + 1$$

trial values are required.

We now need to find a value ε which is no greater than the difference between any two distinct attainable ratios. Suppose ρ, ρ', are attainable ratios, where $\rho \neq \rho'$ and

$$\rho = \frac{w}{p}, \qquad \rho' = \frac{w'}{p'}.$$

Then

$$\left|\frac{w}{p} - \frac{w'}{p'}\right| \doteq \left|\frac{wp' - w'p}{pp'}\right| < \left|\frac{1}{(np^*)^2}\right|.$$

It is sufficient to carry out a binary search to the point at which the optimal ratio is known to be in an interval of length not exceeding

$$\varepsilon = \frac{1}{(np^*)^2}.$$

At this point the smallest value in the interval is chosen for a trial value ρ. Either Case 1 or Case 3 must result. In either case, the maximum weight initial set I which is determined is ρ-maximal.

It now follows that an upper bound on the total number of trial values required is:

$$2\log_2 n + \log_2 w^* + 2\log_2 p^* + 1.$$

Each trial value requires an $O(m^2 n)$ network flow computation. Suppose that we are dealing with a population of problems over which w^*, p^* are fixed, and only m and n vary. Then the overall running time required to find a ρ-maximal initial set is $O(m^2 n \log n)$. In any case, even without this supposition, the computation is bounded by a polynomial in the number of bits required to specify an instance of the problem.

8. The directed linear ordering problem

A problem closely related to the sequencing problem is that of optimally ordering the nodes of a weighted digraph. It was, in fact, this problem which Adolphson and Hu [1] dealt with. They established the equivalence of the two problems in the case that the digraph G is a rooted tree. Our purpose here is to make clear the equivalence of the two problems for arbitrary acyclic digraphs.

Let $G = (N, A)$ be an acyclic digraph, with a "width" $p_j > 0$ assigned to each node $j \in N$ and a weight q_{ij} assigned to each arc $(i, j) \in A$. The nodes are to be placed on the real line in the interval $[0, \sum p_j]$, consistent with width restrictions, in such a way that all arcs are directed from "left-to-right". The object is to minimize the weighted sum of the arc lengths.

More precisely, let x_j be the coordinate assigned to node j. If $(i, j) \in A$, then it is required that $x_i < x_j$. If j is the left-most node, then $x_j = p_j$. If nodes i and j are

placed at consecutive positions on the line, then $x_j = x_i + p_j$. The value of the objective function is

$$Z = \sum_{(i,j)\in A} q_{ij}\,(x_j - x_i).$$

Formally, let $q_{ij} = 0$, for all $(i,j) \notin A$. Then

$$\begin{aligned} Z &= \frac{1}{2}\Big[\sum_i \sum_j q_{ij}\,(x_j - x_i)\Big] \\ &= \frac{1}{2}\Big[\sum_i \sum_j q_{ij} x_j - \sum_i \sum_j q_{ij} x_i\Big] \\ &= \frac{1}{2}\Big[\sum_j \sum_i q_{ij} x_j - \sum_j \sum_i q_{ji} x_j\Big] \\ &= \sum_j w_j x_j, \end{aligned}$$

where

$$w_j = \frac{1}{2}\Big[\sum_{(i,j)\in A} q_{ij} - \sum_{(j,i)\in A} q_{ji}\Big]. \tag{8.1}$$

The x_j values can now be interpreted as completion times of jobs in a corresponding sequencing problem with the same digraph G for precedence constraints. Hence it follows that equations (8.1) yield a simple means for reducing a directed linear ordering problem to an equivalent sequencing problem on the same digraph G, with the same p_j values.

As a very simple example, consider the linear ordering problem on the digraph in Fig. 4, with arc weights as indicated by numerical values on the arcs. Each $p_j = 1$. Consider the solution $x_j = j$, where

$$\begin{aligned} Z &= 3(3-1) + 2(3-2) + 1(4-2) + 1(5-3) + 6(5-4) \\ &= 18. \end{aligned}$$

By equations (8.1) the values of w_j for the equivalent sequencing problem are $w_1 = w_2 = 3/2$, $w_3 = 2$, $w_4 = -5/2$, $w_5 = 7/2$. For the feasible solution $C_j = j$, we have:

$$\begin{aligned} Z &= \tfrac{3}{2}(1) + \tfrac{3}{2}(2) + 2(3) - \tfrac{5}{2}(4) + \tfrac{7}{2}(5) \\ &= 18. \end{aligned}$$

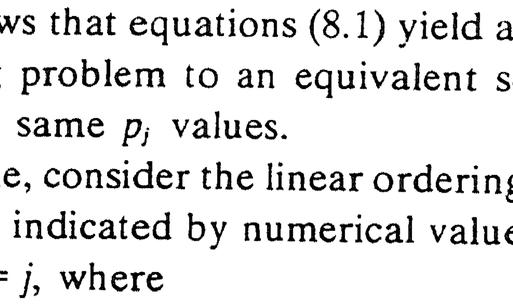


Fig. 4. Example linear ordering problem.

Equations (8.1) also provide the key for obtaining a converse reduction from the sequencing problem to the linear ordering problem. But whereas the w_j's are uniquely determined by the q_{ij}'s as independent variables, the converse is not necessarily the case. The q_{ij}'s may not be uniquely determined. Or much worse, there may be no solution for the q_{ij}'s at all.

As a simple example of this difficulty, let $G = (N, A)$ contain a single arc $(1, 2)$, with $w_1 = w_2 = 1$. Then

$$w_1 = -\tfrac{1}{2} q_{12} = 1,$$

$$w_2 = \tfrac{1}{2} q_{12} = 1,$$

an obvious contradiction.

One way to resolve this problem is to add an $(n+1)^{st}$ node to G, with arcs $(j, n+1)$, for all j. "Slack" weights $q_{j,n+1}$ can always be found to resolve any inconsistency. For our example, we now have

$$w_1 = -\tfrac{1}{2}[q_{12} + q_{13}] = 1,$$

$$w_2 = \tfrac{1}{2}[q_{12} - q_{23}] = 1.$$

One solution is $q_{12} = 2$, $q_{13} = -4$, $q_{23} = 0$.

Since node $n+1$ must always be assigned a position at the extreme right, its position is fixed. The effect of the slack weights $q_{j,n+1}$ is to add a linear cost of the form

$$\sum_j q_{j,n+1}\, x_j$$

to the objective function Z. If the problem is redefined to include such a term in the objective function, then for every sequencing problem on a digraph G, there is an equivalent linear ordering problem on the identical digraph. Otherwise, it may be necessary to find an equivalent linear ordering problem on an $(n+1)$ node digraph.

Notice that when a reduction is made back from the $(n+1)$-node linear ordering problem to a sequencing problem, there is an $(n+1)^{st}$ job which must follow all the others. Since its completion time is fixed, this job simply contributes a constant to the objective function.

It follows that all of the results in this paper apply with equal validity to the directed linear ordering problem. In particular, this problem is NP-complete, even if all node widths are unity. The directed linear ordering problem can be solved in $O(n \log n)$ time, if the digraph is series parallel. And Sidney's decomposition technique can be adapted to the problem.

References

[1] D. Adolphson and T.C. Hu, Optimal linear ordering, SIAM J. Appl. Math. 25 (1973) 403–423.
[2] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).

[3] K.R. Baker, Single machine sequencing with weighting factors and precedence constraints, unpublished paper (1971).
[4] M.L. Balinski, On a selection problem, Management Sci. 17 (1970) 230–231.
[5] R.W. Conway, W.L. Maxwell and L.W. Miller, Theory of Scheduling (Addison-Wesley, Reading, MA, 1967).
[6] M.R. Garey, D.S. Johnson and L. Stockmeyer, Some simple NP-complete problems, Theoret. Comput. Sci. (to appear).
[7] W.A. Horn, Single machine job sequencing with treelike precedence ordering and linear delay penalties, SIAM J. Appl. Math. 23 (1972) 189–202.
[8] R.M. Karp, On the computational complexity of combinatorial problems, Networks 5 (1975) 45–68.
[9] E.L. Lawler, R.E. Tarjan and J. Valdez, Analysis and isomorphism of series parallel digraphs (to appear).
[10] E.L. Lawler, Optimal cycles in doubly weighted directed linear graphs, in: P. Rosenstiehl, ed., Theory of Graphs (Dunod, Paris; Gordon and Breach, NY, 1961) 209–214.
[11] E.L. Lawler, Combinatorial Optimization: Networks and Matroids (Holt, Rinehart and Winston, NY, 1976).
[12] J. Rhys, Shared fixed cost and network flows, Management Sci. 17 (1970) 200–207.
[13] J.B. Sidney, Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs, Operations Res. 22 (1975) 283–298.
[14] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logist. Quart. 3 (1956) 59–66.
[15] J. Vuillemin, A data structure for manipulating priority queues, University of Paris XI, 182 (1976).

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 4, No. 4, November 1979
Printed in U.S.A.

FAST APPROXIMATION ALGORITHMS FOR KNAPSACK PROBLEMS*†

EUGENE L. LAWLER

University of California at Berkeley

Fully polynomial approximation schemes for knapsack problems are presented. These algorithms are based on ideas of Ibarra and Kim, with modifications which yield better time and space bounds, and also tend to improve the practicality of the procedures. Among the principal improvements are the introduction of a more efficient method of scaling and the use of a median-finding routine to eliminate sorting. The 0-1 knapsack problem, for n items and accuracy $\epsilon > 0$, is solved in $(n \log(1/\epsilon) + 1/\epsilon^4)$ time and $O(n + 1/\epsilon^3)$ space. The time bound is reduced to $O(n + 1/\epsilon^3)$ for the "unbounded" knapsack problem. For the "subset-sum" problem, $O(n + 1/\epsilon^3)$ time and $O(n + 1/\epsilon^2)$ space, or $O(n + (1/\epsilon^2)\log(1/\epsilon))$ time and space, are achieved. The "multiple choice" problem, with m equivalence classes, is solved in $O(n \log n + mn/\epsilon)$ time and $O(n + m^2/\epsilon)$ space.

1. Introduction. The "0-1" knapsack problem is as follows: Given n pairs of positive integers, (p_j, a_j) and a positive integer b, find $x_1, x_2, \ldots, x_n$ so as to

$$\text{maximize} \quad P = \sum_j p_j x_j$$

$$\text{subject to} \quad A = \sum_j a_j x_j \leqslant b, \quad x_j \in \{0, 1\}.$$

We may think of j as indexing *items*, with associated *profits* p_j and *weights* a_j. The object is to find the most profitable possible selection of items which can be made to fit into a knapsack with capacity b. One variation of the problem permits items to be chosen with repetition. That is, x_j is permitted to be any nonnegative integer. This is sometimes called the "unbounded" knapsack problem.

There are well-known methods for solving the 0-1 knapsack problem which have worst-case running time of $O(nb)$ [2]. However, these do not qualify as "polynomial-bounded" algorithms, because the running time is not bounded by a polynomial in the length of the encoding of input data. Thus $O(n \log b)$ would be a polynomial bound, but not $O(nb)$. In fact, the problem is known to be NP-complete, even in the case of the "subset-sum" problem, where $p_j = a_j$, for all items. Hence it is very unlikely that any polynomial-bounded algorithm exists.

However, it is possible, within polynomial time, to find a solution which is arbitrarily close to optimum. An approximation scheme for this purpose is an algorithm which receives two inputs: one is the encoded set of data for a problem instance (i.e., pairs (p_j, a_j), and b), the other is a number ϵ, $0 < \epsilon \leqslant 1$, which prescribes the degree of accuracy required. The algorithm then produces a solution with profit P, such that if P^* is the value of an optimal solution,

$$P^* - P \leqslant \epsilon P^*.$$

* Received July 19, 1977; revised December 6, 1978.
AMS 1970 subject classification. Primary 90C10.
IAOR 1973 subject classification. Main: Nonlinear programming. Cross reference: Computational analysis.
Key words. Knapsack problem, approximation algorithms, NP-completeness, polynomial-time computation, integer programming.
† The research reported herein was supported in part by the National Science Foundation Grant MCS 76-17605, and in part by the Mathematical Center, Amsterdam, The Netherlands, where the author was a visitor, January-February 1977.

If for every fixed ϵ, the algorithm operates in time bounded by the length of the encoded input, the algorithm is a "polynomial time approximation scheme." If the algorithm operates in time bounded by a polynomial in the length of the encoded input and in $1/\epsilon$, it is said to be "fully polynomial" [5].

Ibarra and Kim [6] have presented fully polynomial approximation schemes for the 0-1 knapsack problem, and variations. Their most efficient scheme utilizes a lower bound P_0, $P_0 \leq P^* \leq 2P_0$, to separate items according to profits into a class of "small" items and a class of "large" items. The problem is then solved for the large items only, with profits scaled by a suitably chosen scale factor. Feasible solutions of large items are then joined with sets of small items, and the feasible solution with the largest total profit is selected.

By noting that it is possible to obtain a bound, independent of n, on the number of large items which need to be considered for their computation, Ibarra and Kim claim a bound of

$$O\big(n \log n + (1/\epsilon^4)\log(1/\epsilon)\big) \tag{1.1}$$

on running time and

$$O(n + 1/\epsilon^3) \tag{1.2}$$

on space requirements.

In this paper we elaborate on the Ibarra-Kim approach, introducing a number of improvements which yield better time and space bounds and also tend to enhance the practicality of the procedures. Among the modifications proposed are the use of a median-finding routine to eliminate sorting and a more efficient scaling technique. A number of other modifications are proposed, including alternative data structures and a different method for carrying out the large-item computation.

As a result of these changes, we are able to obtain the following bounds. For the 0-1 problem: $O(n \log(1/\epsilon) + 1/\epsilon^4)$ time and $O(n + 1/\epsilon^3)$ space. For the unbounded problem: $O(n + 1/\epsilon^3)$ time and space. For the subset-sum problem: $O(n + 1/\epsilon^3)$ time and $O(n + 1/\epsilon^2)$ space, or $O(n + (1/\epsilon^2)\log(1/\epsilon))$ time and space. The "multiple-choice" problem, with m equivalence classes, is solved in $O(n \log n + mn/\epsilon)$ time and $O(n + m^2/\epsilon)$ space.

The organization of this paper is as follows. In §2 a basic optimization procedure is described. Modifications of this procedure are presented in §§3–11. These yield various approximation schemes for the 0-1 knapsack problem. In §§12 through 15, algorithms are outlined for related problems, including the unbounded problem, the subset-sum problem, and "multiple-choice" knapsack problems. Concluding sections, 16 and 17, indicate possible further extensions and directions for future research.

Comment. Complexity estimates are based on the assumption that computer word length is sufficient to accommodate numbers as large as P^*, b, and n. Arithmetic operations on numbers as large as these are assumed to require constant time. Thus, factors such as $\log b$ do not appear in bounds such as (1.1) and (1.2). However, we shall *not* assume word length on the order of $1/\epsilon$ or n bits (numbers as large as $2^{1/\epsilon}$ or 2^n).

2. A basic optimization procedure. We begin with the description of an optimization algorithm for the 0-1 knapsack problem. All of the approximation algorithms presented in this paper are modifications or adaptations of this basic procedure.

One way to solve the 0-1 knapsack problem is to generate a list of all feasible combinations of profit and weight. Each such combination is represented by a pair

(P, A), for which there is a subset of items S with

$$\sum_{j \in S} p_j = P,$$

$$\sum_{j \in S} a_j = A \leqslant b.$$

The value of P^* is then given by a pair (P, A) for which P is maximum.

The list can be generated in n iterations as follows. Initially place only the pair (0, 0) in the list. Then, at iteration j, form from each pair (P, A) a "candidate" pair $(P + p_j, A + a_j)$, if $A + a_j \leqslant b$, and place the new pair in the list if it does not duplicate an existing one. The result is that at the end of iteration j, each pair in the list represents a feasible profit-weight combination for some subset of items $S \subseteq \{1, 2, \ldots, j\}$, and each such subset is represented by a pair.

This procedure is unnecessarily inefficient, because it generates many pairs which are not needed to determine an optimal solution. It clearly does not affect the computed value of P^* if "dominated" pairs are discarded. That is, if (P, A) is in the list, one may eliminate any pair (P', A'), where $P \geqslant P'$, and $A \leqslant A'$. After dominated pairs are eliminated, each remaining pair (P, A) satisfies the following conditions at the end of iteration j: P is the largest attainable profit for a subset of items $S \subseteq \{1, 2, \ldots, j\}$, with weight at most equal to A, and A is the smallest attainable weight for a subset of items with profit at least equal to P.

The procedure is now revised as follows. At the end of each iteration, the pairs (P, A) are in strictly increasing order of P and of A. To perform iteration j, produce a candidate pair $(P + p_j, A + a_j)$ for each pair (P, A), provided $A + a_j \leqslant b$. Since the list is in increasing order of A, the production of candidate pairs can be terminated whenever a pair (P, A) is reached for which $A + a_j > b$. Then merge the existing list and the list of candidates, discarding dominated pairs in the process. This is easily accomplished, because of the ordering of the P's and A's. At the end of iteration n, the last pair in the list is (P^*, A^*), where A^* is the minimum attainable weight for an optimal solution.

There are various data structures and list merging techniques which are suitable for processing the list of pairs. In any case, it is clear that at each iteration the running time and space requirements are linearly proportional to the number of pairs existing in the list at the beginning of the iteration. An upper bound on the number of pairs in the list is $\min\{P^*, b\}$. Hence an upper bound on the running time for the overall procedure, in the worst case, is $O(n \min\{P^*, b\})$, and an upper bound on space requirements is $O(n + \min\{P^*, b\})$. (The term n in the space bound accounts for the storage of input data.)

Up to this point, we have ignored the problem of constructing an optimal solution, i.e., determining a set S^* corresponding to the last pair (P^*, A^*) in the list at the end of iteration n. There are at least two methods for doing this.

A straightforward method is to convert each pair (P, A) to a triple (P, A, S), where S is a list of indices of items such that

$$\sum_{j \in S} p_j = P, \qquad \sum_{j \in S} a_j = A.$$

Initially, only the triple $(0, 0, \emptyset)$ is placed in the list. Thereafter, at iteration j, each candidate triple is of the form $(P + p_j, A + a_j, S \cup \{j\})$. This has the effect of increasing the space bound to $O(n \min\{P^*, b\})$, since S may be $O(n)$ in size. Forming

the set $S \cup \{j\}$ may be assumed to require $O(n)$ time, thereby also adding an extra factor of n to the time bound, running it up to $O(n^2 \min\{P^*, b\})$.

We choose not to provide an explicit representation of the set S corresponding to a pair (P, A). Instead, we propose to construct S by "backtracing" through a secondary data structure in the form of a rooted tree.

The necessary data structures are indicated in Figure 1. Each entry in the list of (P, A) pairs has four components: a P value, an A value, a pointer to the next entry in the list, and a pointer to a node in the rooted tree. Each tree node has two components: an item index j and a pointer to its parent node.

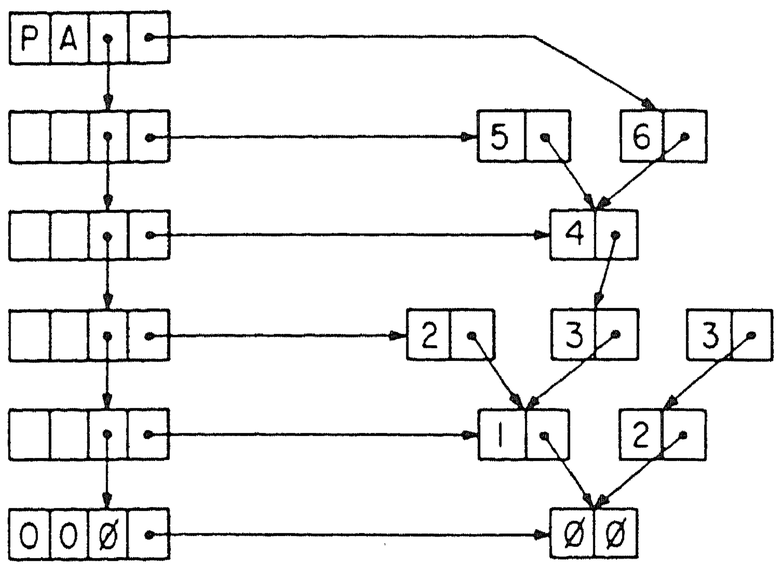


FIGURE 1. Data structures for list of (P, A) pairs and tree for backtracing.

To find the set S for an entry (P, A) in the list, one goes to the tree node indicated by its pointer and then reads off the item indices associated with nodes on the path to the root. Thus, for the entry at the head of the list in Figure 1, $S = \{6, 4, 3, 1\}$. It is easily seen that S can be constructed in $O(n)$ time.

Initially the list contains only the pair (0, 0) and the tree pointer for this pair is directed to the root of the tree. Thereafter, whenever a pair $(P + p_j, A + a_j)$ is added to the list of (P, A) pairs, the tree pointer for $(P + p_j, A + a_j)$ is directed to a new tree node with associated item index j. The pointer for this new node is directed to the node pointed to by (P, A). It is clear that these operations can be implemented in constant time for each new pair $(P + p_j, A + a_j)$, or in $O(n \min\{P^*, b\})$ time overall. Moreover, the tree requires $O(n \min\{P^*, b\})$ space.

3. Scaling of profit space. One way to make the computation more efficient is to reduce the number of distinct P (or A) values which may occur in the pairs (P, A). The simplest method to accomplish this is to replace each p_j coefficient by

$$q_j = \left\lfloor \frac{p_j}{K} \right\rfloor,$$

where K is a suitably chosen scale factor. Then P^*/K replaces P^* in the time and space bounds given above.

How large can we make K, and be assured that the solution we obtain differs from the optimum by no more than ϵP^*? We note the inequality

$$Kq_j \leq p_j < K(q_j + 1). \tag{3.1}$$

It follows that for any set S,

$$\sum_{j \in S} p_j - K \sum_{j \in S} q_j < K|S|.$$

Hence if we can insure that

$$K|S^*| \leqslant \epsilon P^*,$$

where S^* is an optimal solution, then K will be a valid scale factor: the solution found by the computation outlined in the previous section will be within the prescribed accuracy $\epsilon > 0$.

But surely $|S^*| < n$ and $P^* \geqslant p_{\max}$, where

$$p_{\max} = \max_j \{p_j\}.$$

(We can assume $a_j \leqslant b$ for all items.) Hence we may choose

$$K = \frac{1}{n} \epsilon p_{\max}. \tag{3.2}$$

Now $P^* \leqslant np_{\max}$, so

$$P^*/K \leqslant n^2/\epsilon.$$

Substituting n^2/ϵ for P^* in the bounds obtained in the previous section, we obtain time and space bounds of $O(n^3/\epsilon)$, with assurance that the relative error of the solution we obtain does not exceed ϵ, as specified. (Hereafter we ignore the role of b in time and space bounds.) We have thus obtained a fully polynomial approximation scheme.

4. A lower bound on P^*. It is evident that a lower bound on P^* better than $P_{\max}$ will enable us to increase the size of the scale factor K and thereby improve the efficiency of the computation.

If we relax the conditions $x_j \in \{0, 1\}$ to $0 \leqslant x_j \leqslant 1$, a linear programming problem results. It can be solved quite simply. First, sort the items in nonincreasing p_j/a_j ratio order, so that, without loss of generality,

$$\frac{p_1}{a_1} \geqslant \frac{p_2}{a_2} \geqslant \cdots \geqslant \frac{p_n}{a_n}.$$

Then place the items in the knapsack in order until either (a) the items are exhausted or (b) the capacity is exactly used up or (c) it is necessary to fractionalize one item to use up the capacity exactly. In cases (a) and (b), an optimal solution is obtained, and it is unnecessary to proceed further.

So suppose case (c) occurs and

$$a_1 + a_2 + \cdots + a_j < b$$

but

$$a_1 + a_2 + \cdots + a_j + a_{j+1} > b.$$

We assert that

$$P_0 \leqslant P^* \leqslant 2P_0, \tag{4.1}$$

where

$$P_0 = \max\{p_1 + p_2 + \cdots + p_j, p_{\max}\},$$

and where

$$p_{\max} = \max_j \{p_j\},$$

as before. This is because

$$p_1 + p_2 + \cdots + p_j \leqslant P^*, \qquad p_{j+1} \leqslant p_{\max} \leqslant P^*,$$

but

$$p_1 + p_2 + \cdots + p_{j+1} > P^*.$$

Replacing $p_{\max}$ by P_0 in (3.2), we obtain

$$K = \epsilon P_0 / n, \tag{4.2}$$

and find that

$$\frac{P^*}{K} \leqslant \frac{2n}{\epsilon}.$$

The computation can now be carried out in $O(n^2/\epsilon)$ time and space, exclusive of the time required to sort the items in p_j/a_j order.

But sorting is not necessary to compute P_0. This can be done in $O(n)$ time by employing a median-finding algorithm as follows. First compute the ratio p_j/a_j for all items. Then find the median of these ratios. (All ratios are considered to be distinct; if ties occur, the item with the smaller index is considered to have the smaller ratio.) Suppose the median ratio is r and let

$$J = \{j \mid p_j/a_j \geqslant r\}. \tag{4.3}$$

If $\sum_{j\in J} a_j > (<) b$, find the median ratio in (the complement of) J until the largest set J is found such that $\sum_{j\in J} a_j \leqslant b$.

Case (a) above occurs if J contains all items. Case (b) occurs if

$$\sum_{j\in J} a_j = b.$$

Otherwise, case (c) occurs and

$$P_0 = \max\left\{ \sum_{j\in J} p_j, p_{\max} \right\}.$$

There are median-finding routines which require only $O(n)$ time [3]. This procedure requires $O(\log n)$ applications of such a routine, but these are carried out over sets which contain $n, n/2, n/4, \ldots$ elements. It is thus evident that the computation of P_0 requires only $O(n)$ time and space.

In the next section, we shall have need for a procedure which will fill the knapsack to any desired capacity b', $0 \leqslant b' \leqslant b$, just as we filled it to capacity b in computing P_0. This means finding the smallest ratio r such that $\sum_{j\in J} a_j \leqslant b'$, where J is defined as in (4.3). We shall let $\phi(b')$ denote the total profit of the items so placed in the knapsack:

$$\phi(b') = \sum_{j\in J} p_j.$$

5. Separation of items. The existence of the lower bound P_0 enabled us to reduce the time bound from $O(n^3/\epsilon)$ to $O(n^2/\epsilon)$. A technique of Ibarra and Kim enables us to reduce the bound still further, to $O(n/\epsilon^2)$.

Comment. The reader may question whether n/ϵ^2 is "better" than n^2/ϵ. The bounds stated are intended to emphasize asymptotic behavior in n, rather than ϵ. The algorithm we are about to describe will, in fact, be bounded by $O(n^2/\epsilon)$, as well as $O(n/\epsilon^2)$.

The Ibarra-Kim approach is as follows: First compute P_0, as described in the previous section. Use P_0 to determine a threshold value T. Items with $p_j \leqslant T$ are considered "small," and those with $p_j > T$ "large." Solve the problem, using the large

items only, with some appropriately chosen scale factor K. This yields a final list of (Q, A) pairs, where Q denotes total scaled profit.

For each pair (Q, A) in the final list, fill in the remaining knapsack capacity $b - A$ with small items, as indicated in the previous section. These small items yield total profit $\phi(b - A)$. The approximate solution, a combination of large and small items, is chosen to yield profit P, where

$$P = \max_{(Q, A)} \{KQ + \phi(b - A)\}. \tag{5.1}$$

Intuitively, it seems reasonable to divide the permissible error equally between the small items and the large items. This suggests setting $T = \frac{1}{2}\epsilon P_0$, which has the following result. When space $b - A$ is filled with small items, at most one item with profit no greater than $\frac{1}{2}\epsilon P_0$ (half the estimated permissible error) will be omitted. The scale factor K should then be chosen so that the total error contributed by the large items is also no more than one half the permissible amount. Below we justify this intuitive argument, and prove that this choice of T also enables us to maximize the scale factor K.

Suppose there exists an optimal solution in which the large items contribute profit P_L and weight A_L and the small items P_S and A_S. Let Q_L be the sum of the scaled profit coefficients contributing to P_L, using scale factor K. It should be apparent, without need of proof, that the final list for the large-item computation must contain a pair (Q, A), dominating (Q_L, A_L), i.e.,

$$Q \geqslant Q_L, \qquad A \leqslant A_L.$$

Thus P, determined by (5.1), must be such that

$$P \geqslant KQ + \phi(b - A) \geqslant KQ_L + \phi(b - A). \tag{5.2}$$

The number of large items contributing to the optimal solution cannot exceed P_L/T, where

$$P_L/T \leqslant P^*/T. \tag{5.3}$$

Employing the inequality (3.1),

$$Kq_j \leqslant p_j < K(q_j + 1),$$

with (5.3), we obtain

$$P^* = P_L + P_S < K(Q_L + P^*/T) + P_S. \tag{5.4}$$

From (5.2) and (5.4) it follows that

$$P^* - P < KP^*/T + P_S - \phi(b - A). \tag{5.5}$$

But $b - A \geqslant b - A_L$. If our procedure exactly filled the capacity $b - A$ with small items, then $\phi(b - A) \geqslant P_S$. If it left some unused capacity, the profit of the item which could not be fit in was no greater than T, and

$$\phi(b - A) + T \geqslant P_S. \tag{5.6}$$

From inequalities (5.5) and (5.6), we obtain

$$P^* - P < KP^*/T + T. \tag{5.7}$$

It follows that T and K should be chosen to insure that

$$KP^*/T + T \leqslant \epsilon P^*,$$

which can be done by letting

$$K/T = \lambda\epsilon,$$

and

$$T = (1-\lambda)\epsilon P_0 \leq (1-\lambda)\epsilon P^*,$$

for any λ, $0 \leq \lambda < 1$. Assuming $T \neq 0$, this means

$$K = \lambda(1-\lambda)\epsilon^2 P_0.$$

Since we wish to maximize K, we choose $\lambda = \frac{1}{2}$, yielding

$$K = \tfrac{1}{4}\epsilon^2 P_0, \qquad T = \tfrac{1}{2}\epsilon P_0. \tag{5.8}$$

This confirms our intuition as to the correct choice of T and K.

Comment. In [4], the choice of K and T was, in effect:

$$K = \tfrac{2}{9}\epsilon^2 P_0, \qquad T = \tfrac{2}{3}\epsilon P_0,$$

corresponding to a choice of $\lambda = \frac{1}{3}$.

Observe that

$$\lceil P^*/K \rceil \leq 2P_0/\tfrac{1}{4}\epsilon^2 P_0 = 8/\epsilon^2,$$

so the size of scaled profit space is now $O(1/\epsilon^2)$, instead of $O(n/\epsilon)$. Time and space for the large-item computation are bounded by $O(n/\epsilon^2)$, since there are n iterations.

We have yet to discuss the computation of the ϕ-values at the end of the large-item computation. This can be accomplished in $O(n \log(1/\epsilon))$ time, as we shall show in the next section. Hence the overall time and space bounds for the procedure are $O(n/\epsilon^2)$. Now notice that if $T < 1$ (all items "large") as determined by (5.8), then certainly $K < 1$ in (4.2). Hence there is no instance in which the method of the previous section provides a useful scale factor (greater than unity) and the present one does not. Moreover, if

$$T = \tfrac{1}{2}\epsilon P_0 < 1,$$

then one should simply solve the problem optimally, which can be done in $O(n/\epsilon)$ time, since $P^* < 4/\epsilon$.

Next notice that P^*/T can be replaced in (5.7) by n, yielding

$$P^* - P < Kn + T.$$

If we assume that error is to be divided equally between small and large item computations, then

$$K = \epsilon P_0/2n,$$

which results in

$$P^*/K \leq 4n/\epsilon$$

and an $O(n^2/\epsilon)$ computation, as in §4.

These observations suggest that (5.8) should be modified to become

$$\left.\begin{aligned} K &= \max\left\{\frac{1}{4}\epsilon^2 P_0,\ \frac{1}{2n}\epsilon P_0,\ 1\right\}, \\ T &= \begin{cases} \epsilon P_0 & \text{if} \quad K = 1, \\ \frac{1}{2}\epsilon P_0 & \text{if} \quad K > 1. \end{cases} \end{aligned}\right\} \tag{5.9}$$

Equations (5.9) assure a computation which is time and space bounded by both $O(n^2/\epsilon)$ and $O(n/\epsilon^2)$. Moreover, whenever $T < 1$ in (5.9), an *optimal* solution is

obtained in $O(n/\epsilon)$ time. Although in succeeding sections we shall present ideas leading to substantially better asymptotic performance with respect to n, we shall not be able to improve on $O(n^2/\epsilon)$, for asymptotic performance with respect to ϵ.

6. Computation of ϕ-values. It is easy to compute the ϕ-values in $O(n + 1/\epsilon^2)$ time, as in [4], once the small items have been placed in p_j/a_j ratio order. However, we eliminated sorting in the computation of P_0, so we do not have an ordered set of small items available as a byproduct. The procedure we propose is as follows.

First find the median p_j/a_j ratio for the set of all small items. Let this ratio be r and let

$$J = \{j \mid p_j/a_j \geq r\},$$

as before. If $\sum_{j\in J} a_j > b$, throw away the complement of J and repeat, continuing to throw away half-sets until a ratio r is found, such that $\sum_{j\in J} a_j \leq b$. Then search the list of pairs until a pair $(\bar{Q}, \bar{A})$ is found, such that

$$\bar{A} = \max_{(Q,A)} \left\{ A \mid \sum_{j\in J} a_j \leq b - A \right\},$$

and

$$\phi(b - \bar{A}) = \sum_{j\in J} p_j.$$

At this point one ϕ-value has been found in $O(n)$ time, exclusive of the time required to find the pair $(\bar{Q}, \bar{A})$. A set J, $|J| \leq n/2$, has been determined, as has a complementary set $\bar{J}$, $|\bar{J}| \leq n/2$, within the set of remaining small items. Proper subsets of J determine $\phi(b - A)$ for $A > \bar{A}$. Proper subsets of $\bar{J}$, joined with all the items in J, determine $\phi(b - A)$ for $A < \bar{A}$. Thus there are now two subproblems of the same character as the original one.

The difficulty in estimating the remaining time required for the procedure is that we cannot be sure how many pairs are involved for each subproblem: about $4/\epsilon^2$ for each or $8/\epsilon^2$ for one and none for the other? We can confirm that time is bounded by $O(n \log(1/\epsilon))$ by the following analysis.

Let

$T(m, n)$ = the time required for median finding, given m pairs and n small items.

Then

$$T(m, n) \leq cn + \max_{1\leq q\leq m} \left\{ T\left(m - q, \frac{n}{2}\right) + T\left(q - 1, \frac{n}{2}\right) \right\},$$

where c is a constant. Assume $T(m, n) \leq cn \log_2 m$. Then

$$T(m, n) \leq cn + \max_q \left\{ \frac{cn}{2} [\log_2(m - q) + \log_2(q - 1)] \right\}$$

$$\leq cn + cn \log_2\left(\frac{m-1}{2}\right) \leq cn \log_2 m,$$

as required.

Thus, median-finding requires at most $O(n \log(1/\epsilon))$ time, since m is $O(1/\epsilon^2)$. Other operations required in determining the ϕ-values (such as locating pairs $(\bar{Q}, \bar{A})$) are bounded by $O((1/\epsilon^2)\log(1/\epsilon))$.

7. Discarding superfluous items. The time and space bounds can be further reduced by the following observations.

The number of distinct q_j values that can exist for the large items is bounded by

$$\left\lceil \frac{P^*}{K} \right\rceil - \left\lfloor \frac{T}{K} \right\rfloor \leqslant \frac{8}{\epsilon^2} - \frac{2}{\epsilon}.$$

The number of large items with scaled profit q_j which can be contained in any feasible solution is at most

$$n_j = \left\lfloor \frac{P^*}{Kq_j} \right\rfloor.$$

Hence for any scaled profit q_j, one need only consider the n_j items with smallest weight for use in the large-item computation.

For q_j values in the interval $(2/\epsilon, 4/\epsilon]$, the average value of n_j is about $3/\epsilon$. For the interval $(4/\epsilon, 8/\epsilon]$, the average value of n_j is about $3/2\epsilon$. For each successive interval, the average value of n_j is half as large, but the interval contains twice as many q_j values. There are at most $\log_2(P^*/T) \leqslant \log_2(4/\epsilon)$ intervals. Hence the number of items which must be considered for the large-item computation is bounded by about

$$\frac{6}{\epsilon^2} \log_2\left(\frac{4}{\epsilon}\right). \tag{7.1}$$

Identifying the items which need be considered for the large-item computation is simple. Place the large items in at most $\max\{n, 8/\epsilon^2\}$ buckets, each bucket containing items with the same q_j value. Then apply a median-finding routine to each bucket q_j, to identify the n_j items with smallest weight. This can be done in $O(n)$ time.

We can now substitute $(1/\epsilon^2)\log(1/\epsilon)$ for n in the time and space bounds for the large-item computation obtained in §5. This yields a time bound of $O(n \log(1/\epsilon) + (1/\epsilon^4)\log(1/\epsilon))$ and a space bound of $O(n + (1/\epsilon^4)\log(1/\epsilon))$ for the overall computation. (Note that $O(n)$ space is required to store the input data.)

8. An improved method of scaling. The method of scaling we have employed up to this point is unnecessarily conservative. The same fixed error, K, has been permitted each scaled profit coefficient q_j. It is more effective to produce coefficients for which the permissible error is somewhat in proportion to the size of the coefficient. This can be done as follows.

Let

$$T = \tfrac{1}{2}\epsilon P_0, \qquad K = \tfrac{1}{2}\epsilon T = \tfrac{1}{4}\epsilon^2 P_0, \tag{8.1}$$

as in §5. If p_j lies in the interval $(2^k T, 2^{k+1}T]$, $k = 0, 1, 2, \ldots, \lfloor \log_2(P^*/T) \rfloor$, then let

$$q_j = \lfloor P_j / 2^k K \rfloor 2^k. \tag{8.2}$$

Now notice that

$$\begin{aligned} Kq_j \leqslant p_j &< Kq_j + 2^k K \\ &\leqslant Kq_j + \tfrac{1}{2}\epsilon 2^k T \leqslant Kq_j + \tfrac{1}{2}\epsilon p_j. \end{aligned}$$

The above inequality is parallel to (3.1). It follows that we have, in place of inequality (5.4),

$$P^* = P_L + P_S < KQ_L + \tfrac{1}{2}\epsilon P_L + P_S. \tag{8.3}$$

By reasoning similar to that used before, we obtain

$$P^* - P < \tfrac{1}{2}\epsilon P^* + T = \tfrac{1}{2}\epsilon P^* + \tfrac{1}{2}\epsilon P_0. \tag{8.4}$$

Let $m = \lfloor \log_2(P^*/T) \rfloor$. The size of scaled profit space is bounded by

$$\left\lfloor \frac{P^*}{2^m k} \right\rfloor 2^m \leqslant \frac{P^*}{K} \leqslant \frac{8}{\epsilon^2},$$

as before. However, the number of distinct q_j values obtained from each p_j-interval $(2^k T, 2^{k+1}T]$ is at most $2/\epsilon - 1$, independent of k. Thus for the scaled profit-space interval $(2/\epsilon, 4/\epsilon]$, the number of q_j values is about $2/\epsilon$, and the average value of n_j is about $3/\epsilon$. For the interval $(4/\epsilon, 8/\epsilon]$, the number of q_j values is still about $2/\epsilon$, and the average value of n_j is about $3/2\epsilon$, and so on. It follows that the number of items which must be considered for the large-item computation is bounded by about

$$\frac{6}{\epsilon^2}\left(1 + \frac{1}{2} + \cdots + \frac{T}{P^*}\right) < \frac{12}{\epsilon^2}.$$

We have thus eliminated the log factor in (7.1).

Comment. We have dispensed with a more general argument, in which $T = (1 - \lambda)\epsilon P_0$, $K = \lambda \epsilon T$, as this carries through exactly as in §5. Nor shall we confirm that there is no advantage in choosing an integer $a > 2$, and letting

$$q_j = \left\lfloor \frac{P_j}{a^k K} \right\rfloor a^k,$$

if p_j lies in the interval $(a^k T, a^{k+1}T]$. Note that the previous scaling technique is the case in which $a > P^*/T$.

Since there are now $O(1/\epsilon^2)$ items for the large-item computation, the time bound can be reduced to $O(n \log(1/\epsilon) + 1/\epsilon^4)$ and the space bound to $O(n + 1/\epsilon^4)$. In the next section we shall show how the space bound can be further reduced to $O(n + 1/\epsilon^3)$, while maintaining the time bound.

9. Modification of the large-item computation. We propose to modify the large-item computation to reduce the space required for backtracing. Our plan is as follows.

First sort at most n_j items with a given q_j value into nondecreasing order of weight. This can be done, for all q_j values, in $O((1/\epsilon^2)\log(1/\epsilon))$ time. Now notice that there are at most $n_j + 1$ possibilities for each q_j value. Either no items are chosen, the first (smallest-weight) item is chosen, the first two items are chosen, . . . , or all (at most n_j) items are chosen. The large-item computation is now carried out in iterations over distinct q_j values (instead of individual items) in strictly decreasing order of q_j.

Initially the list contains only the pair (0, 0). At the end of iteration i, each pair (Q, A) indicates the smallest weight A attainable for a subset of items chosen from the i largest q_j values, with total scaled profit at least Q. It also indicates the largest total scaled profit Q attainable for a subset of items chosen from the i largest q_j values, with total weight not exceeding A.

Suppose an iteration is for scaled profit q_j and there are n_j items with this scaled profit. For each pair (Q, A) in the list, n_j candidate pairs are formed, corresponding to the choice of 1 item, 2 items, . . . , n_j items. These pairs are placed in n_j separate candidate lists. The $n_j + 1$ lists are then merged by means of n_j pairwise merges. Each list entering into a pairwise merge is $O(1/\epsilon^2)$ in length, and the resulting list is also $O(1/\epsilon^2)$ in length, dominated entries being eliminated in the course of the merge. It follows that the running time for each iteration is bounded by $O(n_j/\epsilon^2)$. The running time for the large-item computation is thus bounded by $O(1/\epsilon^4)$, since $\sum n_j$ is $O(1/\epsilon^2)$.

Now let us consider the space bound. At each iteration, the space required for candidate lists is at most $O(n_j N_j)$, where N_j is the length of the list of pairs produced by the previous iteration. This space is bounded by $O(1/\epsilon^3)$. The number of new

entries added to the list of pairs, and hence the number of nodes added to the tree used for backtracing, is $O(N_j)$. But N_j is at most 1 for the largest interval, 2 for the second-largest, . . . , $2/\epsilon^2$ for the second-smallest, and $4/\epsilon^2$ for the smallest. This is because the q_j values in each interval are a power of 2, which reduces the effective size of the scaled profit space from $P^*/K \leq 8/\epsilon^2$ to $2^{-k}P^*/K \leq 2^{-k}(8/\epsilon^2)$, for iterations over q_j values in interval k. The number of q_j values (iterations) for each interval is at most $T \leq 2/\epsilon$. It follows that the total number of nodes in the tree is bounded by a number proportional to

$$\frac{2}{\epsilon}\left[1 + 2 + \cdots + \frac{4}{\epsilon^2}\right] \approx \frac{16}{\epsilon^3}. \tag{9.1}$$

Hence total space is bounded by $O(n + 1/\epsilon^3)$.

Comment. Each of the n_j candidate pairs obtained from a pair (Q, A) can be viewed as corresponding to the choice of a "multiple" item. The indexing scheme and backtracing procedure must be slightly modified. It is not difficult to do this, while staying within the time and space bounds. Hereafter, we shall not mention necessary modification of the indexing scheme and backtracing procedure, assuming the reader can supply details.

10. Summary of algorithm. We now summarize the steps of the approximation algorithm for the 0-1 knapsack problem:

1. Compute p_j/a_j ratios for all items. Compute P_0, using a median-finding routine as indicated in §4: $O(n)$ time and space.
2. Determine the threshold T and scale factor K by (8.1). Compute q_j for all large items, using equation (8.2): $O(n)$ time and space.
3. Determine the $O(1/\epsilon^2)$ items to enter into the large-item computation, by applying a median-finding routine to find the at most n_j items with smallest weight, for each q_j value: $O(n)$ time and space.
4. Sort the at most n_j items for each q_j value in order of nonincreasing weight: $O((1/\epsilon^2)\log(1/\epsilon))$ time and $O(1/\epsilon^2)$ space.
5. Carry out the modified version of the large-item computation described in the previous section: $O(1/\epsilon^4)$ time and $O(1/\epsilon^3)$ space.
6. Compute $\phi(b - A)$ for each pair (Q, A) in the final list, employing a median-finding procedure, as described in §6: $O(n \log(1/\epsilon) + (1/\epsilon^2)\log(1/\epsilon))$ time and $O(n + 1/\epsilon^2)$ space.
7. Find a pair (Q, A) for which $KQ + \phi(b - A)$ is maximum: $O(1/\epsilon^2)$ time and space.
8. Find the set of large items in the approximate solution by backtracing: $O(1/\epsilon)$ time and $O(1/\epsilon^3)$ space. The set of small items in the approximate solution is readily available as a byproduct of Step 6.

In succeeding sections we shall indicate how the steps of the algorithm should be modified for other versions of the knapsack problem.

11. "Bootstrapping" the algorithm. A further analysis of the space bound shows that space is bounded by a constant times

$$P^*/\epsilon^3 P_0.$$

Similarly, time for the large-item computation is bounded by a constant times

$$(P^*)^2/\epsilon^4 P_0^2.$$

It follows that an improvement in the quality of the lower bound P_0 can yield a reduction in the bounds by at least a linear scale factor.

One way to obtain a better lower bound is to use the algorithm itself to produce

approximate solutions which provide better lower bounds. A "bootstrapping" procedure is as follows. Begin with the lower bound P_0 with accuracy $\epsilon_0 \leqslant \frac{1}{2}$. Use P_0 to obtain a threshold T_1 and scale factor K_1 for accuracy $\epsilon_1 < \epsilon_0$. Apply the approximation algorithm to obtain an approximate solution with profit P_1 and relative error ϵ_1. Proceed through successive iterations, with accuracies $\epsilon_0 > \epsilon_1 > \epsilon_2 > \cdots > \epsilon_N = \epsilon$.

At successive iterations, the lower bounds P_i, thresholds T_i and scale factors K_i are determined by the relations:

$$P_i \geqslant (1 - \epsilon_{i-1})P^*,$$

$$T_i = \tfrac{1}{2}\epsilon_i P_{i-1} \geqslant \tfrac{1}{2}\epsilon_i(1 - \epsilon_{i-1})P^*,$$

$$K_i = \tfrac{1}{4}\epsilon_i^2 T_i \geqslant \tfrac{1}{4}\epsilon_i^2(1 - \epsilon_{i-1})P^*.$$

The running time at iteration i is then bounded by a constant times

$$\frac{(P^*)^2}{\epsilon_i^4 P_{i-1}^2} \leqslant \frac{P^*}{\epsilon_i^4(1 - \epsilon_{i-1})^2}. \tag{10.1}$$

If we perform one iteration, as in the preceding, with $\epsilon_0 = \frac{1}{2}$, $\epsilon_1 = \epsilon$, then the quantity (10.1) becomes $4P^*/\epsilon^4$. If we perform N iterations, a lower bound on the quantity (10.1) is given by P^*/ϵ^4. It follows that no matter how such a bootstrapping scheme is arranged, we can expect to improve the time bound for the large-item computation by no more than a linear scale factor less than four.

A practical advantage of bootstrapping is that early iterations are likely, in practice, to be quite short, lengthening considerably with each successive iteration. This enables one to halt the procedure, when desired, with an approximate solution known to have accuracy ϵ_i, where i was the last iteration.

12. The unbounded problem. Recall the unbounded knapsack problem is the case in which the variables x_j are not restricted to 0, 1 values, but may be nonnegative integers. A number of simplifications can be made in this case.

First, it is evident that the computation of P_0 and of the ϕ-values is now much more straightforward. One need only identify a single item with maximum p_j/a_j ratio, which can be done with a single scan through the items. A small item with maximum p_j/a_j ratio is all that is needed to compute all the ϕ-values. This can be done in $O(n + 1/\epsilon^2)$ time.

It is evident we need retain only one large item for each q_j value for the large-item computation, i.e., one with minimum weight. However, we must provide for all possible n_j multiplicities of each such item, where

$$n_j = \left\lfloor \frac{P^*}{Kq_j} \right\rfloor \leqslant \left\lfloor \frac{4}{\epsilon} \right\rfloor.$$

Ibarra and Kim propose doing this by providing n_j identical copies of the item. A more sensible procedure is to provide only $\lfloor \log_2 n_j \rfloor$ additional copies by "doubling". That is, let the ith copy of item j be such that

$$p_j^{(i)} = 2^i p_j, \qquad a_j^{(i)} = 2^i a_j.$$

All necessary copies of items can be found in $O((1/\epsilon^2)\log(1/\epsilon))$ time. It is now necessary to retain only the smallest-weight items, or copy of an item, for each q_j value. This leaves $O((1/\epsilon)\log(1/\epsilon))$ items for the large-item computation. The large-item computation now proceeds by iteration over items or q_j values (there is no difference), from largest to smallest. It is evident that this can be carried out in $O(1/\epsilon^3)$ time. Finally, we note that the secondary data structures used for backtracing

in the 0-1 problem can be eliminated. List entries can be of the form (Q, A, j), where j is the index of the item identified with the iteration at which the entry is formed, i.e., candidate entries are of the form $(P + p_j, A + a_j, j)$. Solutions can be constructed by simple backtracing using the item indices. Hence we conclude that the unbounded knapsack problem can be solved in $O(n + 1/\epsilon^3)$ time and $O(n + 1/\epsilon^2)$ space.

13. "Subset-sum" problem. The "subset-sum" problem is as follows: Given n positive integers $p_1, p_2, \ldots, p_n$, and an integer b, does there exist a subset S, such that

$$\sum_{j \in S} p_j = b?$$

This can obviously be reduced to a 0-1 knapsack problem of the form

$$\begin{aligned} \text{maximize} \quad & \sum_j p_j x_j \\ \text{subject to} \quad & \sum_j p_j x_j \leqslant b, \\ & x_j \in \{0, 1\}. \end{aligned}$$

This the type of problem for which we propose to find an approximate solution with accuracy $\epsilon > 0$.

We first observe that it is a simple matter to compute P_0. The knapsack may be filled with items in arbitrary order. This clearly requires only $O(n)$ time.

We next observe that it is possible to carry out the large-item computation without consideration of item weights. That is, it is possible to process only lists of scaled profits Q, $Q \leqslant b/K$, instead of pairs (Q, A). However, we must insure that each entry in our lists is feasible. That is, if S is the set corresponding to Q, where $KQ \leqslant b$, then

$$\sum_{j \in S} p_j \leqslant b.$$

In order to guarantee this, we propose to round up in scaling, rather than rounding down, i.e., replace (8.2) by

$$q_j = \left\lceil \frac{p_j}{2^k K} \right\rceil 2^k.$$

When this is done, all of the preceding error analysis goes through as before.

Notice that no existing list entry need ever be eliminated by dominance. Hence the data structure used for backtracing requires only $O(1/\epsilon^2)$ space. (In fact, the secondary data structure can be dispensed with entirely, and the pointers replaced by pointers to other list entries.)

Let us now make a selection of items for the large-item computation. First place the large items in buckets, according to their scaled profits q_j and eliminating any surplus items (more than n_j) in any bucket. This leaves at most $O(1/\epsilon^2)$ items. The situation now differs from the ordinary 0-1 problem in that the items in each bucket are indistinguishable: they can all be considered to have the same weight. We now want to provide for the choice of any possible number of the items in any bucket. The procedure is as follows.

Start with the smallest and work upward. If bucket q_j contains an odd number of items, say $2k + 1$, place k "multiple" items, each with scaled profit $2q_j$, in bucket $2q_j$, discarding any extra items if the capacity of bucket $2q_j$ is exceeded. This leaves one item in bucket q_j. If bucket q_j is nonempty and contains $2k + 2$ items, do the same thing, leaving two items. This process requires $O(1/\epsilon^2)$ time for all buckets.

We are now left with at most two items with any given scaled profit q_j. The

large-item computation can be carried out in a straightforward fashion, in $O(1/\epsilon^3)$ time and $O(1/\epsilon^2)$ space. The small-item computation is $O(n)$. This yields overall time and space bounds of $O(n + 1/\epsilon^3)$ and $O(n + 1/\epsilon^2)$.

It is also possible to solve the subset-sum problem in $O(n + (1/\epsilon^2)\log(1/\epsilon))$ time and space by applying the computation proposed by Karp [7] to the $O((1/\epsilon)\log(1/\epsilon))$ large items. This computation involves the consideration of "intervals" of attainable P-values. We shall not give details of this computation, referring the reader to the reference.

14. Multiple-choice problems. Suppose the n items are partitioned into m equivalence classes and it is stipulated that no more than one item (or multiples of one item) may be chosen from each equivalence class. Such a problem is sometimes called a *multiple-choice* knapsack problem.

The author has developed an approximation algorithm for the unbounded multiple-choice problem with time and space bounds of $O(n + (1/\epsilon^6)\log(1/\epsilon))$ and $O(n + (1/\epsilon^4)\log(1/\epsilon))$. However, this algorithm is rather complicated and very likely can be improved upon. Hence we shall limit our discussion to the 0-1 multiple choice problem.

As in the case of the ordinary 0-1 problem, we wish to obtain a bound P_0, where $P^* \leqslant P_0 \leqslant 2P^*$. A method for doing this has been proposed by d'Atri [1] and is as follows.

We begin by relaxing the restrictions $x_j \in \{0, 1\}$, $j = 1, 2, \ldots, n$, to $0 \leqslant x_j \leqslant 1$. If S_i, $i = 1, 2, \ldots, m$, denotes the set of indices of items in the ith equivalence class, then we obtain a linear programming problem of the form

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_j x_j \leqslant b, \\ & \sum_{j \in S_i} x_j \leqslant 1, \qquad i = 1, 2, \ldots, m, \\ & 0 \leqslant x_j \leqslant 1, \qquad j = 1, 2, \ldots, n. \end{aligned}$$

For any feasible solution to this linear programming problem, the profit-weight contribution of the items in the ith equivalence class corresponds to a point in the convex hull of the set of points $\{(a_j, p_j) \mid j \in S_i\} \cup \{0, 0\}$. Moreover, by dominance relations, this point can be assumed to lie on the "upper boundary" of the convex hull, indicated by the darkened line in Figure 2.

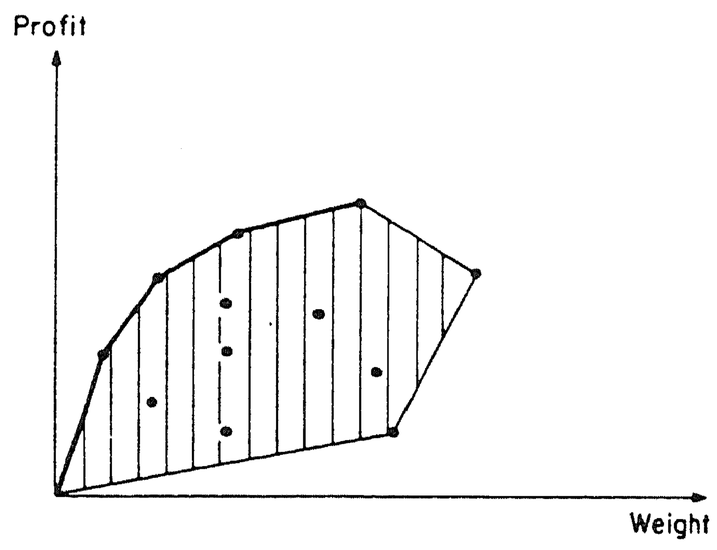


FIGURE 2. Convex hull of (a_j, p_j) pairs in a given equivalence class.

Let the n_i corner points of the upper boundary for equivalence class i be designated $(a_{j(1)}, p_{j(1)}), (a_{j(2)}, p_{j(2)}), \ldots, (a_{j(n_i)}, p_{j(n_i)})$. These points can be identified first by sorting the items into nondecreasing order of the ratios p_j/a_j and then selecting out the desired items by observing that

$$\frac{p_{j(k)} - p_{j(k-1)}}{a_{j(k)} - a_{j(k-1)}} > \frac{p_{j(k+1)} - p_{j(k)}}{a_{j(k)} - a_{j(k)}} > 0, \qquad k = 1, 2, \ldots, n_i - 1,$$

where $a_{j(0)} = p_{j(0)} = 0$. All corner points for all equivalence classes can be identified in $O(n \log n)$ time. We leave details to the reader.

Now, for $k = 1, 2, \ldots, n_i$, let

$$\bar{a}_{j(k)} = a_{j(k)} - a_{j(k-1)},$$

$$\bar{p}_{j(k)} = p_{j(k)} - p_{j(k-1)}.$$

With these new coefficients, we obtain a linear programming problem of the form

$$\begin{aligned} \text{maximize} \quad & \sum \bar{p}_j \bar{x}_j \\ \text{subject to} \quad & \sum \bar{a}_j \bar{x}_j \leq b, \\ & 0 \leq \bar{x}_j \leq 1. \end{aligned}$$

We assert that the optimal value of the objective function for this new problem is equal to that of the previous one.

Now suppose we solve this new linear programming problem, as in §4, obtaining an optimal solution with at most one fractional variable. It is not hard to see that this optimal solution has the property that, for each S_i, $\bar{x}_{j(k)} = 1$ implies $\bar{x}_{j(k-1)} = 1$ $(k > 1)$, and $\bar{x}_{j(k)} = 0$ implies $\bar{x}_{j(k+1)} = 0$ $(k < n_i)$. For each S_i let

$$p_i' = \begin{cases} p_{j(k)}, & \text{if } \bar{x}_{j(k)} = 1 \text{ and either } k = n_i \text{ or } \bar{x}_{j(k+1)} < 1, \\ 0, & \text{if } \bar{x}_{j(1)} < 1. \end{cases}$$

Then, by reasoning similar to that used in §4, the desired bound is

$$P_0 = \max\{p_1' + p_2' + \cdots + p_m', p_{\max}\}.$$

Because of the multiple-choice structure of the problem, there appears to be no useful way to separate items into "large" and "small" items, as in §5. However, we can proceed as in §4, taking

$$K = \epsilon P_0/m,$$

so that

$$P^*/K \leq 2m/\epsilon.$$

The list of pairs (Q, A) is processed with iteration over equivalence classes, rather than single items, making the computation rather similar to the large-item computation in §9.

Initially the list contains only the pair (0, 0). At the end of iteration i, each pair (Q, A) is identified with a feasible solution containing items chosen from equivalence classes 1 through i. Suppose there are n_i items in equivalence class i. To perform iteration i, form n_i candidate items for each pair (Q, A) existing in the list at the end of iteration $i - 1$. These candidate pairs are placed in n_i separate candidate lists. The $n_i + 1$ lists are then merged, eliminating dominated entries.

Iteration i requires $O(n_i m/\epsilon)$ time, $O(m/\epsilon)$ space, and at most $O(m/\epsilon)$ nodes are added to the tree used for backtracing. Taking account of the $O(n \log n)$ time

required to compute P_0, overall time and space requirements are bounded by $O(n \log n + mn/\epsilon)$ and $O(n + m^2/\epsilon)$, respectively. The number of items which need be considered from each equivalence class is bounded by the number of distinct q_j values, which is $O(m/\epsilon)$. Hence a time bound of $O(n \log n + m^2/\epsilon^2)$ can also be obtained.

These results represent a very significant improvement over those in [4].

15. Separable nonlinear functions. One considerable generalization of the knapsack problem is:

$$\text{maximize} \quad \sum_{j=1}^{m} p_j(x_j)$$

$$\text{subject to} \quad \sum_{j=1}^{m} a_j(x_j) \leqslant b,$$

$$x_j \text{ nonnegative integer},$$

$$x_j \leqslant n_j.$$

Here p_j and a_j are arbitrary real-valued functions, $j = 1, 2, \ldots, m$.

By evaluating each function at each feasible integer point, one obtains $n = \sum n_j$ items for a 0-1 multiple-choice knapsack problem, with m equivalence classes. This can be solved, for any prescribed relative error $\epsilon > 0$, in $O(n \log n + mn/\epsilon)$ time, using the procedure of the previous section.

Note that there is nothing in our theory which cannot accommodate real-valued profits and weights. (Of course, items with negative profits or weights may be neglected, or may cause a knapsack problem to become unbounded, under certain obvious conditions.)

16. Further extensions. The knapsack problem arises in many applications. However, it is a greatly specialized version of more general integer programming models for which there is a real need for approximation algorithms.

Certainly the techniques discussed in this paper fall far short of providing a fully polynomial approximation algorithm for multi-constraint problems. The principal reason is that they involve rescaling profit (objective) space instead of weight (constraint) space. Until a new technique is devised, there seems to be no reasonable way to apply the present approach to multi-constraint problems.

That is, unless we are willing to modify our views of optimization and approximation. For example, it is clearly possible to scale weight space to obtain an approximate knapsack solution of the following type. Given $\delta > 0$, find a subset of items S, such that

$$\sum_{j \in S} p_j > P^*$$

and

$$\sum_{j \in S} a_j < (1 + \delta)b.$$

At first glance, the above may seem like an unnatural form of approximation. Possibly this is because we have been taught that constraints are inviolate. But suppose the knapsack problem is being used as a simplified model for, say, project scheduling. A manager seeks to choose projects for a certain period, subject to certain resource constraints (knapsack capacity). The profits associated with the items are real and hard. The constraints are soft and flexible. He certainly wants to earn P^* dollars, if possible. Which type of approximation is more reasonable?

If the notion of constraint approximation is accepted, then it seems feasible to move ahead with the application of known techniques to multi-constraint problems.

17. Concluding remarks. It is certainly possible that the time and space bounds presented here can be improved upon. Aside from improvments in factors of $1/\epsilon$, or $\log(1/\epsilon)$, there are a number of open questions and directions for future research which suggest themselves.

We have made a few simple assumptions in making time and space bounds. Among these are that arithmetic operations can be performed in constant time on integer operands as large as P^* and b. In the case of the 0-1 knapsack problem, operations on integers as large as n are assumed, for the purpose of finding medians. Can this assumption be removed?

Perhaps a more interesting question is: Can a 0-1 approximation algorithm be found which is "strictly linear" in n, instead of order $n \log(1/\epsilon)$? More generally, is it possible to establish that the 0-1 problem is inherently more complex than the unbounded problem?

Acknowledgment. The author wishes to thank Alexander Rinnooy Kan and David Lichtenstein for reading preliminary drafts of this paper, and for their helpful comments and suggestions. He is also indebted to Gianfranco d'Atri for his communication of the method of bounding for the multiple-choice problem.

References

[1] d'Atri, G. (1977). The Generalized Knapsack Problem. Communication at the annual meeting of CNR-GNIM, Rimini, Italy.

[2] Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press.

[3] Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L. and Tarjan, R. E. (1973). Time Bounds for Selection. *J. Comput. System Sci.* 7 448–461.

[4] Chandra, A., Hirschberg, D. and Wong, C. (1976). Approximate Algorithms for Some Generalized Knapsack Problems. *Theor. Comput. Sci.* 3 293–304.

[5] Garey, M. R. and Johnson, D. S. (1976). Approximation Algorithms for Combinatorial Problems: an Annotated Bibliography. In *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, ed. 41–52. Academic Press, New York.

[6] Ibarra, O. H. and Kim, C. E. (1975). Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *J. Assoc. Comput. Mach.* 22 463–468.

[7] Karp, R. M. (1975). The Fast Approximate Solution of Hard Combinatorial Problems. *Proc. 6th Southeastern Conference on Combinatorics, Graph Theory, and Computing*. Utilitas Mathematica Publishing, Winnipeg, 15–31.

[8] Sahni, S. (1975). Approximate Algorithms for the 0/1 Knapsack Problem. *J. Assoc. Comput. Mach.* 22 115–124.

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, UNIVERSITY OF CALIFORNIA, BERKELEY, CALIFORNIA 94720

Efficient Implementation of Dynamic Programming Algorithms for Sequencing Problems

Eugene L. Lawler
Computer Science Division, University of California, Berkeley, CA 94720, USA

Dynamic programming has proved to be a fruitful approach for the solution of a variety of sequencing problems, including single-machine sequencing problems and assembly line balancing problems. However, certain technical difficulties have been experienced in fully exploiting the reduction in computational complexity which should result from the existence of precedence constraints in these problems. In particular, no completely satisfactory technique has existed for generating "feasible" sets of jobs, as determined by precedence constraints. In this paper we present a simple computer implementation of the dynamic programming algorithms which overcomes these difficulties. The implementation permits sequencing problems with precedence constraints to be solved in $O(Kn)$ time, where K is the number of feasible sets, and in $O(n+k_{\max})$ space, where k_m is the number of feasible sets of size m and $k_{\max} = \max_m k_m$. It is also shown how similar techniques can be applied to improve the efficiency of dynamic programming computations for the traveling salesman problem, when the network is sparse or when there are precedence constraints.

Report BW 106/79, Afdeling Mathematische Besliskunde, Mathematisch Centrum, Amsterdam (1979). Reprinted with permission.

1. Introduction. Dynamic programming has proved to be a fruitful approach for the solution of a variety of sequencing problems [4]. However, certain technical difficulties have been encountered by Baker and Schrage [2], Held, Karp and Shareshian [6], and others, in implementing the dynamic programming algorithms when the sequencing problems are to be solved subject to precedence constraints. In this paper we present a simple and efficient computer implementation which overcomes these technical difficulties.

The general type of sequencing problem we are concerned with is to find a permutation π of the n elements of a set N such that the value of a specified objective function $f(\pi)$ is minimized. When *precedence constraints* are specified by an acyclic digraph $G = (N, A)$, the minimization is to be carried out over *admissible* permutations, where a permutation π is admissible only if $(i, j) \in A$ implies that i precedes j in π.

It is well known that a number of important special cases of this general problem can be solved by dynamic programming. We are concerned with those cases in which dynamic programming calls for the computation of $F(N)$ by recursion over subsets $S \subseteq N$, subject to equations of the general form

$$F(S) = \min_{j \in S}\{g(F(S-j), S, j)\}, \qquad (1.1)$$
$$F(\emptyset) = 0,$$

where g is a readily computed function derived from the objective function f. (Here, and below, we let $S + j$ denote $S \cup \{j\}$ and $S - j$ denote $S - \{j\}$.)

For example, suppose n jobs are to be sequenced for processing by a single machine. For each job j, $j = 1, 2, \ldots, n$, there is a specified processing time $p_j \geq 0$ and a penalty function f_j. A given permutation π induces a completion

time C_j^π for each job j. The objective is to find an admissible permutation π which minimizes

$$f(\pi) = \sum_{j=1}^{n} f_j(C_j^\pi).$$

For this problem,

$$g(F(S-j), S, j) = F(S-j) + f_j(\sum_{k \in S} p_k),$$

and $F(S)$ is the cost of an optimal sequence for the jobs in S, where the first job starts at time $t = 0$ [5, 7].

A similar treatment can be given to problems in which

$$f(\pi) = \max_j f_j(C_j^\pi),$$

by letting

$$g(F(S-j), S, j) = \max\{F(S-j),\ f_j(\sum_{k \in S} p_k)\}.$$

This approach is of no value, if the functions f_j are monotone nondecreasing, since such problems can be solved much more efficiently by other means [8]. However, dynamic programming can be worthwile for related minmax problems. For example, whereas the "cumulative cost" or "initial resource requirement" problems can be efficiently solved for series parallel precedence constraints [1, 11, 12], these same problems are NP-hard for arbitrary precedence constraints, and dynamic programming may then be worthwhile.

Another problem admitting of solution by recursion equations of the general form (1.1) is the linear arrangement or one-dimensional module placement problem [9]. For this problem,

$$g(F(S-j), S, j) = F(S-j) + c(S, N-S),$$

where $c(S, N-S)$ is the capacity of cutset $(S, N-S)$ in a specified graph. The bin packing, one dimensional stock cutting and assembly line balancing problems also yield to this approach [5, 6]. For these problems,

$$g(F(S-j), S, j) = F(S-j) + \Delta(F(S-j), p_j),$$

where p_j is a specified parameter and Δ is a certain readily computed function.

Let us consider the running time required to solve equations (1.1) for $F(N)$, subject to the assumption that g can be computed in constant time. In the absence of precedence constraints, there is an equation for each subset $S \subseteq N$ and $F(N)$ can be computed in time proportional to

$$\sum_{k=1}^{n} k \binom{n}{k} = n2^{n-1},$$

or $O(n2^n)$ time. Although this time bound is exponential, it does represent very much less running time than would be required for an exhaustive examination of $n!$ permutations.

When precedence constraints are added, the computational complexity is in principle reduced. Let us call a set $S \subseteq N$ *feasible* if $j \in S$ implies that all predecessors of j in the digraph $G = (N, A)$ are contained in S. It is well known that equations (1.1) need be solved only for feasible sets S, with the minimization in each equation only over $j \in S$ such that $S - j$ is also feasible. It follows that the time required to compute $F(N)$ should be bounded by $O(Kn)$, where K is the number of feasible sets. Moreover, it has been shown that in practice K is often very much smaller than 2^n [2].

However, there are a number of questions of implementation that need to be resolved, in order to take complete advantage of the reduction in complexity that should result from precedence constraints. In particular, one must have an efficient procedure for generating the K feasible sets and, for a given feasible set S, identifying and locating in memory the feasible sets $S - j$.

Various schemes have been proposed for generating and addressing feasible sets. Baker and Schrage [2], for example, proposed an approach whereby each element j is assigned an integer label a_j. Each feasible set S is then assigned an index equal to the sum of the labels of the elements contained within it. Ideally, these indices should provide a one-one mapping of the feasible sets onto the integers $0, 1, \ldots, K-1$, so that no space is wasted when these indices are used to determine memory addresses. It remains an open question whether a labeling with this property exists for all precedence digraphs, and it appears that the labeling procedure proposed in [2] does waste an indefinite amount of space for some digraphs. In the opinion of the present author, the labeling approach is inherently more cumbersome and time consuming than is necessary to effect an efficient implementation of the dynamic programming algorithms.

In this paper we present a simple and efficient computer implementation whereby the dynamic programming computations can be carried out in $O(Kn)$ time, and more significantly, in only $O(n+k_{\max})$ space, where k_m is the number of feasible sets of size m and $k_{\max} = \max_m k_m$. Not only does this implementation overcome the difficulties encountered in [2, 6], but it provides an answer to the observation in [2] that "There are a number of algorithms in the literature for generating subsets ... However, we are unaware of any efficient algorithms for generating subsets subject to precedence restrictions."

In the final section of this paper we indicate how the implementation can be extended to solve the traveling salesman problem. The approach presented may be especially useful in cases where the network over which the problem to be solved is sparse, or where precedence constraints exist.

2. Generating feasible sets. We assume that we are dealing with a computer with binary word length at least n. This is actually not a very restrictive assumption, and it is one which appears to have been made, at least implicitly, by previous authors. Note that at least $\log_2 K$ bits are required to address K feasible sets. Thus, if $K = \frac{1}{1024}2^n$, a word length of $n-10$ is required to store a single address. From a strictly theoretical point of view, our assumption enables us to obtain time and space bounds that are smaller by a factor of n than those which would otherwise be obtained.

We shall represent the precedence digraph $G = (N, A)$ by its adjacency matrix and store each column of the matrix in a separate word. Thus, column 4 of the adjacency matrix of the digraph pictured in Figure 1 is stored as

$$\begin{matrix}(1 & 1 & 0 & 0 & 0 & 0).\\ 1 & 2 & 3 & 4 & 5 & 6\end{matrix} \tag{2.1}$$

We shall represent a set S by its incidence vector on N and store this vector in a single word. The incidence vector for S has numerical value

$$n(S) = \sum_{j \in S} 2^{n-j}.$$

Thus the set $S = \{1, 2, 3\} \subseteq \{1, 2, \ldots, 6\}$ is represented by the vector

$$\begin{matrix}(1 & 1 & 1 & 0 & 0 & 0),\\ 1 & 2 & 3 & 4 & 5 & 6\end{matrix} \tag{2.2}$$

and this vector has value 56.

We assert that each feasible set of size $m + 1$ is of the form $S + j$, where S is a feasible set of size $m, j \notin S$, and there is no arc (i, j) in G such that $i \notin S$. Given a set S of size m, it is easy to verify whether $S + j$ satisfies these conditions. First check that element j of the incidence vector for S is zero. Then compare the incidence vector for S with column j of the adjacency matrix for G, to see if the incidence vector for S has a one in each position that column j has a one. Thus, for example, comparison of the vectors (2.1) and (2.2) shows that $\{1, 2, 3, 4\}$ is a feasible set for the digraph in Figure 1.

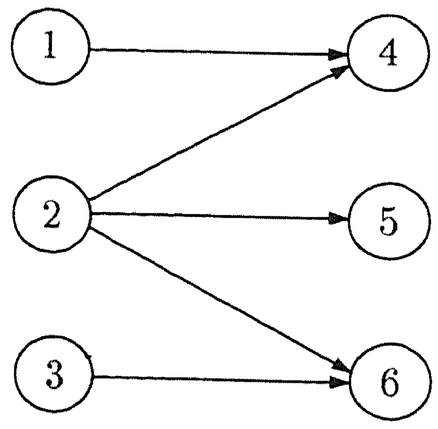


Figure 1: Digraph for example.

Exactly how these tests for $S+j$ should be made depends upon the machine language instructions available in the computer we are dealing with. In any case, these tests can be assumed to be carried out, and the incidence vector for $S+j$ formed, in constant time.

We shall generate feasible sets in order of size, beginning with $\emptyset$, then generating feasible sets of size one, size two, ..., and ending with the set N. Suppose that there are k_m feasible sets of size m, and that these are $S_i, i = 1, 2, \ldots, k_m$, where $n(S_i) < n(S_{i+1}),\ i = 1, 2, \ldots, k_m - 1$. The k_{m+1} feasible sets of size $m + 1$ are generated as follows.

First make one list of all feasible sets of the form $S_i + 1$ and a second list

of all feasible sets of the form $S_i + 2$, with both lists in increasing order of the index i, and therefore in increasing numerical order for the sets contained in them. Then merge the two lists, *eliminating duplicate entries in the course of the merge*. Each of the two lists merged contains no more than k_{m+1} entries, and the list resulting from the merge also contains no more than k_{m+1} entries. Next generate a list of feasible sets of the form $S_i + 3$ and merge it (again eliminating duplicate entries) with the list previously obtained. Continue in this way until the list $S_i + n$ has been merged. The final list contains the k_{m+1} feasible sets of size $m + 1$.

As an example, consider the digraph depicted in Figure 1. There are five feasible sets of size three, and from these we obtain the list indicated below (dispensing with brackets and commas in the representation of sets):

S_i	S_i+1	S_i+2	S_i+3	S_i+4	S_i+5	S_i+6
123				1234	1235	1236
124			1234		1245	
125			1235	1245		
235	1235					2356
236	1236				2356	

Let us ignore the list for $S_i + 2$ since it is empty. Merging lists $S_i + 1$ and $S_i + 3$, we obtain a list containing 1234, 1235, 1236, a duplicate entry for 1235 being discarded. Merging this list with $S_i + 4$, we obtain 1234, 1235, 1236, 1245, eliminating a duplicate entry for 1234. Merging this list with $S_i + 5$, we obtain 1234, 1235, 1236, 1245, 2356, eliminating duplicate entries for 1235 and 1245. Merging with $S_i + 6$, we obtain 1234, 1235, 1236, 1245, 2356, eliminating duplicate entries for 1236, 2356. The final list contains all five feasible sets of size four.

Each list containing feasible sets of the form $S_i + j$ can be obtained in $O(k_m)$ time and space, exclusive of space required to store the adjacency matrix of G. Hence the generation of all n lists can be carried out in $O(nk_m)$ time and $O(k_m)$ space. Each list entering a merge contains no more than k_{m+1} entries. Hence each merge can be carried out with no more than $2k_{m+1} - 1$ numerical comparisons and in $O(k_{m+1})$ time and space. Hence all $n - 1$ merges can be carried out in $O(nk_{m+1})$ time and $O(k_{m+1})$ space. Taking account of time and space requirements for both list generation and merging, the list of all feasible sets of size $m + 1$ can be obtained in $O(n\max\{k_m, k_{m+1}\})$ time and $O(\max\{k_m, k_{m+1}\})$ space.

It follows from the above that the time required to generate all feasible sets is bounded by $O(Kn)$. If feasible sets of size m are outputted as soon as those of size $m+1$ have been obtained, the space required to generate all feasible sets is bounded by $O(n+k_{\max})$, where $O(n)$ space is required to store the adjacency matrix of G. Otherwise space is bounded by $O(K)$.

The key idea which has enabled us to attain these time and space bounds is the generation of lists of the form $S_i + j$ and then merging them, with elimination of duplicate entries. This is precisely the same technique employed by the

present author to attain small time and space bounds for fast approximation algorithms for knapsack problems [10].

3. Solving the recurrence equations. It is a simple matter to compute the values $F(S)$ in the course of generating the feasible sets, as follows.

Suppose with each set S_i, $i = 1, 2, \ldots, k_m$, there is recorded the value $F(S_i)$. When forming the list of feasible sets of the form $S_i + j$, make a single entry in the list for each feasible set $S_i + j$ consisting of its incidence vector (in one word) and the computed value $g(F(S_i), S_i + j, j)$ (in a second word). In the course of merging lists, whenever a given set is found to be duplicated, retain the list entry with the smaller g-value. When all lists have been merged, the g-value recorded for each $(m+1)$-element set S in the final list is the value of $F(S)$, as determined by (1.1).

If we seek only to compute the value $F(N)$, then it is possible to discard all feasible sets of size m (and their computed F-values) as soon as those of size $m+1$ have been obtained. The entire computation can thus be carried out in $O(Kn)$ time and $O(n + k_{\max})$ space. However, if we wish to construct an optimal sequence π for which $f(\pi) = F(N)$, then it is necessary to elaborate the procedure a bit, especially if we are to attain the same time and space bounds.

4. Constructing an optimal sequence. The most straightforward way to construct an optimal sequence is simply to record with each feasible set S a sequence $\pi(S)$ yielding the value $F(S)$.

Suppose with each set S_i, $i = 1, 2, \ldots, k_m$, there is recorded a sequence $\pi(S_i)$ yielding the value $F(S_i)$. When forming the list of feasible sets of the form $S_i + j$, the sequence $\pi(S_i), j$ (sequence $\pi(S_i)$, followed by j) is made part of the list entry for $S_i + j$ (in addition to the incidence vector for $S_i + j$ and the value $g(F(S_i), S_i + j, j)$). When all lists have been merged, as described in the previous section, the desired sequence $\pi(S)$ is recorded in the entry for each $(m+1)$-element set S in the final list.

The difficulty with this straightforward approach is the amount of space required to store each $\pi(S)$. Since $\lceil \log_2 n \rceil$ words of length n are required to store a permutation of n elements, the space bound is increased to $O(n + k_{\max} \log n)$. Moreover, the time bound is increased to $O(Kn \log n)$, because of the time required to record sequences in list entries and the extra time required to merge lists with the larger entries.

The time bound is easily restored to $O(Kn)$ by using pointers. Instead of recording the sequence $\pi(S_i), j$ in the entry for $S_i + j$, record the index j and a pointer to the sequence $\pi(S_i)$. Then, when the list of $(m+1)$-element feasible sets has been obtained, use the index and the pointer recorded in the entry for each set S to obtain the sequence $\pi(S)$.

It is possible to obtain a space bound of $O(n + k_{\max})$, while maintaining the time bound at $O(Kn)$. However, we admit that the "divide and conquer" technique we shall present may be of more theoretical than practical significance.

Virtually all of the problems for which recursion equations of the form (1.1)

have been formulated can be solved equally well by constructing an optimal sequence in one direction or the other, i.e. first-to-last or last-to-first. For example, in the case of the single-machine problem discussed in Section 1, let

$$\overline{F}(S) = \max_{j \in S} \{\overline{F}(S-j) + f_j(P - \sum_{k \in S-j} p_k)\}, \tag{4.1}$$

with $\overline{F}(\emptyset) = 0$, where $P = \sum_{j=1}^{n} p_j$. If equations (4.1) are solved for sets $S, S-j$ which are feasible with respect to the precedence digraph $\overline{G}(N, \overline{A})$ obtained by reversing the directions of all arcs in $G = (N, A)$, then $\overline{F}(S)$ is the cost of an optimal sequence for the jobs in S, with the completion of the last job at time P, and the other jobs immediately preceding.

Let $\mathcal{S}_m$ ($\overline{\mathcal{S}}_m$) denote the family of sets of size m which are feasible with request to G ($\overline{G}$). Note that $S \in \mathcal{S}_m$ if and only if $N - S \in \overline{\mathcal{S}}_{n-m}$, hence $k_m = |\mathcal{S}_m| = |\mathcal{S}_{n-m}|$. It should be clear that, for any m,

$$F(N) = \min\{F(S) + \overline{F}(N-S) | S \in \mathcal{S}_m\}. \tag{4.2}$$

Assume, for simplicity, that n is a power of two. Taking $m = n/2$, it is a simple matter to compute $F(N)$ by (4.2). Generate a list for $\mathcal{S}_{n/2}$, in increasing numerical order of the sets contained within it, and a list for $\overline{\mathcal{S}}_{n/2}$, in decreasing numerical order. Form the sums indicated in (4.2) and choose the smallest. This can be done in $O(Kn)$ time and $O(n + k_{\max})$ space.

Let S^* be a set in $\mathcal{S}_{n/2}$ yielding a minimum value in (4.2). There is an optimal sequence in which the jobs in S^* precede the jobs in $N - S^*$. We now, in effect, have two problems, each with $n/2$ jobs. We can now apply equation (4.2) with respect to each of these subproblems to obtain four problems each with $n/4$ jobs, and so forth.

The time required to solve the first subproblem of size $n/2$ is proportional to $(n/2)K_1$ and the time required to solve the second subproblem is proportional to $(n/2)K_2$ where $K_1 + K_2 \leq K + 1$. ($K_1 + K_2$ can be expected to be considerably less than K in practice.) It follows that the time required to construct an optimal sequence by this method is bounded by a function of order

$$Kn + (K+1)\tfrac{n}{2} + (K+3)\tfrac{n}{4} + \cdots + (K - n + 1) \ ,$$

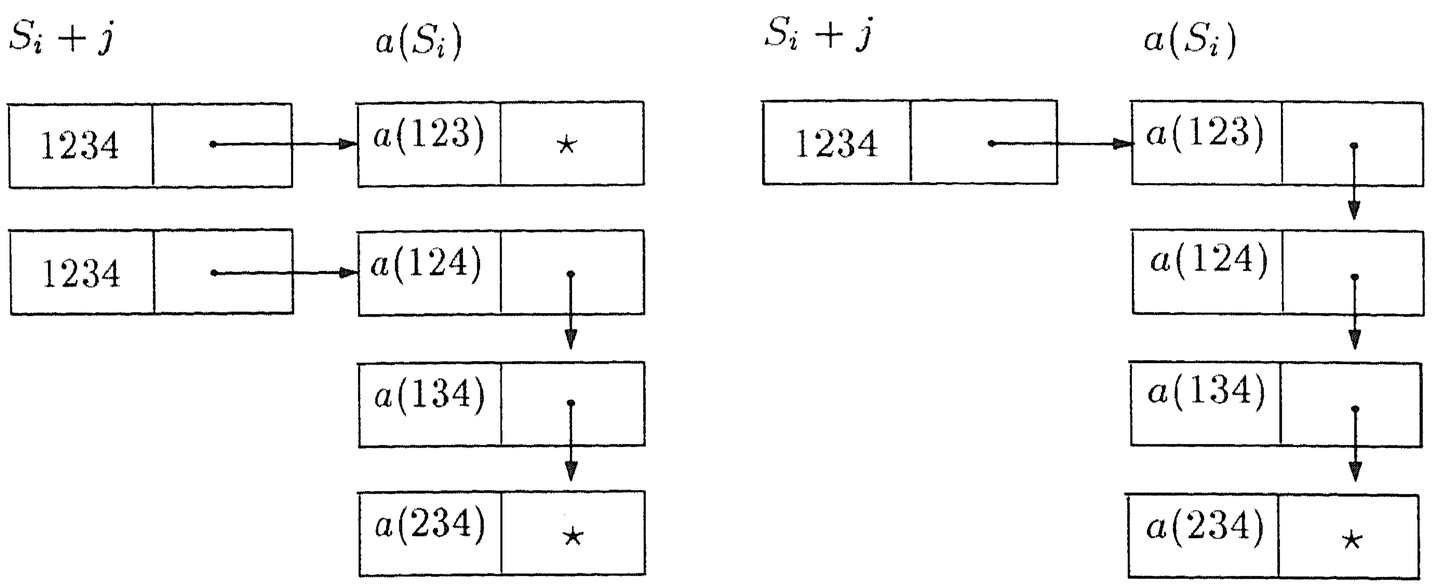
which is, of course, $O(Kn)$.

It is not hard to verify that, if this approach is properly implemented, space requirements are bounded by $O(n + k_{\max})$.

5. Providing for repetitive computations. There may be situations in which it is desired to carry out computations for a large number of different objective functions, but with respect to the same precedence digraph in each case. It may then be desirable to generate the feasible sets once only, and to record the feasible sets in memory in such a way that repetitive computations can be carried out as quickly as possible.

As soon as feasible sets of size $m + 1$ have been generated, we propose to assign them to consecutive memory addresses, immediately following those for sets of size m. With each feasible set S we shall store a pointer to a list of the

addresses $a(S-j)$ of feasible sets $S-j$. This data structure is indicated in Figure 2.



(a) Data structure before elimination of duplicate list entry.

(b) Data structure after elimination of duplicate list entry.

Figure 2

This data structure is generated by a simple modification of the procedure presented in Section 2. When the list of feasible sets of the form $S_i + j$ is generated, each entry in the list is provided with a pointer to a record containing the address $a(S_i)$ and a special symbol $*$, as shown in Figure 2(a). When duplicate entries are found in the course of merging list $S_i + j$, the entry in the list $S_i + j$ is retained, but the pointer in the other entry is substituted for "$*$" in the record specified by the pointer in the entry for $S_i + j$. This procedure is indicated in Figure 2.

It is thus possible, within $O(Kn)$ time, to generate all feasible sets, assign them to consecutive memory locations, and to provide, for each set S, a list of addresses of sets $S-j$. The total space required is proportional to the total number of pairs $(S, S-j)$. Space may, of course, be compacted somewhat by reforming the lists of addresses $a(S-j)$ and eliminating pointers.

6. Solving the traveling salesman problem. It is well known that the dynamic programming approach can be applied to the traveling salesman problem [3, 5]. Let H be an $(n+1)$-node network on node set $N \cup \{0\}$ over which the problem is to be solved. For $S \subseteq N$, let $F(S, j)$ denote the length of a shortest path from node 0 to node j which passes through each of the nodes in S. Then the problem is solved by computing $F(N, 0)$ by the recursion equations

$$F(S, j) = \min_{i \in S}\{F(S-i, i) + d_{ij}\}, \qquad (6.1)$$
$$F(\emptyset, j) = d_{0j},$$

where d_{ij} is the length of arc (i, j) in H.

If H is complete, the time required to solve equations (6.1) for $F(N, 0)$ is

$O(n^2 2^n)$. However, if H is sparse or if there are precedence constraints specified by an acyclic digraph $G = (N, A)$, the complexity of the computation may be considerably reduced.

Let us call a path *admissible* if it passes through nodes in an order consistent with the precedence constraints specified by G, and let us call a pair (S, j) *feasible* if there exists an admissible path in H from node 0 to node j which passes through the nodes in S.

A feasible pair (S, j) is said to have size m if $|S| = m$. Feasible pairs can be generated in order of size by a simple adaptation of the procedure given in Section 2. Each feasible pair of size $m+1$ is of the form $(S+i, j)$, where (S, i) is a feasible pair of size m, $j \notin S+i$, there is an arc (i, j) in H, and there is no arc (k, j) in G such that $k \notin S+i$. If there are K feasible pairs, they can be generated in $O(Kn)$ time and $O(n + k_{\max})$ space, where k_m is the number of feasible pairs of size m and $k_{\max} = \max_m k_m$.

The computation of the values $F(S, j)$ can be carried out in the course of state generation. Moreover, the divide-and-conquer technique of Section 4 can be applied. Let

$$\begin{aligned} &\overline{F}(S, j) = \min\{\overline{F}(S - i, i) + d_{ji}\}, \\ &\overline{F}(\emptyset, j) = d_{j0}. \end{aligned} \tag{6.2}$$

Let $\mathcal{S}_m$ ($\overline{\mathcal{S}}_m$) denote the family of feasible pairs with respect to H and G ($\overline{H}$ and $\overline{G}$). It is not necessarily true that $(S, j) \in \mathcal{S}_m$ if and only if $(N-(S+j), j) \in \overline{\mathcal{S}}_{n-m-1}$. Nevertheless, we have by analogy with (4.2), for any m,

$$\begin{aligned} F(N, 0) = \min\{&F(S, j) + \overline{F}(N - (S + j), j) | \\ &(S, j) \in \mathcal{S}_m, (N - (S + j), j) \in \overline{\mathcal{S}}_{n-m-1}\}. \end{aligned} \tag{6.3}$$

Using the approach of Section 4, equations (6.3) can be used to construct an optimal sequence in $O(Kn)$ time and $O(n+k_{\max})$ space. (In fact, there may be considerable advantage to this approach, resulting from the fact that not both (S, j) and $(N - (S + j), j)$ may be feasible.) In the (conventional) case that H is complete and G is empty, note that

$$k_{\max} = k_{n/2} = \tfrac{n}{2}\binom{n}{n/2} \approx \tfrac{\sqrt{n}}{2} 2^n,$$

and an optimal sequence can be constructed in $O(n^{1/2} 2^n)$ space.

Acknowledgements. The author wishes to thank Kenneth Baker for suggesting this research with his presentation at the POPCORN Festival (Prominent Open Problems in Combinatorial Optimization (Relevant or Not)), held at the Mathematisch Centrum, Amsterdam, April 6, 1979, and Jan Karel Lenstra and Alexander Rinnooy Kan for organizing the festival. This research was supported in part by NSF Grant MCS76-17605 and by NATO Special Research Grant 9.2.02 (SRG.7).

References

1. H.M. Abdel-Wahab, T. Kameda. Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Oper. Res. 26* (1978), 141–158.

2. K.R. Baker, L.E. Schrage. Finding an optimal sequence by dynamic programming: an extension to precedence-related tasks. *Oper. Res. 26* (1978), 111–120.

3. R.E. Bellman. Dynamic programming treatment of the traveling salesman problem. *J. Assoc. Comput. Mach. 9* (1962), 61–63.

4. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math. 5* (1979), 287–326.

5. M. Held, R.M. Karp. A dynamic programming approach to sequencing problems. *J. SIAM 10* (1962), 196–210.

6. M. Held, R.M. Karp, R. Shareshian. Assembly-line balancing – dynamic programming with precedence constraints. *Oper. Res. 11* (1963), 442–459.

7. E.L. Lawler. On scheduling problems with deferral costs. *Management Sci. 11* (1964), 280–288.

8. E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci. 19* (1973), 544–546.

9. E.L. Lawler. The quadratic assignment problem: a brief review. In: B. Roy (ed.), *Combinatorial Programming: Methods and Applications*, Reidel, Dordrecht (1975), 351–360.

10. E.L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res. 4* (1979), 339–356.

11. E.L. Lawler. Sequencing problems with series parallel precedence constraints. *Proc. Summer School in Combinatorial Optimization*, Urbino, Italy, 1978, to.appear.

12. C.L. Monma, J.B. Sidney. Sequencing with series-parallel precedence constraints. *Math. Oper. Res. 4* (1979), 215–224.

The Sciences

The Great Mathematical Sputnik of 1979

by Eugene L. Lawler

This article is reprinted by permission from *The Sciences*
and is from the September/October 1980 issue.
Individual subscriptions are $25.00 per year. Write to:
The Sciences, 2 East 63rd Street, New York, NY 10021
or call 1-800-THE NYAS.

The Great Mathematical

A Russian discovery launched by the Western press

Under the front-page headline, "A Soviet Discovery Rocks World of Mathematics," *The New York Times* of November 7, 1979 announced an event which its readers could easily believe had the importance of the launching of Sputnik. "A surprise discovery by an obscure Soviet mathematician," said the *Times,* "has rocked the world of mathematics and computer analysis . . . Apart from its profound theoretical interest, the new discovery may be applicable in weather prediction, complicated industrial processes, petroleum refining, the scheduling of workers at large factories. . ." Furthermore, confided the *Times,* "the theory of [secret] codes could eventually be affected by the Russian discovery, and this fact has obvious importance to intelligence agencies everywhere." One could almost hear alarm bells ringing in the offices of the CIA and the National Security Agency.

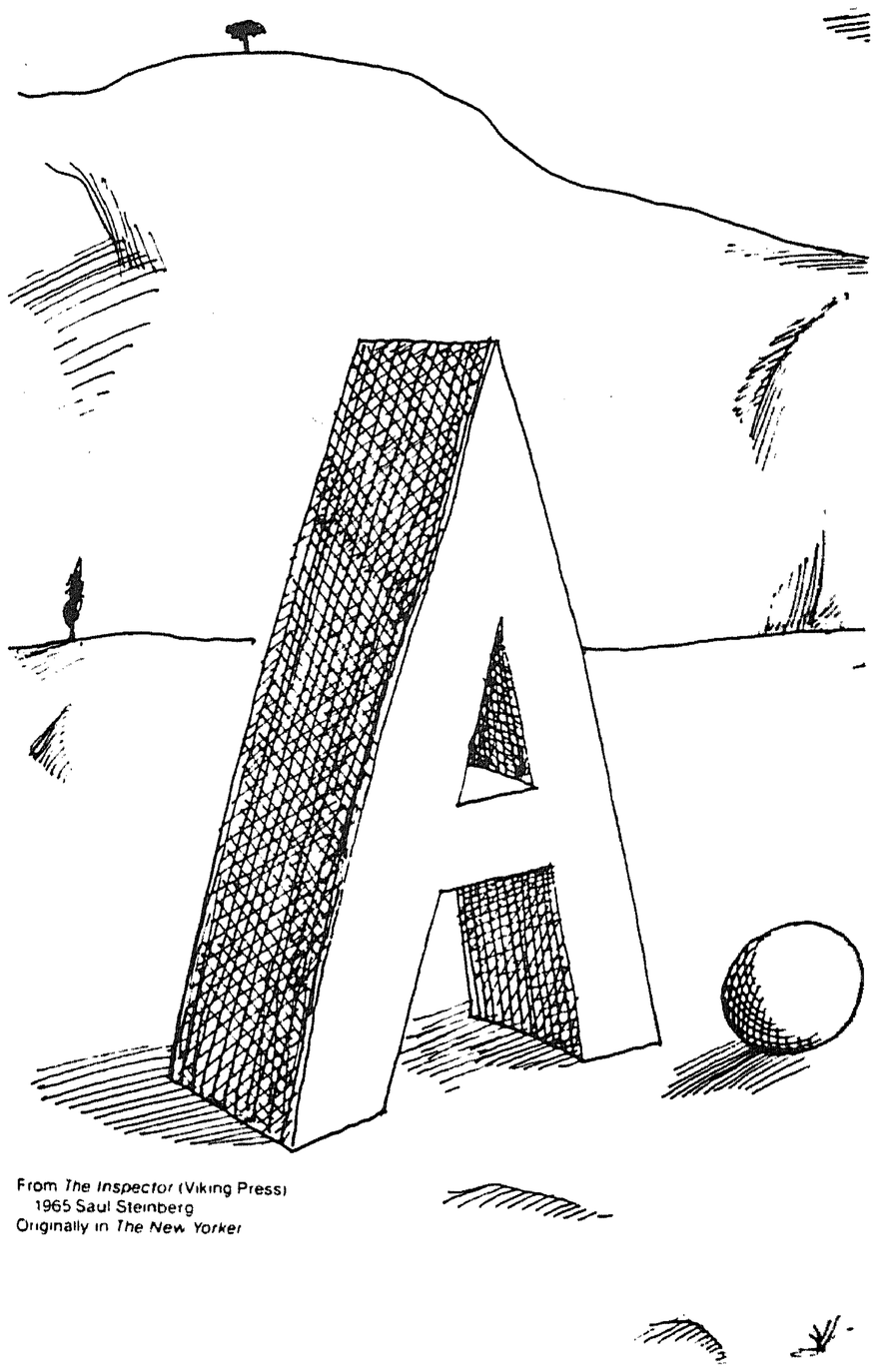
In England, the *Guardian* broke a somewhat different version of the story, under the headline, "Soviet Answer to 'Traveling Salesmen.' " Choosing to emphasize the human side of the story, the *Guardian* said: "A young Soviet mathematician, apparently totally unknown to any of the world's senior practitioners, has found an answer to one of the most baffling problems in computer calculation. But his obscurity is such that his discovery went unnoticed for 10 months in the mathematical world, although work on the problem has been going on for years. The apparent breakthrough was achieved by L.G. Khachian, and was published in a Soviet journal, *Doklady,* last January. Few people in the West read the journal and it was only after rumors of the discovery circulated at a conference in Germany that anyone in the mathematical world at large had even a hint that someone had come up with an answer to what is known in the trade as the 'Traveling Salesman' problem."

So it was that readers of the *Times* and the *Guardian,* and a multitude of other daily newspapers, were told of the Russian mathematical sputnik. But what was this new sputnik? The 1957 sputnik was simply thrown into space, a fact which anyone could understand. This 1979 sputnik was—what? A mathematical formula? If a mathematician were to throw it onto a blackboard, what sense could any ordinary person make of it? The *Times* story was so garbled and bereft of technical information, and the *Guardian* was so blatantly wrong, that even professional mathematicians were perplexed and misled.

Eugene L. Lawler is a professor of computer science at the University of California in Berkeley. He was credited by the press as the "discoverer" of the Russian discovery.

The Talk of the Town

The public unveiling of Khachian's results actually occurred with the October 4 cover story of *Science News:* "Linear Programming: Solid New Algorithm." This piece began with a flowery prologue describing abstract mathematics as a "sort of dream." But despite some inaccurate and misleading illustrations, the account was generally correct, and did not seriously misstate the significance of the achievement.

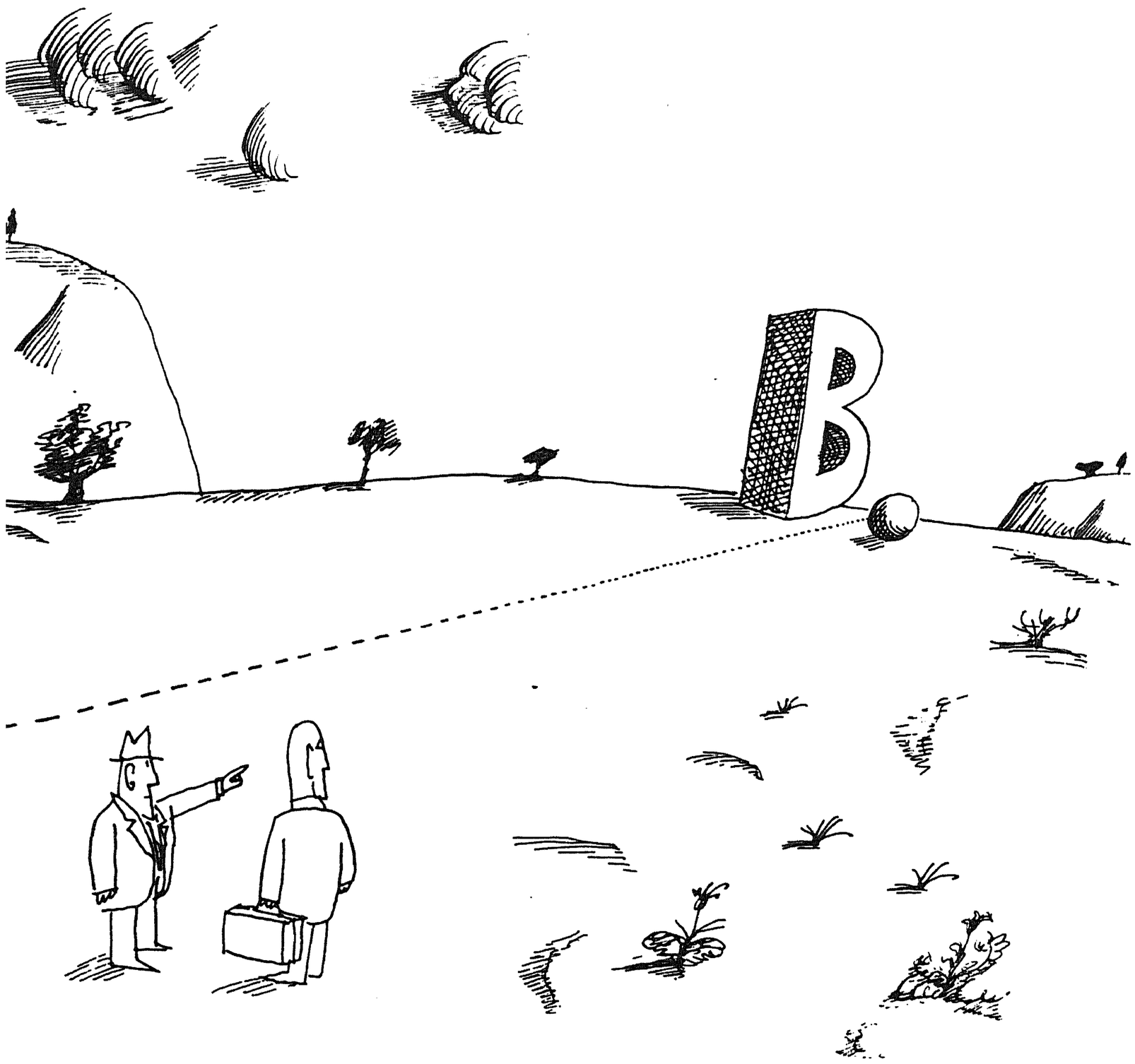


From *The Inspector* (Viking Press)
1965 Saul Steinberg
Originally in *The New Yorker*

Sputnik of 1979

by Eugene L. Lawler

Shortly after the *Science News* story appeared, the journal *Science* published an article about the algorithm by staff writer Gina Bari Kolata. Kolata researched her article well, talking to a number of mathematicians for background information, including Ronald Graham at Bell Laboratories, Laslo Lovasz of the University of Szeged in Hungary, and myself. The article was generally accurate and well balanced. One can only speculate as to how the article came to be seriously misinterpreted. However, it did contain one somewhat mushy paragraph with the words "Khachian's result...is tied to what is said to be the major unsolved problem in computer science...the traveling salesman problem." (Here my ellipses are purposely chosen to radically change the meaning of the passage, in order to show how it could be misinterpreted.) Most significantly, the article appeared under the title, "Mathematicians Amazed by Russian's Discovery."



Newspaper science writers are more assiduous readers of *Science* than mathematicians are of *Doklady*. And when an authoritative publication like *Science* says that mathematicians are "amazed" by a Russian discovery, any reporter can smell a story. What happened next was almost inevitable. The *Guardian* story, communicated by its Washington correspondent, appears to have been based on nothing more than a very careless misreading of the *Science* story. The *London Telegraph* rewrite offered the observation that there are few traveling salesmen in communist countries (a fact which the *Chicago Tribune* found worthy of editorial comment).

The *Times* story appears to have been based on certain unshakable preconceptions of its writer, Malcolm W. Browne. Browne telephoned George Dantzig of Stanford University, a great pioneering authority on linear programming, and tried to force him into various admissions. Dantzig's version of the interview bears repeating. "What about the Traveling Salesman problem?" asked Browne. "If there is a connection, I don't know what it is," said Dantzig. ("The Russian discovery proposed an approach for [solving] a class of problems related to the 'Traveling Salesman Problem,' " reported Browne.) "What about cryptography?" asked Browne. "If there is a connection, I don't know what it is," said Dantzig. ("The theory of codes could eventually be affected," reported Browne.) "Is the Russian method practical?" asked Browne. "No," said Dantzig. ("Mathematicians describe the discovery...as a method by which computers can find solutions to a class of very hard problems that has hitherto been attacked on a hit-or-miss basis," reported Browne.)

Science is mailed to the 130,000 members of the American Association for the Advancement of Science. Immediately upon publication of its story, my phone began ringing, as did those of Dantzig, Graham, Peter Gacs of the University of Rochester, and almost everyone else mentioned in the story. Lovasz, back in Hungary, was protected from this onslaught, as of course was Khachian, still in relative isolation in Moscow. When the *Times* story appeared, the deluge predictably increased, and predictably changed character. One caller asked me to explain how in the world the Russians allowed such an important secret to escape their borders. A Beverly Hills lawyer asked if this new development might affect the immigration status of a client. Graham was called by a radio talk show host and asked if it wasn't true that the Russian discovery would change the lives of every man, woman and child in America.

When *Time* magazine called, I was apprehensive. A cover story? Perhaps Khachian was to be Man of the Year? But no. It was not hard to convince the *Time* man that the story was, well, just a bit overblown, and perhaps *Time* should pass it up. His parting question was to ask if I could recommend a good book on linear programming.

Artful Ellipsoids

L.G. Khachian had not solved the Traveling Salesman problem, or anything like it. If he had, that *would* have deserved the front page of the *Times* and the *Guardian* (more on that Traveling Salesman later).

What Khachian *had* done was to propose a remarkable new algorithm, or computational procedure, for solving linear programming problems. And, as every modern M.B.A. knows, linear programming has innumerable applications in economic modeling and business planning. It is used to allocate resources, plan production, schedule workers, plan investment portfolios and formulate marketing (and military) strategies. The versatility and economic impact of linear programming in today's industrial world is truly awesome.

In order to understand how large corporations use linear programming to minimize costs or maximize profits, let's consider an application for consumers: buying food at the supermarket. We want our food purchases to cost as little as possible, but we insist that they provide us with all the nutrients required to sustain healthful life.

We must first decide what our daily nutritional requirements are: so many calories, so much protein, fat, and carbohydrates, so many units of each of several vitamins, minerals and so forth. Let us suppose that we decide upon 30 such nutritional components, and our daily requirements of each of them. We then go to the supermarkets in which there are, say, 1,000 different foods to choose from. For each of the foods, we record its price per unit and the amount per unit of our daily requirements of each of the 30 different nutritional components which the food supplies. When we are done, we have a table of 31,000 numbers.

Our table of numbers provides the input data for a linear programming problem with 1,000 "decision variables" indicating the amounts of each of the foods to be purchased, and 30 "constraints" determined by the nutritional requirements. Each constraint is linear, in that it involves simply a summation of the 1,000 variables, appropriately weighted by coefficients specified by our table of numbers. Our problem also has a linear "objective function," determined by a summation of the 1,000 variables, with coefficients specified by the prices of the foods. An "optimal" solution to the linear programming problem minimizes the objective function, that is, makes our purchases of food as cheap as possible, subject to the nutritional constraints. The resulting diet may not be tasteful—consisting of, say, peanut butter sandwiches, mustard greens and buttermilk. But there is no doubt that it will be "rational" and "optimal" within the limitations of the model we have formulated.

A linear programming problem like ours, with 1,000 variables and 30 constraints, is by no means large by present-day standards. Problems of much larger size and complexity are routinely formulated and solved, quite

successfully, by the "simplex" method of computation invented by George Dantzig in 1947.

The simplex method can be visualized in geometric terms. Imagine a space whose dimensionality is determined by the number of variables, a dimensionality of 1,000 in the case of our diet problem. (There is really no magic in working in high-dimensional spaces. Mathematicians don't try very hard to visualize them, and usually content themselves with drawing suggestive sketches of three- or even two-dimensional examples.) Within this space imagine a polyhedron (a multidimensional polygon) whose flat sides or "faces" are determined by the linear constraints of the problem and whose corner points or "vertices" correspond to possible solutions. The simplex computation proceeds from one vertex of this polyhedron to another, continually improving the value of the solution, until it finally arrives at a vertex corresponding to an optimal solution.

The computational method that Khachian proposed is quite different, but can also be described in terms of the geometry of the same high-dimensional space. At a given point in the computation, an optimal solution is known to lie within an ellipsoid (a multidimensional ellipse, sort of egg-shaped). The center of this ellipsoid is tested to see if it is an optimal solution. If it is not, it is possible to slice the ellipsoid in two with a plane defined by one of the linear constraints of the problem. An optimal solution is known to lie within one of the "semiellipsoids" which result from this slicing. This semiellipsoid is then artfully surrounded by a new ellipsoid, and the process repeated. The ellipsoids get smaller and smaller, and eventually the center of one of them turns out to yield an optimal solution.

Do It Yourself (1962), by Andy Warhol Photo courtesy of Leo Castelli Gallery, N.Y.

Pathological Problems

In order to understand why the new algorithm is so remarkable, we must understand how applied mathematicians and computer scientists measure computational efficiency.

One accepted practice is to determine the number of computational steps (and thus the running time) which an algorithm will require, in the worst possible case, in a problem of a given size. If the worst-case running time increases no faster than a polynomial function of problem size, an algorithm is said to be "polynomially bounded." (A polynomial function of problem size is, for example, the square, cube or any fixed power of problem size.)

It is a mathematical fact that a polynomial algorithm can be guaranteed to outperform (in the worst case) a nonpolynomial algorithm, for sufficiently large problems. It is also an *empirical* fact that polynomial algorithms tend to be efficient in practice, and usually outperform nonpolynomial algorithms. Beginning about 15 years ago, both theoretical and practical computer scientists began to accept that polynomial algorithms are, virtually by definition, "good" and "efficient" algorithms.

Suppose we concern ourselves only with linear programming problems in which the coefficients are whole numbers and not fractions, and suppose we measure problem size by the total number of digits in all the coefficients, for example, by the total number of digits in the table of 31,000 numbers for the diet problem. Subject to these ground rules, the simplex method is not polynomially bounded. In 1967, Victor Klee of the University of Washington and George Minty of Indiana concocted a "pathological" class of linear programming problems for which the most commonly employed version of the simplex method must carry out a horrendously long sequence of computational steps before arriving at an optimal solution.

The ellipsoid method is polynomially bounded and that is what gives it such enormous importance to theoreticians. But this does not mean that it is better than the simplex method. The worst case behavior of the ellipsoid method is better, but the average or typical running time is certainly a good deal worse. Dantzig has estimated that problems which are routinely solved within half an hour by the simplex method would require fifty million years to solve by the new method. Some preliminary computational experiments have confirmed these pessimistic projections, and have also indicated some technical difficulties, such as "numerical instability." This latter defect can be overcome by performing extremely precise arithmetic on numbers with, say, hundreds of thousands of digits to the right of the decimal point. But this is a cure which is almost worse than the disease.

It is possible that the new algorithm can be modified

(Continued on page 34)

Mathematical Sputnik

Continued from page 15

and improved so that it can become a practical computational method. But this is a matter for much further investigation. And by the time that happens, it is also entirely possible that someone will have found a way to modify the simplex method so that it, too, will be polynomially bounded.

In order to fully understand what the *Times* called the "profound theoretical interest" of Khachian's result—its alleged relation to the Traveling Salesman problem and how "the theory of codes could eventually be affected"—a bit more background information is required. It is an irresistible temptation for me to begin by quoting the opening paragraphs of my own textbook, *Combinatorial Optimization*: "Combinatorial analysis is the mathematical study of the arrangement, grouping, ordering or selection of discrete objects, usually finite in number. Traditionally, combinatorialists have been concerned with the questions of existence or of enumeration. That is, does a particular type of arrangement exist? Or how many such arrangements are there?

"Quite recently, a new line of combinatorial investigation has gained increasing importance. The question asked is not 'Does the arrangement exist?' or 'How many arrangements are there?' but rather 'What is a *best* arrangement?' The existence of a particular type of arrangement is usually not in question, and the number of such possible arrangements is irrelevant. All that matters is finding an optimal arrangement, whether it be one in a hundred or one in an effectively infinite number of possibilities."

Specialists in combinatorial optimization have a large stable of pet problems whose colorful names tend to belie their considerable technological and economic importance. There are the "Chinese Postman's problem," the "Knapsack problem" and even the "Homosexual Marriage problem." An enduring favorite is the Traveling Salesman problem, possibly because of its deceptively simple statement: A salesman must visit each of several specified cities. How can he find, with reference to a highway map or mileage table, a tour which will enable him to visit each city exactly once and to return to his home city with the smallest total mileage on the odometer of his car?

In recent years, theoretical computer scientists have become fascinated with issues of computational complexity: Which problems are easy to solve? Which problems are inherently difficult? And why? They have pored over the stable of problems in combinatorial optimization, analyzing them and classifying them, in much the same way that physicists try to bring order into the collection of particles in their subatomic zoo.

P and NP

A very useful approach to the classification of problems according to their inherent computational complexity was provided in the early 1970s by theoretical results of Stephen Cook of Toronto, coupled with insights of Richard Karp of Berkeley. (Essentially the same results were obtained independently by a Russian, L.A. Levin, now at MIT.) Under this approach, problems which can be solved in polynomially bounded time are placed in a class called P. Another class of problems called NP (for "nondeterministic polynomial") is defined. Roughly speaking, the problems in NP are those for which it is possible to prove the correctness of a solution in polynomially bounded time, if one is able to guess what the solution is. Every problem in P is also in NP.

It has not been proven that there are problems in NP which are not in P, but it is known that there are very special problems in NP called NP-complete problems. If any NP-complete problem can be solved in polynomially bounded time then *all* problems in NP can be solved in polynomially bounded time. A specially formulated version of the Traveling Salesman problem is known to be NP-complete. Hence a polynomially bounded solution to the Traveling Salesman problem would imply that P = NP.

The P = NP question can reasonably be said to be the major open problem in theoretical computer science today. It is truly a baffling problem, one which many investigators have tackled and come away from essentially empty-handed. It has even attracted the attention of mathematical logicians, some of whom have speculated on the relations between the P = NP question and the axiomatic foundations of mathematics itself.

Although no one has been able to prove that P is not equal to NP, nearly all computer scientists conjecture that this is the case, and a certain amount of circumstantial evidence has accumulated to support this conjecture. Much current research, including some research in cryptography, proceeds on the premise that certain problems in NP are difficult to solve in practice.

The problem of cryptography is to encode messages so that it will be exceedingly difficult and time-consuming for an outsider to unscramble the encoded text. A number of novel proposals for "public key" encryption systems have been made by computer scientists. One proposal is based on the assumption that factorization of large numbers, a problem in NP, is very difficult. Others are based on the assumption that still other problems in NP are hard.

Khachian emphatically did not discover a polynomially bounded algorithm for the Traveling Salesman problem. Had he done so, as the *Guardian* claimed and the *Times* strongly implied, the effect would have been devastating. With P proved equal to NP, a possibility discounted by nearly all theoreticians, innumerable technical papers instantly would become dead letters. The theory of computational complexity would require top-to-bottom rethinking. One of the lesser issues would be that various proposals for data encryption would appear ill-advised. And if the polynomially bounded algorithm proved to be efficient in a practical sense, its economic impact quite possibly would exceed that of the simplex method.

What Khachian did do was to answer a much smaller question in complexity theory. Linear programming had been known to be a problem in NP. It had not been shown to be NP-complete, and there was some evidence that it could not be. Some investigators, including myself, thought that perhaps linear programming was a problem of intermediate difficulty: not in P, but also not NP-complete. Khachian squeezed linear programming into P and thereby resolved the issue.

But there is even some question as to how much recognition should be given to Khachian for this smaller result. It appears that the ellipsoid algorithm was developed by three other Russians, D.B. Judin, A.Z. Nemirovsky and N.Z. Shor, in the context of more general problems than linear programming. In a 1976 paper, Judin and Nemirovsky explicitly suggested the application of the ellipsoid approach to linear programming. In a 1977 paper (not referenced by Khachian), Shor worked out almost the precise formulas employed by Khachian. What Khachian did was to supply the final bit of argumentation necessary to obtain the polynomial

boundedness result. Because of this mixture of contributions (and in deference to the fact that Khachian is of Armenian, rather than Russian, extraction), perhaps the ellipsoid algorithm should be referred to as the "Soviet" method.

Without Apology

On November 11, under the headline "Soviet Mathematician Is Obscure No More," the *Times* reported that it had located Khachian. The *Times* Moscow correspondent found that he was "a relaxed, friendly young man," a "kandidat" who performs research at the computer center of the Academy of Sciences. Khachian was reported as saying that he was "somewhat surprised" by the enthusiastic response his paper had had in the West. "All this glory has fallen on him quite unexpectedly," his "teacher and mentor," G.S. Pospelov, was quoted as saying.

One might say that Khachian and Pospelov just don't understand Western journalism as well as they understand mathematics.

In fact, for mathematicians who might have been unfamiliar with the inner workings of the press, the behavior of the *Times* provided a kind of higher education. After Malcolm Browne's first article appeared on November 7, mathematicians telephoned the *Times*, wrote letters to the editors and reporters involved, and invited the *Times* to seminars on the ellipsoid method at IBM and in New York City. But the paper printed three follow-up articles, the last of which repeated the mistake of the first, saying, "Mr. Khachian's method is believed to offer an approach for the linear programming of computers to solve so-called 'Traveling Salesman' problems."

Only much later, on March 21, 1980, did the *Times* print a retraction. And the retraction was so artfully constructed that the uninformed might have mistaken it for a simple update. Six paragraphs down, the piece did admit that the *Times* had been mistaken in its linking of Khachian's work to the Traveling Salesman problem. But the headline read "A Russian's Solution in Math Questioned" and the subhead read "Americans Who Studied Khachian Linear Programming Method Express Doubt on Scope." That made it sound as though poor Khachian had exaggerated the importance of his work, and Western mathematicians had cut him down to size. The lead sentence of the piece reinforced that impression: "American mathematicians who have studied the new Soviet Method for solving a difficult class of computational problems known as linear programming problems say that the feat announced last November, while important, is far from the seminal achievement originally portrayed."

To which one can only nod, and add with a sigh, "Originally portrayed . . . by the *newspapers*." □

Annals of Discrete Mathematics 16 (1982) 189–200

FLOW NETWORK FORMULATIONS OF POLYMATROID OPTIMIZATION PROBLEMS*

E.L. LAWLER

Computer Science Division, University of California, College of Engineering, Berkeley, CA 94720, USA

C.U. MARTEL

Department of Electrical and Computer Engineering, University of California, Davis, CA 95616, USA

In the 'polymatroidal' network flow model, capacity constraints are imposed by polymatroid rank functions on the sets of arcs directed into and out of each node. It is shown that a variety of matroid optimization problems can be easily formulated and solved in terms of this model. Among these problems are (poly)matroid intersection, matroid partitioning, problems of gammoids and linking systems, and problems formulated by Iri, Krogdahl, and Fujishige. It is shown that simple proofs of known min–max theorems for these problems are easily obtained as corollaries of the max-flow min-cut theorem for polymatroidal network flows, previously proved by the authors.

1. Introduction

In the 'classical' network flow model, flows are constrained by the capacities of individual arcs. In the 'polymatroidal' network flow model, flows are constrained by the capacities of *sets* of arcs, where these capacities are imposed by polymatroid rank functions on the sets of arcs directed into and out of each node. Yet, as the authors have shown in another paper [10] the essential features of the classical model are retained; the augmenting path theorem, the integral flow theorem, and the max-flow min-cut theorem all yield to straightforward generalization. Moreover, a maximal integral flow can be computed efficiently, provided that there is a 'feasibility' oracle available for each capacity function.

We believe that the polymatroidal network flow model provides a satisfying generalization and unification of both network flow theory and much of the theory of (poly)matroid optimization. In this paper we demonstrate the usefulness of the model by providing network flow formulations of (poly)matroid

* This research was supported by National Science Foundation Grant MCS 78-20054.

intersection, matroid partitioning, problems of gammoids and linking systems, and problems formulated by Iri, Krogdahl and Fujishige. In fact, virtually every known problem in (poly)matroid optimization known to the authors can be easily formulated in terms of their network flow model, with the conspicuous exception of the polymatroid matching problem, solved by Lovász for the case of linearly represented polymatroids [11]. (Polymatroid matching cannot be formulated as a polymatroidal network flow problem, just as (nonbipartite) graphic matching cannot be formulated as an ordinary network flow problem.)

We assert that, when the maximal flow algorithm described in [10] is applied to each of the networks described in this paper, it either specializes to a known algorithm or is competitive with special algorithms which have been developed for the problems in question. We shall not elaborate on this point, but instead emphasize the ease with which min–max theorems can be proved, as corollaries of the general max-flow min-cut theorem proved in [10].

Remark. The polymatroidal network flow model described in this paper was formulated independently by Hassin [7], who considered a more general model in which costs are associated with flows in individual arcs and in which supermodular set functions impose lower bounds on flows through subsets of arcs. In [7] Hassin proved a (circulation) version of the max-flow min-cut theorem and also a more general version of the integrality theorem stated here. (These theorems require that the submodular upper bounds on flow and the supermodular lower bounds satisfy certain conditions.) He also developed algorithms for finding optimal flows, but gave no complexity estimates.

2. The polymatroidal network flow model

A *polymatroid* (E, ρ) is defined by a finite set of *elements* E and a *rank function* $\rho : 2^E \to \mathbb{R}^+ \cup \{\infty\}$ satisfying the following properties:

$$\rho(\emptyset) = 0\,, \tag{2.1}$$

$$\rho(X) \leqslant \rho(Y) \quad (X \subseteq Y \subseteq E)\,, \tag{2.2}$$

$$\rho(X \cup Y) + \rho(X \cap Y) \leqslant \rho(X) + \rho(Y) \quad (X \subseteq E,\ Y \subseteq E)\,. \tag{2.3}$$

Inequality (2.2) states that the rank function is monotone and (2.3) asserts that it is submodular. If also ρ is integer-valued and $\rho(\{e\}) = 0$ or 1 for all $e \in E$, then the polymatroid is a *matroid*. If (E, ρ) is matroid and $I \subseteq E$ is such that $|I| = \rho(I)$, then I is an *independent* set.

A *polymatroidal flow network* is a directed multigraph with a *source* s, a *sink* t

and two *capacity functions* α_j and β_j for each node j. Each function α_j (β_j) satisfies properties (2.1)–(2.3) with respect to the set of arcs A_j (B_j) directed out from (into) node j. Thus (A_j, α_j) and (B_j, β_j) are polymatroids. (Comment: We permit multiple arcs from one node to another. Hence A_j and B_j may be arbitrarily large finite sets.)

A *flow* in the network is an assignment of real numbers to the arcs of the network. Thus a flow is given by a function $f : E \to \mathbb{R}$, where E is the set of arcs. Such a function can be extended to subsets of arcs in a natural way, i.e.,

$$\begin{aligned} &f(\emptyset) = 0\,, \\ &f(X) = \sum_{e \in X} f(e) \quad (\emptyset \neq X \subseteq E)\,. \end{aligned} \tag{2.4}$$

(We continue to write $f(e)$ instead of the more cumbersome $f(\{e\})$.)

A flow is said to be *feasible* if

$$f(A_j) = f(B_j)\,, \quad j \neq s, t\,, \tag{2.5}$$

$$f(X) \leqslant \alpha_j(X) \quad \text{for all } j \text{ and } X \subseteq A_j\,, \tag{2.6a}$$

$$f(X) \leqslant \beta_j(X) \quad \text{for all } j \text{ and } X \subseteq B_j\,, \tag{2.6b}$$

$$f(e) \geqslant 0 \quad \text{for all arcs } e\,. \tag{2.7}$$

Eq. (2.5) imposes the customary flow conservation law at each node other than the source and sink. Inequalities (2.6a) and (2.6b) assert that capacity constraints are satisfied on sets of arcs, and (2.7) simply demands that the flow through each arc be nonnegative. Our objective is to find a feasible flow which has maximum value, i.e., one which maximizes

$$v = f(A_s) - f(B_s) = f(B_t) - f(A_t)\,.$$

Comment. It is possible to generalize the polymatroidal network flow model to provide for costs on arc flows and lower bounds on arc flows as imposed by supermodular set functions. However these generalizations are unnecessary for our present purposes.

3. Integrality and max-flow min-cut theorems

We now state two theorems proved in [10]. Strictly speaking, the statements of these theorems should allow for the possibility of the nonexistence of a

maximal flow (which occurs if the maximum flow value is unbounded). However, we ignore this possibility, since it does not occur in any of the problems considered here.

Integrality Theorem. *If all capacity functions are integer-valued, then there exists a maximal flow which is integral.*

An *arc-partitioned cut* (S, T, L, U) is defined by a partition of the nodes into two sets S and T, with $s \in S$ and $t \in T$, and by a partition of the forward arcs across the cut into two sets L and U. The capacity of an arc partitioned cut is defined as

$$c(S, T, L, U) = \sum_{i \in S} \alpha_i (U \cap A_i) + \sum_{j \in T} \beta_j (L \cap B_j) .$$

As in the case of ordinary flow networks, the value v of any feasible flow f is equal to the net flow across any cut, i.e.,

$$v = f(L) + f(U) - f(B) ,$$

where B is the set of backward arcs, and clearly

$$v \leqslant c(S, T, L, U) .$$

Max-Flow Min-Cut Theorem. *The maximum value of a flow is equal to the minimum capacity of an arc-partitioned cut.*

4. (Poly)matroid intersection

The (unweighted) *matroid intersection* problem is as follows: Given two matroids (E, ρ_1) and (E, ρ_2), find a largest set $I \subseteq E$ such that I is independent in each of the matroids.

This problem can be formulated and solved as a flow problem as shown in Fig. 1. There are two nodes, s and t, and each arc from s to t corresponds to an element $e_i \in E$. The two capacity functions are determined by the two matroid

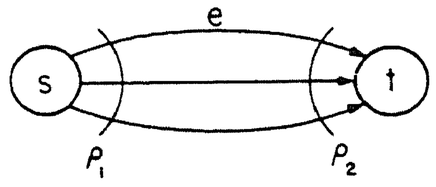


Fig. 1. Network for poly(matroid) intersection.

rank functions: $\alpha_s = \rho_1$, $\beta_t = \rho_2$. Since these capacity functions are integer-valued, there exists a maximal flow which is integral. Any such integral maximal flow corresponds to a solution to the matroid intersection problem.

Any partitioned cut (S, T, L, U) must have $S = \{s\}$, $T = \{t\}$ and is thus determined simply by a partition of E into two subsets L and $U = E - L$. The duality theorem for this problem thus follows as an immediate corollary of the max-flow min-cut theorem.

Matroid Intersection Duality Theorem. $\max|I| = \min_{L \subseteq E}\{\rho_1(E-L) + \rho_2(L)\}$.

In the *polymatroid intersection* problem, the capacity functions are simply the rank functions of the two polymatroids, and the duality theorem generalizes in an obvious way.

5. Matroid partitioning

A general version of the *matroid partitioning problem* is as follows: Given k matroids $M_j = (E, \rho_j)$, $j = 1, \ldots, k$, find a largest set $I \subseteq E$ such that I can be partitioned into k sets $I_1, \ldots, I_k$, where I_j is independent in (E, ρ_j).

A flow network for this problem is shown in Fig. 2. There is a node for each element $e_i \in E$, $i = 1, \ldots, n$, and a node for each matroid M_j plus a source s and a sink t. There is an arc (e_i, M_j) for each e_i and M_j. The flow into each node M_j is constrained by the capacity function $\beta_j = \rho_j$. Each arc (s, e_i) has unit capacity; for convenience we assume that these unit capacities are imposed by the capacity function α_s, where $\alpha_s(X) = |X|$, for all $X \subseteq A_s$. There are no other capacity constraints.

It follows from the integrality theorem that there exists a maximal flow which corresponds to an optimal solution to the partitioning problem. In order to determine an optimal dual solution we reason as follows. An arc-partitioned cut with finite capacity must have all nodes M_j in T. Let us denote by A the subset of nodes e_i in S; the nodes in $E - A$ are in T. For given S and T, L and

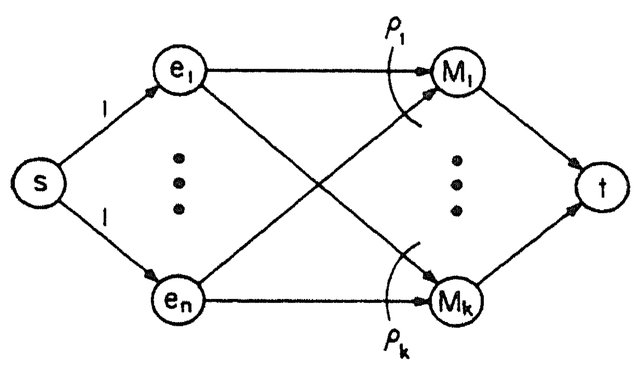


Fig. 2. Network for matroid partitioning.

U must be chosen as follows, in order for the arc-partitioned cut to have finite capacity:

$$L = \{(e_i, M_j) \mid e_i \in A\}, \qquad U = \{(s, e_i) \mid e_i \in E - A\},$$

as shown in Fig. 3. Thus the minimum capacity of an arc-partitioned cut is

$$\min_{A \subseteq E} \left\{ |E - A| + \sum_j \rho_j(A) \right\}.$$

This capacity is strictly less than $|E|$ if and only if

$$|A| > \sum_j \rho_j(A),$$

for some $A \subseteq E$. We have thus proved the following well-known result.

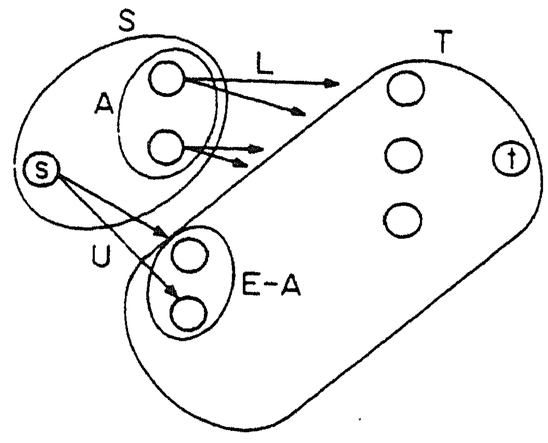


Fig. 3. Cut in proof of the matroid partitioning duality theorem.

Matroid Partitioning Duality Theorem (Edmonds and Fulkerson [2]). *Let I be a feasible solution to the matroid partitioning problem. Then*

$$\max|I| = \min_{A \subseteq E} \left\{ |E - A| + \sum_j \rho_j(A) \right\}.$$

Moreover, E is a feasible solution if and only if

$$|A| \leq \sum_j \rho_j(A),$$

for all $A \subseteq E$.

6. A problem of Krogdahl

The unweighted version of a problem considered by Krogdahl [9] and Schrijver [13] is as follows: Given k matroids $M_j = (E, \rho_j)$, $j = 1, \ldots, k$, and l matroids $M'_j = (E, \rho'_j)$, $j = 1, \ldots, l$, find a largest set I such that I can be partitioned into k subsets $I_1, \ldots, I_k$, where I_j is independent in M_j, and I can also be partitioned into l subsets $I'_1, \ldots, I'_l$, where I'_j is independent in M'_j.

It is well known that a set I can be so partitioned if and only if I is independent in both M and M', where M (M') is the *sum* of matroids $M_1, \ldots, M_k$ ($M'_1, \ldots, M'_l$, respectively). Thus this is actually a matroid intersection problem in which independence in each of the matroids M, M' can be determined by solving a matroid partitioning problem. Hence it is not surprising that the flow network for this problem, as shown in Fig. 4, is much like two networks for the matroid partitioning problem joined together. Each element e_i is represented by two nodes, e_i and e'_i, and an arc (e_i, e'_i), as shown in the center of the network. Each arc (e_i, e'_i) has unit capacity and we assume this capacity is imposed by a capacity function at the tail of the arc.

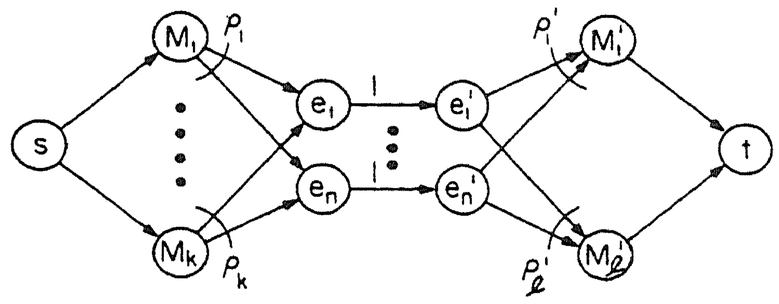


Fig. 4. Network for Krogdahl's problem.

In order for an arc-partitioned cut to have finite capacity, all of the nodes $M_1, \ldots, M_k$ must be in S and all of the nodes $M'_1, \ldots, M'_l$ must be in T. Thus we need only consider how the nodes $e_1, \ldots, e_n, e'_1, \ldots, e'_n$ are assigned to S and T:

(i) If $e'_i \in S$, then each of the arcs (e'_i, M'_j) must be in L. (There are no capacity constraints at the tails of these arcs.)

(ii) If $e_i \in T$, then each of the arcs (M_j, e_i) must be in U. (There are no capacity constraints at the heads of these arcs.)

(iii) If $e_i \in S$, $e'_i \in T$, then the arc (e_i, e'_i) must be in U, and its contribution to the capacity of the arc-partitioned cut is unity, independent of the other arcs in the cut.

It follows that there exists an arc-partitioned cut (S, T, L, U) of minimum capacity in which, for no arc (e_i, e'_i), the case holds that $e_i \in T$, $e'_i \in S$. If this were so, either e_i could be moved to S or e'_i could be moved to T without increasing the capacity. (Either the new L or the new U would be a proper subset of the old.) We thus see that an arc-partitioned cut simply partitions the arcs (e_i, e'_i) into three classes: A', A, and $E - (A \cup A')$, corresponding to the cases $e'_i \in S$, $e_i \in T$ and $e_i \in S$, $e'_i \in T$, indicated above, and as shown in Fig. 5.

By reasoning similar to that in the previous section, we now have the following result.

Theorem 6.1. *Let I be a feasible solution to the problem of Krogdahl. Then*

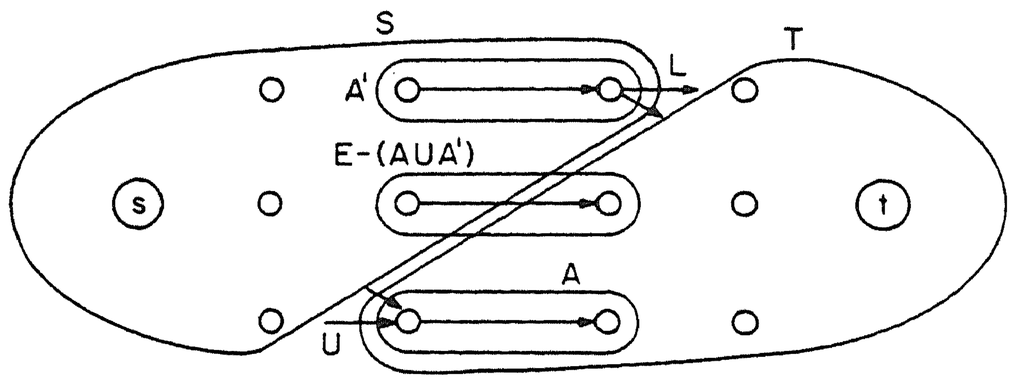


Fig. 5. Cut in proof of Theorem 6.1.

$$\max|I| = \min_{A,A' \subseteq E} \left\{ |E - (A \cup A')| + \sum_j \rho_j(A) + \sum_j \rho'_j(A') \right\}.$$

Moreover, E is a feasible solution if and only if for all $A, A' \subseteq E$,

$$|A \cup A'| \leq \sum_j \rho_j(A) + \sum_j \rho'_j(A').$$

A special case of interest is that of finding a largest common partial transversal of two families of subsets of E, $\mathscr{E} = (E_1, E_2, \ldots, E_k)$, and $\mathscr{E}' = (E'_1, \ldots, E'_l)$. Here let

$$\rho_j(X) = \begin{cases} 1 & \text{if } X \subseteq E_j, \\ 0 & \text{otherwise}, \end{cases}$$

and define ρ'_j similarly. Then the appropriate duality theorem follows.

As another special case, let $l = 1$ and k be arbitrary. Then the problem becomes that of finding a largest set I'_1, such that I'_1 is independent in M'_1 and I'_1 is partitionable into subsets $I_1, \ldots, I_k$, where I_j is independent in M_j. Since

$$\rho'_1(E - A) \leq |E - (A \cup A')| + \rho'_1(A'),$$

we have

$$\max|I'_1| = \min_{A \subseteq E} \left\{ \rho'_1(E - A) + \sum_j \rho_j(A) \right\}.$$

If $\rho'_1(X) = |X|$, for all $X \subseteq E$, then this result further specializes to the Edmonds–Fulkerson theorem.

7. A problem of Fujishige

Fujishige [5] has formulated and solved the following problem: Let $G = (V, E)$ be a digraph with two disjoint sets $V_1, V_2 \subseteq V$ of sources and sinks,

respectively. Let (V_1, ρ_1) and (V_2, ρ_2) be polymatroids, and let each arc $(i, j) \in E$ be assigned a capacity c_{ij}. The problem is to find a maximum value flow from the sources in V_1 to the sinks in V_2 subject to the constraints that the net flows out of the source nodes in V_1 and the net flows into the sink nodes in V_2 are feasible with respect to the polymatroids (V_1, ρ_1), (V_2, ρ_2), and all arc capacity constraints are respected.

This problem can easily be formulated as a polymatroidal network flow problem by adding a dummy source s and a dummy sink t, as shown in Fig. 6.

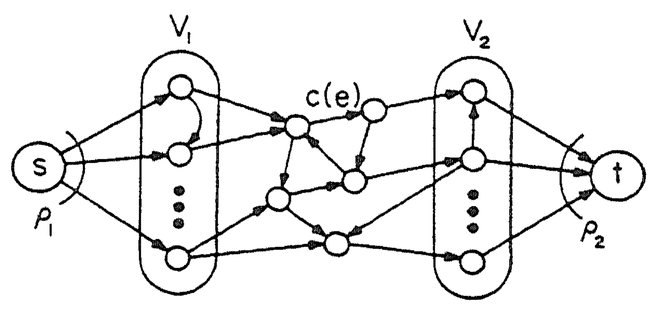


Fig. 6. Network for a problem of Fujishige.

There is an arc (s, i) to each node i in V_1 and we let $\alpha_s = \rho_1$. Similarly, there is an arc (j, t) from each node j in V_2 to t and we let $\beta_t = \rho_2$. We impose capacity constraints on the arcs (i, j) in G by letting

$$\alpha_i(X) = \sum_{(i,j)\in X} c_{ij} \quad (X \subseteq A_i)\,.$$

An arc-partitioned cut for this network is indicated in Fig. 7. The forward arcs across this cut are of three kinds:

(i) Arcs of the form (s, i) where $i \in V_1 \cap T$. All such arcs are in U, if the cut is to have finite capacity, and their contribution to the capacity of the cut is $\rho_1(V_1 \cap T)$.

(ii) Arcs of the form (j, t), where $j \in V_2 \cap S$. All such arcs are in L, if the cut is to have finite capacity, and their contribution to the capacity of the cut is $\rho_2(V_2 \cap S)$.

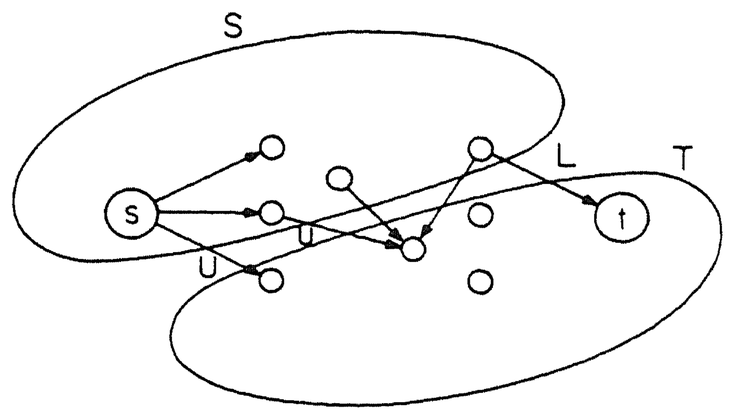


Fig. 7. Cut in proof of Theorem 7.1.

(iii) Each arc $e = (i, j)$ of G, such that $i \in S$, $j \in T$. The total contribution of these arcs to the capacity of the cut is

$$\sum_{i \in S, j \in T} c_{ij},$$

independent of the contributions of the arcs considered in (i) and (ii) above.

We thus have the following duality result.

Theorem 7.1 (Fujishige [4]). *Let v be the maximum feasible flow value. Then*

$$v = \min\left\{\rho_1(V_1 \cap S) + \rho_2(V_2 \cap T) + \sum_{i \in S, j \in T} c_{ij}\right\},$$

where the minimum is taken over all partitions S, T of V.

8. Gammoids and linking systems

Problems involving gammoids and linking systems are easily formulated as network flow problems. One such problem considered by Brualdi [1] and Schrijver [13] is as follows: Let $G = (V, E)$ be a digraph, V_1, V_2 be disjoint subsets of V, and $M_1 = (V_1, \rho_1)$, $M_2 = (V_2, \rho_2)$ be matroids. Find a pair of subsets $I_1 \subseteq V_1$, $I_2 \subseteq V_2$, such that $k = |I_1| = |I_2|$ is as large as possible, I_j is independent in M_j, $j = 1, 2$, and there are k node-disjoint paths linking I_1 to I_2.

This problem is clearly very much like that of Fujishige, except that, instead of arc capacities, there are node capacities (i.e., there may be at most one unit of flow through each node $i \in V$). In order to impose these constraints, we set

$$\alpha_i(X) = 1 \quad (\emptyset \neq X \subseteq A_i)$$

for each node $i \in V$. The following result is now easy to obtain, by the same reasoning as in the previous section.

Theorem 8.1 (Brualdi [1]). *The maximum of $k = |I_1| = |I_2|$ is equal to*

$$\min\{\rho_1(V_1 - I_1) + \rho_2(V_2 - I_2) + |X|\},$$

where $X \subseteq V$ intersects each path from I_1 to I_2.

9. Independent assignments

The unweighted version of the 'independent assignment' problem considered by Iri and Tomizawa [8] is as follows: Given a bipartite graph $G = (V_1, V_2; E)$ and two matroids $M_1 = (V_1, \rho_1)$, $M_2 = (V_2, \rho_2)$, find a matching in G such that

the vertices $I_1 \subseteq V_1$, $I_2 \subseteq V_2$ covered by the matching are independent in M_1 and M_2.

This problem, which can also be formulated as a matroid intersection problem, is clearly a special case of the problem of Fujishige, where $V = V_1 \cup V_2$, and $c_{ij} = \infty$ for each arc $(i, j) \in E$, $i \in V_1$, $j \in V_2$. In order for an arc-partitioned cut (S, T, L, U) to have finite capacity, there can be no arcs (i, j) such that $i \in S$, $j \in T$. Thus the vertices $V_1' \cup V_2'$, where $V_1' = T \cap V_1$, $V_2' = S \cap V_2$, must provide a covering of the arcs of G. Define the *rank* of such a covering $V_1' \cup V_2'$ as $\rho_1(V_1') + \rho_2(V_2')$. We then have the following result.

Theorem 9.1. *The maximum size of an 'independent assignment' is equal to the minimum rank of a covering of edges by vertices.*

The well-known König–Egervary theorem is clearly a corollary, for the case that (V_j, ρ_j), $j = 1, 2$, are free matroids, i.e., $\rho_j(X) = |X|$, for all $X \subseteq V_j$.

10. The Rado–Hall problem

Let $M = (E, \rho)$ be a matroid and let $\mathscr{E} = (E_1, \ldots, E_k)$ be a family of subsets of E (with repetitions allowed). The problem is to find a largest partial transversal which is independent in M.

This is a matroid intersection problem, in which the two matroids are M and the transversal matroid induced by $\mathscr{E}$. The problem can also be viewed as a special case of the independent assignment problem in which the graph $G = (V_1, V_2; A)$ and the two matroids $M_1 = (V_1, \rho_1)$, $M_2 = (V_2, \rho_2)$ are as follows: $V_1 = E$, $V_2 = \mathscr{E}$, $(i, j) \in A$ if and only if $e_i \in E_j$, $\rho_1 = \rho$, $\rho_2(V_2') = |V_2'|$ for all $V_2' \subseteq V_2$. It is now easy to obtain the following result.

Rado–Hall Theorem (Rado [12]). *The size of a largest independent partial transversal is*

$$\min_{\mathscr{E}' \subseteq \mathscr{E}} \{|\mathscr{E} - \mathscr{E}'| + \rho(E')\},$$

where

$$E' = \bigcup_{E_j \in \mathscr{E}'} E_j .$$

Moreover, there exists an independent transversal if and only if the union of any k sets in $\mathscr{E}$ has matroid rank at least k.

Since this theorem specializes to the well-known Hall theorem in the case that M is a free matroid, it is known as the Rado–Hall theorem.

11. Concluding remarks

We believe that the formulations given in this paper demonstrate the usefulness of the polymatroidal network flow model. It should be emphasized that other models, e.g., those of Fujishige [5] and Edmonds and Giles [3], are equivalent in the sense that any problem which can be formulated and solved in terms of one model can be formulated and solved in terms of one of the others. (We assert, but do not prove here, that the Edmonds–Giles model can be reformulated in terms of the polymatroidal network flow model, with costs on the arcs.) We believe that the advantage of our model is that it permits problem formulations to be particularly simple and transparent. For example, although one can, in principle, reformulate the matroid partition problem as a matroid intersection problem, it seems simpler and more direct to formulate it directly in terms of the network given in Section 5.

References

[1] R.A. Brualdi, Menger's theorem and matroids, J. London Math. Soc. 4 (1971) 46–50.
[2] J. Edmonds and D.R. Fulkerson, Transversals and matroid partition, J. Res. Nat. Bur. Standards B69 (1965) 147–153.
[3] J. Edmonds and R. Giles, A min–max relation for submodular functions on graphs, Ann. Discrete Math. 1 (1977) 185–204.
[4] S. Fujishige, An algorithm for finding an optimal independent linkage, J. Oper. Res. Soc. Japan 20 (1977) 59–75.
[5] S. Fujishige, Algorithms for solving the independent flow problems, J. Oper. Res. Soc. Japan 21 (1978) 189–204.
[6] A. Frank, An algorithm for submodular functions on graphs, preprint, 1981.
[7] R. Hassin, On network flows, Ph.D. dissertation, Yale University, 1978.
[8] M. Iri and N. Tomizawa, An algorithm for finding an optimal independent assignment, J. Oper. Res. Soc. Japan 19 (1976) 32–57.
[9] S. Krogdahl, A combinatorial proof for a weighted matroid intersection algorithm, Univ. of Tromsø, Comp. Sci. Rept. 17, Tromsø, 1976.
[10] E.L. Lawler and C.U. Martel, Computing maximal 'polymatroidal' network flows, Math. Oper. Res., to appear.
[11] L. Lovász, The matroid matching problem, in: Algebraic Methods in Graph Theory, Colloq. Math. Soc. János Bolyai 25 (1978) 495–517.
[12] R. Rado, A theorem on independence relations, Quart. J. Math. (Oxford) 13 (1942) 83–89.
[13] A. Schrijver, Matroids and linking systems, Math. Centrum Tract 88, Math. Centrum, Amsterdam, 1978.
[14] P. Schönsleben, Ganzzahlige Polymatroid-Intersektions-Algorithmen, Thesis, ETH Zürich, 1980 (in German).
[15] U. Zimmermann, Minimization of some nonlinear functions over polymatroidal flows, preprint, 1981.

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 7, No. 3, August 1982
Printed in U.S.A.

COMPUTING MAXIMAL "POLYMATROIDAL" NETWORK FLOWS*†

E. L. LAWLER‡ AND C. U. MARTEL§

In the "classical" network flow model, flows are constrained by the capacities of individual arcs. In the "polymatroidal" network flow model introduced in this paper, flows are constrained by the capacities of *sets* of arcs. Yet the essential features of the classical model are retained; the augmenting path theorem, the integral flow theorem and the max-flow min-cut theorem are all shown to yield to straightforward generalization. We describe a maximal flow algorithm which finds augmenting paths by labeling arcs instead of nodes, as in the case of the classical model. As a counterpart of a known result for the classical model, we prove that the number of augmentations required to achieve a maximal value flow is bounded by the cube of the number of arcs in the network, provided each successive augmentation is made along a shortest augmenting path, with ties between shortest paths broken by lexicography.

1. Introduction. In the "classical" network flow model, flows are constrained by the capacities of individual arcs. In the "polymatroidal" network flow model introduced in this paper, flows are constrained by the capacities of *sets* of arcs. Yet the essential features of the classical model are retained; the augmenting path theorem, the integral flow theorem and the max-flow min-cut theorem are all shown to yield to straightforward generalization. We describe a maximal flow algorithm which finds augmenting paths by labeling arcs instead of nodes, as in the classical model. As a counterpart to a result of Edmonds and Karp [3] for the classical model, we prove that the number of augmentations required to achieve a maximal value flow is bounded by the cube of the number of arcs in the network, provided each successive augmentation is made along a shortest augmenting path, with ties between shortest paths broken by lexicography.

We believe that the theory of polymatroidal network flows provides a satisfying generalization and unification of both classical network flow theory and much of the theory of matroid optimization. In particular, various well-known algorithms for (poly)matroid intersection [1], [8]–[10], [13], matroid partitioning [2], [10] and other matroid optimization problems [5], [7] can be shown to be equivalent, or very nearly equivalent to specializations of the maximal flow algorithm presented in this paper. These specializations are obtained by applying the maximal flow algorithm to appropriately formulated polymatroidal flow networks. Moreover, the application of the max-flow min-cut theorem to these networks yields simple proofs of known duality theorems for these matroid optimization problems. These observations, as well as generalizations of the polymatroidal flow model to provide for costs and lower bounds on arc flows, will be reported in papers in preparation by the authors.

The plan of this paper is as follows. §2 deals with preliminaries, in the form of some simple lemmas concerning polymatroidal rank functions. In §3 the polymatroidal network flow model is formulated and in §4 the concept of an augmenting path is

* Received February 9, 1981.
AMS 1980 subject classification. Primary: 90B10. Secondary: 05B35.
OR/MS Index 1978 subject classification. Primary: 484 Networks/graphs/flow algorithms.
Key words. Matroid, polymatroid, flow network, algorithm, optimization, max-flow min-cut theorem.
† This research was supported by National Science Foundation Grant MCS 78-20054.
‡ University of California, Berkeley.
§ University of California, Davis.

introduced. §§5 through 7 present the maximal flow algorithm and provide proofs of the augmenting path theorem and max-flow min-cut theorem with reference to that algorithm. §§8 through 10 concern properties of shortcut-free augmenting paths and provide a proof of the integrality theorem. In §§11 and 12 the bound on the number of augmentations is proved, and various issues of computational complexity are discussed.

2. Some polymatroidal preliminaries. We assume that the reader is familiar with the basic concepts of network flow theory and with at least some of the ideas of matroid optimization, as presented in [10]. In this section we present a few results concerning polymatroids which are needed in the remainder of the paper.

A *polymatroid* (E, ρ) is defined by a finite set of *elements* E and a *rank function* $\rho: 2^E \to \mathbb{R}^+$ satisfying the properties

$$\rho(\emptyset) = 0, \tag{2.1}$$

$$\rho(X) \leqslant \rho(Y) \qquad (X \subseteq Y \subseteq E), \tag{2.2}$$

$$\rho(X \cup Y) + \rho(X \cap Y) \leqslant \rho(X) + \rho(Y) \qquad (X \subseteq E,\ Y \subseteq E). \tag{2.3}$$

Inequalities (2.2) state that the rank function is monotone and inequalities (2.3) assert that it is submodular. If ρ is also integer-valued and $\rho(\{e\}) = 0$ or 1 for all $e \in E$, then the polymatroid is a *matriod*.

We shall be dealing with polymatroids whose elements are arcs of a network. We shall assign values of "flow" to these arcs, which is equivalent to specifying a function $f: E \to \mathbb{R}$. This function can be extended to subsets of E in a natural way, i.e.,

$$f(\emptyset) = 0,$$

$$f(X) = \sum_{e \in X} f(e) \quad (\emptyset \neq X \subseteq E). \tag{2.4}$$

Such an extended flow function f will be said to be *feasible* with respect to the rank function ρ if for all $X \subseteq E$,

$$f(X) \leqslant \rho(X). \tag{2.5}$$

A feasible function f *saturates* X if (2.5) holds with equality. An individual element e will be said to be *saturated* if there is some saturated set in which it is contained.

The following three lemmas apply with respect to any polymatroid (E, ρ) and any feasible function f.

LEMMA 2.1. *If Y is a saturated set, then*

$$\rho(X) - f(X) \geqslant \rho(X \cap Y) - f(X \cap Y),$$

$$\rho(X) - f(X) \geqslant \rho(X \cup Y) - f(X \cup Y),$$

for all $X \subseteq E$.

PROOF.

$$\rho(X) + \rho(Y) \geqslant \rho(X \cup Y) + \rho(X \cap Y), \quad \text{by submodularity,}$$

$$f(X) + f(Y) = f(X \cup Y) + f(X \cap Y), \quad \text{by (2.4).}$$

Hence

$$\rho(X) - f(X) + \rho(Y) - f(Y) \geqslant \rho(X \cup Y) - f(X \cup Y) + \rho(X \cap Y) - f(X \cap Y),$$

and the result follows from the saturation of Y and the feasibility of f. ∎

LEMMA 2.2. *If X and Y are saturated sets, then so are $X \cap Y$ and $X \cup Y$.*

PROOF. Apply Lemma 2.1 with X a saturated set. ∎

LEMMA 2.3. *If $e \in E$ is saturated, then there is a unique minimal saturated set $S(e)$ containing e. Moreover, for each $e' \in S(e)$, $e' \neq e$, it is the case that $f(e') > 0$.*

PROOF. Suppose $S(e)$ and $S'(e)$ are distinct minimal saturated sets containing e. By Lemma 2.2, $S(e) \cap S'(e)$ is also a saturated set containing e, and neither $S(e)$ nor $S'(e)$ can be minimal.

Now suppose $S(e)$ is the unique minimal saturated set containing e and there is an element $e' \neq e$ in $S(e)$ such that $f(e') = 0$.

$$\begin{aligned} f(S(e) - \{e'\}) &\leqslant \rho(S(e) - \{e'\}), && \text{by feasibility} \\ &\leqslant \rho(S(e)), && \text{by monotonicity} \\ &= f(S(e)), && \text{by assumption} \\ &= f(S(e) - \{e'\}), && \text{since } f(e') = 0. \end{aligned}$$

It follows that $S(e) - \{e'\}$ is also saturated and $S(e)$ cannot be the minimal saturated set containing e. ∎

3. Polymatroidal flow networks. We shall consider only the simplest type of flow network, namely one in which there is a single *source* s and a single *sink* t. Our objective will be to find a maximum-value flow from s to t.

For each node j of the network there are specified two *capacity functions* α_j and β_j. The function α_j (β_j) satisfies properties (2.1)–(2.3) with respect to the set of arcs A_j (B_j) directed out from (into) node j. Thus (A_j, α_j) and (B_j, β_j) are polymatroids. (Comment: We permit there to be multiple arcs from one node to another. Hence A_j and B_j may be arbitrarily large finite sets.)

A *flow* in the network is an assignment of real numbers to the arcs of the network. We let a flow be represented by a function $f: 2^E \to \mathbb{R}$, obtained as in (2.4), where E is the set of arcs. A flow f is *feasible* if

$$f(A_j) = f(B_j), \qquad j \neq s, t, \tag{3.1}$$

$$f \text{ is feasible for } \alpha_j, \beta_j, \qquad \text{for all nodes } j, \tag{3.2}$$

$$f(e) \geqslant 0, \qquad \text{for all arcs } e. \tag{3.3}$$

(We write $f(e)$ in place of the more cumbersome $f(\{e\})$.)

Equations (3.1) impose the customary flow conservation law at each node other than the source and sink. Property (3.2) indicates that capacity constraints are satisfied on sets of arcs, and (3.3) simply demands that the flow through each arc be nonnegative. Our objective is to find a feasible flow of maximum value, i.e., one which maximizes

$$v = f(A_s) - f(B_s) = f(B_t) - f(A_t). \tag{3.4}$$

If, for a given feasible flow f, an arc $e = (i, j)$ is saturated with respect to α_i, we shall say that the *tail* of e is saturated and denote the minimal saturated set containing e by $T(e)$, where $T(e) \subseteq A_i$. Similarly, if e is saturated with respect to β_j, we shall say that the *head* of e is saturated and denote the minimal saturated set containing e by $H(e)$, where $H(e) \subseteq B_j$.

In the case of an ordinary flow network in which there is a specified capacity c_{ij} for each arc $e = (i, j)$, we can define $\alpha_i(e) = \beta_j(e) = c_{ij}$, and then extend the functions α_i, β_i to sets as in (2.4). The resulting capacity functions are modular, i.e., satisfy (2.3)

with equality. Note that in this special case the head of an arc e is saturated if and only if its tail is saturated, and $H(e) = T(e) = \{e\}$.

REMARK. We could choose to formulate the network flow model in terms of capacity functions which are simply nonnegative and submodular. All of the results in this paper would then follow, in virtually the same form. This is due to the fact that for every nonnegative submodular function ρ there is polymatroidal rank function $\bar{\rho}$ in the relation

$$\bar{\rho}(\emptyset) = 0,$$
$$\bar{\rho}(X) = \min\{\rho(Y) | X \subseteq Y \subseteq E\}, X \neq \emptyset,$$

where $\bar{\rho}$ has the same effect as a capacity function as ρ. We prefer to assume capacity functions are monotone, as well as submodular, because it slightly simplifies definitions and seems somewhat more intuitively appealing. Lemma 2.3 indicates the effect of monotonicity.

4. Augmenting paths. With respect to a given feasible flow f, an *augmenting path* is an undirected path of distinct arcs (but not necessarily distinct nodes) from s to t such that

(4.1) each backward arc e in the path is nonvoid, i.e., $f(e) > 0$; and

(4.2) if the head (tail) of a forward arc e in the path is saturated, then the following (preceding) arc in the path is a backward arc contained in $H(e)(T(e))$.

In an ordinary flow network the minimal saturated set containing a saturated arc e is simply $\{e\}$, and since repetitions of arcs are not allowed, (4.2) does not permit any forward arc to be saturated. Thus, in this specialization our definition almost exactly coincides with the accepted notion of an augmenting path, the only (inconsequential) difference being that we permit repetitions of nodes.

We shall want to use augmenting paths in the customary way. That is, for some strictly positive δ, we want to increase the flow through each forward arc by δ and decrease the flow through each backward arc by δ, and thereby obtain an augmented flow which is feasible. It is not readily apparent that this can be done in our generalization.

LEMMA 4.1. *For any augmenting path there exists a strictly positive value of δ by which the flow can be augmented.*

PROOF. There are two types of constraints on δ. First, the flow through each backward arc must remain nonnegative, and (4.1) assures us that there is a strictly positive value of δ for which this is possible. Second, for each node j and each $X \subseteq A_j$ (and similarly for each $X \subseteq B_j$) the resulting flow f' must be such that

$$f'(X) \leqslant \alpha_j(X).$$

Let $\mu(X)$ denote the number of forward arcs in X minus the number of backward arcs. Then we must have

$$f'(X) = f(X) + \delta\mu(X) \leqslant \alpha_j(X). \tag{4.3}$$

The only way in which (4.3) could fail to permit δ to be strictly positive would be for X to be saturated by f and for $\mu(X)$ to be strictly positive. But if X is saturated and contains forward arcs $e_1, e_2, \ldots, e_l$, then the tails of these forward arcs are saturated and $T(e_i) \subseteq X$, $i = 1, \ldots, l$. By (4.2), each e_i must be paired with a distinct backward arc $e_i' \in T(e_i)$. It follows that $\mu(X) \leqslant 0$ if X is saturated, and the constraints (4.3) permit δ to be strictly positive. ∎

5. Maximal flow algorithm. Augmenting paths can be found by means of a labeling procedure which is much like that employed for ordinary flow networks. The principal difference is that labels are applied to arcs instead of nodes. We present below an algorithm for computing maximum-value flows in which labeling is carried out in a "breadth-first" manner so that when an augmenting path is found it contains as few arcs as possible. We do not need shortest augmenting paths for our present purposes, but such paths will be useful later on.

The label applied to each arc has three components. The first component is either "+" or "−" indicating whether the arc is forward or backward. The second component is an arc name or index, for use in backtracing to find an augmenting path. The third component is a *level number* used to carry out labeling in a breadth-first manner. (The level of an arc is its position in a shortest path from s.) An arc is said to be *at* level l if it is labeled and its level number is l.

Maximal Flow Algorithm

Step 0. (Initialize flow.) Let f be any feasible flow, possibly the zero flow.

Step 1. (Label arcs at level 1.) To each nonvoid arc directed into s apply the label $(-, *, 1)$ and to each arc directed out from s whose tail is unsaturated apply the label $(+, *, 1)$. (No other arcs are labeled and no arcs are scanned.)

Set l to zero.

Go to Step 3.

Step 2. (Label arcs at level $l + 1$.) As long as there remains a labeled arc at level l which is unscanned, choose such an arc e and scan it as follows:

(2.1) If e has a "+" label and its head is saturated, then apply the label $(-, e, l + 1)$ to all unlabeled arcs in $H(e)$.

(2.2) If e has a "−" label and its tail is saturated, then apply the label $(+, e, l + 1)$ to all unlabeled arcs e' such that $e \in T(e')$.

(2.3) If e has a "+" label, its head is unsaturated, and e is directed into node j, or if e has a "−" label and e is directed out from node j, then apply the label $(+, e, l + 1)$ to all unlabeled arcs directed out from j whose tails are unsaturated and the label $(-, e, l + 1)$ to all unlabeled nonvoid arcs directed into j.

Step 3. (Check for augmenting path.) If there is an arc e labeled "−" which is directed out from t or there is an arc labeled "+" which is directed into t and whose head is unsaturated, go to Step 5. (An augmenting path has been found.)

Step 4. (Check for maximality of flow.) If there is no arc at level $l + 1$, then stop. (There is no augmenting path and the flow is maximal.) Otherwise set l to $l + 1$ and return to Step 2.

Step 5. (Augment flow.) Starting at arc e, identified in Step 3, find an augmenting path by backtracing. (If e has the label $(+, e', l + 1)$ then e' is the second-to-last arc in the path, the label on arc e' indicates the third-to-last arc, and so on.) Determine the maximum amount δ by which the flow can be augmented. Augment the flow to obtain a new feasible flow f and return to Step 1. (We defer until later a discussion of how to find δ.)

6. Augmenting path theorem. We asserted in Step 4 of the maximal flow algorithm that if the labeling procedure fails to find an augmenting path, then no augmenting path exists. This fact is by no means evident. The alert reader may even suspect that the labeling procedure may be defective, in that it permits a given arc to be given only one type of label ("+" or "−"), whereas both types might be applicable. We shall now prove that if the procedure fails to find an augmenting path then not only is there no augmenting path, but the flow is in fact maximal.

THEOREM 6.1 (AUGMENTING PATH THEROEM). *A flow is maximal if and only if it admits no augmenting path.*

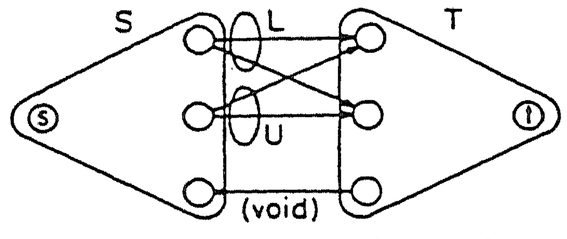


FIGURE 1. Cut (S, T) with sets of forward arcs L, U.

PROOF. If there is an augmenting path then Lemma 4.1 shows that the flow cannot be maximal. So suppose that the labeling procedure fails to find an augmenting path and let us show that this implies that the flow is maximal. The discussion which follows is with reference to the labels existing at the termination of the procedure.

Let us partition the nodes of the network into two sets, S and T. S is to contain node s, together with all nodes i such that either there is an arc directed from i with a "$-$" label or there is an arc directed into i with a "$+$" label whose head is unsaturated. All other nodes (including necessarily t) are in the set T.

We have thus defined a cut (S, T). Each "backward" arc (i, j), where $i \in T, j \in S$, must be void, else it would have received a "$-$" label and i would be in S. Let us partition the "forward" arcs (i, j), where $i \in S, j \in T$ into two sets U and L. Set U is to contain all unlabeled forward arcs and L is to contain all forward arcs which are labeled (either "$+$" or "$-$"). We thus have the situation indicated in Figure 1.

Consider any node $i \in S$ and any arc $e \in U \cap A_i$. The tail of e is saturated, else a "$+$" label would have been applied to e, either in Step 1, if $i = s$, or in Step (2.3), if $i \neq s$. Moreover, we assert that $T(e) \subseteq U \cap A_i$. Suppose there were some arc $e' \in T(e)$, with $e' \notin U$. The following cases exhaust all possibilities, if e' is not in U:

(i) e' is unlabeled and directed into a node $j \in S$. This is not possible, because a "$-$" label would have been applied to e' in Step 1, if $j = s$, or in Step (2.3), if $j \neq s$, when some arc incident to j was scanned. (Note that e' is nonvoid, by Lemma 2.3.)

(ii) e' has a "$-$" label. This is not possible, because scanning e' would apply a "$+$" label to e in Step (2.2).

(iii) e' has a "$+$" label. If so, e' could only have received its label in Step (2.2) when an arc $e'' \in T(e')$ was scanned. But $e'' \in T(e)$, else we would have $e' \in T(e) \cap T(e') \not\subseteq T(e')$, in contradiction to Lemma 2.2. But if $e'' \in T(e)$, then e would have received a "$+$" label in Step (2.2) when e'' was scanned. Hence this case cannot occur.

Since $T(e) \subseteq U \cap A_i$, for each $e \in U \cap A_i$, $U \cap A_i$ is the union of saturated sets and is itself a saturated set.

Now consider any node $j \in T$ and the set $L \cap B_j$. If any arc $e \in L \cap B_j$ has a "$-$" label, then this label could only have been applied in Step (2.1) when some arc e' with a "$+$" label was scanned and $e \in H(e')$. It follows that if $H(e) \subseteq L \cap B_j$, for each arc $e \in B_j$ with a "$+$" label, then $L \cap B_j$ is the union of saturated sets. So suppose $e \in B_j \cap L$, e has a "$+$" label, and there is an arc $e' \in H(e)$, with $e' \notin L$. The following cases exhaust all possibilities, if e' is not in L:

(i) e' is unlabeled. This is not possible, because a "$-$" label would have been applied to e' in Step (2.1) when e was scanned.

(ii) e' has a "$-$" label and is directed from a node $i \in T$. This is not possible, because $i \in S$ if e' is labeled "$-$".

(iii) e' has a "$+$" label and is directed from a node $i \in T$. The tail of e' is saturated, else e' was labeled in Step (2.3) with $i \in S$. Since the tail of e' is saturated, it could only have received its label in Step (2.2) when some arc $e'' \in T(e')$ with a "$-$" label was scanned. But then $i \in S$. Hence this case cannot occur.

It follows that $L \cap B_j$ is the union of saturated sets and is itself a saturated set.

We have shown that $f(U \cap A_i) = \alpha_i(U \cap A_i)$, for each $i \in S$, and $f(L \cap B_j) = \beta_j(L \cap B_j)$, for each $j \in T$. Since each arc (i, j), with $i \in T, j \in S$, is void, it follows

that the net flow across the cut is

$$f(U) + f(L) = \sum_{i \in S} f(U \cap A_i) + \sum_{j \in T} f(L \cap B_j)$$
$$= \sum_{i \in S} \alpha_i(U \cap A_i) + \sum_{j \in T} \beta_j(L \cap B_j).$$

The flow is therefore maximal and there can be no augmenting path. ∎

7. Max-flow min-cut theorem. The proof of Theorem 6.1 clearly indicates the form of a max-flow min-cut theorem for polymatroidal network flows, which we now proceed to state.

An *arc-partitioned cut* (S, T, U, L) is defined by a partition of the nodes into two sets S and T, with $s \in S$, $t \in T$, and by a partition of the forward arcs across the cut into two sets U and L. The *capacity* of such an arc-partitioned cut is defined as

$$C(S, T, U, L) = \sum_{i \in S} \alpha_i(U \cap A_i) + \sum_{j \in T} \beta_j(L \cap B_j).$$

As in the case of ordinary flow networks, the value v of any feasible flow f is equal to the net flow across any cut, i.e.,

$$v = f(U) + f(L) - f(B),$$

where B is the set of backward arcs, and clearly

$$v \leqslant c(S, T, U, L). \tag{6.1}$$

THEOREM 7.1 (MAX-FLOW MIN-CUT THEOREM). *The maximum value of a flow is equal to the minimum capacity of an arc-partitioned cut.*

PROOF. Let f be a maximal flow. (Such a flow clearly exists, since the flow problem is a linear programming problem with a nonempty bounded set of feasible solutions.) Apply the labeling procedure of the maximal flow algorithm and construct an arc-partitioned cut, as in the proof of Theorem 6.1. The capacity of this cut is equal to the value v of the flow f. The theorem now follows from (6.1). ∎

8. Shortcut-free augmenting paths. An augmenting path can be *shortcut* if some portion of it can be removed to obtain a shorter augmenting path. For example, suppose $P = (e_1, \ldots, e_p)$ contains a forward arc e_i and a backward arc e_k, with $i + 1 < k$ and $e_k \in H(e_i)$. Then removing arcs $e_{i+1}, \ldots, e_{k-1}$ from P yields a shorter augmenting path $P' = (e_1, \ldots, e_i, e_k, \ldots, e_p)$. The labeling procedure of the maximal flow algorithm finds shortest augmenting paths and these are certainly shortcut-free.

In the next section we show that the *capacity* of a shortcut-free augmenting path, i.e., the maximum amount δ by which the flow can be augmented along it, can be determined by computing the "capacities" of successive pairs of arcs in the path. The integrality theorem for polymatroidal flows then follows immediately. In order to obtain these results, we first provide a few more definitions.

An ordered pair of arcs $(e, \bar{e})$ is said to be *admissible* with respect to a given flow f if, when e is scanned in the labeling procedure, $\bar{e}$ is eligible to receive a label from e. More specifically, admissible pairs are of three kinds, corresponding to the ways in which e' might be labeled in Steps (2.1)–(2.3) of the maximal flow algorithm:

(i) A pair $(e, \bar{e})$ is a *saturated head pair* at node j if both e and $\bar{e}$ are directed into j, the heads of e and $\bar{e}$ are saturated and $\bar{e} \in H(e)$.

(ii) A pair $(e, \bar{e})$ is a *saturated tail pair* at node j if both e and $\bar{e}$ are directed from j, the tails of e and $\bar{e}$ are saturated and $e \in T(\bar{e})$.

(iii) a pair $(e, \bar{e})$ is an *unsaturated pair* at node j if either:

(a) e and $\bar{e}$ are both directed into j, the head of e is unsaturated and $\bar{e}$ is nonvoid;

(b) e and $\bar{e}$ are both directed from j, e is nonvoid and the tail of $\bar{e}$ is unsaturated;

(c) e is directed into j, $\bar{e}$ is directed from j, and the head of e and the tail of $\bar{e}$ are unsaturated;

(d) e is directed from j, $\bar{e}$ is directed into j and both e and $\bar{e}$ are nonvoid.

Let $P = (e_1, \ldots, e_p)$ be an augmenting path with respect to f. Clearly each pair of arcs (e_i, e_{i+1}), $i = 1, \ldots, p-1$, is admissible with respect to f. In order for arcs e_1 and e_p each to be contained in two admissible pairs in the path (one pair for the head and one for the tail of each arc), we introduce two *virtual* arcs * and **, incident with nodes s and t, respectively. Hereafter when we refer to the consecutive pairs of arcs in P we mean by convention to include the pairs $(*, e_1)$ and $(e_p, **)$ which are considered to be unsaturated. This convention enables us to avoid the statement of exceptions and special cases.

LEMMA 8.1. *Let P be a shortcut-free augmenting path. With reference to any node j and the consecutive pairs of arcs in P:*

(i) *P contains at most one unsaturated pair at j.*

(ii) *All unsaturated head pairs at j precede the unsaturated pair, in any, followed by all saturated tail pairs.*

(iii) *If P contains saturated head pairs $(e_1, \bar{e}_1), \ldots, (e_k, \bar{e}_k)$ at node j, in that order, then the sets*

$$H(e_1) \cup \ldots \cup H(e_i), \qquad 1 \leqslant i \leqslant k,$$

remain saturated after augmentation of the flow along P.

(iv) *If P contains saturated tails pairs $(e_1, \bar{e}_1), \ldots, (e_k, \bar{e}_k)$ at node j, in that order, then the sets*

$$T(\bar{e}_i) \cup \ldots \cup T(\bar{e}_k), \qquad 1 \leqslant i \leqslant k,$$

remain saturated after augmentation of the flow along P.

PROOF. (i) Suppose P were to contain two unsaturated pairs $(a, \bar{a})$, $(b, \bar{b})$ in that order. Then $(a, \bar{b})$ would be admissible and P would have a shortcut.

(ii) Suppose an unsaturated pair $(a, \bar{a})$ were to precede a saturated head pair $(b, \bar{b})$. Then $(a, \bar{b})$ would be admissible and P would have a shortcut. Similar analyses of other cases completes the proof of (ii).

(iii) If a saturated head pair $(a, \bar{a})$ is followed either by another saturated head pair $(b, \bar{b})$ or by an unsaturated pair $(b, \bar{b})$ in P, then $\bar{b} \notin H(a)$, else P would have a shortcut. It follows that for each i, $1 \leqslant i \leqslant k$,

$$\mu(H(e_1) \cup \ldots \cup H(e_i)) > 0,$$

where μ is defined as in the proof of Lemma 4.1. Therefore the set $H(e_1) \cup \ldots \cup H(e_i)$ must remain saturated after augmentation.

(iv) Proof is similar to that for (iii). ∎

9. Augmenting path capacity and the integrality theorem. With respect to a given flow f, let us define the value $\delta(e, \bar{e})$ for an admissible arc pair $(e, \bar{e})$ at node j as follows.

$$\delta(e, \bar{e}) = \min\{\delta_1(e, \bar{e}), \delta_2(e, \bar{e})\},$$

where

$$\delta_1(e,\bar{e}) = \begin{cases} f(e) & \text{if } e \in A_j, \\ \min\{\beta_j(X) - f(X) | e \in X \subseteq B_j - \{\bar{e}\}\} & \text{if } e \in B_j. \end{cases}$$

$$\delta_2(e,\bar{e}) = \begin{cases} \min\{\alpha_j(X) - f(X) | \bar{e} \in X \subseteq B_j - \{e\}\} & \text{if } \bar{e} \in A_j, \\ f(\bar{e}) & \text{if } \bar{e} \in B_j. \end{cases}$$

By definition, $\delta_1(*,\bar{e}) = \delta_2(e,**) = +\infty$.

Let us define $\delta(P)$ to be the minimum of the values $\delta(e_i, e_{i+1})$ over all consecutive pairs of arcs (e_i, e_{i+1}), $i = 0, \ldots, p$, in an augmenting path $P = (e_1, \ldots, e_p)$. (Here $e_0 = *$, $e_{p+1} = **$, as before.) An arc pair (e_i, e_{i+1}) is said to be *critical* if $\delta(P) = \delta(e_i, e_{i+1})$.

THEOREM 9.1. *If P is a shortcut-free augmenting path with respect to flow f, the maximum possible augmentation of the flow along P is $\delta(P)$. Furthermore, after augmentation of the flow by $\delta(P)$, each critical pair in P is inadmissible with respect to the augmented flow.*

PROOF. Let δ be the maximum amount of augmentation possible along P. We first show that $\delta \geqslant \delta(P)$.

If δ is determined by the flow through a backward arc in P, then clearly $\delta \geqslant \delta(P)$, from the definition of $\delta(P)$. So suppose δ is determined by an inequality of the form

$$f'(X) = f(X) + \delta\mu(X) \leqslant \alpha_j(X),$$

where $\mu(X)$ is defined as in the proof of Lemma 4.1. By that lemma, $\delta > 0$, hence we can assume that X is unsaturated by f and that $\mu(X) \geqslant 1$. Hence P contains at least one consecutive pair of arcs $(e,\bar{e})$ such that $\bar{e}$ is a forward arc, $\bar{e} \in X \cap A_j$ and $e \notin X$. Let $(e,\bar{e})$ be the first such arc pair in P. Let $(e_1,\bar{e}_1), \ldots, (e_k,\bar{e}_k)$ be the saturated tail pairs at node j in P. If the tail of $\bar{e}$ is unsaturated, let

$$X' = X \cup T(\bar{e}_1) \cup \ldots \cup T(\bar{e}_k), \tag{9.1}$$

and if the tail of $\bar{e}$ is saturated, let

$$X' = (X \cap T(\bar{e}_i)) \cup T(\bar{e}_{i+1}) \cup \ldots \cup T(\bar{e}_k), \tag{9.2}$$

where $\bar{e}_i = \bar{e}$. By assumption, X is saturated by f', hence X' is also saturated, by Lemmas 2.2 and 8.1. X' has been constructed to have exactly one forward arc which is not paired with a backward arc, so $\mu(X') = 1$. It follows that

$$f'(X') = f(X') + \delta = \alpha_j(X').$$

Because $\bar{e} \in X' \subseteq A_j - \{e\}$, we have

$$\delta = \alpha_j(X') - f(X') \geqslant \delta(e,\bar{e}) \geqslant \delta(P).$$

We shall now show that augmentation by an amount $\delta(P)$ renders each critical pair inadmissible with respect to the augmented flow. (Consequently $\delta \leqslant \delta(P)$.) Let $(e,\bar{e})$ be a critical arc pair at node j. If e or $\bar{e}$ is a backward arc and $\delta(e,\bar{e}) = f(e)$ or $\delta(e,\bar{e}) = f(\bar{e})$, then $(e,\bar{e})$ obviously becomes inadmissible. So suppose $\bar{e}$ is a forward arc and

$$\delta(e,\bar{e}) = \alpha_j(X) - f(X),$$

form some X such that $\bar{e} \in X \subseteq A_j - \{e\}$. If $\mu(X) \geqslant 1$ we are done. So suppose $\mu(X) \leqslant 0$. From the definition of $\delta(e,\bar{e})$ it must be the case that $\bar{e}$ is a forward arc in P. Now construct the set X' as in (9.1) or (9.2), depending upon whether or not the tail of

$\bar{e}$ is unsaturated. By applying Lemma 2.1, we see that

$$\alpha_j(X') - f(X') \leqslant \alpha_j(X) - f(X) = \delta(\dot{e}, \bar{e}) = \delta(P).$$

Hence augmentation by an amount $\delta(P)$ renders $(e, \bar{e})$ inadmissible. ∎

From the definition of $\delta(P)$, it is evident that if the capacity functions and the flow f are integer-valued, then $\delta(P)$ is also a positive integer. The theorem below then follows immediately.

THEOREM 9.2 (INTEGRALITY THEOREM). *If all capacity functions are integer-valued, then there is a maximal flow f which is integral. Moreover, f can be obtained by a finite number of augmentations along shortcut-free augmenting paths, beginning with the zero flow.*

10. The splicing lemma. We now prove a technical lemma needed in the following section. For the purpose of this lemma, (e, e), for any arc e, is by definition an admissible pair with respect to any flow. (Note also that if e is directed from i to j, there are actually two such admissible pairs, one at i and one at j.)

LEMMA 10.1 (SPLICING LEMMA). *Suppose flow f is augmented along a shortcut-free path P to obtain flow f'. If $(e, \bar{e})$ is an admissible pair with respect to f' but not with respect to f, then P contains two arcs x and $\bar{x}$, with x preceding $\bar{x}$ (and possibly with $x = \bar{e}$ or $\bar{x} = e$), such that $(e, \bar{x})$ and $(x, \bar{e})$ are admissible pairs with respect to f.*

PROOF.

Case 1. $(e, \bar{e})$ is a saturated head pair at j with respect to f'. In this case, $\bar{e} \neq **$ and $F'(\bar{e}) > 0$.

(i) If the head of e is unsaturated by f, then $f(\bar{e}) = 0$, else $(e, \bar{e})$ would be admissible with respect to f. Because $f'(\bar{e}) > 0$, $\bar{e}$ is a forward arc in P. Let $\bar{x}$ be the arc immediately following $\bar{e}$. Then $(e, \bar{x})$ and $(x, \bar{e})$, where $x = \bar{e}$, are admissible pairs with respect to f.

(ii) If the head of e is saturated by f and $f(\bar{e}) > 0$, then $\bar{e} \notin H(e)$, else $(e, \bar{e})$ would be admissible with respect to f. (Here and below $H(e)$ and $H'(e)$ denote head sets with respect to f and f'.) But $H(e)$ is not saturated by f', else we would have $H'(e) \subseteq H(e)$ and $\bar{e} \notin H'(e)$. It follows that P contains at least one backward arc $\bar{x}$, where $\bar{x} \in H(e)$. Let $\bar{x}$ be the last such arc in P and let x be the immediately preceding arc.

(iii) If the head of e is saturated by f and $f(\bar{e}) = 0$, then $\bar{e}$ must be a forward arc on P, and P must contain a backwards arc $\bar{x}$, where $\bar{x} \in H(e)$. As in (ii) let $\bar{x}$ be the last such arc in P. If $\bar{e}$ does not precede $\bar{x}$ then the set X constructed in (ii) is saturated by f' and $\bar{e} \notin X$. Thus $(e, \bar{e})$ would not be admissible with respect to f'. Therefore $\bar{e}$ precedes $\bar{x}$ and the result follows.

If $(x, \bar{x})$ is either an unsaturated head pair or a saturated head pair with $\bar{e} \in H(x)$ then we are done; $(e, \bar{x})$ and $(x, \bar{e})$ are admissible with respect to f. So suppose $(x, \bar{x})$ is a saturated head pair with $\bar{e} \notin H(x)$. By Lemma 8.1, P contains no unsaturated pairs at the same node, prior to $(x, \bar{x})$. Let $(x_1, \bar{x}_1), \ldots, (x_i, \bar{x}_i)$ be the saturated head pairs in P, up to and including $(x, \bar{x})$. By Lemmas 2.2 and 8.1, $X = H(e) \cup H(x_1) \cup \ldots \cup H(x_i)$ is saturated by f. By construction, $\mu(X) \geqslant 0$ so X remains saturated by f'. But $e \in X$, hence $H'(e) \subseteq X$ and $\bar{e} \in X$. Thus for some x_i, $1 \leqslant i \leqslant k - 1$, $\bar{e} \in H(x_i)$. Now choose x to be x_i and $(x, \bar{e})$ is admissible with respect to f.

Case 2. $(e, \bar{e})$ is a saturated tail pair with respect to f'. The proof for this case is similar to that for Case 1.

Case 3. $(e, \bar{e})$ is an unsaturated pair with respect to f'. We consider the case that $(e, \bar{e})$ is an unsaturated pair at a node into which both e and $\bar{e}$ are directed; the proofs for other cases are similar. If the head of e is unsaturated by f or if $f(\bar{e}) = 0$, then the argument in Case 1 (i) applies. So suppose the head of e is saturated by f and $f(\bar{e}) > 0$.

Then there is a backward arc $\bar{x}$ in P such that $\bar{x} \in H(e)$ (possibly $\bar{x} = e$). Let x be the arc immediately preceding $\bar{x}$ in P. Clearly $(x, \bar{e})$ and $(e, \bar{x})$ are admissible pairs with respect to f. ∎

11. Bounding the number of augmentations. We now wish to obtain a polynomial bound on the number of augmentations required to compute a maximal flow, using the splicing lemma as a tool. This requires a bit more notation. With respect to a given feasible flow f, let $\sigma_j(e)$ denote the number of (nonvirtual) arcs in a shortest path from node s to node j, such that each successive pair of arcs in the path is admissible with respect to f and e is the last arc in the path (either $e \in A_j$ or $e \in B_j$). Similarly, let $\tau_j(e)$ denote the length of a shortest path from node j to node t which begins with arc e. Recall that by convention each augmenting path starts with virtual arc * and ends with **. Let $\sigma_s(*) = \tau_t(**) = 0$.

LEMMA 11.1. *Suppose flow f is augmented along a shortest augmenting path P to obtain flow f'. For each node j and arc e,*

$$\sigma_j(e) \leqslant \sigma'_j(e), \qquad \tau_j(e) \leqslant \tau'_j(e),$$

where σ', τ' and σ, τ denote path lengths with respect to f' and f.

PROOF. Assume that for some j and e,

$$\sigma'_j(\bar{e}) < \sigma_j(\bar{e})$$

and let

$$\sigma'_j(\bar{e}) = \min_{i,x} \{\sigma'_i(x) | \sigma'_i(x) < \sigma_i(x)\}.$$

There is some arc e preceding $\bar{e}$ in a shortest path from s to j with respect to f' (else $\bar{e} = *, j = s$ and we would have $\sigma'_s(*) = 0 < \sigma_s(*) = 0$). Let k be the node to which both e and $\bar{e}$ are incident. By assumption, $\sigma'_k(e) \geqslant \sigma_k(e)$, so $(e, \bar{e})$ is not admissible with respect to f (else $\sigma'_j(e) \geqslant \sigma_j(e)$). Since $(e, \bar{e})$ is admissible with respect to f', we know by the splicing lemma that P contains arcs x, $\bar{x}$ such that $(e, \bar{x})$ and $(x, \bar{e})$ are admissible pairs at node k. It must be that $\sigma_k(x) \leqslant \sigma_k(e)$, else P would not be a shortest augmenting path with respect to f. But then

$$\sigma_j(\bar{e}) \leqslant \sigma_k(x) + 1 \leqslant \sigma_k(e) + 1 \leqslant \sigma'_k(e) + 1 = \sigma'_j(\bar{e}),$$

which contradicts our assumption that $\sigma_j(\bar{e}) > \sigma'_j(\bar{e})$. The proof that $\tau_j(e) \leqslant \tau'_j(e)$ is similar. ∎

COROLLARY 11.2. *If augmentations are made along shortest augmenting paths, then the number of arcs in successive augmenting paths is nondecreasing.*

PROOF. The length of a shortest augmenting path is $\min_e \{\tau_s(e)\}$. Apply Lemma 11.1 and note that

$$\min_e \{\tau'_s(e)\} \geqslant \min_e \{\tau_s(e)\}. \qquad ∎$$

Suppose we carry out successive augmentations along shortest augmenting paths. The augmentations made along paths with the same number of arcs will be said to constitute a *phase* of the maximal flow computation. There can be no more than m phases, where m is the number of arcs in the network, since no path can contain more than m arcs.

In order to bound the number of augmentations within a phase we introduce a lexicographic ordering to break ties between shortest paths. The arcs are indexed arbitrarily. Given two paths P and P' with the same number of arcs, the rule to determine if P is *lexicographically smaller* than P', written $P \lessdot P'$, is as follows. If the

index of the last arc in P is smaller than the index of the last arc in P', then $P \leqslant P'$. If these arcs are the same, then compare the indices of the next-to-last arcs, and so on. If all arcs are the same, then $P = P'$ and $P' \leqslant P$.

It is easy to modify the maximal flow algorithm so as to insure that each augmentation is made along a lexicographically minimal shortest path. Modify Step 2 so that the arcs at level l are scanned in order of increasing index. Also modify the algorithm so that the arc e identified in Step 3 (as the last arc of an augmenting path) has the smallest possible index. These are the only changes necessary.

LEMMA 11.3. *Suppose flow f is augmented along a lexicographically minimal shortest augmenting path P to obtain flow f' and that $\sigma_j(e) = \sigma_j'(e)$, for some given node j and arc e. If Q, Q' are lexicographically minimal paths realizing the values $\sigma_j(e)$, $\sigma_j'(e)$ respectively, then $Q \leqslant Q'$.*

PROOF. Assume that for some j and $\bar{e}$, $\sigma_j(\bar{e}) = \sigma_j'(\bar{e})$ and that $Q \not\leqslant Q'$. Further assume, without loss of generality, that the next-to-last arcs in Q and Q' are different, these being e and e', respectively. Since $Q \not\leqslant Q'$ the index of e is greater than the index of e', therefore $(e', \bar{e})$ is not an admissible pair with respect to f. Since $(e', \bar{e})$ is admissible with respect to f' but not f, by the splicing lemma P contains a pair $(x, \bar{x})$ such that $(x, \bar{e})$, and $(e', \bar{x})$ are admissible with respect of f. The index of e' is greater than or equal to the index of x, else P is not a lexicographically minimal path. Hence the index of e is greater than the index of x, by our previous observation. But the index of e is less than or equal to the index of x, else Q is not lexicographically minimal. This contradicts our assumption that $Q \not\leqslant Q'$. ∎

LEMMA 11.4. *If augmentations are made along lexicographically minimal shortest augmenting paths, then an arc pair $(e, \bar{e})$ which is critical in a path P cannot appear in any later path P' in the same phase of the computation.*

PROOF. By contradiction. Assume that a path P with respect to flow f contains a critical pair $(e, \bar{e})$ which is contained in a later path P'' with respect to flow f'', in the same phase. Let k be the node to which both e and $\bar{e}$ are incident and let j be the other node to which $\bar{e}$ is incident. We know that $(e, \bar{e})$ is not admissible with respect f', the flow existing after augmentation along P. The lexicographically minimal shortest path Q' realizing $\sigma_j'(\bar{e})$ contains some arc x as its next-to-last arc, with index $x >$ index e. But we must also have index $e >$ index x, else either P'' or Q' is not lexicographically minimal. This yields the desired contradiction. ∎

THEOREM 11.5. *If augmentations are made along lexicographically minimal shortest augmenting paths, then a maximal flow is achieved with at most m^3 augmentations, where m is the number of arcs in the network.*

PROOF. There are at most m^2 possible arc pairs. At least one pair is critical in each augmenting path and cannot be contained in a later augmenting path in the same phase. Hence there are at most m^2 augmentations in each phase. There are at most m phases and so at most m^3 augmentations in all. ∎

This result can be compared with that of Edmonds and Karp [3] for the classical network flow model. In that case there are at most n phases, where n is the number of nodes, and at most m augmentations per phase.

12. Complexity of the maximal flow algorithm. The maximal flow algorithm requires that certain computations be carried out with respect to the capacity functions and a given feasible flow f. Until this point, we have not discussed how these computations can and should be carried out. There are two types of problems we should like to be able to solve. In general polymatroidal terms these are as follows:

(i) *The saturation problem*. Given a polymatroid (E, ρ), a feasible function f and an element $e \in E$, is e saturated by f?

(ii) *The δ problem*. Given a polymatroid (E, ρ), a feasible function f and an element $e \in E$, what is the maximum value δ such that f' is feasible, where $f'(e) = f(e) + \delta$ and $f'(e') = f(e')$ for $e' \neq e$?

We must be able to solve the saturation problem in order to perform scanning and labeling in Steps 1 and 2 of the algorithm. Notice that any procedure solving the saturation problem in time c can be used to find $S(e)$, the unique minimal saturated set containing e, in time $c|E|$. Simply set $f(e') = 0$, for each $e' \neq e$, one at a time. Then $e' \in S(e)$ if and only if e becomes unsaturated.

If we can solve the δ problem for each capacity function α_j or β_j, then we can compute $\delta(e_i, e_{i+1})$ for each consecutive pair of arcs in an augmenting path $P = (e_1, \ldots, e_p)$, as required to determine the path capacity $\delta(P)$ in Step 5. For example, suppose (e_i, e_{i+1}) is a saturated head pair at node j. Then we can solve the δ problem for the polymatroid (B_j, β_j), $e_i \in B_j$, and $\bar{f}$, where $\bar{f}(e_{i+1}) = 0$ and $\bar{f}(e) = f(e)$ for $e \neq e_{i+1}$. The smaller of this value of δ and $f(e_{i+1})$ is then $\delta(e_i, e_{i+1})$. Notice also that any procedure solving the δ problem also solves the saturation problem. (An element $e \in E$ is saturated if and only if $\delta = 0$.)

If ρ is arbitrary and defined only by an explicit listing of values, i.e., $\rho(A)$ for each $A \subseteq E$, we cannot expect to be able to solve these problems efficiently, unless possibly $|E|$ is small. However, if ρ has some special structure, we may be more fortunate. For example, if ρ is a function whose value is determined by the cardinality of the set $A \subseteq E$. In this case there are numbers $s_1 \geqslant s_2 \geqslant \cdots > s_{|E|} \geqslant 0$ such that

$$\rho(A) = \sum_{i=1}^{|A|} s_i .$$

Both the saturation problem and the δ problem can be solved in $O(|E|\log|E|)$ time, with a sort of the values $f(e)$. This situation arises in the solution of the scheduling problem dealt with in [11], [12].

In order to be able to state a general bound on the complexity of the maximal flow algorithm, we shall suppose there are black-box subroutines (or "oracles") to which we can pass on much of the burden of the computation. For example, we might suppose that there is a subroutine, operating in time d, to solve the δ problem for each polymatroid (A_j, α_j), (B_j, β_j), $j = 1, \ldots, n$. Then we have the following result.

THEOREM 12.1. *Suppose for each capacity function there is a subroutine for solving the δ problem in time d. Then a maximal flow can be computed in time $O(m^5 d)$, where m is the number of arcs in the network.*

PROOF. At most m^3 augmentations are needed. Each augmentation requires that each of the m arcs be scanned at most once. The scanning of an arc e requires that the saturation problem be solved, in time d, and (in the worst case) that $H(e)$ or $T(e)$ be found, in time $d|A_j|$ or $d|B_j|$. Scanning and labeling can thus be seen to require at most $O(m^2 d)$ time per augmentation. Determining $\delta(P)$ for each augmenting path P requires at most $O(md)$ time. The time for this, and all other operations, is dominated by the time required for scanning and labeling. ∎

In the case of matroid optimization algorithms, it is common to assume the existence of a subroutine which determines whether or not a given subset of elments is independent in a matroid. A natural generalization of such a subroutine is one which tests a given function for feasibility with respect to ρ. Suppose there exists such a subroutine and that it runs in time c. Let us also suppose that f and ρ are integer-valued. We can then proceed as follows.

To solve the saturation problem, let f' be such that $f'(e) = f(e) + 1$ and $f'(e') = f(e')$,

$e' \neq e$, and apply the feasibility test to f'. Element e is unsaturated if and only if f' is feasible. To solve the δ problem, carry out a bisection search on the values of δ in the interval $[f(e), \rho(e)]$. For each trial value of δ, test whether f' is feasible, where $f'(e) = f(e) + \delta$, $f'(e') = f(e)$, for $e' \neq e$. $O(c \log_2 \rho(e))$ time is thus sufficient to solve the δ problem. (Moreover, with proper modification of the procedure, this bound can be achieved, without prior knowledge of $\rho(e)$.) This yields the following theorem.

THEOREM 12.2. *Suppose for each capacity function there is a subroutine for testing the feasibility of a flow in time c. If all capacity functions are integer-valued, then a maximal flow can be computed in time* $O(m^4 c(m + \log r))$, *where r is the maximum value of any capacity function for any single arc.*

PROOF. By analysis similar to that in the proof of Theorem 12.1, given the observations preceding this theorem. ∎

13. Concluding remarks. The polymatroidal network flow model as formulated here was suggested by research of one of the authors on a machine scheduling problem [11], [12]. The authors wish to acknowledge that the same network flow model (generalized to accommodate lower bounds and costs on arc flows) was independently formulated in the doctoral thesis of Rafael Hassin [6], supervised by Alan Hoffman. In the thesis Hassin uses a different approach to prove a generalized max-flow min-cut theorem and to show the existence of an integral optimal solution. However, for the most part, Hassin's thesis concerns rather different questions than those dealt with in the present paper and we believe there is very little overlap. The polymatroidal network flow model also bears at least some similarity to a model of Edmonds and Giles [4] in which submodular set functions are assigned to nodes instead of arcs. However, there appears to be very little, if any, relationship between the network flow model studied in this paper and the notion of "flows" in matroids (except that both provide generalizations of the classical network flow model), cf. Welsh [14].

References

[1] Aigner, M. and Dowling, T. A. (1971). Matching Theory for Combinatorial Geometries. *Trans. Amer. Math. Soc.* **158** 231–245.

[2] Edmonds, J. (1965). Minimum Partition of a Matroid into Independent Subsets, *J. Res. Nat. Bur. Standards* **69B** 67–72.

[3] ——— and Karp, R. M. (1972). Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. Assoc. Comput. Mach.* **19** 248–264.

[4] ——— and Giles, R. (1977). A Min-Max Relation for Submodular Functions on Graphs. In *Studies in Integer Programming* (Proc. Workshop on Programming Bonn, 1975) P. L. Hammer, E. L. Johnson and B. H. Korte, eds., *Annals Discrete Math.* **1** 185–204.

[5] Fujishige, S. (1978). Algorithms for Solving the Independent Flow Problem. *J. Oper. Res. Soc. Japan* **21** 189–204.

[6] Hassin, R. (1978). On Network Flows. Ph.D. dissertation, Yale University.

[7] Iri, M. and Tomizawa, N. (1976). An Algorithm for Finding an Optimal Independent Assignment Set. *J. Oper. Res. Soc. Japan* **19** 32–57.

[8] Krogdahl, S. (1975). A Combinatorial Proof of Lawler's Matroid Intersection Algorithm. Unpublished.

[9] Lawler, E. L. (1975). Matroid Intersection Algorithms. *Math. Programming* **9** 31–56.

[10] ———. (1976). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston.

[11] Martel, C. U. (1980). Generalized Network Flows with an Application to Multiprocessor Scheduling. Ph.D. thesis, University of California, Berkeley.

[12] ———. (to appear). Scheduling Uniform Machines with Release Times, Deadlines and Due Times. *J. Assoc. Comput. Mach.*

[13] Schönsleben, P. (1980). Gazzahlige Polymatroid–Intersektions–Algorithmen. Thesis, ETH, Zürich.

[14] Welsh, D. J. A. (1976). *Matroid Theory*. Academic Press.

LAWLER: ELECTRONICS RESEARCH LABORATORY, COLLEGE OF ENGINEERING, UNIVERSITY OF CALIFORNIA, BERKELEY, CALIFORNIA 94720

MARTEL: DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UNIVERSITY OF CALIFORNIA, DAVIS, CALIFORNIA 95616

AT PLAY IN THE FIELDS OF SCHEDULING THEORY

E.L. Lawler
University of California, Berkeley

J.K. Lenstra
Mathematisch Centrum, Amsterdam

A.H.G. Rinnooy Kan
Erasmus University, Rotterdam

Machine scheduling theory is something of a jungle, encompassing a bewilderingly large variety of problem types, as the most cursory examination of the journals reveals. It is also a marvelous playground for the algorithm designer and the complexity analyst, in that every known trick of combinatorial optimization can be applied somewhere, to one problem or another. This is an account of our explorations of this jungle-playground. Not incidentally, we shall describe a computer program we have used to help us guide our way. We conclude with some speculations about how similar, possibly more sophisticated, programs could be useful aids for researchers in other fields.

When we began our collaboration several years ago, we decided to focus our attention on *machine* scheduling problems. This meant that we excluded from consideration such worthy topics as project scheduling, timetabling, and cyclic scheduling of manpower. We also decided to concentrate on strictly *deterministic* models. Even so, this left us with an enormous number of problem types to study.

Very early on in our investigations, we decided we needed a uniform system of classification for the problems which had appeared in the literature. Starting from the classification scheme of Conway, Maxwell and Miller [1], after much debate we settled on a scheme which suited our purposes. This classification system is detailed elsewhere [4], and for present purposes can be summarized as encompassing *machine environment* (single machine, parallel machines, open shop, flow shop, job shop), *job characteristics* (independent vs. precedence constrained, etc.), and *optimality criterion* (makespan, flowtime, maximum lateness, total tardiness, etc.).

An immediate payoff was the consummate ease with which we could communicate problem types. Visitors to our offices were sometimes baffled to hear exchanges such as: "Since $1|r_j|\Sigma C_j$ is NP-hard, does that imply that $1|pmtn,r_j|\Sigma C_j$ is NP-hard too?" "No, that's easy, remember?" "Well, $1|d_j|\Sigma C_j$ is easy and that implies $1|pmtn,d_j|\Sigma C_j$ is easy, so what do we know about $1|pmtn,r_j,d_j|\Sigma C_j$?" "Nothing."

As this discussion indicates, one of our objectives was to demark as clearly as possible the boundary line between *easy* problems (solvable in polynomial time) and *NP-hard* problems. But because of the huge number of problem types and the relationships between them, it was easy to become confused. One could spend an hour trying to determine the status of a particular problem, only to realize that the issue had already been resolved - the problem was a generalization of a known NP-hard problem and therefore NP-hard as well, or a specialization of a known easy problem and therefore easy as well.

The idea of using the computer as an aid began as a joke. The afternoon of September 22, 1975, Dick Karp, Ben Lageweg, Gene Lawler and Jan Karel Lenstra met in the Mathematical Center in Amsterdam to decide on a gift to present to Alexander Rinnooy Kan on the occasion of his upcoming promotion to doctorate. Somebody made the amusing suggestion of a bound volume consisting of a computer tabulation of all the thousands of problem types with a notation for the status of each one: * for easy, ! for NP-hard, and ? for unresolved.

We were well aware that the problems in our classification system admitted of a natural partial ordering. Job shops are more general than flow shops. Precedence constrained problems are more general than

OPTIMA, Mathematical Programming Society Newsletter, Number 7, June 1982, pp. 1-3.
Reprinted by permission.

problems with independent jobs. Maximum lateness is a more general optimality criterion than makespan. And so on. All that was required to produce the tabulation was to feed the computer all results in the form of known easy problems and known NP-hard problems (ignoring results that were clearly dominated by others), let the computer take account of the partial ordering, and let it churn out a properly annotated listing.

That afternoon at the Math Center, the group speculated on what the score would be: how many *'s, !'s and ?'s would the tabulation contain? A playful attempt was made to obtain an estimate by generating a few random chains in the partial order, with everyone testing his expertise to see how far up a chain he could prove easiness and how far down NP-hardness. (Lenstra has since made the generation of random chains part of one of his stock lectures, with a member of the audience throwing a die.)

The next inevitable suggestion someone made was: "Why not have the computer list the maximal easy and minimal hard problems, and the minimal and maximal open ones as well? Wouldn't that give us a clearer picture of the situation?"

A suitable program was forthwith written by Lageweg and an initial run was made. The results were startling, for the number of easily resolvable cases it revealed in the listings of minimal and maximal open problems. During the next few weeks Lageweg, Lawler and Lenstra knocked off many targets of opportunity. The number of question marks in the tabulation was considerably smaller when, on January 28, 1976, a handsomely bound volume was presented to Alexander [5].

During the past six years there have been many developments, and Alexander's volume is now thoroughly outdated. The most impressive progress has been made in the area of preemptive scheduling of parallel machines. An elegant algorithm due to Gonzalez and Sahni (for $Q|pmtn|C_{max}$, the problem of minimizing makespan in preemptive scheduling of uniform parallel machines) [3] spawned a whole host of derivative algorithms for related problems.

At the present time, the score for 4,536 problem types stands at 81% NP-hard, 9% easy and 10% open [7]. This particular split is an artifact of our classification system, but it is certainly true that several subareas have been pretty well cleaned up. For example, the status of almost all single machine problems is known. Though open problems are still occasionally resolved, it is safe to say that nearly all the cream has been skimmed.

The problems which remain are mostly rather difficult. It is possible that they are neither NP-hard nor easy, provided that $P \neq NP$ (which we believe). One of the frustrations of the theory is that there is no way of proving such a result at present. For those who might care to accept a challenge, we mention two classic open problems:
(1) $P3|prec,p_j = 1|C_{max}$, the problem of minimizing makespan for unit-time jobs subject to arbitrary precedence constraints on three identical parallel machines: known to be easy for two machines and NP-hard for an arbitrary number of machines.
(2) $1||\Sigma T_j$, the problem of minimizing total tardiness on a single machine: known to admit of a pseudopolynomial algorithm, hence not NP-hard in the strong sense (unless $P = NP$). Is this problem easy or is it NP-hard in the ordinary sense?

A few words about the significance of NP-completeness theory are in order. It is not always true that polynomial algorithms are *good* and that problems that admit of such algorithms are *easy* to solve in practice. It is perhaps even less true that NP-hard problems are invariably *hard* in a practical sense. Yet there is enough correspondence with reality to make the notions of *easy* and *NP-hard* more than a polite fiction. Some NP-hard problems are *really hard* to solve. For example, no one has yet solved to optimality a certain notorious 10-machine 10-job job shop problem, small as this problem instance is. (The best published solution has a makespan of 972 [8]. Lageweg has found a solution of 935. The best known lower bound is 865.)

The primary usefulness of the concept of NP-hardness is the direction it gives to the algorithm designer. With knowledge that a problem is NP-hard, he can abandon any attempt to reformulate the problem as, say, a simple network flow problem or a graphic matching problem. Instead he can concentrate his energies on developing an efficient enumerative optimization method or a well-behaving approximation algorithm. It is in this way that NP-completeness theory has probably had more impact on combinatorial optimization than any other theoretical development of the past ten years. We were pleased to observe, during the NATO Advanced Study and Research Institute on Deterministic and Stochastic Scheduling (Durham, England, July 1981) [2], that the methodology carries over to computational questions about stochastic scheduling as well.

One of our hopes when we began our project was that we might be able to determine the boundary line between easy and NP-hard problems sufficiently closely that we could gain meaningful insight into the properties that make a scheduling problem of one type or the other. This we have been able to do to some extent. When dealing with a practical scheduling problem (which is invariably NP-hard), we found it increasingly easy to detect the particular features of the problem which were responsible for its computational intractability, since they would correspond to the crucial ingredients of an NP-hardness proof. These features then suggested certain relaxations that should be made to obtain lower bounds for a branch-and-bound procedure, or directions that could be taken in designing a heuristic.

Now to return to a discussion of the computer program and the benefits we have received from it. First, the program has provided an orderly form of record keeping for research results. Confusions and oversights have been greatly reduced. Second, the program has helped us focus our research. Listings of minimal and maximal open problems have made it easy to choose the most interesting and important ones to work on. And finally, the automatic scorekeeping has been motivational and introduced a healthy competition into our work.

A frivolous idea which occurred to us was that the computer might be programmed to produce another type of score, namely the minimum number of open problems whose resolution would resolve all remaining open problems. Alas, we found that the calculation of this score is itself an NP-hard problem [6]. We have made no attempt to devise an algorithm for its solution.

We believe that computer programs similar to ours could be applied equally well to other well-structured areas of knowledge and research. Certainly allied areas of combinatorial optimization such as location theory and, more ambitiously, algorithmic graph theory are candidates. Even the broad area of mathematical programming might be susceptible, as well as inventory theory, queueing theory, or even organic chemistry.

It would not be difficult to create a sort of automated encyclopedia. Given such a system for the field of mathematical programming, the user could make queries of the form: "What is known about a problem with such-and-such objective function and so-and-so constraints?" The system might answer: "Nothing has been reported on this specific problem, but these results have been obtained for more general and more special cases. Moreover, the following computer codes are available . . ." The program would be knowledgeable of problem relationships which might be unknown to the user, even if their usefulness would be contingent on future theoretical developments. For example, it would know that maximization of a posynomial in bivalent variables is equivalent to the min-cut problem of network flow theory.

There are other types of question-answering facilities it would be useful to have. For example, for a book on scheduling theory we are writing, we should like to state a few simple rules that will enable the reader to comprehend the status of large

subclasses of problems. It would be nice to be able to verify these rules by asking the system questions of the form: "Are there easy problems involving the nonpreemptive scheduling of parallel machines which do not have the objective of minimizing flowtime?" or "Are there any problems which are known to be NP-hard when preemption is permitted but easy when it is not?"

At some future date it may be possible to have computers search for problem transformations themselves. At this time, such an undertaking appears to be beyond the capabilities of artificial intelligence. Should this development come to pass, the computer would truly be an automated research assistant.

REFERENCES

1. R.W. Conway, W.L. Maxwell, L.W. Miller (1967) *Theory of Scheduling*, Addison-Wesley, Reading, MA.
2. M.A.H. Dempster, J.K. Lenstra, A.H. G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht.
3. T. Gonzalez, S. Sahni (1978) Preemptive Scheduling of Uniform Processor Systems. *J. Assoc. Comput. Mach. 25*, 92-101.
4. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979) Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Ann. Discrete Math. 5*, 287-326.
5. B.J. Lageweg, E. L. Lawler, J.K. Lenstra (1976) Machine Scheduling Problems: Computations, Complexity and Classification. Report BN 30, Mathematisch Centrum, Amsterdam (out of print).
6. B.J. Lageweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1981) Computer Aided Complexity Classification of Combinatorial Problems. Report BW 137, Mathematisch Centrum, Amsterdam.
7. B.J. Lageweg, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1981) Computer Aided Complexity Classification of Deterministic Scheduling Problems. Report BW 138, Mathematisch Centrum, Amsterdam.
8. G. McMahon, M. Florian (1975) On Scheduling With Ready Times and Due Dates to Minimize Maximum Lateness. *Oper. Res. 23*, 475-482.

Information Processing Letters 17 (1983) 73–76
North-Holland

24 August 1983

A FASTER ALGORITHM FOR FINDING EDGE-DISJOINT BRANCHINGS *

Po TONG and E.L. LAWLER
Computer Science Department, University of California, Berkeley, CA 94720, U.S.A.

Communicated by Michael A. Harrison
Received 26 April 1982
Revised 2 April 1983

An $O(k^2mn)$ algorithm is proposed for finding k edge-disjoint branchings in a directed multigraph with m edges and n vertices. With appropriate preprocessing the time bound can be reduced to $O(kmn + k^3n^2)$. Our proposed algorithm runs faster than any previously known algorithm and provides yet another constructive proof of Edmonds' Branchings Theorem.

Keywords: Algorithm, branching, directed multigraph, edge-connectivity

1. Introduction

In this paper $G = (V, E)$ is assumed to be a finite directed multigraph with at least two vertices. A subgraph T of G is a *tree rooted from r* if each vertex of T is reachable from r, each vertex of T other than r has indegree one, and r has indegree zero. T is a *branching* (rooted from r) if T spans the vertices of G, i.e., $V(T) = V$, where $V(T)$ is the set of vertices of T.

A *cut* of G determined by a set S, $\emptyset \neq S \subset V$, is the set of all edges extending from S to $V - S$. Let $\delta_G(S)$ denote the number of edges in the cut and define

$$c_G(r) = \min\{\delta_G(S) \mid r \in S \neq V\}.$$

Thus $c_G(r)$ is the minimum number of edges which can be deleted from G in order to make at least one vertex unreachable from r.

By Menger's Theorem, there exist at least $c_G(r)$ edge-disjoint paths from r to any vertex of G. Moreover, there is at least one vertex for which this is the maximum number of such paths. It follows that there can be no more than $c_G(r)$ edge-disjoint branchings in G. Edmonds' Branchings Theorem [1] states that there are indeed this many.

Theorem 1. *There exist* $c_G(r)$ *edge-disjoint branchings in* G.

Based on Edmonds' proof, Tarjan [2] presented an $O(k^2m^2)$ algorithm for constructing a maximum set of edge-disjoint branchings, where $k = c_G(r)$, $m = |E|$. Lovász [3] provided an elegant proof of the theorem which also suggested an algorithm with $O(k^2m^2)$ running time. In the present paper we describe an $O(k^2mn)$ algorithm, where $n = |V|$, and then show how with appropriate preprocessing the bound can be reduced to $O(kmn + k^3n^2)$. The algorithm provides yet another constructive proof of Edmonds' theorem.

In [4] an algorithm with an $O(k^2mn)$ time bound was also proposed. However, we believe that the algorithm is faulty; examples can be produced for which the algorithm yields an incorrect result.

2. A constructive proof

Let T be a tree rooted from r and let $G - T$ denote the directed multidigraph obtained by de-

* This research was partially supported by National Science Foundation Grant MCS78-20054.

leting the edges of T from G. By letting T be a tree containing the single vertex r, Theorem 2 below is seen to generalize Theorem 1. As in [1,3] we propose a constructive proof of this stronger version of Theorem 1.

Theorem 2. *If* $c_G(r) \geq k$ *and* $c_{G-T}(r) \geq k-1$, *then there exist* k *edge-disjoint branchings in* G, *one of which contains* T.

Proof. The theorem is trivially true for $k = 0$ and all n, and also for $n = 2$ and all k. So let $k \geq 1$ and $n \geq 3$ be given and let us assume the theorem is true for $k-1$ and n and also for k and $2, 3, \ldots, n-1$. We shall show that this will imply the theorem to be true for k and n. The theorem will then have been proved by double induction.

Our strategy will be to add one edge at a time to the tree, always checking to see if the enlarged tree maintains the property that $c_{G-T}(r) \geq k-1$. If we are able to turn T into a branching, then by the inductive hypothesis there are $k-1$ edge-disjoint branchings in $G-T$ and we are done. Thus we must consider the possibility that we try to add an edge e to T and discover that $c_{G-T-e}(r) = k-2$. (Note that $G-T-e$ is used to denote the multidigraph obtained by deleting the edges in T and also e.)

Note that if T is not a branching, then there is at least one edge from a vertex in T to a vertex not in T, else $c_G(r) = 0 < k$. Our rule for choosing such an edge e will be *to avoid the choice of an edge directed from* r, *if possible*.

Suppose $c_{G-T}(r) = k-1$ and $c_{G-T-e}(r) = k-2$, where e has been chosen according to our rule. We assert that for any set S, with $r \in S$, such that $\delta_{G-T-e}(S) = k-2$ we must have $|S| \geq 2$ and $|V-S| \geq 2$. First note that e must be in the cut determined by S, since $\delta_{G-T}(S) = k-1$. And since $\delta_G(S) \geq k$, there must be at least one edge e′ of T in the cut. Because e and e′ are directed to different vertices, we must have $|V-S| \geq 2$. If e is directed from r, then by our rule for the choice of e, there are no edges in $G-T$ directed from vertices other than r in T to vertices not in T. This means that

$$\delta_{G-T}(S) \geq \delta_{G-T}(S \cup V(T)) = \delta_G(S \cup V(T)) \geq k,$$

a contradiction. It follows that e is directed from a vertex other than r and $|S| \geq 2$.

We now apply a divide-and-conquer strategy. A multidigraph G_1 is obtained from G by condensing the vertices in S into a single vertex r′. The edges directed into r′ are then removed and the edges of T which remain form a tree T_1 rooted from r′. A multidigraph G_2 is formed from G by condensing the vertices in $V-S$ into a single vertex t. Self loops at t are removed. The edges of T which remain in G_2 do not necessarily form a tree rooted from r in G_2 because there may be more than one edge of T directed into t. (There must be at least one such edge.) If this is the case, we delete from G_2 all but one edge e′ of T directed into t, where e′ is directed from a vertex which can be reached from r by a path in T, all of whose vertices are in S. The edges of T which remain in G_2 form a tree T_2 rooted from r.

Clearly

$$c_{G_1}(r') \geq k, \qquad c_{G_1-T_1}(r') \geq k-1,$$

$$c_{G_2}(r) \geq k, \qquad c_{G_2-T_2}(r) \geq k-1.$$

Because $|S| \geq 2$, $|V-S| \geq 2$, G_1 and G_2 each contain no more than $n-1$ vertices. It follows, by inductive hypothesis, that G_1 and G_2 each have k edge-disjoint branchings, with T_i contained in one of the branchings in G_i, $i = 1, 2$. We assert that it is possible to combine these two sets of branchings so as to obtain the desired k edge-disjoint branchings for G. When we have established this assertion, we shall have completed the proof of the theorem.

At this point we consider an example in the form of the multidigraph G, with $c_G(r) = 3$, as shown in Fig. 1. Bold edges indicate a tree T rooted from r, with $c_{G-T}(r) = 2$. Suppose we try to

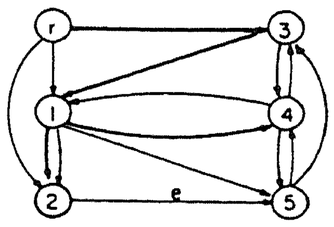


Fig. 1.

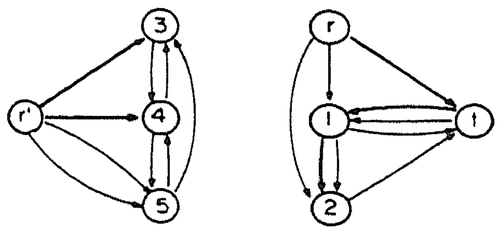


Fig. 2.

add edge e to T. Then $\delta_{G-T-e}(S) = 1$, with $S = \{r, 1, 2\}$. G_1 and G_2 are as shown in Fig. 2, with T_1 and T_2 indicated by bold edges. G_1 and G_2 each contain three edge-disjoint branchings, as indicated by bold, solid and dashed edges in Fig. 3. T_1 and T_2 are contained in the bold-edge branchings. Taking the union of the edges in the two bold-edge branchings and the union of the edges in the solid and dashed edge branchings, we obtain three edge-disjoint branchings, as indicated in Fig. 4.

Each branching from G_1 is edge-disjoint from exactly $k-1$ of the k branchings from G_2 and vice versa. We assert that a desired branching for G is obtained by combining each nondisjoint pair of branchings. The assertion is easily seen to be true for those pairs which do not contain T_1 and T_2. To prove the assertion for the other case, let B_1 and B_2 be branchings for G_1 and G_2 respectively such that $T_1 \subseteq B_1$ and $T_2 \subseteq B_2$. For G, B_1 is a forest (i.e., a union of vertex-disjoint rooted trees) whose roots are all covered by $T = T_1 \cup T_2 \subset B_1 \cup B_2$. For G, B_2 is a forest whose roots (except r) are all in $V - S$. Since B_1 and B_2 have exactly one edge in common, $B_1 \cup B_2$ contains $n-1$ edges. To show that $B_1 \cup B_2$ is a branching for G, it remains to prove that every vertex in G is reachable from r using edges in $B_1 \cup B_2$. This reachability property follows from the following three facts:

(1) Using edges in B_2, every vertex in S is reachable from either r or a root of B_2 in $V - S$.

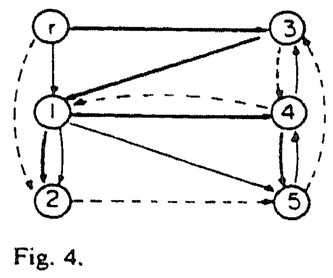


Fig. 4.

(2) Using edges in B_1, every vertex in $V - S$ is reachable from a root of B_1.

(3) Every root of B_1 is covered by T and hence is reachable from r using edges in $T \subset B_1 \cup B_2$.

We have shown that $B_1 \cup B_2$ is a branching for G such that $T \subset B_1 \cup B_2$. This finishes the proof of Theorem 2. □

3. The algorithm and its complexity

In order to determine the maximum number of edge-disjoint branchings in G, we compute $c_G(r)$. As indicated in [2,4], this can be done with $n-1$ network flow computations, in which r is taken as a source and each of the other $n-1$ vertices is separately taken as a sink. Each of the edges is given a flow capacity of one. The $n-1$ flow problems can be solved 'simultaneously' by first trying to find a flow of value one in each of them, then a flow of value two, and so forth, until one of the $n-1$ flows reaches its maximum value. As indicated in [4], the entire computation can be carried out in O(kmn) time (assuming $m \geq n$), where $k = c_G(r)$.

To find k edge-disjoint branchings in G, where $k \leq c_G(r)$, one can apply a recursive procedure clearly suggested by the constructive proof of Theorem 2. The inputs to this recursive procedure are G, T and k, where T is the tree with one vertex, r.

The procedure attempts to add one edge e at a time to T (always choosing an edge not directed from r, if possible), until T becomes a branching with $c_{G-T}(r) = k-1$ or $c_{G-T-e}(r) = k-2$. In the

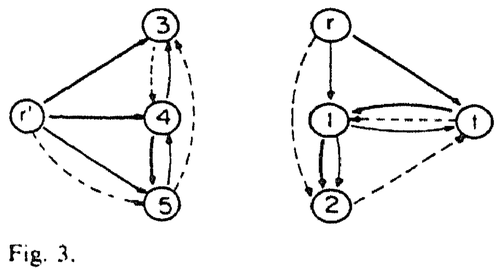


Fig. 3.

first case, the procedure outputs T as one of the k branchings and calls itself with inputs $G - T$, T', $k - 1$, where T' is any tree containing a single edge from r, in order to obtain the remaining $k - 1$ branchings. In the second case, the procedure forms G_1, G_2 and T_1, T_2, as indicated in the previous section, and then calls itself twice, once with inputs G_1, T_1, k, and once with inputs G_2, T_2, k. The k pairs of edge-disjoint branchings obtained are then combined to obtain k edge-disjoint branchings for G.

The computation of $c_{G-T-e}(r)$ can be accomplished with a single network flow computation by making r a source and the vertex v into which e is directed a sink. (Note that for any S such that $\delta_{G-T-e}(S) = k - 2$, it must be the case that v is in $V - S$.) Each augmentation requires O(m) time, so the computation of $c_{G-T-e}(r)$ requires O(km) time.

Let us now establish an upper bound on the total amount of time required for network flow computations by the recursive procedure.

Examining an edge e for possible inclusion in a tree takes O(km) time. Each such edge examined either is successfully added to a tree or causes a splitting into subproblems. Since only O(kn) edges are added to trees, the total time for successful additions is $O(k^2mn)$. Each subproblem contains at least three vertices (if $n \geq 3$). A split into subproblems decreases the quantity

$$\sum\{n_i - 3 \mid n_i \text{ is the number of vertices in subproblem } i\}$$

by at least one. Since this quantity is at most $n - 3$ and at least zero, there can be at most $n - 3$ splittings, giving a bound of O(kmn) for unsuccessful edge examinations. Thus the overall running time is $O(k^2mn)$.

4. Further refinements

With some additional processing, it is possible to reduce the number of edges in G to exactly $k(n - 1)$, while preserving the property that $c_G(r) = k$. Let the vertices of G be r, v_1, $v_2, \ldots, v_{n-1}$. Find a flow from r to v_1 of value k and then eliminate from G all those edges into v_1 with zero flow. After the elimination, for any $S \subset V$ with $r \in S$, if $v_1 \in S$, then clearly $\delta_G(S)$ remains unchanged; if $v_1 \notin S$, then since there is a flow of value k from r to v_1, so $\delta_G(S) \geq k$. Hence the property $c_G(r) = k$ is preserved after the elimination. Continue the elimination process for $v_2, v_3, \ldots, v_{n-1}$, until finally exactly k edges remain in each of the vertices v_i, $i = 1, 2, \ldots, n - 1$.

This procedure requires O(kmn) time and allows us to replace m by kn in the $O(k^2mn)$ bound obtained in the previous section. Thus the overall time bound can be reduced to $O(kmn + k^3n^2)$. Of course, any branching algorithm with an appearance of m in its running time can take advantage of this procedure.

We have implicitly assumed that each edge is represented by a distinct record. If the number of edges is large, which will necessarily be the case if k is large, it may be more reasonable to represent each set of parallel edges by a single record, with the number of edges specified as part of the record. Suppose this is the case, and that there is available a network flow algorithm with time bound p(m, n) (where now m is at most $O(n^2)$). Then the recursive procedure we have described can be implemented to run in O(knp(m, n)) time.

Acknowledgment

The authors wish to thank an anonymous referee for pointing out that their original proof of the running time of the algorithm was incorrect and for suggesting the proof that appears in Section 3.

References

[1] J. Edmonds, Edge-disjoint branchings, in: R. Rustin, ed., Combinatorial Algorithms (Algorithmics Press, New York, 1972) pp. 91–96.
[2] R. Tarjan, A good algorithm for edge-disjoint branchings, Inform. Process. Lett. 3 (1975) 51–53.
[3] L. Lovász, On two minimax theorems in graph theory, J. Combin. Theory Ser. B 21 (1976) 96–103.
[4] Y. Shiloach, Edge-disjoint branchings in directed multigraphs, Inform. Process Lett. 8 (1979) 24–27.

Preemptive Scheduling of Uniform Machines Subject to Release Dates

J. Labetoulle

Centre National d'Etudes des Télécommunications
Issy les Moulineaux, France

E.L. Lawler

University of California
Berkeley, U.S.A.

J.K. Lenstra

Centre for Mathematics and Computer Science
Amsterdam, The Netherlands

A.H.G. Rinnooy Kan

Erasmus University
Rotterdam, The Netherlands

We shall be concerned with finding optimal preemptive schedules on parallel machines, subject to release dates for the jobs. Two polynomial-time algorithms are presented. The first algorithm minimizes maximum completion time on an arbitrary number of uniform machines. The second algorithm minimizes maximum lateness with respect to due dates for the jobs on an arbitrary number of identical machines or on two uniform machines. A third algorithm for minimizing maximum lateness on an arbitrary number of uniform machines is briefly discussed. *NP*-hardness is established for the problem of minimizing total weighted completion time on a single machine.

1. Introduction

We consider scheduling problems in which n independent jobs $J_1, \ldots, J_n$ have to be processed on m parallel machines $M_1, \ldots, M_m$. Each machine can handle at most one job at a time and each job can be executed on at most one machine at a time. Each job J_j becomes available for processing at its release date r_j. It has an execution requirement p_j and possibly also a due date or deadline d_j and a

weight w_j. Unlimited preemption is allowed: the processing of any job may arbitrarily often be interrupted and resumed at the same time on a different machine or at a later time on any machine. The machines are assumed to be uniform, i.e., each machine M_i has a speed s_i, and complete execution of J_j on M_i would require p_j/s_i time units. If all speeds are equal, the machines are identical; if $m = 1$, we have a single machine. We assume that all numerical data r_j, p_j, d_j, w_j, s_i are integers.

A feasible schedule defines a completion time C_j and a lateness $L_j = C_j - d_j$ for each J_j. We may choose to minimize the maximum completion time $C_{max} = \max_{1 \le j \le n}\{C_j\}$, the maximum lateness $L_{max} = \max_{1 \le j \le n}\{L_j\}$, the total completion time $\sum C_j = \sum_{j=1}^{n} C_j$, or the total weighted completion time $\sum w_j C_j = \sum_{j=1}^{n} w_j C_j$.

When scheduling jobs subject to release dates, one can distinguish between three types of algorithms. An algorithm is on-line if at any time only information about the available jobs is required. It is nearly on-line if in addition the next release date has to be known. It is off-line if all information is available in advance.

In Section 2 we consider the minimization of C_{max} on m uniform machines. For the case that all release dates are equal, Horvath, Lam and Sethi [8] derived a closed form expression for the optimum value of C_{max}. Gonzalez and Sahni [6] proposed an $O(m \log m + n)$ algorithm which produces a schedule meeting this value and containing at most $2(m-1)$ preemptions. For the case that the release dates are arbitrary, Sahni and Cho [18] gave an $O(n \log n + mn)$ off-line algorithm to determine if there exists a schedule in which no job is completed after a common deadline. We will present an $O(n^2)$ nearly on-line algorithm to minimize C_{max}; Sahni and Cho [17] independently developed an $O(mn \log n + m^2 n)$ nearly on-line algorithm that is very similar to ours. We will indicate how to obtain an $O(n \log n + mn)$ off-line implementation of our algorithm. These methods can also be used to minimize L_{max} in the case of equal release dates.

In Section 3 we consider the minimization of L_{max}. For the case of equal release dates, Horn [7] proposed an $O(n^2)$ algorithm to minimize L_{max} on m identical machines. For the case of arbitrary release dates, he gave an off-line algorithm, based on a network flow computation, to determine if there exists a schedule in which no job is completed after its deadline. Bruno and Gonzalez [3] adapted this feasibility test to the case of two uniform machines. We will extend both methods by presenting polynomial-time algorithms to minimize L_{max}. Martel [13,14] recently proposed a feasibility test for the case of m uniform machines, based on a polymatroidal network flow model,

and used it to obtain a polynomial-time algorithm to minimize L_{max}. We will discuss these results as well.

In Section 4 we consider the minimization of $\sum C_j$ and $\sum w_jC_j$. For the case of equal release dates, Bruno and Gonzalez [5] proposed an $O(n \log n + mn)$ algorithm to minimize $\sum C_j$ on m uniform machines. It is well known that in the case of identical machines allowing preemptions will not decrease the optimal value of $\sum w_jC_j$ [15]. It follows that $\sum w_jC_j$ is minimized on a single machine by scheduling the jobs in order of nonincreasing ratios w_j/p_j [19], and that the problem on two identical machines is already *NP*-hard [2,12]. For the case of arbitrary release dates, $\sum C_j$ is minimized on a single machine by an obvious on-line extension of the above ordering rule [1]; we will establish *NP*-hardness for the problem of minimizing $\sum w_jC_j$.

In Section 5 we conclude by indicating a major open problem and some important recent developments in the area of preemptive scheduling.

2. Maximum Completion Time

We first consider the problem of minimizing the *maximum completion time* C_{max} on m uniform machines. The jobs and the machines are assumed to be ordered in such a way that $r_1 \le \cdots \le r_n$ and $s_1 \ge \cdots \ge s_m$.

We will describe a nearly on-line algorithm that considers the time intervals $R_k = [r_k, r_{k+1}]$ in order of increasing k. For each successive interval R_k $(k = 1,\ldots,n-1)$, denote the remaining execution requirement of J_j at r_k by $p_j^{(k)}$ $(j = 1,\ldots,k)$ and renumber the jobs so that $p_1^{(k)} \ge \cdots \ge p_k^{(k)}$. The subalgorithm to be applied in each interval determines the amounts by which the $p_j^{(k)}$ are to be decreased within R_k. At time r_n, all jobs are available, and it is well known [8] that the minimum time for their completion is given by

$$C^*_{max} = r_n + \max\{\max_{1 \le l \le m-1}\{\textstyle\sum_{j=1}^{l} p_j^{(n)} / \sum_{i=1}^{l} s_i\}, \quad (1)$$
$$\textstyle\sum_{j=1}^{n} p_j^{(n)} / \sum_{i=1}^{m} s_i\}.$$

The portion of an optimal schedule within any interval R_k can be constructed by applying the Gonzalez-Sahni algorithm [6] to the quantities $p_j^{(k)} - p_j^{(k+1)}$ determined by our subalgorithm. Similarly, a schedule for the final interval $[r_n, C^*_{max}]$ can be constructed by applying the same algorithm to the quantities $p_j^{(n)}$. Since both the subalgorithm and the schedule construction procedure require $O(n)$ time for each interval, the algorithm requires $O(n^2)$ time overall; it introduces $O(mn)$ preemptions into the optimal schedule.

Our algorithm has the property that the remaining execution requirements passed on to the next interval will be as *evenly* distributed as possible. More specifically, for each k there is no way to process the jobs before r_k that could lead to a smaller value for *any* of the partial sums $\sum_{j=1}^{l} p_j^{(k)}$ $(l = 1,\ldots,k)$. This immediately implies the correctness of the algorithm, since each of these partial sums appearing in (1) is as small as it could possibly be.

Rather than giving an inductive proof of this property, we will settle for a simpler correctness proof of the entire algorithm. This proof will also serve to introduce algorithmic refinements, by which the optimum value C^*_{max} can be determined in $O(n \log n + mn)$ time. An actual schedule can be constructed by applying the Sahni-Cho algorithm [18], using C^*_{max} as a common deadline for the jobs. This off-line approach requires $O(n \log n + mn)$ time and introduces $O(mn)$ preemptions into the optimal schedule.

Let us consider an interval R_k for fixed k. Given the $p_j^{(k)}$ $(j = 1,\ldots,k)$, we have to determine the $p_j^{(k+1)}$ to be passed on to the next interval R_{k+1}.

Suppose that at time r_k the jobs $J_1, \ldots, J_v$ are available and not yet completed, with $p_1^{(k)} \geq \cdots \geq p_v^{(k)} > 0$. For ease of notation, we drop the superscripts. Thus, denote the given $p_j^{(k)}$ by p_j and the unknown $p_j^{(k+1)}$ by q_j $(j = 1,\ldots,v)$, and let $t = r_{k+1} - r_k$. For purposes of exposition, we assume for the time being that, if $m < v$, machines $M_{m+1}, \ldots, M_v$ with $s_{m+1} = \cdots = s_v = 0$ are added to the model.

The p_j can be viewed as defining a staircase pattern as in Figure 1. The q_j will be chosen in such a way that they define a similar pattern.

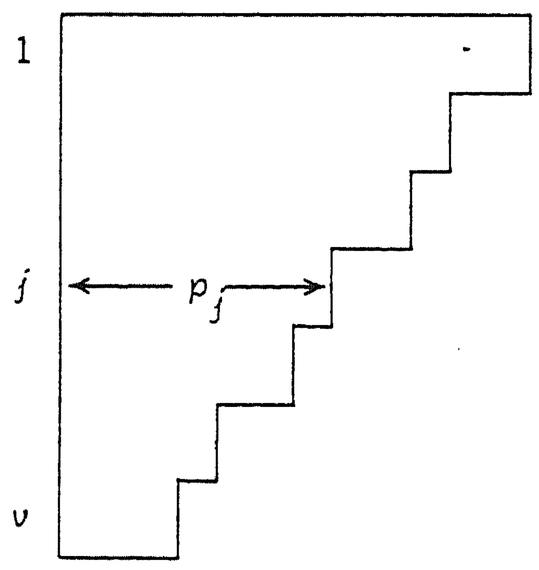


Figure 1. Staircase pattern at r_k.

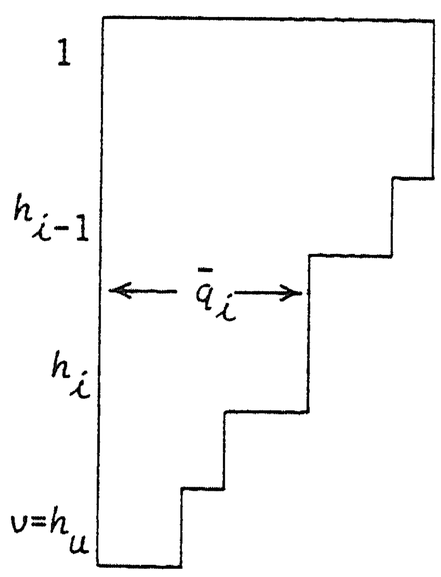


Figure 2. Staircase pattern at r_{k+1}.

As illustrated in Figure 2, such a staircase can be characterized by a

sequence $((h_1, \bar{q}_1),\ldots,(h_u, \bar{q}_u))$, where $\bar{q}_i = q_j$ for each J_j with $h_{i-1} + 1 \le j \le h_i$ $(i = 1,\ldots,u;\ h_0 = 0;\ h_u = v)$. A first condition for feasibility is that

$$\bar{q}_i > \bar{q}_{i+1} \quad (i = 1,\ldots,u-1). \tag{2}$$

The staircase $((h_1, \bar{q}_1),\ldots,(h_u, \bar{q}_u))$ will be constructed in such a way that, for $i = 1,\ldots,u-1$, the capacity of $M_{h_{i-1}+1}, \ldots, M_{h_i}$ will be fully utilized to decrease $p_{h_{i-1}+1}, \ldots, p_{h_i}$ to $\bar{q}_i$. A second condition for feasibility is therefore that

$$\sum_{j=h_{i-1}+1}^{l} q_j = (l-h_{i-1})\bar{q}_i \ge \sum_{j=h_{i-1}+1}^{l} p_j - t\sum_{j=h_{i-1}+1}^{l} s_j \tag{3}$$
$$(l = h_{i-1} + 1,\ldots,h_i;\ \ i = 1,\ldots,u),$$

with the corners of the staircase, except possibly the last one, corresponding to strict equalities:

$$\sum_{j=h_{i-1}+1}^{h_i} q_j = (h_i-h_{i-1})\bar{q}_i = \sum_{j=h_{i-1}+1}^{h_i} p_j - t\sum_{j=h_{i-1}+1}^{h_i} s_j \tag{4}$$
$$(i = 1,\ldots,u-1).$$

A third condition for feasibility is of course that

$$0 \le q_j \le p_j \quad (j = 1,\ldots,v). \tag{5}$$

We tentatively construct the first step of the staircase by setting

$$h_1 = 1,\ \bar{q}_1 = p_1 - ts_1.$$

Generally, having found i tentative steps $(h_1, \bar{q}_1),\ldots,(h_i, \bar{q}_i)$ with $h_i < v$ and $\bar{q}_1 > \cdots > \bar{q}_i$, we construct the $(i+1)$-st tentative step by setting

$$h_{i+1} = h_i+1,\ \bar{q}_{i+1} = p_{h_{i+1}} - ts_{h_{i+1}}. \tag{6}$$

If $\bar{q}_i > \bar{q}_{i+1}$ and $\bar{q}_i \ge 0$, the staircase $((h_1, \bar{q}_1),\ldots,(h_{i+1}, \bar{q}_{i+1}))$ satisfies (2) and (4); we increment i by one and, if h_i is still smaller than v, construct the next step.

Suppose now that $\bar{q}_i \le \bar{q}_{i+1}$ or $\bar{q}_i < 0$. In the latter situation, there is excess capacity on $M_{h_{i-1}+1},\ldots,M_{h_i}$; in both cases, some of the capacity of these machines has to be devoted to processing J_{h_i+1} if (2) and (4) are to be satisfied. We therefore reconstruct the i-th step so as to include J_{h_i+1} as well: h_i is incremented by one, and $\bar{q}_i$ is recalculated according to

$$\bar{q}_i = \Big(\sum_{j=h_{i-1}+1}^{h_i} p_j - t\sum_{j=h_{i-1}+1}^{h_i} s_j\Big)/(h_i-h_{i-1}) \tag{7}$$

(cf. (4)). As a result, it may now be that $\bar{q}_{i-1} \leq \bar{q}_i$ ($\bar{q}_{i-1} < 0$ cannot occur). In this case, we reconstruct the $(i-1)$-st step so as to include the current i-th step: h_{i-1} is increased to h_i, and $\bar{q}_{i-1}$ is recalculated as in (7). We continue until once more $\bar{q}_1 > \ldots > \bar{q}_i$; the adjusted staircase $((h_1, \bar{q}_1), \ldots, (h_i, \bar{q}_i))$ includes one more job and may have fewer steps than before. If h_i is still smaller than v, we construct the next step according to (6).

The process is terminated as soon as $h_i = v$. If $\bar{q}_i < 0$, we reset $\bar{q}_i = 0$ and note that only in this situation the last corner of the staircase does not correspond to a strict equality.

We have to verify that the resulting staircase $((h_1, \bar{q}_1), \ldots, (h_u, \bar{q}_u))$ and the corresponding remaining execution requirements $q_1, \ldots, q_v$ indeed satisfy the feasibility conditions (2) - (5). For (2) and (4), this is obvious. To see that (3) must be true, note that each $\bar{q}_i$ is initially defined by an equality constraint and can only increase thereafter. To verify (5), it is sufficient to show that $\bar{q}_i \leq p_{h_i}$. Subtracting (3) for $l = h_i - 1$ from (4), we find $\bar{q}_i \leq p_{h_i} - ts_{h_i}$, which implies the desired result.

Let us now analyze the running time of the subalgorithm. The number of step constructions as in (6) is exactly v. The number of step reconstructions as in (7) is at most $v-1$, since during each adjustment two steps are collapsed into one. It follows that the process terminates in $O(v)$ time. This presupposes that the given values p_j are ordered; but since the relative order of the remaining execution requirements does not change, we can maintain an ordered list of these values and insert the value of the job that becomes available at r_k in $O(v)$ time. Hence the subalgorithm determines the values q_j for each interval in $O(v)$ time. As has been indicated above, the Gonzalez-Sahni algorithm [6] can be applied to construct an actual schedule in each interval in $O(v)$ time as well. We thus have arrived at a nearly on-line algorithm that requires $O(n^2)$ time overall.

We now intend to prove the correctness of the algorithm.

We note first that not only does the relative order of the remaining execution requirements remain invariant, but also the following stronger property holds: as soon as two remaining execution requirements become equal, they will remain equal. To see this, suppose that $p_j = p_{j+1}$ at time r_k, and let $h_i = j$. According to (6), we set $\bar{q}_{i+1} = p_{j+1} - ts_{j+1}$. But $\bar{q}_i \leq p_j - ts_j \leq p_j - ts_{j+1} = \bar{q}_{i+1}$, and we have to reconstruct the i-th step so as to include J_{j+1} as well.

This leads us to define the rank of an available job J_j at time r_k as the value h_i for which $h_{i-1}+1 \leq j \leq h_i$. The rank of a job at time r_u is defined analogously as its step height that would be found if the

subalgorithm were to be applied in the interval $[r_n, C^*_{max}]$. A job will be called *critical* if its rank is at most $m-1$ and *noncritical* otherwise. The rank of a job cannot decrease; in particular, once a job becomes noncritical, it never becomes critical again. It follows from (4) that in any interval the fastest h_l machines are exclusively processing the longest h_l critical jobs. A critical job is processed continuously from its release date until it either is completed or becomes noncritical.

These observations suggest the following correctness proof for the algorithm. First, suppose that the schedule ends at C^*_{max} with the simultaneous completion of l critical jobs ($l < m$). At any time when l' of these jobs are available, they are processed by the fastest l' machines. In this case, the schedule is clearly optimal.

Alternatively, suppose that the schedule ends with the simultaneous completion of m noncritical jobs. Let r_k be the last release date just prior to which there is idle time on some machine. Ignoring the jobs that are available but noncritical at time r_{k-1}, we conclude that the portion of the schedule for the remaining jobs has a structure as illustrated in Figure 3.

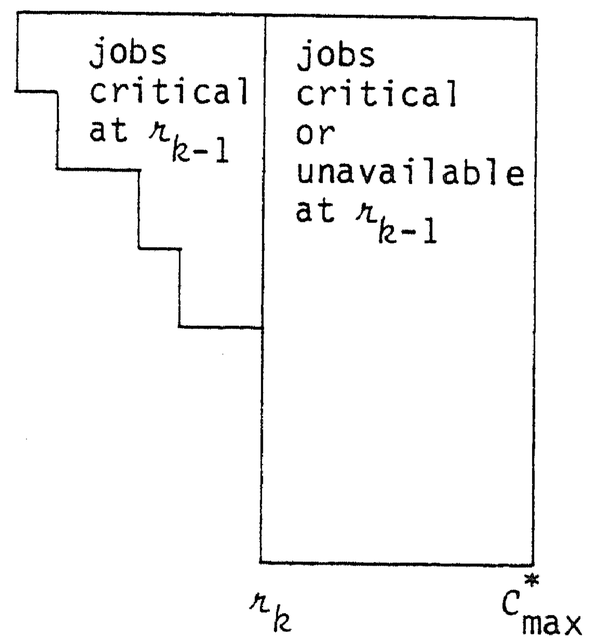


Figure 3 - Simultaneous completion of noncritical jobs.

Before r_k, the available critical jobs are processed by the fastest machines. Between r_k and C^*_{max}, there is no idle time. It follows that the schedule is optimal for the jobs under consideration and *a fortiori* that C^*_{max} is the minimum time to complete all the jobs.

Let us use the new terminology to describe a more efficient implementation of the subalgorithm. We will reduce the running time by dealing more carefully with the noncritical jobs, circumventing the need to introduce machines of speed zero.

Consider the situation after a typical application of the

subalgorithm, as illustrated in Figure 4.

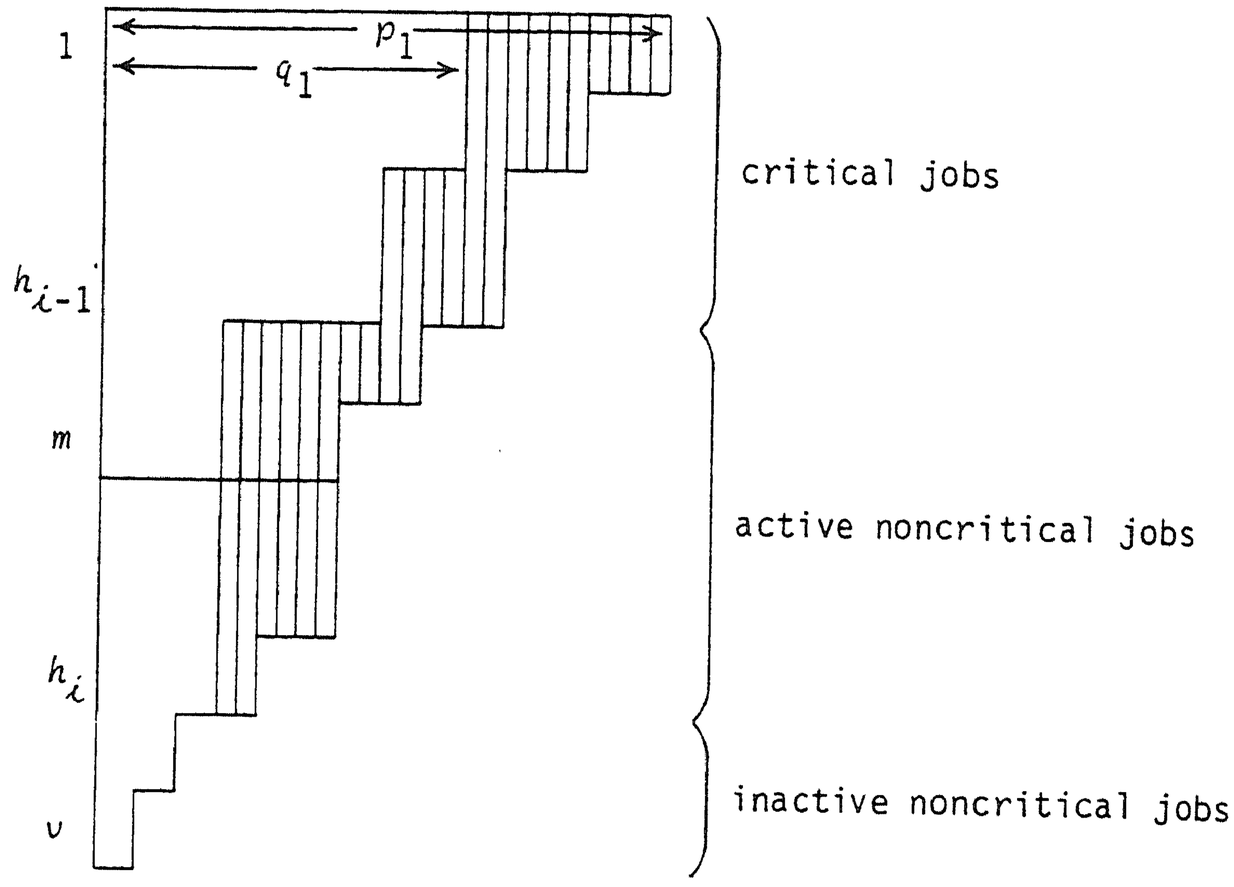


Figure 4 - Staircase patterns at r_k and r_{k+1}.

The noncritical jobs of lowest rank, i.e., $J_{h_{l-1}+1},\ldots,J_{h_l}$ where $h_{l-1}+1 \leq m \leq h_l$, will be called *active*. In the interval R_k, their remaining execution requirements are reduced by machines $M_{h_{l-1}+1}, \ldots, M_m$ to a common amount $\bar{q}_l$. The remaining noncritical jobs, i.e., $J_{h_l+1},\ldots,J_v$, will be called *inactive*. In R_k, their remaining execution requirements are not reduced at all, since $\bar{q}_l > \bar{q}_{l+1} = p_{h_{l+1}}$ (note that $s_{h_{l+1}} = 0$).

As a first refinement, the subalgorithm does not have to deal with the active noncritical jobs separately, since their remaining execution requirements will remain equal throughout. They can easily be handled simultaneously by straightforward generalizations of (6) and (7). As a second refinement, the subalgorithm can be terminated as soon as either $h_l = v$ or $h_l \geq m$ and $\bar{q}_l > p_{h_l+1}$.

Rather than maintaining an ordered list of all remaining execution requirements, we have to do so only for the largest $m-1$ of them. We simply record the number of active noncritical jobs, their common remaining execution requirement, and the lowest index of any of them. Finally, we maintain a priority queue for the remaining requirements of the inactive noncritical jobs.

At each release date, the execution requirement of the job that

becomes available is, depending on its size, inserted either in the ordered list in $O(m)$ time or in the priority queue in $O(\log n)$ time. The staircase computations for the longest $m-1$ jobs and the active noncritical jobs require $O(m)$ time in each interval and $O(mn)$ time overall. The queue operations require $O(\log n)$ time in each interval and $O(n \log n)$ time overall, since once an inactive job becomes active and is withdrawn from the queue, it remains active throughout. Hence successive applications of the modified subalgorithm determine the value C^*_{max} in $O(n \log n + mn)$ time. As has been indicated above, the Sahni-Cho algorithm [18] can be applied to construct an actual schedule in the interval $[r_1, C^*_{max}]$ in $O(n \log n + mn)$ time as well. We thus have arrived at an off-line algorithm that requires $O(n \log n + mn)$ time overall.

3. Maximum Lateness

We now consider the problem of minimizing the *maximum lateness* L_{max} on m identical machines.

A relaxed version of this problem is to test a *trial value* of L_{max} for feasibility. That is, for a given value y, one has to determine whether or not there exists a schedule for which $L_{max} \leq y$. This condition is equivalent to the requirement that no job J_j is completed after an *induced deadline* $d_j + y$. Sahni [16] proposed an off-line algorithm for the case of equal deadlines that requires $O(n \log mn)$ time and introduces at most $n-2$ preemptions. He also showed that there can be no nearly on-line algorithm for the case of arbitrary deadlines. Horn [7] proposed a network flow algorithm for the latter case. He suggested that one might conduct a search for the optimum value of L_{max}, but offered no upper bound on the number of trial values that have to be tested. Our contribution here is to obtain such a bound and to show that it is polynomial in the problem size.

Horn's approach is as follows. Suppose y is a trial value for L_{max}. Let $\{e_1, \ldots, e_{2n}\}$ $(e_1 \leq \cdots \leq e_{2n})$ be the ordered collection of release dates r_j and induced deadlines $d_j + y$; if a release date and a deadline are equal, the smaller index is to be assigned to the release date. Further, define the time interval $E_k = [e_k, e_{k+1}]$ for $k = 1, \ldots, 2n-1$.

A flow network is constructed with job vertices $J_1, \ldots, J_n$, interval vertices $E_1, \ldots, E_{2n-1}$, a source vertex S and a sink vertex T. There is an arc (J_j, E_k) of capacity $e_{k+1} - e_k$ if and only if $r_j \leq e_k$ and $e_{k+1} \leq d_j + y$. In addition, there is an arc (S, J_j) of capacity p_j for $j = 1, \ldots, n$ and an arc (E_k, T) of capacity $m(e_{k+1} - e_k)$ for $k = 1, \ldots, 2n-1$. Now, a maximum value flow is found in $O(n^3)$ time

[9]. It should be evident that the trial value y is feasible if and only if the maximum flow value is $P = \sum_{j=1}^{n} p_j$. If the maximum flow value is indeed P, a feasible schedule is easily constructed: for each interval E_t, read off the flows through the arcs (J_j, E_t) and apply McNaughton's schedule-construction procedure [15]. The resulting schedule contains at most $O(n^2)$ preemptions.

Notice that there are certain *critical* trial values of L_{max}. These are the n^2 values y such that $d_j + y = r_t$ for some pair J_j and J_t. The vertex-arc structure of the network remains unchanged for all trial values between two successive critical values.

We propose to find the optimum value of L_{max} in two phases. In the first phase, the largest infeasible critical value y_0 is determined. A bisection search for y_0 requires the testing of $\log_2 n^2 = O(\log n)$ trial values, or $O(n^3 \log n)$ time overall.

In the second phase, a maximum value flow and a minimum capacity cut are found in the network with capacities induced by the value y_0. Next, a value $y_1 > y_0$ is determined in such a way that the capacity of this cut is increased to exactly P. The procedure is then repeated in the network induced by y_1. This process yields a sequence of increasing trial values y_l. It terminates when the minimum cut capacity is exactly P, i.e., at an iteration z where y_z is the first feasible trial value and therefore the optimum value of L_{max}. We shall show that $z = O(n^2)$.

Suppose a minimum cut with capacity $P_0 < P$ is found in the network for y_0. Consider how the capacity of this cut is changed when y_0 is increased by some positive amount δ. The capacity $e_{t+1} - e_t$ of an arc (J_j, E_t)

(a) stays the same if e_t and e_{t+1} are both release dates or both deadlines;

(b) increases by δ if e_t is a release date and e_{t+1} is a deadline;

(c) decreases by δ if e_t is a deadline and e_{t+1} is a release date.

A similar situation holds for the capacities of the arcs (E_t, T), except that they change by $m\delta$ or $-m\delta$ rather than by δ or $-\delta$. All arcs whose capacities are increased are incident with a vertex E_t of type (b), of which there are at most n. If (E_t, T) is in the cut, then no (J_j, E_t) can be a forward arc in the cut, so that the cut capacity increase in all arcs incident with E_t is at most $n\delta$ (assuming $m \leq n$). It follows that the capacity of the cut is increased by $\mu_0\delta$, where μ_0 is an integer *multiplier* with $1 \leq \mu_0 \leq n^2$. Accordingly, we set $\delta = (P - P_0)/\mu_0$, $y_1 = y_0 + \delta$, and repeat.

Each cut in the network can be characterized by a pair (μ, P'),

where μ is its multiplier and P' its capacity. When y_l is increased to y_{l+1}, the multipliers of cuts do not change, although their capacities, of course, do. Suppose that the minimum cut found at iteration i has multiplier μ_l and capacity P_l, and consider the replacement of y_l by y_{l+1}. Each cut with multiplier $\mu \geq \mu_l$ will have its capacity increased to at least P. Hence $\mu_{l+1} < \mu_l$, unless $z = i + 1$. It follows that there can be at most $O(n^2)$ iterations.

Alternatively, note that the desired value $y_z = y_0 + \delta$, where δ is a ratio of the form $(P-P')/\mu$; here P, P' and μ are integers with $0 \leq P' < P$ and $1 \leq \mu \leq n^2$. A bisection search can determine the desired ratio $(P-P')/\mu$ with the testing of $O(\log n^4 P) = O(\log n + \log p_{max})$ values, where $p_{max} = \max_{1 \leq j \leq n}\{p_j\}$. Specifically, the search is conducted over the interval $(0, P)$ until the remaining interval of uncertainty (α, β) is no longer than $1/n^4$, since this is the smallest possible difference between two ratios. Given a minimum cut in the network for $y_0 + \alpha$, δ is chosen such that the capacity of this cut is exactly P. Then $y_z = y_0 + \delta$.

We have indicated two different means for minimizing maximum lateness in the second phase, one requiring $O(n^2)$ network flow computations and the other $O(\log n + \log p_{max})$. By running these procedures in parallel, we arrive at an algorithm that requires $O(n^3 \min\{n^2, \log n + \log p_{max}\})$ time overall.

Bruno and Gonzalez [3] showed that essentially the same feasibility test can be employed in the case of two uniform machines. Under the assumption that $s_1 \geq s_2$, each arc (J_j, E_k) has capacity $s_1(e_{k+1} - e_k)$ and each arc (E_k, T) has capacity $(s_1 + s_2)(e_{k+1} - e_k)$.

The first phase in our search for the optimum value of L_{max} is as before. In the second phase, the arc capacities change by $\pm s_1\delta$ and $\pm(s_1 + s_2)\delta$, instead of $\pm\delta$ and $\pm m\delta$. Since there are now $s_1 n^2$ possible values for the multiplier μ, the first bound on the number of iterations is $O(s_1 n^2)$. The second bound is $O(\log s_1^2 n^4 P) = O(\log s_1 + \log n + \log p_{max})$. Hence the algorithm requires $O(n^3 \min\{s_1 n^2, \log s_1 + \log n + \log p_{max}\}$ time overall.

Martel [13,14] recently developed an analogous approach to the minimal common generalization of the problems discussed so far, i.e., minimizing maximum lateness on m uniform machines.

To solve the feasibility problem with respect to induced deadlines, one constructs the same network as in the case of identical machines, with the flow through an arc (J_j, E_k) corresponding to the amount of processing of job J_j in interval E_k. The capacity constraints are to be modified as follows. Assume that $s_1 \geq \cdots \geq s_m$. For each subset

$\{J_j | j \in X\}$ and each vertex E_k, it is required that the total flow through the set of arcs $\{(J_j, E_k) | j \in X\}$ is no more than $\rho(k, |X|)$, where

$$\rho(k, l) = (e_{k+1} - e_k) \sum_{j=1}^{\min\{l,m\}} s_j.$$

Further, each arc (E_k, T) has capacity $\rho(k, m)$. This is a special case of the *polymatroidal* network flow model [11], in which the capacities are defined by nonnegative and submodular functions, one for each set of arcs entering (or leaving) a specific vertex; the model derives its name from the fact that such a set function corresponds to the rank function of a polymatroid. Traditional notions such as augmenting paths and labeling techniques can be extended to find a maximum value flow for the scheduling model in $O((m^2n^3 + n^4)(m + \log n))$ time [13,14].

To determine the optimum value of $L_{\max}$, one again uses the concept of critical trial values so as to arrive at a polynomial-time algorithm that requires $O(n^2 + n \log s_1 + \log(\max_j\{d_j\} + P))$ calls to the feasibility routine [14]. Admittedly, the degree of the polynomial is on the high side. By way of compensation, we should add that the investigation of polymatroidal network flow models was inspired by this scheduling problem and has yielded a useful generalization and unification of classical network flow theory and much of the theory of matroid optimization [11].

4. Total Weighted Completion Time

We finally consider the problem of minimizing the *total completion time* $\sum C_j$ or the *total weighted completion time* $\sum w_jC_j$.

Let us first assume that all release dates are equal. Bruno and Gonzalez [5] proposed a simple algorithm to minimize $\sum C_j$ on m uniform machines: order the jobs according to nondecreasing execution requirements, and schedule each successive job preemptively so as to minimize its completion time. This algorithm is illustrated in Figure 5. Obviously, it requires $O(n \log n + mn)$ time and introduces at most $(m-1)(n - \frac{m}{2})$ preemptions.

The Bruno-Gonzalez algorithm not only minimizes $\sum C_j$ but also $\sum_{j=1}^{l} C_j$ for $l = 1,\ldots,n-1$. Further, it minimizes $\sum w_jC_j$ provided that the weights are *agreeable*, i.e., $p_j < p_k$ implies $w_j \geq w_k$ [5].

A characteristic feature of the algorithm is that at each point in time the fastest machines are working on the jobs with the shortest remaining execution requirements. One may consider a straightforward extension to the case of arbitrary release dates, in which at each subsequent release date the above rule is applied to the available jobs.

$m = 3,\ s_1 = 3,\ s_2 = 2,\ s_3 = 1$

$n = 4,\ p_1 = 3,\ p_2 = 8,\ p_3 = 8,\ p_4 = 10$

optimal schedule obtained by Bruno-Gonzalez algorithm:

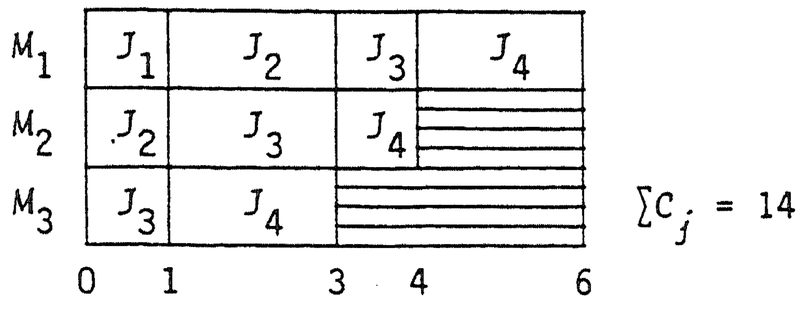


Figure 5 - Example with m uniform machines, all $r_j = 0$, $\sum C_j$ criterion.

In contrast to the algorithm described in Section 2, the resulting algorithm has the property that the remaining execution requirements passed on to the next interval will be as *unevenly* distributed as possible. Unfortunately, it may produce non-optimal schedules, as is illustrated in Figure 6. The example shows in fact that no on-line algorithm will be able to minimize $\sum C_j$ even on two identical machines.

For the case of a single machine, it has been pointed out in Section 1 that when all release dates are equal $\sum w_j C_j$ is minimized in $O(n \log n)$ time by scheduling the jobs in order of nonincreasing ratios w_j / p_j [19]. Again, an obvious extension to the case of arbitrary release dates is to apply the ratio rule at each release date to the remaining execution requirements of the available jobs. This on-line algorithm yields an optimal schedule when the weights are equal or agreeable [1]. Surprisingly [1, p. 82], the problem is *NP*-hard when the weights are arbitrary, as will be shown below.

This result will be obtained by a reduction from the following *NP*-complete problem [4]:

PARTITION: Given a set $T = \{1, \ldots, t\}$ and positive integers $a_1, \ldots, a_t, b$ with $\sum_{j \in T} a_j = 2b$, does there exist a subset $S \subset T$ such that $\sum_{j \in S} a_j = b$?

Given any instance of PARTITION, we define a corresponding instance of the problem of minimizing $\sum w_j C_j$ on a single machine subject to arbitrary release dates as follows:

$n = t + 1;$

$r_j = 0,\ p_j = w_j = a_j \quad (j \in T);$

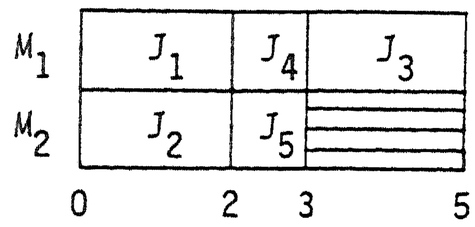
$r_n = b,\ p_n = 1, w_n = 2.$

$n = 5,\ r_1 = r_2 = r_3 = 0,\ r_4 = r_5 = r$

$p_1 = p_2 = p_3 = 2,\ p_4 = p_5 = 1$

(a) $r = 2$

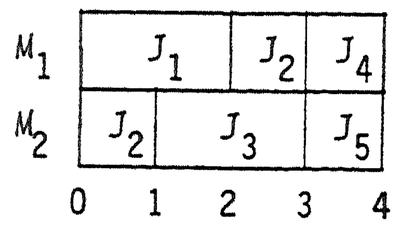
optimal schedule obtained by extended Bruno-Gonzalez algorithm:



$\sum C_j = 15$

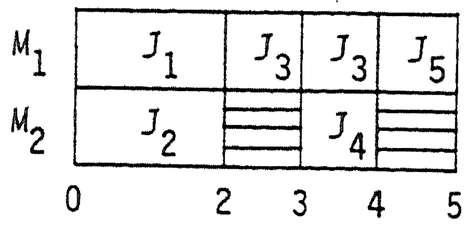
(b) $r = 3$

optimal schedule:



$\sum C_j = 16$

non-optimal schedule obtained by extended Bruno-Gonzalez algorithm:



$\sum C_j = 17$

Figure 6 - Example with two identical machines, $\sum C_j$ criterion.

We claim that PARTITION has a solution if and only if there exists a schedule with value $\sum w_j C_j \leq y$, where

$$y = \sum_{1 \leq j \leq k \leq t} a_j a_k + 3b + 2.$$

With respect to $\{J_j | j \in T\}$, any nonpreemptive schedule without machine idle time is optimal and has value $\sum_{1 \leq j \leq k \leq t} a_j a_k$. Inserting the unit-time job J_n in a schedule for $\{J_j | j \in T\}$ increases the contribution to $\sum w_j C_j$ of the latter set by the total weight of all jobs completed after J_n. Let us denote the index set of all jobs completed before J_n by S, the length of the interval from b until J_n starts by c $(c \geq 0)$, and the length of the interval from the last completion before J_n until J_n starts by d $(d \geq 0)$. We then have for any schedule that

$$C_n = b + c + 1,$$
$$\sum_{j \in S} w_j = b + c - d,$$

$$\sum w_jC_j = \sum_{1\le j\le k\le t} a_ja_k + 2b - \sum_{j\in S} w_j + 2C_n = y + c + d$$

(cf. Figure 7). It follows that there exists a schedule with value y if and only if PARTITION has a solution.

Since PARTITION can be solved in $O(tb)$ time, the above reduction does not exclude the existence of a similar *pseudopolynomial* algorithm [4] for the single machine problem. However, the latter problem is *NP*-hard even with respect to a *unary* encoding [12] (*NP*-hard in the *strong* sense [4]), which implies that it cannot be solved in pseudopolynomial time unless $P = NP$.

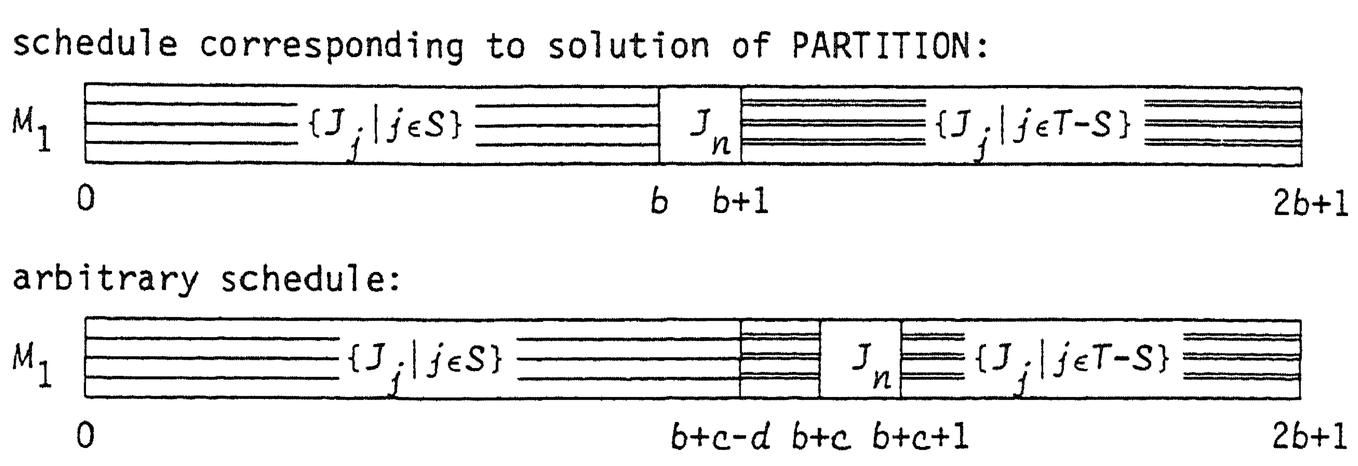


Figure 7 - Reduction from PARTITION to single machine problem, $\sum w_jC_j$ criterion.

This stronger result can be obtained by a reduction from the following unary *NP*-complete problem [4]:

3-PARTITION: Given a set $T = \{1,\ldots,3t\}$ and positive integers $a_1,\ldots,a_{3t},b$ with $\frac{1}{4}b < a_j < \frac{1}{2}b (j \in T)$ and $\sum_{j\in T} a_j = tb$, do there exist t pairwise disjoint subsets $S_i \subset T$ such that $\sum_{j\in S_i} a_j = b$ for $i = 1,\ldots,t$?

The reduction is as follows:

$$n = 4t - 1;$$

$$r_j = 0,\ p_j = w_j = a_j \quad (j \in T);$$

$$r_j = (j-3t)\,(b+1)-1,\ p_j = 1,\ w_j = 2 \quad (j = 3t+1,\ldots,4t-1);$$

$$y = \sum_{1\le j\le k\le 3t} a_ja_k + (t-1)t(\frac{3}{2}b+1).$$

The equivalence proof is left to the reader.

5. Concluding Remarks

The major open problem in the area of preemptive scheduling of uniform machines subject to release dates involves the minimization of $\sum C_j$. It has been pointed out that this problem cannot be solved by an on-line algorithm. We suspect that it cannot be proved *NP*-hard either and conjecture that it is solvable in polynomial time.

An important generalization of the models considered in this paper is the addition of *precedence constraints* between the jobs. It turns out that a number of results for the nonpreemptive scheduling of unit-time jobs subject to precedence constraints can be extended to the preemptive scheduling of jobs with arbitrary processing requirements. For example, polynomial-time algorithms have been obtained for the minimization of C_{max} on an arbitrary number of identical machines subject to release dates and outtree constraints, and for the minimization of L_{max} on two uniform machines subject to release dates and general precedence constraints. Also some *NP*-hardness proofs carry through, e.g., for the above C_{max} problem with intree rather than outtree constraints. The reader is referred to [10] for further details.

Another challenge is to investigate the *stochastic* counterparts of these models, in which the job parameters are random variables and an expected objective value is to be minimized. Initial results for such models are reported in [20].

Acknowledgments

The referee's comments led to an improvement of our running time analysis in Section 3. This research was partially supported by NSF Grants MCS76-17605 and MCS78-20054 and by NATO Special Research Grant 9.2.02 (SRG.7).

References

[1] K.R. Baker (1974) *Introduction to Sequencing and Scheduling*, Wiley, New York.

[2] J. Bruno, E.G. Coffman, Jr., R. Sethi (1974) Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* 17, 382-387.

[3] J. Bruno, T. Gonzalez (1976) Scheduling independent tasks with release dates and due dates on parallel machines. Technical Report 213, Computer Science Department, Pennsylvania State University.

[4] M.R. Garey, D.S. Johnson (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco.

[5] T. Gonzalez (1977) Optimal mean finish time preemptive schedules. Technical Report 220, Computer Science Department, Pennsylvania State University.

[6] T. Gonzalez, S. Sahni (1978) Preemptive scheduling of uniform processor systems. *J. Assoc. Comput. Mach.* 25, 92-101.

[7] W.A. Horn (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21, 177-185.

[8] E.C. Horvath, S. Lam, R. Sethi (1977) A level algorithm for preemptive scheduling. *J. Assoc. Compta. Mach.* 24, 32-43.

[9] A.V. Karzanov (1974) Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl.* 15, 434-437.

[10] E.L. Lawler (1982) Preemptive scheduling of precedence-constrained jobs on parallel machines. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 101-123.

[11] E.L. Lawler, C.U. Martel (1982) Computing maximal "polymatroidal" network flows. *Math. Oper. Res.* 7, 334-347.

[12] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker (1977) Complexity of machine scheduling problems. *Ann. Discrete Math.* 1, 343-362.

[13] C. Martel (1982) Preemptive scheduling with release times, deadlines, and due times. *J. Assoc. Compta. Mach.* 29, 812-829.

[14] C. Martel (1982) Scheduling uniform machines with release times, deadlines and due times. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 89-99.

[15] R. McNaughton (1959) Scheduling with deadlines and loss functions. *Management Sci.* 6, 1-12.

[16] S. Sahni (1979) Preemptive scheduling with due dates. *Oper. Res.* 27, 925-934.

[17] S. Sahni, Y. Cho (1979) Nearly on line scheduling of a uniform processor system with release times. *SIAM J. Compta.* 8, 275-285.

[18] S. Sahni, Y. Cho (1980) Scheduling independent tasks with due times on a uniform processor system. *J. Assoc. Compta. Mach.* 27, 550-563.

[19] W.E. Smith (1956) Various optimizers for single-stage production. *Naval Res. Logist. Quart.* 3, 59-66.

[20] G. Weiss (1982) Multiserver stochastic scheduling. In: M.A.H. Dempster, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1982) *Deterministic and Stochastic Scheduling*, Reidel, Dordrecht, 157-179.

JOURNAL OF ALGORITHMS 8, 216–235 (1987)

Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs*

M. W. BERN,† E. L. LAWLER,† AND A. L. WONG†

Computer Science Division, University of California, Berkeley, California 97420

Received May 30, 1985; revised February 18, 1986

A general problem in computational graph theory is that of finding an optimal subgraph H of a given weighted graph G. The matching problem (which is easy) and the traveling salesman problem (which is not) are well-known examples of this general problem. In the literature one can also find a variety of ad hoc algorithms for solving certain special cases in linear time. We suggest a general approach for constructing linear-time algorithms in the case where the graph G is defined by certain rules of composition (as are trees, series-parallel graphs, and outerplanar graphs) and the desired subgraph H satisfies a property that is "regular" with respect to these rules of composition (as do matchings, dominating sets, and independent sets for all the classes just mentioned). This approach is applied to obtain a linear-time algorithm for computing the irredundance number of a tree, a problem for which no polynomial-time algorithm was previously known.

1. INTRODUCTION

One of the most common types of combinatorial optimization problems is that of finding an optimal subgraph of a given graph. Such problems include the maximum matching problem, the traveling salesman problem, and the Steiner network problem. Sometimes a feasible subgraph may consist only of a subset of vertices, as in the case of the maximum independent set and minimum dominating set problems.

Quite typically, optimal subgraph problems are NP-complete if there is no restriction on the set of problem instances, but are solvable in polynomial time when the graphs are restricted to an appropriate special class.

*A preliminary version of this paper appeared under the title "Why Certain Subgraph Computations Require Only Linear Time" in the proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985.

†Research supported in part by the NSF under Grant MCS-8311422.

In fact, the polynomial-time bound may actually be linear, as in the case of:

- the minimum dominating set problem for trees [5],
- the minimum total dominating set problem for trees [14],
- the maximum disjoint triangle problem for series-parallel graphs [20],
- the traveling salesman problem for Halin networks [6],
- the Steiner problem for Halin networks [22],
- the traveling salesman problem for bandwidth-limited networks [15],

and a great variety of similar problems.

Each of the cited problems is of the general form: Given a graph G drawn from a class Γ, find, from among all subgraphs H of G that satisfy a certain property P, a subgraph that is optimum (where optimum may be in the sense of minimum number of vertices, maximum total weight of vertices and edges with respect to a given weighting, etc.). We assert that each of the problems has two key features that allow it to be solved in linear time: (1) the class Γ of problem instances is definable in terms of certain rules of composition, and (2) the property P is "regular" with respect to these rules of composition. In this paper we show how these two key features provide the basis for a systematic approach to the construction of linear-time algorithms (though not necessarily the same linear-time algorithms presented in the referenced papers). Some parts of our approach are already available in the literature: Ref. [20] treats a wide class of combinatorial problems on series-parallel graphs using a similar set of techniques, though described in very different terms. We shall illustrate the application of our methods by constructing a linear-time algorithm for computing the irredundance number of a tree. This is a problem for which no polynomial-time algorithm was previously known.

2. Graphs Definable by Composition

Let us consider how each of the following classes of graphs can be defined recursively in terms of rules of composition: rooted trees, series-parallel graphs, outerplanar graphs, Halin graphs, and bandwidth-limited graphs.

A rooted tree is a tree $T = (V, E)$ with one vertex r of V designated as its root. If T_1 and T_2 are two rooted trees, then placing an edge between their roots creates a new tree. This suggests the following definition.

DEFINITION 1 (Rooted Trees).

(1.1) The graph K_1 consisting of a single vertex r is a rooted tree and r is its root.

(1.2) If $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are rooted trees with roots r_1 and r_2, respectively, then $T = T_1 \circ T_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(r_1, r_2)\})$ is a rooted tree with root r_1. We say that T_1 is the *parent* tree and T_2 is the *child* tree.

The type of series-parallel graph we are interested in is "two-terminal" series-parallel or "edge" series-parallel, as opposed to "vertex" series-parallel [20].

DEFINITION 2 (Series-Parallel Graphs).

(2.1) The graph $G = (\{u, v\}, \{(u, v)\})$ is series-parallel, with terminals u and v.

(2.2) If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are series-parallel graphs, with terminals u_1, v_1 and u_2, v_2, respectively, then:

(a) The graph obtained by identifying u_1 and u_2 is a series-parallel graph, with v_1 and v_2 as its terminals. This graph is the *series* composition of G_1 and G_2.

(b) The graph obtained by identifying u_1 and u_2 and also v_1 and v_2 is a series-parallel graph, the *parallel* composition of G_1 and G_2. This graph has u_1 $(= u_2)$ and v_1 $(= v_2)$ as its terminals.

Two-connected outerplanar graphs can be viewed as a special kind of series-parallel graph, namely one in which at least one of the graphs entering into a parallel composition operation must be a primitive graph, that is, a single edge extending between two (terminal) vertices. We leave details to the reader.

Halin graphs were devised to provide examples of edge minimal planar 3-connected graphs [10]. The usual method of construction of a Halin graph is described as follows: Start with a tree T, in which each nonleaf vertex has degree of at least three. Embed the tree in the plane and then connect the leaves with a cycle in such a way that the planar embedding remains valid. We shall relax the condition that the nonleaf vertices have degree at least three and give a recursive definition of "protoHalin" graphs with three terminals (t, l, r), a root terminal, and left and right "leaf" terminals. All Halin graphs, as well as some multigraphs that we can view as degenerate cases, can be obtained from some protoHalin graph by adding a single edge between left and right terminals.

DEFINITION 3 (ProtoHalin Graphs).

(3.1) The graph $K_1 = (\{v\}, \varnothing)$ is a protoHalin graph; v is its root, left, and right terminals.

(3.2) If $G = (V_1, E_1)$, $G_2 = (V_2, E_2)$ are protoHalin graphs with terminals (t_1, l_1, r_1) and (t_2, l_2, r_2), respectively, then:

(a) If $G_1 = K_1$, then $G = (\{v\} \cup V_2, E_2 \cup \{(v, t_2)\})$ is a protoHalin graph, with terminals (v, l_2, r_2).

(b) If $G_1 \neq K_1$, then $G = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(t_1, t_2), (r_1, l_2)\})$ is a protoHalin graph, with terminals (t_1, l_1, r_2).

A graph of bandwidth k is defined as one whose adjacency matrix $A = (a_{ij})$ has all its nonzero entries within a band of width k about the diagonal (possibly after a renumbering of the vertices). That is, if $|i - j| > k$ then $a_{ij} = 0$. A recursive definition is as follows:

DEFINITION 4 (Bandwidth k Graphs).

(4.1) Any graph with fewer than k vertices is a graph of bandwidth k. Any graph with exactly k vertices $v_1, v_2, \ldots, v_k$ is a graph of bandwidth k, and $(v_1, v_2, \ldots, v_k)$ is its ordered set of terminals.

(4.2) If G is a graph of bandwidth k with terminals $(v_1, v_2, \ldots, v_k)$, then the graph obtained by creating a vertex v and any subset of the set of edges $\{(v_1, v), (v_2, v), \ldots, (v_k, v)\}$ is also a graph of bandwidth k and $(v_2, v_3, \ldots, v_k, v)$ is its ordered set of terminals.

In each of the four definitions above, the first rule defines a finite number of *primitive graphs* that serve as base cases, and the second rule gives instructions about how to manufacture further graphs by certain rules of composition. We abstract the essential characteristics of the four definitions above in the following definition. A class of graphs Γ is a class of *decomposable* graphs if Γ is given by a set of recursive rules that satisfy the following conditions:

(D1) The rules define a finite number of primitive graphs.

(D2) Each graph in Γ has an ordered (possibly empty) set of special vertices called *terminals*. The number of terminals is bounded by a constant.

(D3) There are a finite number of binary composition operations that act only at terminals, joining terminals either by identifying two terminals or by adding an edge (called an *attachment edge*) between two terminals. A composition operation also determines the terminals of the resulting graph, which must be a subset of the terminals of the composing graphs.

It is important to note that our decomposable definitions do not encompass other types of recursive definitions, for example, a well-known recursive definition of chordal graphs. A second important cautionary note: given a graph G and a set of rules defining a class of decomposable graphs Γ, it may be a difficult problem to recognize whether or not G belongs to Γ. We choose to avoid these recognition problems ([19] and [21] are two papers that treat such problems.), and in subsequent sections we shall simply assume that when we are given a graph G of Γ, we are also given a parse tree for G whose size is linear in the size of G. In such a parse tree, each leaf represents a primitive graph of Γ, and each nonleaf node represents a composition operation (as well as a subgraph of G).

3. Homomorphisms

Suppose we have a class of decomposable graphs Γ. By Γ_H we denote the set of graph $\times$ subgraph pairs (G, H) in which G is a graph in Γ and H is a subgraph of G. That is, (G, H) is in Γ_H if and only if G is in Γ and H is a subgraph of G. A subgraph property P is a predicate on Γ_H. To simplify matters, let us now suppose that we are concerned only with subgraphs that consist of subsets of vertices (in which case we will denote subgraphs by S and the set of graph $\times$ subgraph pairs by Γ_S). For example, the predicate P might be true for a pair (G, S) if and only if S is an independent set of vertices of G (that is, no pair of vertices in S are adjacent). This simplification will be in effect in Sections 3 and 4 of this paper; in Section 5, we extend our results to more general subgraph problems.

Assume for the moment that Γ is defined in terms of a single binary composition operator $\circ$ that can be applied to any two graphs G_1 and G_2 in Γ. In the restricted case of vertex subsets, we can unambiguously extend the definition of $\circ$ to Γ_S by writing $(G_1, S_1) \circ (G_2, S_2)$ for $(G_1 \circ G_2, S_1 \cup S_2)$. If $\circ$ is a composition operator that identifies vertices, as in (2.2a) and (2.2b) above, then, with this definition, a vertex of a composed graph is chosen to be in the vertex subset if it is chosen in either of the composing graphs. Let C be a set that has a multiplication operation (not necessarily associative nor commutative) defined on it and denote the operator by the symbol $\bullet$. We say that a mapping h from Γ_S onto C defines a *homomorphism* with respect to the class Γ and the property P if for every (G_1, S_1) and (G_2, S_2) in Γ_S,

(H1) $h(G_1, S_1) = h(G_2, S_2) \Rightarrow P(G_1, S_1) = P(G_2, S_2)$,

(H2) $h((G_1, S_1) \circ (G_2, S_2)) = h(G_1, S_1) \bullet h(G_2, S_2)$.

(Here we have taken the liberty of writing $h(G_1, S_1)$ for $h((G_1, S_1))$ and

$P(G_1, S_1)$ for $P((G_1, S_1))$; we shall use this shorthand throughout.) Condition (H1) requires that elements of Γ_S that map to the same element of C either all satisfy P or all fail to satisfy P. Thus, we may think of property P as extending to the set C, and we may say, more intuitively, that a homomorphism is a mapping that preserves the property and respects the Γ_S composition operation. In general, if Γ_S has k composition operations, $\circ_i$, $1 \le i \le k$, C should also have k multiplication operations $\bullet_i$, $1 \le i \le k$, and for each i, $1 \le i \le k$,

$$\text{(H2')} \quad h((G_1, S_1) \circ_i (G_2, S_2)) = h(G_1, S_1) \bullet_i h(G_2, S_2).$$

If a composition operator of Γ is not defined for all combinations of two graphs in Γ (as in the case of rule (4.2) for bandwidth k graphs), then Eq. (H2), or (H2'), above, need only hold for all combinations of pairs (G_1, S_1) and (G_2, S_2) for which G_1 and G_2 can be composed in Γ. Finally, we shall call a homomorphism with a finite range a *finite homomorphism*.

Two pairs $(G_1, S_1), (G_2, S_2)$ are said to be *equivalent* if $h(G_1, S_1) = h(G_2, S_2)$. Equivalence is clearly an equivalence relation. If h is a finite homomorphism, then the number of equivalence classes is finite and one can write explicit multiplication tables for elements of C, one for each multiplicative operator. (Equivalently, the multiplication tables for C can be viewed as defining operations on equivalence classes. In later sections of this paper, we speak of multiplying equivalence classes.) Thus, for the very simple case of rooted trees and independent sets, we have the single table in Fig. 1, in which equivalence classes are expressed by diagrams of representative elements. In these diagrams, the symbol ⊗ represents a "chosen" vertex, that is, a member of the vertex subset, and the symbol ○ represents an "unchosen" vertex. The three classes of tree × subset pairs (G, S) consist of, respectively, C_1, those pairs in which S is an independent set and

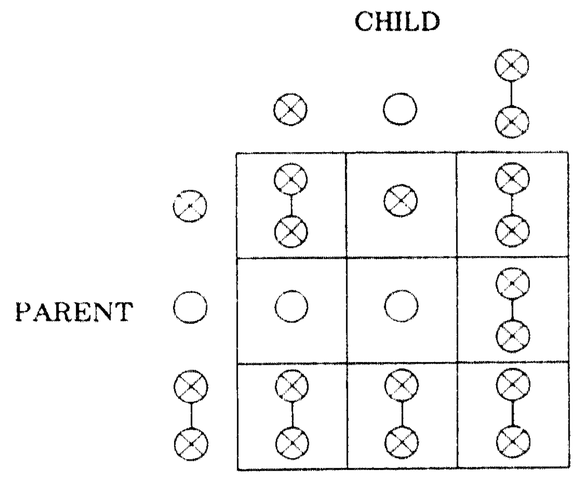


FIG. 1. Multiplication table for independence in trees.

the root r of G is chosen; C_2, those in which S is an independent set and r is unchosen; and C_3, those in which S is not independent. Thus, the entry in the first row and first column shows that the composition of a parent subtree of C_1 with a child subtree of the same class is an element of C_3, that is, the induced subset in the composed tree is not independent. It is straightforward to verify the other multiplication table entries, and thereby confirm that the table gives a homomorphism with respect to trees and independence.

If the composition operators of Γ are defined for all pairs of graphs G_1 and G_2 in Γ, then multiplication tables will be square. If Γ is a class of decomposable graphs with a definition such as (4.2) above, that is, a definition in which one of the graphs entering into a composition is restricted, then multiplication tables may be rectangular. We need not worry about the meaning of multiplications in the set C that do not correspond to graph compositions in Γ.

Multiplication tables for a homomorphism are similar to state transition tables for finite state automata. Instead of entries for state $\times$ input pairs, however, there are entries for $(G, S) \times (G, S)$ pairs. With this observation as justification, we borrow some vocabulary from the theory of finite automata. The *starting classes* (of which there may be more than one, a nonstandard feature of finite state automata) are those equivalence classes which contain a *primitive* (G, S) *pair*, that is, a pair in which the graph G is a primitive graph for the class Γ. *Accepting classes* are equivalence classes that contain (G, S) pairs for which $P(G, S)$ is true. In the example of independence for rooted trees, classes 1 and 2 are the starting classes as well as the accepting classes.

It is also possible to borrow the very useful procedure of state reduction (that is, the Myhill–Nerode theorem) from finite automata theory and apply it to a finite homomorphism. The usual procedure (in [12], for example) must be modified slightly to account for the fact that the equivalence classes act as both input symbols and states. We call two equivalence classes C_i and C_j 0-*distinguishable* if one of C_i and C_j is an accepting class and the other is not. Equivalence classes C_i and C_j are *m-distinguishable* for $m \geq 1$, if they are $(m-1)$-distinguishable, if there exists a class C_k and a multiplicative operator $\bullet_l$ such that $C_i \bullet_l C_k$ and $C_j \bullet_l C_k$ are $(m-1)$-distinguishable, or if there exists a class C_k and a multiplicative operator $\bullet_l$ such that $C_k \bullet_l C_i$ and $C_k \bullet_l C_j$ are $(m-1)$-distinguishable. As in the usual procedure, we may coalesce classes that are not n-distinguishable, where n is the original number of classes. Further reductions may be achieved by eliminating all classes not attainable (by a sequence of left and right multiplications) from the starting classes and by coalescing all classes from which no accepting class is attainable. For any given finite homomorphism, these reduction techniques produce an equiv-

alent, unique, finite homomorphism with a minimum number of equivalence classes.

4. A Linear-Time Dynamic Programming Algorithm

We say that a property or predicate P is *regular* with respect to a class Γ of graphs if it admits of a homomorphism (with respect to Γ) with a finite range and therefore a finite number of equivalence classes. The term "regular" is proposed by analogy with the notion of regular sets. One of our principal results is the following.

THEOREM 1. *Assume that Γ is a class of decomposable graphs for which we can find linear-size parse trees. Further assume that P is a computable property of vertex subsets regular with respect to Γ. Then there is a linear-time algorithm for finding optimal (either minimum or maximum cardinality) subsets satisfying property P in parsed graphs of Γ.*

Proof. Assume G is a graph drawn from Γ and that we have a linear-size parse tree T for G. We prove Theorem 1 by indicating how to construct a linear-time algorithm for solving the optimal subset problem, given a multiplication table for a finite homomorphism with respect to Γ and P. The algorithm is a dynamic programming algorithm which carries out a computation in an order corresponding to a postorder traversal of the nodes of the parse tree T. With each node of T, the algorithm associates "optimal representatives" of each equivalence class. An optimal representative of equivalence class C_k at a node v in T achieves the optimum cardinality of any graph $\times$ subset pair in C_k in which the first element of the pair is the graph G_v, the subgraph of G identified with node v.

At the leaves of T, the optimal representative of an equivalence class will be an optimal primitive $(G; S)$ pair in the class. These optimal representatives can be found in $O(1)$ time simply by exhaustive search, since the number of primitive graphs in Γ is fixed and since P is a computable property. Of course, some classes may contain no primitive pairs; in this case, we use a placeholder symbol, assumed to be less optimal than any feasible graph $\times$ subset pair. At each interior node v of T, we may assume that we have computed an optimal representative of each equivalence class for the graphs identified with the two children of v. From these, we assert that we are able to compute an optimal representative for each equivalence class for the graph identified with v. In order to find an optimal representative for equivalence class C_k, we consider all triples $C_i, C_j, \bullet_l$, such that $C_k = C_i \bullet_l C_j$, and find a best such triple. If we are interested only in the cardinality of an optimal subset, we need only remember the optimal

cardinality achieved for each equivalence class. If we are interested in recovering an optimal subset itself, we use the usual dynamic programming technique of maintaining "back pointers." That is, for each nonleaf node and each class we remember a triple $C_i, C_j, \bullet_l$ that achieved the optimum, and for each leaf and each class, we remember an optimal primitive graph $\times$ subset pair. An overall optimal solution is an optimum of all the optimal representatives of accepting classes at the root of the parse tree.

We now assert that this algorithm is linear in the size $\|G\|$ of G. First, the parse tree for G is linear in $\|G\|$ (by assumption) and can be traversed in linear time. Second, the time required to process a node is $O(1)$ since this time depends only on the size and number of multiplication tables, which in turn depends only on the number of equivalence classes and the number of operators in C. The latter two numbers are both constant with respect to $\|G\|$. □

As a simple example, consider the problem of finding the maximum cardinality of an independent set in a rooted tree G; this problem was solved by ad hoc methods in [7]. For trees, it is quite simple to construct a linear-size parse tree in linear time (simply perform a postorder traversal). Thus, we may assume that the input tree G is already parsed. We recall that the three equivalence classes for pairs (G, S) are C_1, the set of those pairs in which S is an independent set and the root of G is chosen; C_2, the set of those pairs in which S is an independent set and the root of G is not chosen; and C_3, the set of those pairs in which S is not independent. Leaves of the parse tree are identified with single-vertex trees. For a single-vertex tree K_1, the maximum cardinality of a vertex subset S such that (K_1, S) is in C_1 is 1, such that (K_1, S) is in C_2 is 0, and such that (K_1, S) is in C_3 is $-\infty$. Let G_v be the tree identified with some interior node v of the parse tree and let G_1 and G_2 be the trees identified with the children of v. Refering to the multiplication table in Fig. 1, we see that the only way to produce a C_1 pair is by multiplying a C_1 pair with a C_2 pair. Thus the maximum cardinality of S such that (G_v, S) is in C_1 is the sum of the maximum cardinalities achieved by a G_1 pair from C_1 and a G_2 pair from C_2. (By a G_1 pair, we mean any pair (G, S) in which G is G_1.) Similarly, there are two possible ways to produce an element of C_2: $C_2 \bullet C_1$ or $C_2 \bullet C_2$. The cardinality of the best representative of C_2 for G_v is the maximum of the two possible sums in this case. More generally, cardinalities of best representatives for the graph identified with a node can be computed by considering the sums of best cardinalities of appropriate classes for its parent and child subgraphs. We choose a way that produces a graph $\times$ subset pair with a largest subset. (In other words, ties are broken arbitrarily.) Finally, at the root of the parse tree, the solution to the maximum independent set problem is given by the maximum of the accepting classes, that is, C_1 and C_2.

Theorem 1 can be extended to other criteria of optimality besides minimum and maximum cardinality. For example, if the vertices of a tree are given integer weights, essentially the same algorithm as that sketched above can find an independent set of vertices with maximum total weight. The only modification is that at each stage of the computation, we must remember maximum weight representatives of each equivalence class rather than maximum cardinality representatives. We give this observation the status of a corollary.

COROLLARY 1. *Assume that* Γ *is a class of decomposable graphs for which we can find linear-size parse trees. Further assume that* P *is a computable property of vertex subsets regular with respect to* Γ. *Then there is a linear-time algorithm for finding optimal weight (either minimum or maximum) subsets satisfying property* P *in vertex-weighted graphs of* Γ. □

5. More General Subgraph Problems

So far, we have considered only problems that deal with subgraphs that do not contain edges. We shall indicate by example how to generalize Theorem 1 to properties of subgraphs that do contain edges. The problem of finding a minimum cardinality maximal matching asks for a smallest set of edges with disjoint endpoints to which it is impossible to add another edge without violating the disjointness constraint. Minimum cardinality maximal matching is NP-complete for general and bipartite graphs [8]. Using an extension of the techniques of Section 4, we shall show that this problem is linear-time solvable for trees, a result first solved in the form of "independent edge domination" in [23]. To show that the maximal matching property is regular with respect to the class of rooted trees, we must construct a finite homomorphism.

Since the composition operation $\circ$ for rooted trees creates a new edge, the extension of $\circ$ from the class of trees Γ to the set of tree $\times$ subgraph pairs Γ_H must specify whether or not the new edge is chosen to be in the subgraph. We accomplish this extension by creating two operators for Γ_H: $\circ_1$ in which the new edge is not chosen and $\circ_2$ in which the new edge is chosen. (In our subgraph version of matching, if an edge is chosen to be in the matching, then both its end-points must also be chosen.) Examples of each of these operations are given in Fig. 2. Vertices in the matching are denoted by $\otimes$ and edges in the matching are denoted by wiggly lines.

We now define the following set of four equivalence classes for the maximal matching property: C_1 is the set of those pairs (G, H) in which the root r is unmatched and H is a maximal matching for G; C_2 is the set of those pairs in which r is matched and H is a maximal matching for G;

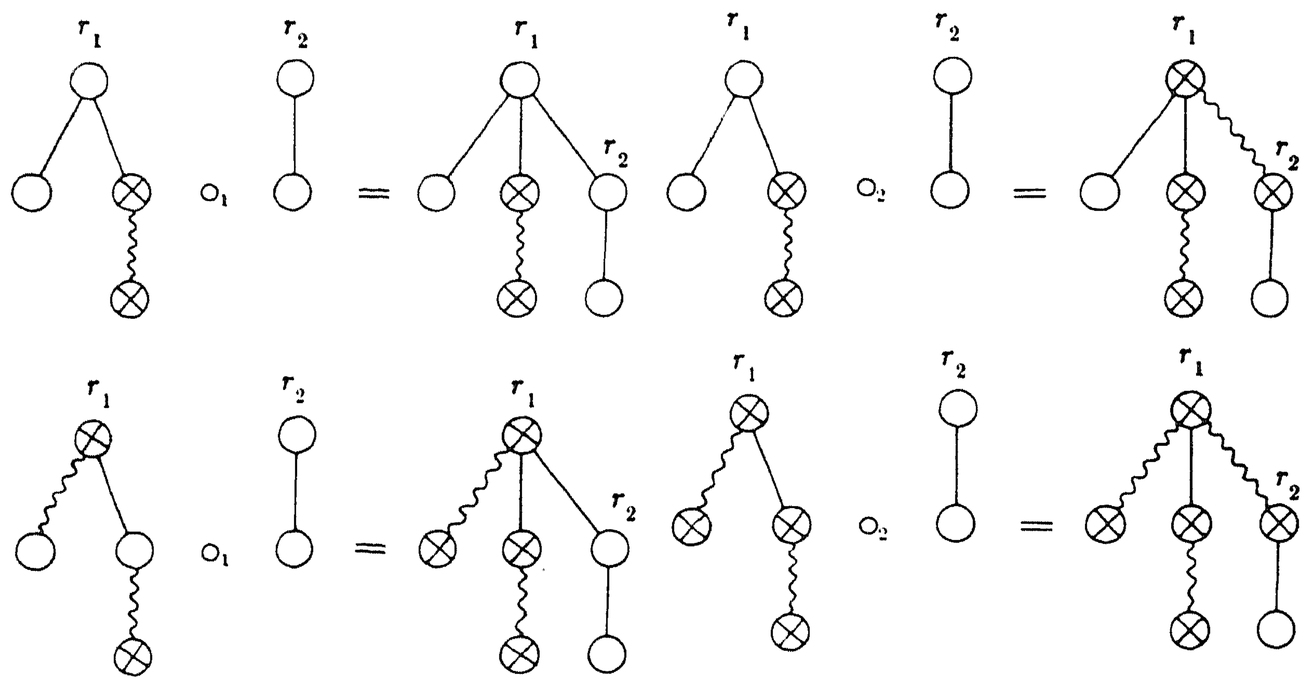


FIG. 2. Examples of $\circ_1$ and $\circ_2$.

C_3 is the set of those pairs in which r is unmatched, H is a matching for G but not a maximal one, and H is a maximal matching for the subgraph of G induced by the vertices in $V - \{r\}$, where V is the vertex set of G; and finally, C_4 is the set of remaining pairs. Intuitively, C_1, C_2, and C_3 contain pairs that can appear as subgraphs of maximally matched pairs in Γ_H and C_4 contains pairs that can never appear as such. Classes C_1 and C_2 contain maximally matched pairs and are therefore the accepting classes. The multiplication tables for the multiplicative operators $\bullet_1$ and $\bullet_2$ corresponding to composition operators $\circ_1$ and $\circ_2$ are given in Fig. 3. The algorithm given in the proof of Theorem 1 can now be used to solve this general subgraph problem. For example, consider determining the best representative of C_2. By inspection of the two multiplication tables, we note that there are altogether six ways of producing C_2, namely: $C_2 \bullet_1 C_1$, $C_2 \bullet_1 C_2$, $C_1 \bullet_2 C_1$, $C_1 \bullet_2 C_3$, $C_3 \bullet_2 C_1$, and $C_3 \bullet_2 C_3$. We choose from these a way that gives a best (minimum number of edges in this case) representative.

As before, this algorithm extends to weighted optimization problems, in which edges, or vertices, or both, carry weights. Thus, we expect that the extent of our techniques includes even problems such as the Steiner problem for Halin networks [22] or the traveling salesman problem for bandwidth-limited networks [15].

The only important difference between our techniques for vertex subset properties and more general subgraph properties lies in the extension of composition operators from the class of decomposable graphs Γ to the set of pairs Γ_H. In order to accomplish this extension, it may be necessary to

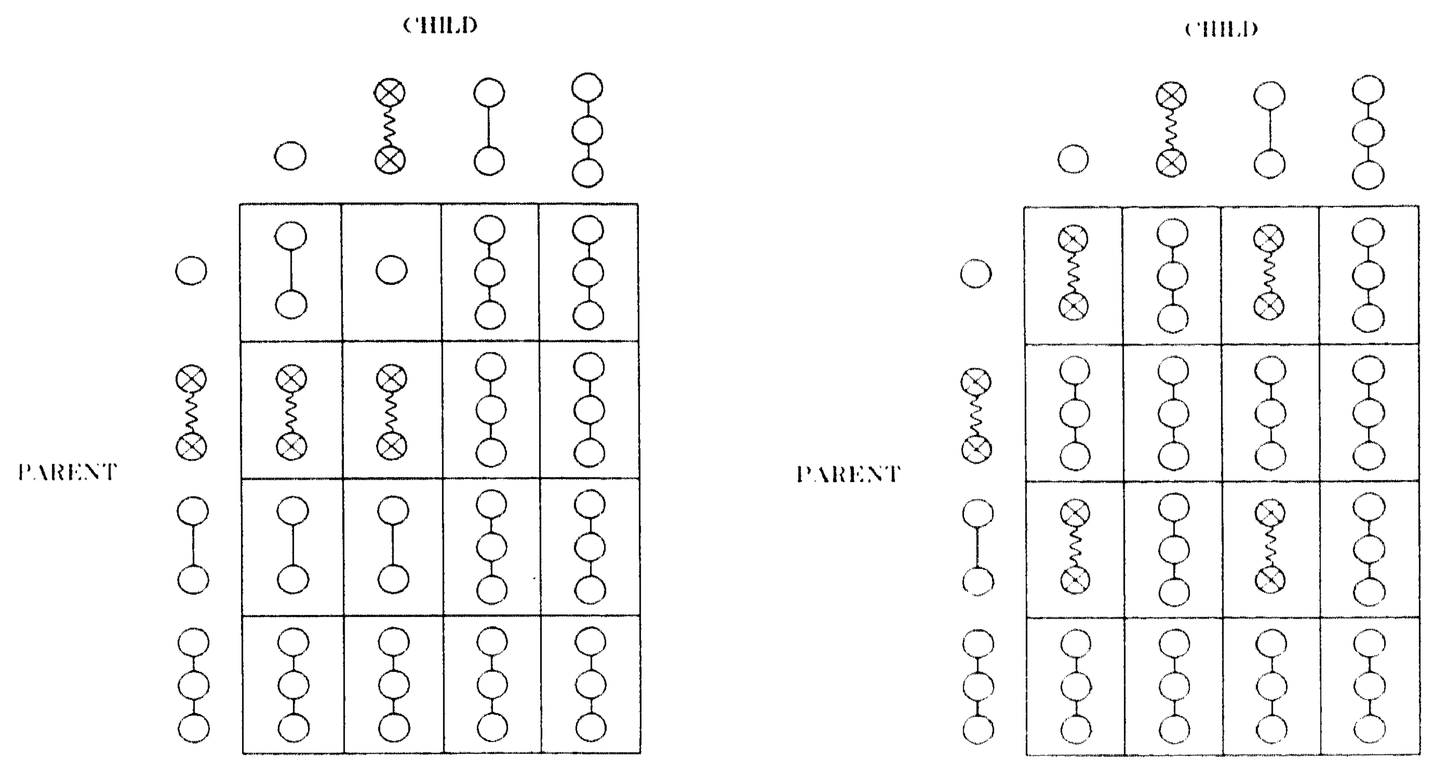


FIG. 3. Maximal matching. Left $\cdot_1$ table; right, $\cdot_2$ table.

replace a single composition operator $\circ$ with as many as 2^k operators on Γ_H, where k is the number of attachment edges created by $\circ$. The definitions of homomorphism and regular property are just as in Section 3, though condition (H2′) is, of course, meant to hold for every one of the enlarged set of composition operators. It should be clear that the proof of Theorem 1 carries over into the more general setting. Unfortunately, the necessity of exponentially enlarging the number of composition operators makes our approach less elegant for general subgraph properties than it is for vertex subset properties.

6. Operations on Regular Properties

It is well known that regular sets are closed under various operations, such as union, intersection, relative complement, reversal, and Kleene closure. The same is true for regular properties (either vertex subset or general subgraph).

THEOREM 2. *The properties that are regular with respect to a given class* Γ *are closed under Boolean operations "and", "or", and "not".*

Proof. The operation "not" follows immediately from definitions. Its set of accepting classes is the complement of the set of accepting classes for the original property.

"And" and "or" are shown by direct product constructions. We sketch the proof for "and"; "or" is similar. Suppose we have two properties P and

Q defined on the same set Γ_H with finite homomorphisms h_1 and h_2 respectively. We think of h_1 and h_2 as mapping graph $\times$ subgraph pairs to sets (that is, equivalence classes) of graph $\times$ subgraph pairs, rather than to elements of abstract sets C. We assume that we have multiplication tables for each of h_1 and h_2 for multiplicative operators corresponding to the same Γ_H composition operators $\circ_i$. Thus the multiplicative operators on the ranges of h_1 and h_2 can be identified according to their corresponding Γ_H composition operators, and these operators can be denoted by $\bullet_i$ regardless of range. To create a homomorphism h_3 with respect to both P and Q, consider all pairwise intersections of equivalence classes in the range of h_1 with equivalence classes in the range of h_2. Under h_3, a pair (G, H) maps to the set $h_1(G, H) \cap h_2(G, H)$. We again reuse the symbols $\bullet_i$ to denote the multiplicative operators for the range of h_3, defined as follows:

$$h_3(G_1, H_1) \bullet_i h_3(G_2, H_2) = (h_1(G_1, H_1) \bullet_i h_1(G_2, H_2)) \cap (h_2(G_1, H_1) \bullet_i h_2(G_2, H_2)).$$

It is straightforward to confirm that h_3 satisfies the two homomorphism conditions, (H1) and (H2). The accepting classes for P and Q are exactly those classes that are intersections of an accepting class for P with an accepting class for Q. □

As an example of the application of Theorem 2, since independence and domination are regular properties for trees, it follows that independent domination is also a regular property for trees. By Theorem 1, there then exists a linear-time algorithm for finding minimum (or maximum) independent dominating sets for trees. Previously, composite property problems such as independent domination have been solved separately from the corresponding problems for the simple properties making up the composite [1, 18].

Other very useful and important operations on vertex subset properties are *max* and *min* (for maximal and minimal). If P is a property of vertex subsets, then we may define $maxP(G, S)$ to be true if and only if $P(G, S)$ is true and $P(G, S \cup \{v\})$ is false for all vertices v in $V - S$. (V is the vertex set of G.) Similarly, $minP(G, S)$ is true if and only if $P(G, S)$ is true and $P(G, S - \{v\})$ is false for all vertices v in S. For *monotone* properties, $maxP$ coincides with another notion of maximal P, that is, $maxP(G, S)$ is true if and only if $P(G, S)$ is true and there is no set S' of vertices in G such that $S \neq S'$, $S \subset S'$, and $P(G, S')$ is true.

THEOREM 3. *If P is a property of vertex subsets that is regular with respect to* Γ, *then so are maxP and minP.*

Proof. Consider only $maxP$; the proof for $minP$ is analogous. Suppose we have a finite homomorphism h_1 for P. We can create a homomorphism

h_2 for $maxP$ by considering a particular refinement of the equivalence classes in the range of h_1. The image $h_2(G, S)$ of a pair (G, S) will be defined by the h_1 equivalence class of (G, S) and by the set of h_1 equivalence classes that are obtainable from (G, S) by the addition of a single vertex to S. If we denote the h_1 equivalence classes by C_i, for $i = 1, 2, \ldots, n$, then we can denote the h_2 equivalence classes by pairs $[C_i, A]$, where A is a subset of $\{C_1, C_2, \ldots, C_n\}$. Note that there are no more than $n \cdot 2^n$ h_2 equivalence classes. (In practice, many of these may be empty or redundant.)

We must now verify that the homomorphism conditions (H1) and (H2) hold for h_2. In order to confirm (H1), that is, $h_2(G_1, S_1) = h_2(G_2, S_2) \Rightarrow maxP(G_1, S_1) = maxP(G_2, S_2)$ for any two pairs (G_1, S_1) and (G_2, S_2), observe that the equivalence classes of h_2 for which $maxP$ is true are exactly those classes $[C_i, A]$ for which P is true for C_i and P is false for every class in A. Thus, any member (G, S) of such a class has property P, and if any vertex is added to S, it no longer has P.

In order to demonstrate (H2), we must first show how to multiply two h_2 equivalence classes. Suppose $h_2(G_1, S_1)$ is the class $[C_1, A_1]$ and $h_2(G_2, S_2)$ is the class $[C_2, A_2]$. Then for each composition operator $\bullet_l$ of h_1 equivalence classes, we reuse the same symbol and define $[C_1, A_1] \bullet_l [C_2, A_2]$ to be

$$\left[C_1 \bullet_l C_2, \{C_i \bullet_l C_2 | C_i \in A_1\} \cup \{C_1 \bullet_l C_j | C_j \in A_2\}\right].$$

We now assert that, with this definition of multiplication, $h((G_1, S_1) \circ_l (G_2, S_2)) = [C_1, A_1] \bullet_l [C_2, A_2]$, that is, (H2) holds for h_2. First note that $h_2((G_1, S_1) \circ_l (G_2, S_2))$ is a subset of the h_1 equivalence class $C_1 \bullet_l C_2$ since (G_1, S_1) is in h_1 class C_1 and (G_2, S_2) is in h_1 class C_2. (We just used the fact that (H2) holds for h_1). Thus we see that the first entry in our definition of $[C_1, A_1] \bullet_l [C_2, A_2]$ makes sense. To complete the proof of our assertion, we must identify which h_1 equivalence classes are obtainable from $(G_1, S_1) \circ_l (G_2, S_2) = (G_1 \circ_l G_2, S_1 \cup S_2)$ by the addition of a single vertex. Notice that any vertex v added to $S_1 \cup S_2$ must come from either G_1 or G_2 (perhaps from both if $\circ_l$ identifies vertices). Assume v comes from G_1. Then

$$\begin{aligned} h_1(G_1 \circ_l G_2, S_1 \cup S_2 \cup \{v\}) &= h_1((G_1, S_1 \cup \{v\}) \circ_l (G_2, S_2)) \\ &= h_1(G_1, S_1 \cup \{v\}) \bullet_l h_1(G_2, S_2) \\ &= C_i \bullet_l C_2 \qquad \text{for some } C_i \in A_1. \end{aligned}$$

Similarly, adding any vertex v from G_2 yields a class that is a product of C_1 with an element of A_2. Conversely, all products of classes in A_1 with C_2

and all products of C_1 with classes in A_2 are obtainable from the graph $G_1 \circ_l G_2$ by the addition of some vertex to $S_1 \cup S_2$. □

The equation

$$[C_1, A_1] \bullet_l [C_2, A_2] = \left[C_1 \bullet_l C_2, \{C_i \bullet_l C_2 | C_i \in A_1\} \cup \{C_1 \bullet_l C_j | C_j \in A_2\}\right]$$

can be used to automatically construct a multiplication table for *maxP* given a multiplication table for P. The resulting table for *maxP* will typically have many redundant or unattainable classes, and obtaining a practical algorithm will require the reduction techniques mentioned in Section 3.

It may be an interesting exercise for the reader to carry out a construction similar to that of Theorem 3 for regular sets. That is, given a language L over the alphabet $\{0, 1\}$, let us say that a string x is *maximal* in L if and only if there is no distinct string $y \in L$, $|y| = |x|$, such that y has a 1 in each position that x has a 1. ($|x|$ is the length of x.) Let us denote by *maxL* the set of all maximal strings in L. The assertion to be proved is that if L is regular, then *maxL* is regular too.

Finally, it is natural to ask whether an extension of Theorem 3 holds for general subgraph properties. The most useful definition of the *max* operation on general subgraph properties is probably as follows: $maxP(G, H)$ is true if and only if $P(G, H)$ is true and $P(G, H')$ is false for all subgraphs H' of G such that $H \neq H'$ and H is a subgraph of H'. Unfortunately, an extension of Theorem 3 does not hold for this definition of *max* and arbitrary properties P. An extension of Theorem 3 does hold, given a different definition of *max*: $maxP(G, H)$ is true if and only if $P(G, H)$ is true and $P(G, H')$ is false for all distinct subgraphs H' of G obtained by adding an edge (and, if necessary, one or both of its endpoints) of G to H. (Perhaps this notion of maximality could be more properly called "edge criticality.") The proof of this extension is analogous to the proof of Theorem 3, but more complicated; we shall sketch the essential difference in the case of a class of graphs Γ, such as rooted trees, that has only a single composition operator $\circ$ that creates only a single new edge. As in Section 5, the composition operators induced on Γ_H are $\circ_1$ and $\circ_2$, the first leaving the attachment edge out of the subgraph, and the second choosing the attachment edge. The corresponding multiplication operators on P equivalence classes are $\bullet_1$ and $\bullet_2$. Following the construction in the proof of Theorem 3, we define the equivalence classes for *maxP* as pairs $[C_i, A]$, where C_i is a P equivalence class (that is, an equivalence class induced by a homomorphism with respect to P and Γ) and A is a subset of P equivalence classes, interpreted as the set of equivalence classes obtainable by the addition of a single edge (and its endpoints, if necessary) to the subgraph. We observe that an edge added to a pair (G, H) resulting from a

composition $(G_1, H_1) \circ (G_2, H_2)$ in Γ_H must lie entirely within one of G_1 or G_2 or must be the (unique) attachment edge. Thus, $\bullet_1$ multiplication of two pairs, $[C_1, A_1]$ and $[C_2, A_2]$, should consider the possibility of adding the attachment edge in determining the subset of C_t classes obtainable by edge addition. $[C_1, A_1] \bullet_1 [C_2, A_2]$ is defined to be

$$\left[C_1 \bullet_1 C_2, \{C_i \bullet_1 C_2 | C_i \in A_1\} \cup \{C_1 \bullet_1 C_j | C_j \in A_2\} \cup \{C_1 \bullet_2 C_2\}\right].$$

The definition of $\bullet_2$ multiplication of pairs $[C_1, A_1]$ and $[C_2, A_2]$ is just as in the proof of Theorem 3; in this case, the attachment edge is known to be chosen, so we need not consider the possibility of adding it.

7. The Irredundance Problem for Trees

We now apply our theoretical machinery to the irredundance number problem. A vertex v is said to *dominate* (or *cover*) itself and each of its neighbors. A subset of vertices dominates all vertices that are dominated by at least one of its members. A subset S of the vertices of a graph G is said to be *irredundant* if there is no proper subset S' of S that dominates the same set of vertices as S; otherwise S is *redundant*. A vertex v is redundant in a set S if $S - \{v\}$ dominates the same set of vertices as S. An irredundant set, even a maximal irredundant set, need not be a dominating set, in that certain vertices may be left uncovered. The *irredundance number* of a graph is defined as the minimum cardinality of a maximal irredundant set [4]. This number has received a certain amount of attention because of several inequalities relating it to domination and independence [2–4].

Computing the irredundance number of a general graph or even a bipartite graph is an NP-complete problem [17]. The computation of the irredundance number for the case of trees was for several years an open question [11]. We now indicate how to construct a linear-time algorithm for this problem, starting with a finite homomorphism for (nonmaximal) irredundance. This homomorphism maps tree $\times$ subset pairs to seven equivalence classes. The first five classes, in addition to the requirements listed next, all require that the subset of chosen vertices be irredundant (these five are thus the accepting classes): C_1 contains pairs (T, S) in which the root of T is not dominated by S; C_2 contains pairs in which the root is unchosen but dominated, and "dominating it again" (that is, composing the pair with another pair in which the root is chosen) will not make any vertices redundant; C_3 contains pairs in which the root is unchosen but dominated, and dominating it again will make some vertex redundant; C_4 contains pairs in which the root is chosen, and dominating the root again will make

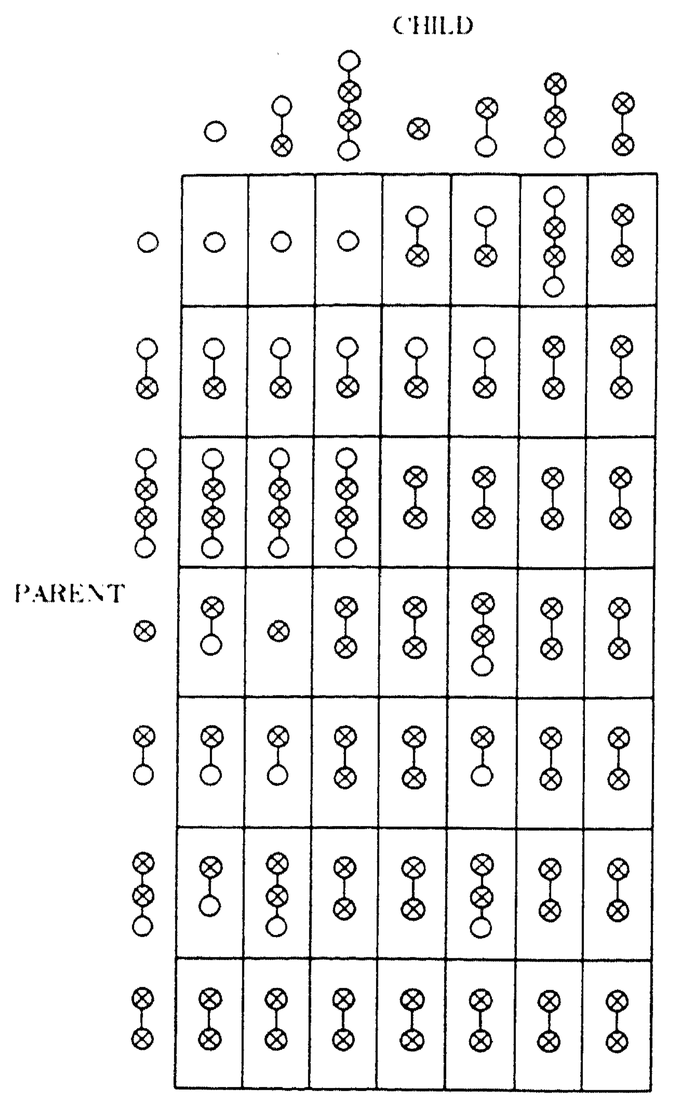


FIG. 4. Irredundance.

it redundant; and C_5 contains pairs in which the root is chosen, and dominating it again will not make it redundant. Note that classes C_1 and C_4 are also starting classes. Equivalence class C_6 contains pairs in which the root is chosen and is redundant, and, finally, C_7 contains pairs in which some vertex other than the root is redundant. Figure 4 gives a multiplication table for these equivalence classes, thus demonstrating that irredundance is regular with respect to the class of rooted trees. As before, the figures in the multiplication table give smallest representatives of each equivalence class. Proving this table correct is quite straightforward using only the defining properties of the seven classes.

Applying the construction indicated in the proof of Theorem 3 to the table in Fig. 4, we obtain a multiplication table for maximal irredundance. (Irredundance is monotone, so the two notions of maximality mentioned above are the same.) The table we have given in Fig. 5 is one obtained by a computer program which simplified a table with $7 \cdot 2^7$ equivalence classes. In Fig. 5, the multiplication table is given as a matrix of numbers with a key below showing class representatives. The starting classes are numbers 4 and 15, and accepting classes are indicated with an asterisk. It would be very

CHILD

PARENT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	2	0	2	0	1	0	2	0	3	0	1	8	9	8	0	9	14
2	0	2	0	0	2	0	2	0	2	0	2	0	2	0	2	0	0	0	0	0	14
3	0	3	0	0	2	0	2	0	3	0	2	0	3	0	3	8	9	8	0	9	14
4	0	6	0	0	2	0	2	0	4	0	2	0	5	0	1	8	9	12	0	13	14
5	0	7	0	0	2	0	2	0	5	0	2	0	5	0	3	8	9	12	0	13	14
6	0	6	0	0	2	0	2	0	6	0	2	0	7	0	1	8	9	10	0	11	14
7	0	7	0	0	2	0	2	0	7	0	2	0	7	0	3	8	9	10	0	11	14
8	0	8	0	0	0	0	0	0	8	0	0	0	8	0	8	8	8	8	0	8	0
9	0	9	0	0	0	0	0	0	9	0	0	0	9	0	9	8	8	8	0	8	0
10	0	10	0	0	0	0	0	0	10	0	0	0	10	0	8	8	8	10	0	10	0
11	0	11	0	0	0	0	0	0	11	0	0	0	11	0	9	8	8	10	0	10	0
12	0	10	0	0	0	0	0	0	12	0	0	0	12	0	8	8	8	12	0	12	0
13	0	11	0	0	0	0	0	0	13	0	0	0	13	0	9	8	8	12	0	12	0
14	0	14	0	0	14	0	14	0	14	0	14	0	14	0	14	0	0	0	0	0	0
15	0	17	18	17	17	17	19	19	15	15	16	16	15	15	0	0	0	20	20	20	0
16	0	0	0	0	0	0	0	0	16	16	16	16	16	16	0	0	0	0	0	0	0
17	0	17	0	17	17	17	0	0	17	17	0	0	17	17	0	0	0	17	17	17	0
18	0	0	0	0	0	0	0	0	18	18	0	0	18	18	0	0	0	19	19	19	0
19	0	0	0	0	0	0	0	0	19	19	0	0	19	19	0	0	0	19	19	19	0
20	0	17	19	17	17	17	19	19	20	20	0	0	20	20	0	0	0	20	20	20	0

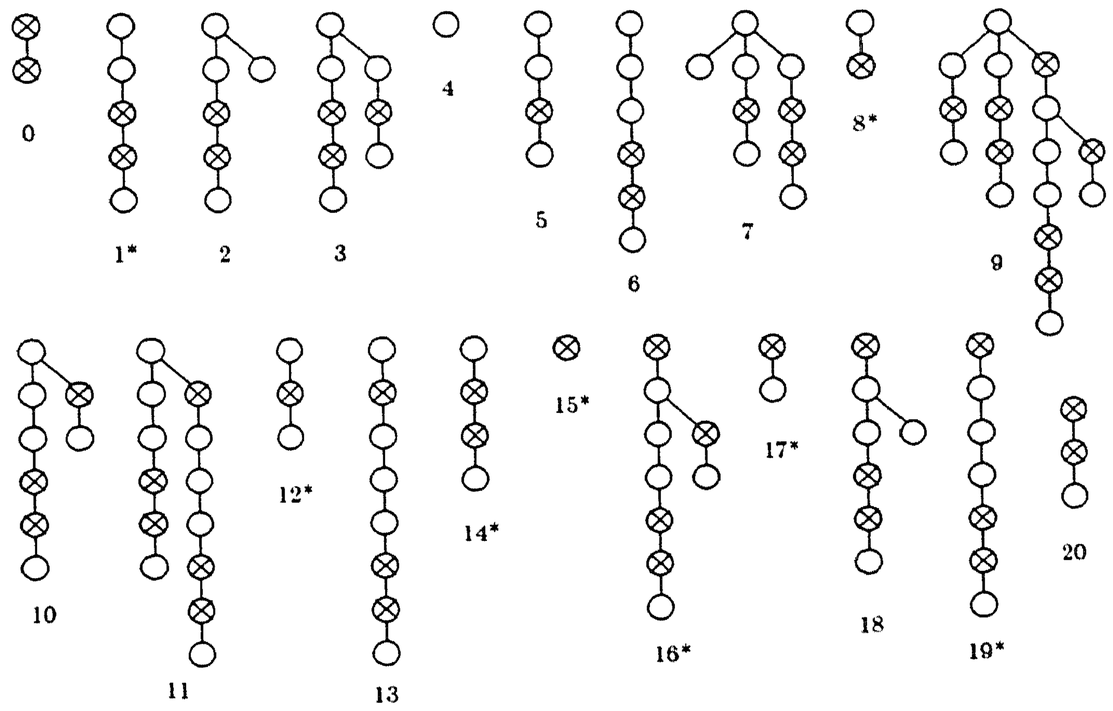


FIG. 5. Maximal irredundance.

difficult to construct such a table without the bootstrapping afforded by Theorem 3. Perhaps worth mentioning is that actually constructing representatives of each equivalence class as we have done is not necessary. In order to apply the dynamic programing algorithm of Section 4, it is sufficient to know the multiplication table or tables and which classes are starting and which accepting.

8. COMMENTS AND CONCLUSIONS

In this paper we have suggested a systematic approach to constructing linear-time algorithms for various optimal subgraph problems. This amounts

to the following:

(1) Obtain a decomposable definition for the class of graphs to which the algorithm is to be applied, and solve the associated parsing problem.

(2) Obtain a finite homomorphism for the property that a subgraph must satisfy in order to be a feasible solution. If this homomorphism cannot be obtained directly, try to obtain it from a combination of simpler homomorphisms, by appropriate application of Theorems 2 and 3.

(3) Apply the basic linear-time dynamic programming algorithm to any given parsed problem instance.

This approach yields algorithms that are different from, but are of the same asymptotic complexity (that is, linear-time) as, those in [1, 5–7, 9, 13–16, 18, 20, 22]. The merit of our approach is that it provides a uniform theory for understanding how optimal subgraph problems can be solved efficiently in a wide variety of special cases, where these problems had previously been attacked by ad hoc methods. Our approach also provides a model in which to work; we expect that researchers will find it easier to create homomorphism multiplication tables than to create entire algorithms. Finally, writers of software could take advantage of inherent modularity in our methods: one program, supplied with the proper multiplication tables, could be used to solve many different subgraph problems.

There are a few open problems that seem worthy of attention. Is there a nice characterization of classes of decomposable graphs in terms of graph grammars? Can our techniques be usefully applied to more general classes of graphs, perhaps by further restricting the subgraph properties? Are there good alternative characterizations of the family of regular properties (with respect to a given class of graphs)? By this is meant characterizations that are analogous to right linear grammars and regular expressions for the family of regular sets. Finally, of course, what significant and challenging open problems might be solved by application of the suggested approach?

ACKNOWLEDGMENTS

The authors wish to thank Stephen Hedetniemi for his enthusiasm over this work and the referee for many valuable comments.

REFERENCES

1. T. BEYER, A. PROSKUROWSKI, S. HEDETNIEMI, AND S. MITCHELL, Independent domination in trees, in "Proceedings, 8th S. E. Conf. on Combinatorics, Graph Theory, and Computing," pp. 321–328, Utilitas Mathematica, Winnipeg, 1977.

2. B. BOLLOBÁS AND E. J. COCKAYNE, Graph theoretic parameters concerning domination, independence, and irredundance, *J. Graph Theory* **3** (1979), 241–250.
3. B. BOLLOBÁS AND E. J. COCKAYNE, The irredundance number and maximum degree of a graph, *Discrete Math.* **49** (1984), 197–199.
4. E. J. COCKAYNE, O. FAVARON, C. PAYAN, AND A. THOMASON, Contributions to the theory of domination, independence, and irredundance in graphs, *Discrete Math.* **33** (1981), 249–258.
5. E. J. COCKAYNE, S. E. GOODMAN, AND S. T. HEDETNIEMI, A linear algorithm for the domination number of a tree, *Inform. Process. Lett.* **4** (1975), 41–44.
6. G. CORNUÉJÒLS, D. NADDEF, AND W. R. PULLEYBLANK, Halin graphs and the Traveling Salesman Problem, *Math. Programming* **26** (1983), 287–294.
7. D. E. DAYKIN AND C. P. NG, Algorithms for generalized stability numbers of tree graphs, *J. Austral. Math. Soc.* **6** (1966), 89–100.
8. M. R. GAREY AND D. S. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," p. 192, Freeman, San Francisco, 1979.
9. S. GOODMAN, S. T. HEDETNIEMI, AND R. TARJAN, B-matchings in trees, *SIAM J. Comput.* **5** (1976), 104–108.
10. R. HALIN, Studies on minimally n-connected graphs, *in* "Combinatorial Mathematics and Its Applications" (D. J. A. Welsh, Ed.), pp. 129–136, Academic Press, New York, 1971.
11. S. M. HEDETNIEMI, S. T. HEDETNIEMI, AND R. LASKAR, Domination in trees: Models and algorithms, *in* "Graph Theory with Applications to Algorithms and Computer Science" (Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall, Eds.), Wiley, New York, 1985.
12. J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," pp. 65–67, Addison–Wesley, Reading, Mass., 1979.
13. T. KIKUNO, N. YOSHIDA AND Y. KAKUDA, A linear algorithm for the domination number of a series-parallel graph, *Discrete Appl. Math.* **5** (1983), 299–311.
14. R. LASKAR, J. PFAFF, S. M. HEDETNIEMI, AND S. T. HEDETNIEMI, On the algorithmic complexity of total domination, *SIAM J. Algebraic Discrete Methods* **5** (1984), 420–425.
15. D. MONIEN AND I. H. SUDBOROUGH, Bandwidth constrained NP-complete problems, *in* "Proceedings, 13th Ann. Symp. on the Theory of Computing," pp. 207–217, 1981.
16. K. S. NATARAJAN AND L. J. WHITE, Optimum domination in weighted trees, *Inform. Process. Lett.* **7** (1978), 261–265.
17. J. PFAFF, R. LASKAR, AND S. T. HEDETNIEMI, "NP-Completeness of Total and Connected Domination and Irredundance for Bipartite Graphs," Tech. Report 428, Clemson University, 1983.
18. J. PFAFF, R. LASKAR, AND S. T. HEDETNIEMI, Linear algorithms for independent domination and total domination in series-parallel graphs, *Congr. Numer.* **45** (1984), 71–82.
19. J. SAXE, Dynamic programming algorithms for recognizing small bandwidth graphs in polynomial time, *SIAM J. Algebraic Discrete Methods* **1** (1980), 363–369.
20. K. TAKAMIZAWA, T. NISHIZEKI, AND N. SAITO, Linear-time computability of combinatorial problems on series-parallel graphs, *J. Assoc. Comput. Mach.* **29** (1982), 623–641.
21. J. VALDES, R. TARJAN, AND E. L. LAWLER, The recognition of series-parallel digraphs, *SIAM J. Comput.* **11** (1982), 289–313.
22. P. WINTER, Generalized Steiner problem in Halin networks, *in* "Proceedings 12th Int. Symp. on Math. Programming," MIT, 1985.
23. S. MITCHELL AND S. HEDETNIEMI, Edge domination in trees, *in* "Proceedings, 8th S.E. Conf. on Combinatorics, Graph Theory, and Computing," pp. 489–509, Utilitas Mathematica, Winnipeg, 1977.

Old Stories

Eugene L. Lawler

What do you say when you are asked to write recollections of the 'old days'? How do you respond, when your own career is quite incomplete, and — you devoutly hope — your best work is yet to be done? How do you respond, when — as your wife reminds you — you can't even remember what you had for lunch? You say, yes, of course. Following are a few stories of the 1950's, . . . , 1980's, with a few speculations about the 1990's.

The Fifties

When I began graduate study at Harvard in 1954, I had little purpose and much innocence. I had completed a three-year baccalaureate at a southern university, in the course of which I had taken every undergraduate and post-graduate offering in mathematics in the catalog. But somehow, I had never heard there was such a thing as 'applied' mathematics. And, certainly, I was unprepared for the higher standards of academic competition I was to encounter in graduate school.

I soon decided that abstract algebra and real and complex analysis were not for me, particularly not when Harvard undergraduates were running circles around me in my classes. Sometimes I thumbed through library copies of journals, looking for something more meaningful. I recall once running across a quaint little journal with the title *Operations Research*. I wish I could say that this was a moment of awakening, but my reaction was only one of bafflement. Awakening came later.

For variety, I elected two courses that were polar opposites. The first of these was W. V. O. Quine's course on mathematical logic. Quine had been away on sabbatical at Oxford. His fame, and the pent-up demand for his

course, attracted an initial attendance of a hundred or more. Curiously, he seemed unperturbed by the small size of the classroom. But I soon understood why. At the end, only a half-dozen of us were left in the class. I passed, but with little comprehension.

About twenty five years after Quine's course, I happened to meet Garrett Birkhoff at a conference. After I had introduced myself as one of his former students, Birkhoff delighted me with the following story. (Quotation is from memory.)

> "You know, many years ago — back in the 1930's — I thought I was interested in the foundations of mathematics. I even wrote a paper or two in the area. We had a brilliant young logician on our faculty who was making a name for himself. One term I happened to lecture in the same classroom, the hour immediately after he did. Each day, before erasing the blackboard, I would take a look to see what he was up to. Well, in September he started out with Theorem 1. Shortly before Christmas he was up to Theorem 747. You know what it was? 'If $x \leqslant y$ and $y \leqslant z$, then $x \leqslant z$'! At that point, something within me snapped. I said to myself, 'There are some things in mathematics that you just have to take for granted!' And I never again had anything to do with the foundations of mathematics."

My other elective was the maiden offering of a course in 'Automatic Data Processing', taught by Kenneth Iverson with Fred Brooks as one of the TA's. Like most of those taking the course, I hoped to learn something about 'giant brains' that could 'think'. Dick Karp, then a Harvard senior, was one of my classmates. We share the same recollection.

One day Iverson announced that he would lecture on 'tape sorting'. My reaction: "Fine, I know what tapes are. They come on reels. There are lots and lots of reels of tape in the Computation Laboratory. It must be a common event for those reels to become disordered. We're going to learn how to sort those reels of tape and put them in order again. Not a terribly interesting issue perhaps, but a necessary bit of housekeeping if one is going to run a computer installation . . . What's this? He's going to use the computer to sort those reels of tape? Strange! Computers aren't robots! How can they sort reels of tape? This doesn't make any sense at all!" About twenty minutes into the lecture, the light dawned. But by then my mental processes had been completely derailed.

I left graduate school for a few years, trying out law school, the army, and employment at a grinding wheel company — a real, honest-to-gosh job shop. I also acquired a family. When I returned to Harvard in 1958, I knew it was the program in *applied* mathematics at the Computation Laboratory that I wanted to enroll in.

'Commander' Howard Aiken was still in charge of the Comp Lab, though

he would shortly take early retirement. The Lab was still turning out volume after volume of tables of mathematical functions (hard as that is to believe in this age of pocket calculators). Aiken took pride in the bound volumes that issued from the Lab. He insisted that technical reports be assembled into leather-bound volumes for presentation to the research sponsor. And he tolerated no errors. Once, when he found Dick Karp had included an errata sheet, Aiken directed his secretary to razor out the offending page from the already bound volumes.

Needless to say, under Aiken's direction the Lab was orderly. And it did have a nice ambiance. The Mark I and Mark IV computers, though quite obsolete, ran 24 hours a day. Each machine was as big as a small house. I enjoyed walking around inside the computers, in order to admire the machinery. The Mark I was much the prettier of the two machines. Great masses of colored wires were neatly bundled into giant cables. And the hundreds of relays made a soothing clickety-clack sound. (The machine had been retrofitted with a relay unit capable of multiplying two 23-decimal digits in seven seconds. I was told that, at its dedication, Aiken declared that the world would never need a faster multiplier.)

My fellow graduate students at the Lab were a capable and interesting lot. I recall that one of them, Fisher Black, was an intrepid participant in experiments with mind-altering drugs, then being conducted in the psychology department under the direction of Professor Timothy Leary. Fisher is today a Wall Street guru, famed for the Black-Scholes model for futures pricing. Another student eventually joined the FBI's Ten Most Wanted List for a brief period. But most of us went on to make our marks in more prosaic ways.

Once again I took a course from Ken Iverson, this time in 'Operations Research'. While I had been away from school, Ken had become obsessed with the invention of a new algorithmic notation. With some grumbling, he led the class through the simplex method, using conventional notation. Then one day he arrived in the lecture hall, with a particularly bright face. "Today, I will show you the *right* way to describe the simplex method!" He then drew a small box on the board, wrote a half-dozen lines of his idiosyncratic notation in the box, and added a single backward arrow to indicate iteration. He stepped back from the board. A few sentences of explanation and he was done. "Questions?" There were none. Only twenty minutes had elapsed, and he was unprepared to any more. "Well," he concluded, "This shows what happens when you have right notation. It takes much less time to explain things than you expect." He then wheeled about and exited the lecture hall a half-hour early. Only years later did I realize that I had been present at the birth of APL.

Iverson's course was my awakening. Here, at last, I found mathematics that was both interesting and useful. I had never worked on combinatorial

problems before, but I found they were something I could really *do*. I read some graph theory, at that time a rather obscure specialty, and I liked it. Ralph Gomory gave a lecture at Harvard on his revolutionary integer linear programming algorithm, and I was intrigued. Willard Eastman wrote a PhD thesis on the solution of the traveling salesman problem by a method that later came to be known as branch-and-bound. I applied the same method to the quadratic assignment problem, and that became a major part of my own PhD thesis.

I recall my years at Harvard with great fondness, and will always remember the kindness and encouragement of the young faculty, particularly Willard Eastman, Robin Esch, Ken Iverson and Gerry Salton.

The Sixties

In 1962 I became assistant professor of electrical engineering at the University of Michigan, where I started off by doing research and teaching in switching theory. At the time, Bob Thrall and George Minty were in the Mathematics department, and Art Burks, Bernie Galler, John Holland, and Anatol Rapaport in the Communication Sciences department. Together, they provided a wealth of stimulation. Thrall took me under his wing and generously made me co-director of the annual Michigan summer conference on operations research. We invited an impressive group of people to lecture at these short courses, including such luminaries as Michel Balinski, Richard Bellman, George Dantzig, Jack Edmonds, and Ray Fulkerson.

One evening I was honored to have George Dantzig as a guest at my house. At the time George was in the midst of developing his concept of a 'compact city', later published as a book with Thomas Saaty. Earnestly, he described his ideas for housing millions of people in what sounded like a gigantic parking structure with forty-foot-high ceilings. People would work in shifts, he explained, and all public facilities would be open around the clock. Combinations of moving walkways and elevators would enable anyone to get from any point in the city to any other point in minutes. For anyone who wanted to experience sunlight, there would be an elevator to a grassy park on the roof of the city. During this discourse, my wife became visibly agitated. A sputtered series of objections ensued, and George responded calmly to all of them. Finally, in exasperation my wife demanded, "And what does *your* wife think of this?" With great sweetness, he smiled and replied. "Compared to her," he said, "you are a kind soul!"

George Dantzig and T. C. Hu organized the most rewarding technical meeting I have ever attended. This was a workshop on 'Integer Programming and Network Flows' held at Lake Tahoe in 1965. Here I learned about matroids, nonbipartite matching, the Chinese Postman's problem, polyhedral characterizations of combinatorial problems, and the

importance of estimating the computational efficiency of algorithms. I heard of more applications of network flow theory than I had dreamed existed. Jack Edmonds was involved in everything. What he had not created, he had improved. I was dazzled by his brilliance.

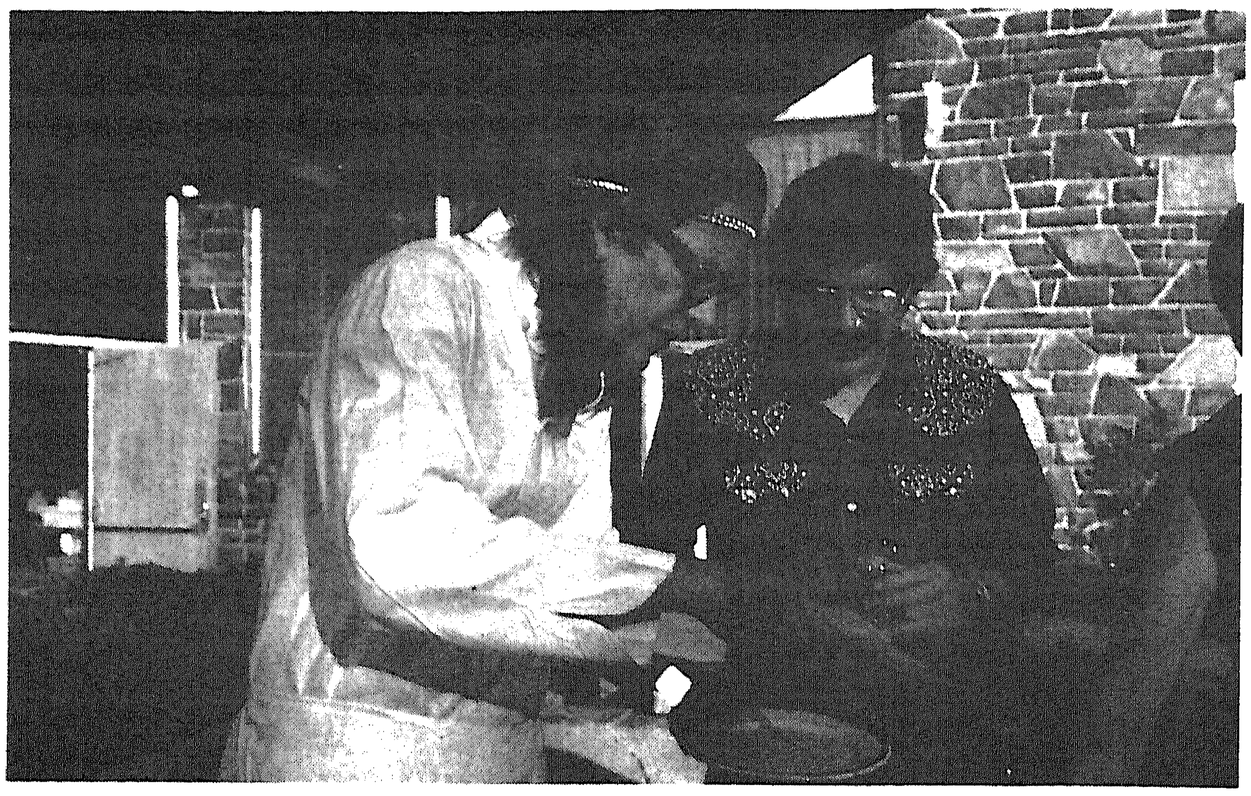
Jack's papers were an endless source of inspiration and exasperation. The theorems were wondrous, but many proofs were inscrutably succinct or seemed entirely lacking. Algorithms were implied, but were far from explicit. Gradually I came to appreciate that bipartite matching was simply a very special case of matroid intersection, and to see that matroid intersection algorithms could be modeled after classic matching and assignment algorithms. When I described such a matroid intersection algorithm in a conference paper, Jack was not amused. "It's all there in my papers!" he insisted. I am sure he was right.

The summer of 1966 I made my first trip to Europe to give a paper at a graph theory meeting in Rome. When I read the program, I was much surprised to find that Dantzig and I were to give papers in the same session on exactly the same problem — the computation of optimal ratio cycles in graphs. But I was also much relieved to find that our solutions to the problem were entirely different: Dantzig employed a variant of the simplex method, whereas I used shortest path computations as the basis for bisection search.

I felt that my contribution to ratio-cycle problem was quite modest. (Years later, Nimrod Meggido provided a vastly more sophisticated and elegant solution, also based on bisection search.) But apparently my paper attracted some attention. A couple of years after the Rome meeting, I was invited to participate in an Italian conference on periodic control. I was dumbfounded. "I don't even know what 'periodic control' is," I demurred. "We want you to talk about the ratio cycle problem," I was told. "Your Rome paper is *the* fundamental result for the discrete case." Naturally, I accepted the invitation to speak, happy that my modest contribution had earned me not one, but two, European junkets.

The academic year 1968-69 was my first sabbatical. I spent it in Berkeley, where I began serious work on my book on combinatorial optimization. Little did I know that this was the beginning of a seven-year ordeal; *Combinatorial Optimization: Networks and Matroids* would not be published until 1976.

Two particularly notable events occurred in the spring semester of 1969, one political and the other technical. Governor Ronald Reagan ordered a military occupation of Berkeley by 2,000 National Guard troops. I became a member of the Berkeley 483, a group of folks arrested *en masse* for the heinous offenses of illegal assembly, blocking a public sidewalk, and creating a public nuisance. I have no documentary evidence of my arrest, because all charges against the 483 were eventually dropped and arrest records



Edmonds, Lawler (Banff, 1977)

expunged. But Dick Karp can bear witness to my brief career as a political criminal; he accompanied my wife when she bailed me out of jail in the morning.

The technical event of the spring of 1969 was Dick Karp's discovery of the theory of *NP*-completeness. Dick commented at the time that the hard part had been done by Steve Cook. Yet it is truly awesome that Dick was able to perceive the significance of Cook's work, and to apply Cook's theorem as he did. I became a handmaiden, contributing some *NP*-completeness proofs for Dick's landmark 1971 paper.

The Seventies

The new decade began with mixed signals. I joined the Berkeley faculty in Electrical Engineering and Computer Sciences. I no longer had to contend with the malorganization of Computer Science at Michigan, and that was good. Berkeley had its own CS turf war, but it was soon resolved, and that was good. I had a war in my own household, and that was very bad.

The new theory of *NP*-completeness had profound implications for both the CS and OR communities. It spawned much new research into complexity theory, including the study of the polynomial hierarchy, *P*space-completeness, and provably exponential problems. Even the purest of mathematicians came to appreciate that there were issues of intellectual substance to be found in theoretical computer science. As a practical matter, we now had a useful new tool to guide us in the development of algorithms for combinatorial problems. However, the Operations Research community was slow to grasp the implications of the new theory. For a period of two or

three years, I made a mini-career of visiting ORSA-TIMS meetings, spreading the gospel. I even organized a rump session at Bernard Roy's 1974 Versailles symposium on Combinatorial Programming. Unfortunately, I gave a rather lame presentation and probably won no converts.

However, two Dutchmen at the Versailles meeting were already converts. Jan Karel Lenstra and Alexander Rinnooy Kan knew *NP*-completeness. Even more impressive, they knew my own work on scheduling theory. I sensed immediately that these eager young graduate students were winners! They easily coaxed me into being a thesis examiner for each of them. This resulted in many visits to Amsterdam, and the beginning of a long and fruitful collaboration. Best of all, Jan Karel and Alex introduced me to their circle of Amsterdam friends, and within this circle I found a new wife.

Jan Karel occupied a dingy office in the old Mathematisch Centrum, adjacent to the Amstel brewery. Amid pungent odors of malt and hops, we set about revising the classification system for scheduling problems devised by Conway, Maxwell and Miller. Soon our technical conversations were laced with shorthand. "We know the status of one-deejay-sum-ceejay and one-arejay-sum-ceejay," I would say, "but what about one-preemption-arejay-deejay-sum-ceejay?" I couldn't speak Dutch, but I *could* speak a new argot!

Jan Karel decided to make Alex an unusual gift on the occasion of his thesis defense: a computer-tabulated listing of the thousands of types scheduling problems we had classified, with each problem annotated with its complexity status. Ben Lageweg, a master programmer, provided a listing. We then observed that it would be amusing to generate a sublisting of the maximal problems solvable in polynomial-time, the minimal *NP*-hard problems, and the minimal and maximal open problems. When Ben obliged with this new listing, we were enthralled with the results. There were numerous open problems that we had previously overlooked. In the next few days, a group of us — Jan Karel, Ben, Dick Karp and I — had the pleasure of knocking off one obscure problem type after another in rapid succession. Alex, of course, was denied the satisfaction of licking this cream. After all, we were preparing a surprise for him!

About this time I made an innocent suggestion to Alex and Jan Karel. "Why don't we write a book on scheduling theory?" I asked. "We have your two theses and my own papers. We can just get out the scissors and pastepot. It shouldn't take much time!" Book-writing, like war-making, often entails grave miscalculations. Fifteen years later, *Scheduling* is still unfinished. (But, with David Shmoys as an added coauthor, our blockbuster *will* be published in 1992.)

The decade of Seventies ended with my once again playing the role of handmaiden to a significant discovery. In January 1979, Rainer Burkard

brought the reprint of a *Doklady* paper to Oberwolfach. The author, somebody named Khachian, purported to have a polynomial-time algorithm for solving linear programming problems. The next week in Amsterdam, sitting at the elbow of Milan Vlach, I scribbled out a translation. I sent this to a long list of people, and before too long, Peter Gács and László Lovász had supplied Khachian's missing proofs and verified his results. Then the uproar began. One thing led to another, and before long newspapers the world over were acclaiming the cataclysmic result: the traveling salesman problem had been solved! I gained a brief moment of notoriety when the *New York Times* cited me on its front page as the Discoverer of the Discoverer. (For those interested in more details of this bizarre episode, I will be happy to send a reprint of my article, 'The Great Mathematical Sputnik of 1979'.)

The Eighties

The Eighties began with a spectacular enrollment boom in computer science. I found myself presiding over an undergraduate program that was mushrooming all out of proportion; it seemed that every Berkeley undergraduate desperately wanted into the CS major. Almost daily, for three years, students left my office in tears. I worried that someday a rejected student would head straight for the Campanile, and following a quick elevator ride, end his life — splat — beneath my window. I even had a call from a US Senator who tried to intervene on behalf of his daughter-in-law. (I listened attentively to the senator, assured him I would 'give serious consideration' to his plea, and rejected her.)

Two or three times, I taught a special topics course in Sequencing and Scheduling. I've always found it hard to force myself to prepare lectures on my own work, so I often got stuck on details. But the students seemed to get the idea anyway. At least two, Barbara Simons and Chip Martel, wrote doctoral theses on open problems I posed in lecture.

A student's first significant research result is always a treasure. A student's first explanation of the same result is something else again. When Chip Martel first dirtied my blackboard with his algorithm for cue-preemption-arejay-deejay-ellmax, I could make neither head nor tails of his ideas. But after much yammering and yakking, we massaged his procedure into a 'polymatroidal' network flow algorithm, thereby transforming a small treasure into a much greater one. The polymatroidal network flow model we developed provides the true *right* way to formulate and solve matroid and polymatroid optimization problems — or so I maintain. (Later, we learned that Chip's scheduling problem can be solved by ordinary network flow techniques. But this is an irrelevant irony. More humbling is the fact that, in the real world, there is about as much of a market for polymatroid optimization algorithms as there is for theorems in mathematical logic.)

[282]

During the 1980's, I was blessed with a number of other outstanding doctoral students. David Shmoys co-edited, with Alex and Jan Karel and myself, the highly successful *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Marshal Bern collaborated with me in the development of a theory of linear-time algorithms for decomposable graphs. Arvind Raghunathan and Huzur Saran, together with Rajeev Motwani, obtained brilliant results on the linkless and knotless embedding of graphs in 3-space.

Other Berkeley students were responsible for two of the most significant developments in mathematical programming in the 1980's. Narendra Karmarkar began work on interior point methods while completing a thesis on another topic under Dick Karp's supervision. Andrew Goldberg, after enrolling at Berkeley for a period, returned to MIT to invent his marvelous new network flow algorithms.

I think there is a moral to be drawn from the work of Karmarkar and Goldberg: No matter how worked-over an area may be, there may still be gold to be mined. Interior point algorithms are not new, and penalty-function computations date back at least to the 1960's. But Karmarkar's work has demonstrated that there is much to be gained by resuscitating these old and discarded methodologies. Numerous very capable investigators have worked on network flow algorithms over the years. Yet Goldberg was able to achieve a breakthrough.

The Nineties

As the 1990's begin to unfold, I find myself turning to computational problems in biology, specifically to the analysis of DNA and protein sequences and the reconstruction of evolutionary histories from molecular data. I believe that in the 1990's, biology will provide a fruitful area of application for combinatorial optimization, and for mathematical programming as a whole. (Perhaps someone in the mathematical programming community will unlock the secret of protein folding?)

We are now able to solve combinatorial problems of a size that would have seemed almost inconceivable even ten years ago. For example, it is now possible to obtain exact or near-optimal solutions to traveling salesman instances with thousands of cities. As the ratio of computing power per dollar increases (at the rate of about 30% a year), the size of solvable problem instances will, of course, continue to increase. However, as parallel architectures become commonplace, it will become increasingly apparent that memory space, not processor cycles, is the bottleneck. It will become increasingly important for algorithm designers to understand the problems of coordination and synchronization of parallel processes.

Software systems for mathematical programming will become more

adaptive, i.e., more 'intelligent', in that these systems will be enabled to make selections from a suite of algorithmic possibilities. We will learn how to design systems that are 'meta-algorithmic', in the sense that they will employ algorithms to design algorithms. Very large collections of data will become readily accessible, either over high speed networks or in the form of CD ROM's, giving mathematical programmers a greatly enhanced opportunity to experiment with realistic data. Visualization of data will become an increasingly important issue, particularly with the advent of a new generation of flat color displays, which is likely to occur about five to seven years from now.

I claim these speculations are neither particularly original nor insightful. To the extent the reader finds them mundane, the reader must accept that certain conclusions are inevitable. Algorithm developers will need to be increasingly computer-sophisticated. Academic departments of Operations Research will have to find ways to dove-tail their programs with Computer Science, else they must remain a step behind, and resign themselves to teaching the applications of software that is developed elsewhere.

A final note, on the $P = ?NP$ question. In 1977, I bet Alex a case of fine champagne that the issue would not be resolved by 1989. I predict that the Decade and the Century will end without resolution of the issue. Sad to say, but it will be many more years, if ever, before we really understand the Mystical Power of Twoness (or what Jan Karel calls the Magical Power of Threeness): 2-SAT is easy, 3-SAT is hard, 2-dimensional matching is easy, 3-dimensional matching is hard . . . Why? Oh, why?

History of Mathematical Programming; A Collection of Personal Reminiscences,
edited by J.K. Lenstra, A.H.G. Rinnooy Kan, and A. Schrijver,
CWI & North-Holland, Amsterdam, 1991, pp. 97-106. Reprinted by permission.

Algorithmica (1994) 12: 327–344
Reprinted by permission

Algorithmica

Sublinear Approximate String Matching and Biological Applications[1]

W. I. Chang[2] and E. L. Lawler[3]

Abstract. Given a text string of length n and a pattern string of length m over a b-letter alphabet, the *k differences approximate string matching problem* asks for all locations in the text where the pattern occurs with at most k differences (substitutions, insertions, deletions). We treat k not as a constant but as a fraction of m (not necessarily constant-fraction). Previous algorithms require at least $O(kn)$ time (or exponential space). We give an algorithm that is sublinear time $O((n/m)k \log_b m)$ when the text is random and k is bounded by the threshold $m/(\log_b m + O(1))$. In particular, when $k = o(m/\log_b m)$ the expected running time is $o(n)$. In the worst case our algorithm is $O(kn)$, but is still an improvement in that it is practical and uses $O(m)$ space compared with $O(n)$ or $O(m^2)$. We define three problems motivated by molecular biology and describe efficient algorithms based on our techniques: (1) approximate substring matching, (2) approximate-overlap detection, and (3) approximate codon matching. Respectively, applications to biology are local similarity search, sequence assembly, and DNA-protein matching.

Key Words. Pattern matching, Edit distance, Suffix tree, Lowest common ancestor, Chernoff bound, Computational biology.

1. Introduction

1.1. Motivation. Pattern matching is one of the classical problems of computer science, and for exact matching many fast algorithms are known. However, in many applications a nonexact or *approximate* match is still meaningful. In the field of molecular biology, for example, a genomic database is likely to contain DNA or protein sequences of many individuals; therefore, matching a piece of DNA against the database must allow a small but significant error due to polymorphism (differences in DNA among individuals of the same species). Furthermore, current DNA sequencing techniques are not perfect, and experimental error can sometimes contribute as much as 5–10% inaccuracies. This problem persists even when two copies of the *same* DNA are compared. The degree to which two sequences match, and the plausibility of a particular alignment

[1] This work was supported in part by NSF Grants CCR-87-04184 and FD-89-02813; by the Human Genome Center, Lawrence Berkeley Laboratory, supported by the Director, Office of Health and Environmental Research, of the U.S. Department of Energy under Contract DE-AC03-76SF00098; and by Department of Energy Grants DE-FG03-90ER60999 and DE-FG02-91ER61190. Earlier versions of this paper appeared as [8] and part of [5].
[2] Cold Spring Harbor Laboratory, P.O. Box 100, Cold Spring Harbor, NY 11724, USA.
[3] Computer Science Division, University of California, Berkeley, CA 94720, USA.

Received September 5, 1991; revised September 2, 1992. Communicated by Alberto Apostolico.

between them, have recently received some statistical analysis of significance (see [24] and [45]). The interpretation of the DNA distance metric as evolutionary distance between species has also been proposed. This is relevant to the construction of evolutionary trees and models [23]. The proposed *sequence-tagged sites* (STS) genomic map database of the Human Genome Project [34], as well as several gene mapping strategies, require the approximate matching of DNA sequences.

We have focussed on the following model of approximate matching: Given a pattern of length m and a text of length n, consisting of letters from an alphabet of size b, and an integer parameter k, find all occurrences of the pattern in the text, allowing in a (partial) match at most k *differences* (substitutions, insertions, or deletions). We treat k not as a constant but as some fraction of m (not necessarily constant-fraction). The text is assumed to be given *on-line* and to be scanned sequentially; the space requirement should be linear in the length of the *pattern.* We note that this simple model has a rich history and interesting combinatorics, and is a natural starting point before more complex, parametric cost functions are to be considered (e.g., [18] and [46]).

1.2. Exact Matching. The problem of locating the first occurrence or all occurrences of a pattern word P in a long text string T has its roots in the implementation of text editors and information retrieval systems in the 1960s to early 1970s. Particularly noteworthy are efficient algorithms of Knuth, Morris, and Pratt (*KMP*) [28], Boyer and Moore (*BM*) (also see [28]), and Karp and Rabin [26]. Each of these algorithms preprocesses the pattern and then scans the text string on-line. Slightly earlier, Weiner [47] solved the complementary problem where the long text string T is fixed and keywords which are given on-line must be located within it, in time proportional to the length of each keyword. Specifically, he constructed a linear-size finite state automaton to report the first occurrences of all substrings of the given text string. An auxiliary data structure used by Weiner, called the *position tree*, soon became widely used and a particular variant of it, the *suffix tree* of McCreight [32], is most widely known for its role in fast implementations of the Lempel–Ziv data compression algorithm (see, for example, [13] and [38]). The suffix tree of a string is a compressed *digital trie* of its suffixes, whose edge labels are substrings (denoted by indices). Recent applications of suffix trees include methods for finding biologically significant motif patterns in DNA [17] and elegant linear-time algorithms for computing:

(1) The longest substrings common to k out of m strings, for all k [21].
(2) The pairwise maximum exact overlaps of m strings [19].

It is interesting to note, however, that no one seemed to know how to use suffix trees to solve the basic string-matching problem in a simple way, using $O(n)$ time and $O(m)$ space, where n, m are respectively the lengths of text T and pattern P (the same time and space constraints as in *KMP*, assuming a fixed, finite alphabet). Such results have been reported for *directed acyclic word graphs* (*DAWG*) [10] and for *suffix automata* [11] which differ from suffix trees in an essential way:

their edge labels are single letters, not strings. We have observed that a linear-time algorithm can be derived from the incremental suffix tree construction given in [32]. (Ukkonen [43] made this observation independently.) This is pedagogically satisfying and unifies several previous results:

(1) One-pass multiple-keyword search [1], using a suffix tree of $P_1\$_1P_2\$_2\cdots$, where P_i are keywords and $\$_i$ are distinct delimiters. Insertions and deletions of keywords are easily accommodated if no keyword contains another.
(2) Sublinear exact matching similar to *BM* but superior when the alphabet is small or m is large.

This special case ($k = 0$) of our sublinear approximate matching algorithm examines on average only $O((n/m)\log_b m)$ letters of the text (b is alphabet size). Such a result was first stated by Knuth *et al.* [28], but their version has to run the basic *KMP* in parallel in order to ensure linearity. Our method is inherently linear (in the worst case) and is easily generalized to handle multiple keywords. Recently Park [35] has found yet another way to achieve the same expected running time (for one keyword), using a suffix tree of the *reverse* of the pattern and facts from string combinatorics.

Our technique uses in a crucial way a set of pointers, called *suffix links*, originally used by McCreight [32] in suffix tree construction. (Subsequent papers paid no attention to these pointers.) By simultaneously scanning the text and walking in the suffix tree of the pattern, our algorithm computes the longest substring of the pattern to appear at each position of the text. We call this information *matching statistics*. In Section 2.1 we give an abstract defintion of the suffix tree, as opposed to definition by construction [32]. This greatly simplifies the description and analysis of the matching statistics algorithm (Section 2.2), which is then used to simplify and to improve upon the current best approximate matching algorithm that is based on dynamic programming (Section 2.3).

1.3. Approximate Matching. Beginning in the 1980s, genetics and DNA sequence analysis research provided the impetus for advances in nonexact string matching. The *k differences approximate string-matching problem* specifies, in addition to T and P, the parameter k of differences (substitutions, insertions, deletions) allowed in a match. The problem is to find all locations in the text where a match *ends*. (So the output is of linear size. This is equivalent to finding where matches begin, by reversing the strings.) Substitutions and *indels* (insertions and deletions) are *edit operations*, and the minimum number needed to transform one string to another is called the *edit distance* or the Levenshtein distance [31].

The classical dynamic programming algorithm of Sellers [41] computes (column by column) an $m + 1$ by $n + 1$ table whose entry $D(i, j)$ is the minimum number of edit operations necessary to transform the length i *prefix* of the pattern into *some* text fragment ending at the jth letter. (Boundary conditions are $D(i, 0) = i$ and $D(0, j) = 0$. There is a match ending at text position j if and only if entry

$D(m, j)$ is at most k.) There is a simple recursive formula giving each entry in terms of the three adjacent entries above or to the left:

$$D(i, j) = \min\{1 + D(i-1, j),\ 1 + D(i, j-1),\ I_{ij} + D(i-1, j-1)\},$$

where $I_{ij} = 0$ if $P[i] = T[j]$; $I_{ij} = 1$ if $P[i] \neq T[j]$. The three expressions in the min correspond respectively to deleting $P[i]$ from the pattern, inserting $T[j]$ into the pattern, and substituting $T[j]$ for $P[i]$. A continuity argument implies (1) adjacent entries along rows and columns differ by at most one; that and the recurrence imply (2) forward diagonals ($\searrow$) are nondecreasing and adjacent entries differ by at most one. More recent methods by Ukkonen [42], Ukkonen and Wood [44], Landau and Vishkin [29], [30], Galil and Giancarlo [14] (survey and refinements), and Galil and Park [15] take advantage of these geometric properties in order to compute $O(kn)$ instead of mn entries.

The locations of the first $k + 1$ *transitions* (x to $x + 1$) along each forward diagonal are sufficient to characterize the solution, by the nondecreasing diagonal monotonicity property (2). Landau and Vishkin's algorithm (*LV*) [29], [30] computes each transition in constant time. Two bottlenecks kept this $O(kn)$ algorithm impractical:

(1) Computing the *lowest common ancestor* (*LCA*) of two nodes of the suffix tree of P required a complicated algorithm. However Schieber and Vishkin [40] found a new constant-time *LCA* algorithm, which we implemented efficiently [4].
(2) It was not known how to compute matching statistics using only the suffix tree of P; however, we now have a simple solution. We have reduced the space requirement of *LV* to $O(m)$ in addition to input/output; it now appears to be the best among $O(kn)$ worst-case algorithms.

When k is a constant-fraction of m, $O(kn)$ dynamic programming methods represent no theoretical improvement over the classical $O(mn)$ algorithm. In this paper we take a different approach. When the error tolerance k is not too large relative to m, $O(n)$ *expected time* (*LET*) or even faster algorithms are possible. The threshold on k for linear expected time is *logarithmic fraction*: $k < k^* = m/(\log_b m + c_1) - c_2$ (for suitable constants c_i). The choice $c_1 = 5.6$ used in the proof is pessimistic; experiments suggest $c_1 = 3$ suffices for $b = 2$; $c_1 = 2$ suffices for $b \geq 4$. In practice, for m in the hundreds the error thresholds k^* in terms of percentage of m are 35 ($b = 64$), 25 ($b = 16$), 15 ($b = 4$), and 7 ($b = 2$).

Our main tool is matching statistics. Computed in a single left-to-right scan of the text, this information is used to eliminate stretches of text where a match cannot possibly occur. Only rarely does the algorithm resort to a dynamic programming subroutine, so the average running time is linear in the length of the text sequence. (An algorithm similar to *LET*, discovered independently but without analysis of threshold, was recently implemented by Jokinen *et al.* [22], and was the fastest for large m and medium k among algorithms they tested.) For $k < k^*/2 - 3$, stretches of text can be skipped altogether (*SET*); this is sublinear

in the sense of Boyer–Moore. When $k = o(m/\log_b m)$ the expected running time is $o(n)$, truly sublinear. Indeed, approximate matches in random text are so infrequent that running time is dominated by the effort to verify mismatches. Our algorithms do so in linear or even sublinear expected time.

Algorithms *LET* and *SET* are described in Sections 3.1 and 3.2. In each case the total work of the dynamic programming subroutine is at most what it would cost to apply the subroutine to the entire text all at once. It therefore does not hurt to apply our algorithms, even for k greater than k^* or $k^*/2 - 3$. These algorithms can be viewed as *approximate string matching via data compression* (compare with [38]). Finally, we discuss recent developments in Section 3.3.

1.4. Biological Applications. In Sections 4.1–4.3 we apply our techniques to several problems motivated by biology. We feel the problem formulations we have chosen are faithful to the original tasks, and hope our algorithms will become useful to biologists.

(k, l) APPROXIMATE SUBSTRING MATCHING PROBLEM. Given T of length n, P of length m, and parameters l and k, find all i such that some substring of T ending at i matches a length l substring of P to at most k differences.

This is a general method for finding *local similarities*. We should mention that matching statistics is already a useful local similarity measure (see [2], [10], and [36]). It can be used to obtain a summary of *all* exact matches between fragments of the text and pattern. A natural extension would be to assign weights to matched words based on frequency. For this purpose, the suffix tree can be any dictionary with suitable statistics.

ρ APPROXIMATE-OVERLAP DETECTION PROBLEM. Given P of length m, Q of length n, and parameter ρ, find all l such that the length l prefix of P matches a suffix of Q to within ρl differences.

Sequence assembly is the task of matching pieces of DNA that are known to be overlapping fragments of a single long DNA sequence. In this application approximate-overlap is done for every pair of fragments and accounts for most of the computing time (98% according to Peltola *et al.* [37]). In this application, the experimental sequencing error contributes about 5% differences to overlapped regions. This is few enough errors ($\rho = 0.05$) for our algorithm to be sublinear on average. Recently Huang [20] has used a simplified (t-tuple) implementation of our method to speed up sequence assembly in a program called *CAP*.

APPROXIMATE CODON MATCHING PROBLEM. $T \in \Sigma^*$ and $P \in \Pi^*$ are from different alphabets. The translation map tr: $\Sigma^t \to \Pi$ via t-tuples is not necessarily one-to-one (i.e., different *codons* may translate to the same letter). A string σ over Σ *translates* to a string π over Π if $\mathrm{tr}(\sigma_1\sigma_2\cdots\sigma_t) = \pi_1$, $\mathrm{tr}(\sigma_{t+1}\cdots\sigma_{2t}) = \pi_2$, etc. Find all locations i such that some substring of T ending at i is within edit distance k of a string that translates to P.

The difficulty comes from having both multiple encodings and framing errors caused by insertions and deletions. In DNA-protein matching, each codon is a 3-tuple of nucleotides (A, C, G, or T), and is translated into one of 20 amino acids. We describe an efficient algorithm based on our methods.

2. Data Structures

2.1. Suffix Trees. Let $P[1, \ldots, m]\$$ be a string over a fixed finite alphabet $\Sigma \cup \{\$\}$ where \$ is a special end-marker that occurs nowhere else. The following is a precise characterization of the suffix tree of a string P, denoted $S(P\$)$.

Substrings of $P\$$ are called *words*; a word w is *branching* if there are different letters x, y such that wx and wy are words. Let w be a word; floor(w) denotes the longest prefix of w that is a branching word; ceiling(w) denotes the shortest extension of w that is either a branching word or a suffix of $P\$$. The following are evident: floor and ceiling are well defined; if w is branching, then floor(w) = ceiling(w) = w; and the empty string, denoted Λ, is a branching word.

If string u is a prefix of string v, $u^{-1}v$ denotes the suffix of v which if appended to u gives v. If string v is not an empty string, v_1 denotes the first letter of v.

Let us define $S(P\$)$. There is a one-to-one correspondence between nodes of $S(P\$)$ and a set of words:

$$\text{root} \leftrightarrow \Lambda,$$

$$\{\text{internal nodes}\} \leftrightarrow \{\text{branching words}\},$$

$$\{\text{leaves}\} \leftrightarrow \{\text{suffixes}\}.$$

The relation *is prefix of* defines a natural partial order on the above set of words that is a tree; this in turn defines the edges of $S(P\$)$. That is, node u is the parent of node v iff (branching word) u is a proper prefix of (branching word or suffix) v, and there is no branching word w that is a proper extension of u and a proper prefix of v. This directed edge from u to v is labeled with a triple (x, l, r) chosen such that $P\$[l] = x$ and $u^{-1}v = P\$[l, \ldots, r]$ (the choice of l, r may not be unique). We write $\text{son}(u, x) = v$, $\text{first}(u, x) = l$, and $\text{len}(u, x) = r - l + 1$, as well as refer to nodes by their corresponding words.

In addition define the function *shift*: {internal nodes other than the root} $\rightarrow$ {internal nodes} given by $\text{shift}(w) = w_1^{-1}w$ ($w \neq \Lambda$). McCreight [32] constructs the suffix tree $S(P\$)$ and the shift function ("suffix links") in $O(m)$ time, using an algorithm very similar to the matching-statistics algorithm given below.

2.2. Matching Statistics. Define the *matching statistics* of text $T[1, \ldots, n]$ with respect to pattern $P[1, \ldots, m]$ to be an integer vector $M(T, P)$ together with a vector $M'(T, P)$ of pointers to the nodes of suffix tree $S(P\$)$, where $M(T, P)_i = l$ if l is the length of the longest word (substring of $P\$$) matching exactly a prefix of $T[i, \ldots, n]$; and $M'(T, P)_i$ points to the node $\text{ceiling}(T[i, \ldots, i + l - 1])$. The goal

is to compute M and M' in $O(n + m)$ time and as little additional storage as possible. In most applications M and M' are not stored all at once. Landau and Vishkin [30] reduced this problem to constructing the suffix tree of $P\$_1 T\$_2$, but that requires either $O(n + m)$ additional space or computing everything twice (by building overlapping $O(m)$-size suffix trees). Galil and Giancarlo [14] gave an automata-based algorithm which scans the text one way and then in reverse; for it to run in $O(m)$ space the text must be scanned four times (again by working on overlapping $2m$-size blocks of text). Although satisfactory automata-based solutions exist [10], [11], since the approximate matching algorithm given in [14] already uses the suffix tree $S(P\$)$ in a different part of the algorithm, it makes sense to try to compute matching statistics directly, using only $S(P\$)$. The following is a very simple linear-time algorithm that computes $M(T, P)_i$ and $M'(T, P)_i$ in order of increasing i, in a single left-to-right scan of the text. Very briefly, the longest match starting at the first position is found by walking down the tree, matching a letter at a time. Subsequent longest matches are found by following suffix links and carefully going down the tree.

MATCHING STATISTICS ALGORITHM (Comments). Variables i, j, k are indices into the text string; the ith iteration of step 3 computes $M(T, P)_i$ and $M'(T, P)_i$; position k of the text had just been scanned, and j is some position between i and k. At all times the following invariant is maintained:

(i) $T[i, \ldots, k-1]$ is a word; $T[i, \ldots, j-1]$ is a branching word.

After step 3.1 the following becomes true:

(ii) $T[i, \ldots, j-1] = \text{floor}(T[i, \ldots, k-1])$ and corresponds to the node v.

After step 3.2 the following becomes true as well:

(iii) $T[i, \ldots, k]$ is not a word.

The key observation is that if $j < k$ after step 3.1, then $T[i, \ldots, k-1]$ is not a branching word, so neither is $T[i-1, \ldots, k-1]$. Indeed, as substrings of P they must have the *same* unique single-letter word extension. We know from iteration $i-1$ that $T[i-1, \ldots, k-1]$ is a word but $T[i-1, \ldots, k]$ is not, so $T[k]$ cannot be this letter. Hence the match cannot be extended.

Together invariants (i) and (iii) imply $M(T, P)_i = k - i$. Step 3.4 uses the suffix link to go from v to shift(v) if v is not the root. Note that (i) is maintained.

```
1.   construct suffix tree S(P$) by McCreight's algorithm [32] (see below)
2.   let v ← root; j ← 1; k ← 1
3.   for i ← 1 to n do
3.1.    while (j < k) and (j + len(v, T[j]) ≤ k) do
           v, j ← son(v, T[j]), j + len(v, T[j])
        end while
```

```
3.2.  if (j = k) then
        while son(v, T[j]) exists and
            (T[k] = P$[first(v, T[j]) + k − j]) do
          k ← k + 1
        if (j + len(v, T[j]) = k) then
            v ← son(v, T[j]); j ← k end if
        end while
      end if
3.3.  M(T, P)_i ← k − i
      if (j = k) M'(T, P)_i ← v
      otherwise M'(T, P)_i ← son(v, T[j])
3.4.  if v is root and (j = k) then
        j ← j + 1; k ← k + 1 end if
      if v is root and (j < k) then
        j ← j + 1 end if
      if v is not root then v ← shift(v) end if
```

The positive integers i, j, and k never decrease and are bounded by n. For every constant amount of work in step 3, at least one of i, j, k is increased in value. The running time is therefore $O(m)$ for steps 1 and 2 and $O(n)$ for step 3.

As a corollary we get an *on-line* exact matching algorithm, based on suffix trees, that uses $O(m)$ space and preprocessing. That is, an exact match ending at position $k - 1$ is found before $T[k]$ is read (as soon as variable k takes on the value $i + m$ in the middle of step 3.2). Furthermore, since m iterations of step 3 take $O(m)$ time, this computation can be made *real-time* (i.e., constant throughput) by buffering m letters of the input text.

For completeness we now sketch very briefly how to construct $S(P\$)$. McCreight's algorithm builds the tree incrementally as it scans the string left to right; it has the same structure as our matching-statistics algorithm. Each iteration of "step 3" adds a leaf (suffix), and possibly a node u above it (splitting an edge below some node v); if so, the next iteration will patch the suffix link shift(u). Transition to the next iteration is accomplished by going from v to shift(v). Then walk down the tree in the same manner as matching statistics. If a node corresponding to the suffix of u already exists, set shift(u) to point to it and continue walking; otherwise, such a node is inserted and the new leaf is added below. Correctness and linear-time arguments are similar to those for matching statistics (for details see [32]). Although a node may have as many edges as there are letters in the alphabet, the tree has altogether a linear number of edges. In practice, suffix trees are *hash coded* in order to be compact; look-up operations son(u, x) are constant time modulo hashing.

2.3. The Landau and Vishkin Algorithm. Vectors M and M' are the main ingredients needed to speed up approximate string matching. Define "jump" $J(i, j)$ as the length of the longest common prefix between $P[i, \ldots, m]$ and $T[j, \ldots, n]$. Jumps can be computed in constant time using the identity $J(i, j) = \min(M_j$, length of word corresponding to $LCA(M'_j$, leaf $P[i, \ldots, m]\$))$. (The min is needed because

M'_j is a "ceiling" and can be the leaf's ancestor.) The following paragraph describes succinctly the dynamic programming component of LV. Recall that the locations of the first $k+1$ transitions (x to $x+1$) along each forward diagonal of table D suffice in characterizing the solution.

Call cell $D(i, j)$ an entry of diagonal $j-i$. However, instead of D, compute column by column a $k+1$ by $n+1$ table L where $L(x, y)$ is the row number of the last x along diagonal $y-x$. Let us first look at D, the original table. We can define $L(x, -1) = -\infty$ because every cell of diagonal $-1-x$ is at least $D(x+1, 0) = x+1$. Likewise we can define $L(x, -2) = -\infty$. It is easy to see $L(0, y) = J(1, 1+y)$. Entry $L(x, y)$ can be computed using jumps and the three cells above and to the left: $\alpha = L(x-1, y-2)$, $\beta = L(x-1, y-1)$, and $\gamma = L(x-1, y)$, which are respectively the row numbers of the last $x-1$ in diagonals $y-x-1$, $y-x$, and $y-x+1$. It can be inferred that along diagonal $y-x$, three cells at rows α, $\beta+1$, and $\gamma+1$ are at most x (respectively, insert $T[\alpha+y-x]$ after $P[\alpha]$, substitute $T[\beta+1+y-x]$ for $P[\beta+1]$, delete $P[\gamma+1]$). Let $i = \max(\alpha, \beta+1, \gamma+1)$; then $L(x, y) = i + J(i+1, i+1+y-x)$.

In order to compute jumps in constant time Landau and Vishkin used $O(n+m)$ space (to build the suffix tree of P concatenated with T). Galil and Giancarlo [14] only needed the suffix tree of P and matching statistics, but their algorithm for the latter (called "Best-Fit" in their paper) required $O(n)$ space. In recent papers Galil and Park [15] and Ukkonen and Wood [44] gave $O(m^2)$-space "practical" algorithms. With a little care Landau and Vishkin's algorithm can be implemented very efficiently in $O(m)$ working storage, using the matching-statistics subroutine given above (keeping at any time only $O(m)$ most recent entries) and Schieber and Vishkin's very fast LCA computation [40]. In our implementation [4] only simple machine instructions are used (such as decrement and complement, but not bit-shift). Logarithm in [40] is replaced by bit magic, using a table of *reversals* of binary representation of numbers. Fewer than 60 machine instructions suffice to compute an LCA.

For a fixed, finite alphabet this appears to be the first practical algorithm to run in $O(m)$ space and $O(kn)$ time in the worst case. For general alphabet the worst-case running time of $O(kn)$ is modulo hashing in $O(m)$ space, or deterministic in $O(bm)$ space.

3. Algorithms for Logarithmic Fraction Error

3.1. Linear Expected Time. We now describe an algorithm for approximate matching that runs in $O(n)$ expected time when:

(1) T is a uniformly random string of length n over a finite alphabet of size b.
(2) The number k of differences allowed in a match is less than the threshold $k^* = m/(\log_b m + c_1) - c_2$ (c_1 and c_2 are constants to be specified later; m denotes pattern length).

The pattern P does not have to be random.

Recall that $M(T, P)_i$ denotes the length of the longest substring of P to be found beginning at position i of text T, and that it is easily computed using the suffix tree $S(P\$)$.

LINEAR EXPECTED TIME ALGORITHM (*LET*). Set $S_1 = 1$ and for $j \geq 1$ compute $S_{j+1} = S_j + M(T, P)_{S_j} + 1$. (The $(j+1)$st start position is obtained by taking the "maximum jump" at the jth start position *plus one more letter*.) For $j = 1, 2, \ldots$, if $S_{j+k+2} - S_j \geq m - k$ apply the Landau–Vishkin algorithm (*LV*) to $T[S_j, \ldots, S_{j+k+2} - 1]$ (call this "work at start position S_j"). All that is required to keep the worst-case running time bounded by $O(kn)$ is for *LV* to remember its state (i.e., the column of table L it last computed) so it can resume from that point.

CLAIM. *This correctly solves k differences approximate matching.*

PROOF. If $T[p, \ldots, p + d - 1]$ matches P and $S_j < p \leq S_{j+1}$, then this string can be written in the form $w_1 x_1 w_2 x_2 \cdots w_{k+1} x_{k+1}$ where each x_l is a letter or empty, and each w_l is a substring of P. It can then be shown by induction that, for every $0 \leq l \leq k+1$, we have $S_{j+l+1} \geq p + \text{length}(w_1 x_1 \cdots w_l x_l)$. In particular, $S_{j+k+2} \geq p + d$ which implies $S_{j+k+2} - S_j > d \geq m - k$. Therefore the above algorithm will perform work at start position S_j and thereby detect there is a match ending at position $p + d - 1$. □

Next we show that the probability of having to perform work at S_1 is small. Since the event $S_{k+3} - S_1 \geq m - k$ implies $S_{k^*+3} - S_1 \geq m - k^*$, it suffices to prove the following lemma.

MAIN LEMMA. *For suitably chosen constants c_1 and c_2, and*

$$k^* = m/(\log_b m + c_1) - c_2,$$

$$\Pr[S_{k^*+3} - S_1 \geq m - k^*] < 1/m^3.$$

PROOF. For ease of presentation let us assume (1) $b = 2$ ($b > 2$ gives slightly smaller c_i's) and (2) k^* and $\lg m$ are integers ($\lg$ denotes log to the base 2).

Let X_j be the random variable $S_{j+1} - S_j$. First note that the X_j's are independent and identically distributed since each position S_{j+1} is beyond the last letter looked at in order to compute the maximum jump at S_j. Also note $S_{k^*+3} - S_1 = X_1 + X_2 + \cdots + X_{k^*+2}$.

Since $X_1 = M(T, P)_1 + 1$ and there are at most m different words of length $\lg m + d$ in P out of $m2^d$ different strings of that length, we have

$$(*) \qquad \Pr[X_1 = \lg m + d + 1] < 2^{-d} \qquad \text{for all integer} \quad d \geq 0.$$

In particular $E[X_1] < \lg m + 3$. Let us consider the probability that $X_1 + X_2 + \cdots + X_{k^*+2} \geq m - k^*$. Let $Y_i = X_i - (m - k^*)/(k^* + 2)$. Choose $c_2 > 2$ so

$$m/(k^* + 2) > m/(k^* + c_2) = \lg m + c_1.$$

Then

$$\begin{aligned} Y_i &< X_i - m/(k^* + 2) + 1 \\ &< X_i - (\lg m + c_1) + 1 \\ &< X_i - \lg m - 3 \end{aligned}$$

provided we choose $c_1 > 4$. This implies $E[Y_i] < 0$, so we can apply the Chernoff bound technique (see [25]) to get

$$\begin{aligned} \Pr[X_1 + \cdots + X_{k^*+2} \geq m - k^*] &= \Pr[Y_1 + \cdots + Y_{k^*+2} \geq 0] \\ &\leq E[e^{tY_1}]^{k^*+2} \end{aligned}$$

for any $t > 0$. Inequality (*) is equivalent to

$$\Pr[Y_1 = \lg m + d + 1 - (m - k^*)/(k^* + 2)] < 2^{-d} \qquad \text{for all integer} \quad d \geq 0.$$

Therefore, for any $t > 0$,

$$E[e^{tY_1}] < \sum_{d=0}^{\infty} e^{t \lg m + td + t - t(m-k^*)/(k^*+2)} \cdot 2^{-d},$$

where the first term of the summation ($d = 0$) bounds

$$E[e^{tY_1} | Y_1 \leq \lg m + 1 - (m - k^*)/(k^* + 2)]$$

and the remaining terms bound

$$E[e^{tY_1} | Y_1 > \lg m + 1 - (m - k^*)/(k^* + 2)] \cdot \Pr[Y_1 > \lg m + 1 - (m - k^*)/(k^* + 2)].$$

Choose $t = (\log_e 2)/2$. Then

$$E[e^{tY_1}] < \sqrt{2}^{\lg m + 1 - (m-k^*)/(k^*+2)} \cdot \sum_{d=0}^{\infty} \sqrt{2}^{-d},$$

which gives $E[e^{tY_1}] < \sqrt{2}^{\lg m + 1 - (m-k^*)/(k^*+2) + 3.6}$; so raising to the power $(k^* + 2)$ yields

$$E[e^{tY_1}]^{k^*+2} < \sqrt{2}^{(k^*+2)\lg m - (m-k^*) + 4.6(k^*+2)}$$
$$< \sqrt{2}^{(5.6-c_1)(k^*+2) - (c_2-2)(c_1 + \lg m)}$$

after some algebra. This is less than $1/m^3$ if $c_1 = 5.6$ and $c_2 = 8$. □

From this we deduce that the expected work at start position S_1 is $O(1)$. This is true for any start position S_j because the event $S_{j+k+2} - S_j \geq m - k$ implies the event $S_{j+k^*+2} - S_j \geq m - k^*$, which occurs with probability less than $1/m^3$. There is no conditioning because the theorem of total expectation implies

$$E[E[\text{work at start position } S_j \text{ given any conditioning}]]$$
$$= E[\text{work at start position } S_j].$$

Hence the expected total work is $O(n)$.

This type of analysis can be applied to more general settings, such as a biased alphabet, if certain assumptions are made about the pattern as well as about the text. In fact, the following is clear from the above proof:

THEOREM 1. *Suppose, for all i, the random variable*

$$Z_i = (\textit{length of longest exact match at position } i \textit{ of text } T \textit{ with some substring of pattern } P[1, \ldots, m])$$

is independent from $T[1, \ldots, i-1]$ *and has the same distribution as* Z_1, *and suppose* $\Pr[Z_1 > E[Z_1] + d \cdot \text{StdDev}(Z_1)]$ *decreases exponentially in d. Then constants* c_1 *and* c_2 *exist such that, for* $k \leq k^* \doteq m/(E[Z_1] + c_1 \cdot \text{StdDev}(Z_1)) - c_2$, *k differences approximate matching takes linear expected time.*

3.2. Sublinear Expected Time. We now derive from the above an algorithm *SET* that is *sublinear* when $k < k^*/2 - 3$, where $k^* = m/(\log_b m + c_1) - c_2$ (same constants as before). Partition the text into regions of length $(m - k)/2$. Any substring of T that matches P must contain the whole of at least one region. Starting from the left end of each region R, compute $k + 1$ "maximum jumps," say ending at some position p. If position p is within R, there can be no match containing the whole of R. If p is beyond R, apply the Landau–Vishkin algorithm (LV) to a stretch of text beginning $(m + 3k)/2$ letters to the left of region R and ending at position p.

It can be shown by a trivial variation of the lemma that $\Pr[p \text{ is beyond } R] < 1/m^3$ (essentially, divide expressions $(m - k^*)$ and $(k^* + 2)$ by 2 everywhere

in the proof), so LV is seldom applied. Therefore, the expected running time of this algorithm is sublinear. On the average only $L = (k + 1)(\log_b m + O(1))$ letters are examined from each region, for a total of $2nL/(m - k)$. (A variation of this algorithm examines only $nL/(m - k - L)$ letters of text, on the average.) A combination of LET (for $k \geq k^*/2 - 3$) and SET (for $k < k^*/2 - 3$) runs in $O((n/m)k \log_b m)$ expected time, for any $k < k^*$. Since edit distance *lower bounds* Hamming distance (no indels) and longest common subsequence metric (no substitutions), our result applies to these models as well.

THEOREM 2. *The average-case complexity of k error (edit distance, Hamming distance, or longest common subsequence metric) approximate string matching is* $O((n/m)k \log_b m)$, *when k is bounded by the threshold* $m/(\log_b m + O(1))$. *In particular, when* $k = o(m/\log_b m)$ *the complexity is* $o(n)$.

3.3. Recent Developments. The choice $c_1 = 5.6$ used to prove the Main Lemma is pessimistic; experiments suggest $c_1 = 3$ suffices for $b = 2$; $c_1 = 2$ suffices for $b \geq 4$. In practice, for m in the hundreds the error thresholds k^* in terms of percentage of m are 35 ($b = 64$), 25 ($b = 16$), 15 ($b = 4$), and 7 ($b = 2$). For the purpose of comparison, the smallest k in terms of percentage of m for which there is a match (using $n = 100m$) are about 85 ($b = 64$), 72 ($b = 16$), 45 ($b = 4$), and 25 ($b = 2$).

The gaps and the mystery of these percentages were motivations for the work described in [7], where the average of row m of table D is shown to be $\Theta(m)$, and conjectured o be $(1 - 1/\sqrt{b} + o(1/\sqrt{b})) \cdot m$. Using the combinatorial lemma from [7], generalized matching statistics from [8], and a *bootstrapping* technique from [5], Chang has recently shown that constants $\rho_b > 0$ exist such that, for all $k < \rho_b m$, k-error approximate matching (edit distance, Hamming distance, or longest common subsequence metric) has average-case complexity $\Theta((n/m)(k + \log_b m))$ [6]. Very briefly, this $m^{O(1)}$ space and preprocessing algorithm extends the notion of matching statistics to nonexact matches; treats the pattern as text, and $\Theta(\log_b m)$-tuples of the text as patterns. It is very different from algorithms based on the *Four-Russians* technique, such as Ukkonen [42] (linear time but exponential space) and more recently Wu *et al.* [48]. The lower bound comes from the obvious $\Omega(kn/m)$ and Yao's classical $\Omega((n/m) \log_b m)$ result for exact matching [49]. Finally, we mention that Myers [33] has independently arrived at a sublinear algorithm based on two important techniques: Divide the pattern into length $\log_b n$ blocks, then:

(1) For each block, generate its "neighborhood" at the given error rate, and look up each element in a linear-size index of the text.
(2) Apply a "doubling trick" in order to extend a block match at the given error rate.

The success of [6] and [33] suggests the combinatorial complexity of string neighborhood is not as high as current bounds would indicate.

4. Biological Applications. Motivation for the following problems are given in the Introduction.

4.1. Substring Matching (Local Similarity)

(k, l) APPROXIMATE SUBSTRING MATCHING PROBLEM. Given T of length n, P of length m, and parameters l and k, find all i such that some substring of T ending at i matches a length l substring of P to at most k differences.

For $k < k^* = l/(\log_b m + c_1) - c_2$ (same constants as in Section 3), the linear and sublinear expected-time algorithms carry over nearly verbatim. The only changes are: $l - k$ appears instead of $m - k$, and the dynamic programming checking phase is more expensive. The Main Lemma and its proof remain correct when $(m - k^*)$ is replaced by $(l - k^*)$, which implies Pr[length of $k + 2$ maximum jumps $\geq l - k$] $< 1/m^3$. (For the sublinear version, Pr[length of $k + 1$ maximum jumps $\geq (l - k)/2$] $< 1/m^3$.) When dynamic programming is invoked, let Q denote the block of text to be matched against all length l substrings of P, and let q be the length of Q. The naive method requires $m - l + 1$ applications of $O(kq)$ dynamic programming. In practice, the following is likely to be an improvement.

Instead of matching Q against each substring of P, consider the reciprocal problem of matching P (as text) against Q. If we *bootstrap* and compute the matching statistics of P against Q, we can further reduce the problem size. Finally, dynamic programming is applied, matching Q against those substrings of P that are not yet eliminated.

4.2. Approximate-Overlap (Sequence Assembly)

ρ APPROXIMATE-OVERLAP DETECTION PROBLEM. Given P of length m, Q of length n, and parameter ρ, find all l such that the length l prefix of P matches a suffix of Q to within ρl differences.

Previous algorithms [37] are $O(\rho mn)$. (The dynamic programming algorithm for approximate string matching can be used to solve this problem: match Q against P; look for entries $D(l, n)$ in the last column that are less than or equal to ρl. Note that $D(l, n)$ is the minimum edit distance between $P[1, \ldots, l]$ and suffixes of Q. See [7] for a proof that this algorithm is $O(\rho mn)$ for random strings.) Our method builds a suffix tree of Q and computes matching statistics of P against Q. For each $i = 1, 2, \ldots, \rho m + 1$, if the length of i maximum jumps starting at $P[1]$ is at least $(i - 1)/\rho$, then apply the dynamic programming (DP) method to the prefix of P covered by those jumps. There is a constant c such that $\rho < 1/(\log_b m + c)$ implies Pr[DP is invoked for i] $< 1/m^3$. The expected running time is $O(\rho m \log_b m)$.

4.3. *Codon Matching (DNA-Protein Matching)*

APPROXIMATE CODON MATCHING PROBLEM. $T \in \Sigma^*$ and $P \in \Pi^*$ are from different alphabets. The translation map $\mathrm{tr}: \Sigma^t \to \Pi$ via t-tuples is not necessarily one-to-one (i.e., different *codons* may translate to the same letter). A string σ over Σ *translates* to a string π over Π if $\mathrm{tr}(\sigma_1\sigma_2\cdots\sigma_t) = \pi_1$, $\mathrm{tr}(\sigma_{t+1}\cdots\sigma_{2t}) = \pi_2$, etc. Find all locations i such that some substring of T ending at i is within edit distance k of a string that translates to P.

If there were no *indels* (insertions or deletions) of single letters from T, matching could be performed after translation, i.e., $\mathrm{tr}(T[1,\ldots,t])\mathrm{tr}(T[t+1,\ldots,2t])\cdots$ against P. If the translation is one-to-one, we can *reverse translate*, i.e., match T against $\mathrm{tr}^{-1}(P[1])\mathrm{tr}^{-1}(P[2])\cdots$. The difficulty is in having both types, multiple encodings and framing errors caused by indels.

The scanning phase of our algorithm computes a modified matching statistics $M_i = \max\ l$ such that $T[i,\ldots,i+l-1]$ *translates* to a substring of P. The suffix tree used is that of P over alphabet Π; t separate passes over the text string T combine to produce the matching statistics (for every possible framing). The "jump" that starts at S_j (as in *LET*) is the *farthest* jump considering all possible framings; the next start position S_{j+1} is one letter beyond: $S_{j+1} = \max\ S_j + i + M_{S_j+i} + 1$ $(0 \le i \le t-1)$. Note that this gives a conservative estimate of the error. If $k+2$ jumps span at least $tm-k$ letters of the text (i.e., $S_{j+k+2} - S_j \ge tm - k$), resort to the checking phase; otherwise there can be no match. It follows from a simple variation of the Main Lemma that if $k < m/(\log_b m + O(1))$, then $\Pr[S_{j+k+2} - S_j \ge tm - k] < 1/m^3$.

The checking phase requires a dynamic programming algorithm for approximate codon matching. Let $\mathrm{ed}'(u, v)$ be the natural generalization of edit distance to this domain where $u \in \Sigma^*$ and $v \in \Pi^*$: find the minimum x such that u is within x differences of a string that translates to v. Let

$$D'(i,j) = \min_{j'} \mathrm{ed}'(P[1,\ldots,i],\ T[j',\ldots,j]),$$

analogous to the dynamic programming formulation of approximate string matching (table D). It is clear that $D'(i,0) = ti$ and $D'(0,j) = 0$. The recurrence for $D'(i,j)$ at first appears complicated because a codon for $P[i]$ may be a noncontiguous subsequence of some block $T[j-l+1,\ldots,j]$ against which $P[i]$ is matched. However, we make the following observation: If $P[i]$ has only one codon w, let D_w be the $t+1$ by $n+1$ dynamic programming table matching T against pattern w, whose top row (boundary condition) is given by row $i-1$ of D'. Then row i of D' is just the bottom row of D_w. If, on the other hand, $P[i]$ has a set of codons, then row i of D' is found by minimizing over that set. The work to compute row i of table D' is $O(tn \cdot \text{number of codons for } P[i])$. The total work to compute D' is therefore within a factor ($t \cdot$ average multiplicity) of classical $O(mn)$ dynamic programming.

Conclusions. We have given a systematic treatment of exact and approximate string matching based on suffix trees and matching statistics, simplifying or improving upon several previous results. The main analytical result is an application of Chernoff bounds to string matching, from which we derive a sublinear expected-time algorithm when the error rate is less than logarithmic fraction. The edit distance model can be replaced by Hamming distance or longest common subsequence metric. Variations of our techniques can be used to give simple and practical solutions to a wide range of problems.

Finally, we would like to point out a difference between our approach and several heuristics used in biology which also consist of a scanning phase and a checking phase. In the absence of a clear definition of "match," these programs are tuned using positive and negative controls, typically data sets with known homology (or lack of it). Since comparison with exhaustive search is generally not available as a control, false negatives cannot be detected except by eye. We hope our techniques can be incorporated into these methods to make them more rigorous.

Acknowledgments. We would like to thank Dan Gusfield, Sampath Kannan, Peter Li, Dalit Naor, Frank Olken, and Tandy Warnow who are our colleagues in computational biology. Members of the Lawrence Berkeley Laboratory Human Genome Center, particularly Cassandra Smith, Sylvia Spengler, and Frank Olken suggested applications of our work. David Aldous, Maxime Crochemore, David Haussler, Dick Karp, Gad Landau, Gene Myers, Kunsoo Park, Esko Ukkonen, and Mike Waterman provided helpful comments and encouragement. Finally, we would like to thank the referees whose comments we found extremely helpful.

References

[1] A. V. Aho and M. J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, *Comm. ACM* **18** (1975), 333–340.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, A Basic Local Alignment Search Tool, *J. Molecular Biology* **215** (1990), 403–410.

[3] A. Apostolico, The Myriad Virtues of Subword Trees, in A. Apostolico and Z. Galil, eds., *Combinatorial Algorithms on Words*, NATO ASI Series F, Vol. 12, Springer-Verlag, New York, 1985, pp. 85–96.

[4] W. I. Chang, Fast Implementation of the Schieber–Vishkin Lowest Common Ancestor Algorithm, Computer program, 1990.

[5] W. I. Chang, Approximate Pattern Matching and Biological Applications, Ph.D. thesis, University of California, Berkeley, August 1991. Also available as Computer Science Division Reports UCB/CSD 91/653–654.

[6] W. I. Chang, Approximate String Matching and Local Similarity, *Proc. Fifth Annual Symposium on Combinatorial Pattern Matching*, Asilomar, CA, June 5–8, 1994, Lecture Notes in Computer Science, Springer-Verlag, Berlin, in press.

[7] W. I. Chang and J. Lampe, Theoretical and Empirical Comparisons of Approximate String Matching Algorithms, *Proc. Third Annual Symposium on Combinatorial Pattern Matching*, Tucson, AZ, April 29–May 1, 1992, Lecture Notes in Computer Science, Vol. 644, Springer-Verlag, Berlin, 1992, pp. 175–184.

[8] W. I. Chang and E. L. Lawler, Approximate String Matching in Sublinear Expected Time, *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, St. Louis, MO, Oct. 22–24, 1990, pp. 116–124.

[9] W. I. Chang and E. L. Lawler, Approximate String Matching and Biological Sequence Analysis (poster), *Human Genome II Official Program and Abstracts*, San Diego, CA, Oct. 22–24, 1990, p. 24.

[10] B. Clift, D. Haussler, R. McConnell, T. D. Schneider, and G. D. Stormo, Sequence Landscapes, *Nucleic Acids Res.* **14**(1) (1986), 141–158.

[11] M. Crochemore, Longest Common Factor of Two Words, *Proc. TAPSOFT '87*, Lecture Notes in Computer Science, Vol. 249, Springer-Verlag, Berlin, 1988, pp. 26–36.

[12] R. F. Doolittle, ed. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, *Methods in Enzymology*, Volume 183, Academic Press, New York, 1990.

[13] E. R. Fiala and D. H. Greene, Data Compression with Finite Windows, *Comm. ACM* **32**(4) (1989), 490–505.

[14] Z. Galil and R. Giancarlo, Data Structures and Algorithms for Approximate String Matching, *J. Complexity* **4** (1988), 33–72.

[15] Z. Galil and K. Park, An Improved Algorithm for Approximate String Matching, *SIAM J. Comput.* **19**(6) (1990), 989–999.

[16] G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures: in Pascal and C*, 2nd edn., Addison-Wesely, Reading, MA, 1991.

[17] D. Gusfield, *Efficient Algorithms for String Manipulation and Pattern Matching*, Lecture Notes, University of California, Davis, 1989.

[18] D. Gusfield, K. Balasubramanian, and D. Naor, Parametric Optimization of Sequence Alignment, *Proc. Third Annual ACM–SIAM Symposium on Discrete Algorithms*, Jan. 1992, pp. 432–439.

[19] D. Gusfield, G. M. Landau, and B. Schieber, An Efficient Algorithm for the All Pairs Suffix–Prefix Problem, *Proc. Sequences 91*, Italy, July 1991.

[20] X. Huang, A Contig Assembly Program Based on Sensitive Detection of Fragment Overlaps, *Genomics* **14**(1) (1992), 18–25.

[21] L. C. Hui, Color Set Size Problem with Applications to String Matching, *Proc. Third Annual Symposium on Combinatorial Pattern Matching*, Tucson, AZ, April 29–May 1, 1992, Lecture Notes in Computer Science, Vol. 644, Springer-Verlag, Berlin, pp. 230–243.

[22] P. Jokinen, J. Tarhio, and E. Ukkonen, A Comparison of Approximate String Matching Algorithms, Manuscript, 1990.

[23] S. Kannan and T. Warnow, Inferring Evolutionary History from DNA Sequences, *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, St. Louis, MO, October 1990, pp. 362–371.

[24] S. Karlin, F. Ost, and B. E. Blaisdell, Patterns in DNA and Amino Acid Sequences and Their Statistical Significance, in M. S. Waterman, ed., *Mathematical Methods for DNA Sequences*, CRC Press, Boca Raton, FL, 1989, pp. 133–157.

[25] R. M. Karp, *Probabilistic Analysis of Algorithms*, Lecture notes, University of California, Berkeley, Spring 1988; Fall 1989.

[26] R. M. Karp and M. O. Rabin, Efficient Randomized Pattern-Matching Algorithms, *IBM J. Res. Develop* **31** (1987), 249–260.

[27] J. D. Kececioglu, Exact and Approximate Algorithms for DNA Sequence Reconstruction, Ph.D. thesis, University of Arizona, Tucson, 1991. Also available as Technical Report TR91-26, Computer Science Department, University of Arizona, Tucson.

[28] D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast Pattern Matching in Strings, *SIAM J. Comput.* **6**(2) (1977), 323–350.

[29] G. M. Landau and U. Vishkin, Fast String Matching with k Differences, *J. Comp. System Sci.* **37** (1988), 63–78.

[30] G. M. Landau and U. Vishkin, Fast Parallel and Serial Approximate String Matching, *J. Algorithms* **10** (1989), 157–169.

[31] V. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Phys. Dokl.* **6** (1966), 126–136.

[32] E. M. McCreight, A Space-Economical Suffix Tree Construction Algorithm, *J. Assoc. Comput. Mach.* **23**(2) (1976), 262–272.
[33] E. W. Myers, A Sublinear Algorithm for Approximate Keyword Matching, Technical Report TR90-25, Computer Science Department, University of Arizona, Tucson, September 1991.
[34] National Center for Human Genome Research, *Understanding Our Genetic Inheritance* (The U.S. Human Genome Project: The First Five Years FY 1991–1995), NIH Publication No. 90-1580, April 1990.
[35] K. Park, Fast String Matching On the Average, Manuscript, 1990.
[36] W. R. Pearson and D. J. Lipman, Improved tools for biological sequence comparison, *Proc. Nat. Acad. Sci. USA* **85** (1988), 2444–2448.
[37] H. Peltola, H. Söderlund, and E. Ukkonen, SEQAID: A DNA Sequence Assembling Program Based on a Mathematical Model, *Nucleic Acids Res.* **12**(1) (1984), 307–321.
[38] M. Rodeh, V. R. Pratt, and S. Even, Linear Algorithms for Data Compression via String Matching, *J. Assoc. Comput. Mach.* **28**(1) (1981), 16–24.
[39] D. Sankoff and J. B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, MA, 1983.
[40] B. Schieber and U. Vishkin, On Finding Lowest Common Ancestors: Simplification and Parallelization, *SIAM J. Comput.* **17**(6) (1988), 1253–1262.
[41] P. H. Sellers, The Theory and Computation of Evolutionary Distances: Pattern Recognition, *J. Algorithms* **1** (1980), 359–373.
[42] E. Ukkonen, Finding Approximate Patterns in Strings, *J. Algorithms* **6** (1985), 132–137.
[43] E. Ukkonen, Personal communications.
[44] E. Ukkonen and D. Wood, Approximate String Matching with Suffix Automata, Report A-1990-4, Department of Computer Science, University of Helsinki, April 1990.
[45] M. S. Waterman, Sequence Alignments, in M. S. Waterman, ed., *Mathematical Methods for DNA Sequences*, CRC Press, Boca Raton, FL, 1989, pp. 53–92.
[46] M. S. Waterman, M. Eggert, and E. Lander, Parametric Sequence Comparisons, *Proc. Nat. Acad. Sci. USA* **89** (1992), 6090–6093.
[47] P. Weiner, Linear Pattern Matching Algorithms, *Proc. IEEE Symposium on Switching and Automata Theory*, 1973, pp. 1–11.
[48] S. Wu, U. Manber, and E. Myers, Improving the Running Times for Some String Matching Problems, Technical Report TR91-20, Computer Science Department, University of Arizona, Tucson, August 1991.
[49] A. C. Yao, The Complexity of Pattern Matching for a Random String, *SIAM J. Comput.* **8** (1979), 368–387.

ELSEVIER

Theoretical Computer Science 182 (1997) 233–244

Theoretical Computer Science

Note

Approximation algorithms for multiple sequence alignment

Vineet Bafna [a,1], Eugene L. Lawler [b,2], Pavel A. Pevzner [c,*]

[a] *Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, USA*

[b] *Computer Science Division, University of California, Berkeley, CA 94720, USA*

[c] *Department of Mathematics and Computer Science, University of Southern California, Los Angeles, CA 90089-1113, USA*

Received January 1995; revised February 1996

Communicated by M. Crochemore

Dedicated to the memory of Eugene Lawler

Abstract

We consider the problem of aligning of k sequences of length n. The cost function is sum of pairs, and satisfies triangle inequality. Earlier results on finding approximation algorithms for this problem are due to Gusfield (1991) who achieved an approximation ratio of $2 - 2/k$, and Pevzner (1992) who improved it to $2 - 3/k$. We generalize this approach to assemble an alignment of k sequences from optimally aligned subsets of $l < k$ sequences to obtain an improved performance guarantee. For arbitrary $l < k$, we devise deterministic and randomized algorithms yielding performance guarantees of $2 - l/k$. For fixed l, the running times of these algorithms are polynomial in n and k.

1. Introduction

Multiple sequence alignment is a fundamental problem in computational molecular biology. Alignments of multiple sequences are commonly computed for the purpose of discovering 'homologous', i.e., evolutionarily or functionally related, regions of the sequences. An *optimal* multiple alignment can be computed by dynamic

* Corresponding author.

[1] The research was supported in part by the National Science Foundation Young Investigator Award, the National Science Foundation under grant CCR-9308567, the National Institute of Health under grant R01 HG00987 and the DOE grant DE-FG03-90ER60999.

[2] Deceased.

programming. However, the running time of dynamic programming algorithms increases rapidly with k, the number of sequences to be aligned. Accordingly, many heuristics and approximation algorithms have been proposed [1, 6, 11–13].

Many objective functions have been suggested for the multiple sequence alignment problem. One of the most widely used is the 'sum-of-pairs' (SP) criterion. The problem of computing an optimal alignment with respect to the sum-of-pairs criterion is NP-hard [20]. The advanced algorithms [12] allow one to construct *optimal* alignments of $k \leqslant 6$ sequences, each of length around 200, the length of an average protein. Many algorithms for k sequences use optimal multiple alignment of $l < k$ sequences with further assembling of these "partial" alignments into an *approximate* alignment of k sequences. Multiple alignment algorithms based on this heuristic are widely used un computational molecular biology [19], and are known to produce meaningful biological results [8]. This approach requires an efficient "assembly" procedure providing an approximate alignment of k sequences close to the optimal one. However, no 'performance guarantee' algorithms for multiple alignment have been known until recently, although a number of heuristics for *suboptimal* multiple alignment have been developed (see the recent review, [6]).

Gusfield [9, 10] achieved an approximation ratio of $2 - 2/k$ by assembling an alignment of k sequences from optimal alignments of pairs of sequences. It is known that models currently employed to align sequences are not quite adequate; thus, for practical sequence alignment it is not always necessary to produce an optimal alignment but only one that is plausible. The Gusfield algorithm produces plausible alignments; a computational experiment with an alignment of 19 sequences gave a suboptimal solution only 2% worse than the optimal one. An obvious direction for improvement is to use optimal alignments of $l > 2$ sequences, and then assemble them to approximately align k sequences. However, devising an efficient "assembling" procedure for an arbitrary l remained an open problem.

Pevzner [15] improved the performance guarantee to $2 - 3/k$ by assembling optimal alignments of triples of strings. This suggests the possibility of achieving a further improvement in the performance guarantee to $2 - l/k$ by assembling l-way alignments. We investigate this possibility, and show that for arbitrary $l < k$ it is possible to obtain such a performance guarantee with a running time that is polynomial in n and k. This result provides an evidence that "assembling of alignments" heuristics, commonly used in computational molecular biology [19] might give "good" suboptimal alignment if assembling is done carefully.

In Sections 2 and 3 we define SP-alignment formally, and outline a heuristic approach to constructing SP-alignments of k sequences by combining alignments of l sequences. In Section 4, we show that the problem of constructing SP-alignments within a desired performance ratio reduces to constructing *balanced* sets of *l-stars*. In Section 5, we use dynamic programming to get some improvement over the brute-force approach. Section 6 deals with constructing small balanced sets to ensure small running time. Finally, in Section 7, we show how to obtain an efficient randomized algorithm for SP-alignment.

2. Definitions

Let $\mathscr{A}$ be a finite *alphabet* and $a_1,\ldots a_k$ be k sequences (strings) over $\mathscr{A}$. For convenience, we assume that each of these strings contains n characters. Let $\mathscr{A}'$ denote $\mathscr{A}\cup\{-\}$, where "−" denotes "space". An *alignment* of strings $a_1,\ldots,a_k$ is specified by a $k\times m$ matrix A, where $m\geqslant n$. Each element of the matrix is a member of $\mathscr{A}'$, and each row i contains the characters of a_i in order, interspersed by $m-n$ spaces.

Given an alignment A we denote A_{ij} a pairwise alignment formed by the rows i and j of A. The score of an alignment is determined with reference to a symmetric matrix D specifiying the dissimilarity or *distance* between elements of $\mathscr{A}'$. We assume the metric properties for distance d, so that $d(x,x)=0$ and $d(x,z)\leqslant d(x,y)+d(y,z)$, for all x,y,z in $\mathscr{A}'$. For a given alignment $A=[a_{ih}]$, the *score* for sequences a_i,a_j is

$$s(A_{ij})=\sum_{h=1}^{m} d(a_{ih},a_{jh}),$$

and the *sum-of-pairs score* (SP-score) for the alignment A is given by $\sum_{i,j}s(A_{ij})$. In this definition the score of alignment A is the sum of the scores of *projections* of A onto all pairs of sequences a_i and a_j. Let $C=[c_{ij}]$ be a $k\times k$ matrix of *weights* where c_{ij} is the "weight" of the pairwise alignment between a_i and a_j. The *weighted sum-of-pairs score* for the alignment A is

$$\sum_{i,j} c_{ij}s(A_{ij}).$$

For notational convenience we use *matrix dot product* to denote scores of alignments. Thus, letting $S(A)=[s(A_{ij})]$ be the matrix of scores of pairs of sequences, the weighted sum-of-pairs score is $CS(A)$. Letting E be the unit matrix consisting of all 1's except the main diagonal consisting of all 0's, the (unweighted) sum-of-pairs score of alignment A is $ES(A)$.

Straightforward dynamic programming, with running time $O((2n)^k)$, solves the weighted sum of pairs alignment problem for k sequences. A number of different variations, and some speedups of the basic algorithm have been devised [16, 17, 21]. Hereafter, we let $g(k,n)$ denote the running time required to obtain an optimal solution to the weighted sum-of-pairs problem for k sequences of length n.

3. Compatible alignments

Given an alignment A on sequences $a_1,\ldots,a_k$ and an alignment A' on some subset of the sequences, we say that A is *compatible* with A' if A aligns the characters of the sequences aligned by A' in the same way that A' aligns them. Feng and Doolittle [7] observed that given any tree in which each vertex is labeled with a distinct sequence a_i, and pairwise alignments specified for each tree edge, there exists an alignment of the k sequences that is compatible with each of the pairwise alignments. A similar result holds for "l-stars", defined as follows:

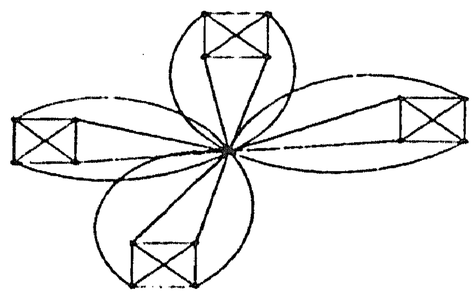


Fig. 1. A 5-star on 17 vertices.

Let V be the set $\{1,2,\ldots,k\}$ representing the sequences $a_1,a_2,\ldots,a_k$, and suppose $l-1|k-1$. An *l-star* $G=(V,E)$ is defined by $r=(k-1)/(l-1)$ cliques of size l whose vertex sets intersect in only one *center* vertex (Fig. 1). Let $A_1\ldots,A_r$, be alignments for the r cliques, with each A_i aligning l sequences. By a construction similar to Feng and Doolittle [7] we have the following lemma:

Lemma 1. *For any l-star and any specified alignments $A_1,\ldots,A_r$ for its cliques, there is an alignment A for the k sequences that is compatible with each of the alignments $A_1,\ldots,A_r$.*

Proof. Assume that the alignment A_i $(1\leqslant i\leqslant r)$ is specified by an $l\times m_i$ matrix with the first row corresponding to the center vertex (string) a_1. We transform matrices $A_1,\ldots,A_r$ into $l\times m^*$ matrices $A_1^*,\ldots,A_r^*$ by "padding" m^*-m_i columns consisting of spaces into A_i for $1\leqslant i\leqslant r$, as follows.

Let $m\leqslant m_i$ be the length of the center string a_1. A_i contains m symbols from a_1 and m_i-m space symbols in the first row. Let $j_{i,1},\ldots,j_{i,m}$ be the positions of m symbols from a_1 in the first row of A_i. Denote $z_{i,l}=j_{i,l+1}-j_{i,l}$ the number of space symbols between lth and $(l+1)$th non-space symbols in the first row of A_i (we assume $z_{i,0}=1$ and $z_{i,m+1}=m_i$). Clearly, $\sum_{0\leqslant l\leqslant m} z_{i,l}=m_i-m$.

Let $z_l=\max_{1\leqslant i\leqslant r} z_{i,l}$ be the maximum spacing between lth and $(l+1)$th non-space symbols in the first row of the matrices $A_1,\ldots,A_r$. Denote $m^*=m+\sum_{l=0}^{m} z_l$ and transform $l\times m_i$ matrix A_i into $l\times m^*$ matrix A_i^* by adding $z_l-z_{i,l}$ "space" columns (i.e. columns consisting of space symbols) between the columns j_l and j_{l+1} of A_i. Matrices $A_1^*,\ldots,A_r^*$ have the same number of columns and union of their rows generates an alignment A compatible with the alignments $A_1,\ldots,A_r$ (see [7] for more details). □

Assign weights to the edges of an l-star G, with center c, as follows.

$$c_{ij}=\begin{cases} k-(l-1) & i=c \text{ or } j=c,\\ 1 & i,j\neq c,\ i \text{ and } j \text{ are contained in the same clique of } G,\\ 0 & \text{otherwise}\end{cases}$$

and let $C(G)=[c_{ij}]$ denote the $k \times k$ matrix of weights. Note that

$$C(G)E=(k-(l-1))(k-1)+\binom{k-1}{l-1}\binom{l-1}{2}=\binom{k}{2}\left(2-\frac{l}{k}\right).$$

The pairwise scores of an alignment inherit the triangle inequality property from the distance matrix D. That is, for any alignment A, $s(A_{ij}) \leqslant s(A_{ik})+s(A_{kj})$, for all i,j,k. This fact was used by Pevzner [15] to prove the following:

Lemma 2. *For any alignment A of the k sequences, and an l-star G, $ES(A) \leqslant C(G)S(A)$.*

Let $C_1,\ldots,C_r$ denote the submatrices of weights for the r cliques of an l-star G. Let $A_1^*,\ldots,A_r^*$ be optimal weighted sum-of-pairs alignments for the r cliques. From Lemma 1 and the fact that $d(-,-)=0$, we obtain the following.

Lemma 3. *Given an l-star G, there is an optimal (weighted with respect to $C(G)$) alignment A_G for the k sequences that is compatible with each of the alignments $A_1^*,\ldots,A_r^*$. Moreover, $C(G)S(A_G)=C_1S(A_1^*)+\ldots+C_rS(A_r^*)$.*

To summarize, for any l-star G we can assemble an alignment A_G, optimal with respect to the weight matrix $C(G)$ specified above, by computing optimal weighted alignments for each clique of G. This can be done in $O(kg(l,n))$ time.

4. Balanced sets of l-stars

Let $\mathcal{G}$ be a collection of l-stars, and let $C(G)$ denote the weight matrix for star G. We say that the collection $\mathcal{G}$ is *balanced* if $\sum_{G\in\mathcal{G}} C(G)=pE$ for some scalar $p>1$.

Lemma 4. *If $\mathcal{G}$ is a balanced set of l-stars, then*

$$\min_{G\in\mathcal{G}} C(G)S(A_G) \leqslant \frac{p}{|\mathcal{G}|}\min_A ES(A).$$

Proof. We use an averaging argument.

$$\begin{aligned}\min_{G\in\mathcal{G}} C(G)S(A_G) &\leqslant \frac{1}{|\mathcal{G}|}\sum_{G\in\mathcal{G}} C(G)S(A_G)\\ &\leqslant \frac{1}{|\mathcal{G}|}S(A)\sum_{G\in\mathcal{G}} C(G)=\frac{p}{|\mathcal{G}|}ES(A).\end{aligned}$$

Here the inequality holds for an arbitrary alignment A, and in particular, it also holds for the optimum alignment. □

Lemmas 2 and 4 motivate the algorithm *Align* (Fig. 2).

Procedure *Align*

1. Construct a balanced set of l-stars, $\mathcal{G}$.
2. For each l-star G in $\mathcal{G}$, assemble an alignment A_G that is optimal with respect to $C(G)$ from alignments that are optimal for each of its cliques (Lemma 3).
3. Choose G with the corresponding alignment A_G such that $C(G) \cdot S(A_G)$ is the minimum over all l-stars in $\mathcal{G}$. Return A_G.

Fig. 2. Deterministic algorithm for multiple alignment.

Theorem 1. *Given a balanced collection of l-stars $\mathcal{G}$, Align returns an alignment with a performance guarantee of $2 - l/k$ in $O(k|\mathcal{G}|g(l,n))$ time.*

Proof. Note that

$$\frac{p}{|\mathcal{G}|} = \frac{C(G)E}{EE} = 2 - \frac{l}{k}.$$

Now, *Align* returns the alignment A_G which is optimal for l-star $G \in \mathcal{G}$, and for which the smallest weighted score, $\min_{G\in\mathcal{G}} C(G)S(A_G)$ is achieved. Lemmas 2 and 4 imply that $ES(A_G) \leqslant C(G)S(A_G) \leqslant (2 - \frac{l}{k}) \min_A ES(A)$. □

5. Optimizing over all l-stars

We have reduced our approximation problem to that of finding an optimal alignment for each l-star in a balanced set. How hard is it to find a balanced set $\mathcal{G}$? A trivial candidate is simply the set of all l-stars, which is clearly balanced by symmetry. Note that for $l = 2$, there are only k l-stars. This fact was exploited by Gusfield [9] to obtain an approximation ratio of $2 - 2/k$. This is really a special case, as for $l > 2$, the number of l-stars grows exponentially with k making the algorithm computationally infeasible. Pevzner [15] solved the case of $l = 3$, by mapping the problem to weighted matching on graphs.

In this section, we show that it is not necessary to exhaustively compute alignments for all possible l-stars. Dynamic programming provides a shortcut. Specifically, we prove the following:

Theorem 2. *For all k, l, it is possible to compute an alignment with a performance guarantee of $2 - l/k$ in $O(k^{l+1}(2^k + kg(l,n)))$ time.*

Proof. For simplicity, consider at first the case when $l - 1 | k - 1$. Fix a center vertex c. Consider an arbitrary subset Q of $l - 1$ sequences from $V \setminus c$. Denote $opt(Q)$ to be the optimum score of a weighted alignment of the sequences in Q along with c such that the weight of all edges incident to c is $k - l + 1$ and the weight of the remaining edges

is 1. For each choice of a center vertex c and for each of the $\binom{k-1}{l-1}$ possible cliques $Q \subseteq V \setminus c$, compute $opt(Q)$. This computation can be done in time $O(k\,k^l\,g(l,n))$.

Next, for all $Q \subseteq V \setminus c$, such that $|Q|$ is a multiple of $l-1$, denote $s(Q)$ as the minimum alignment score among all l-stars over the vertices in $Q \cup c$ with center vertex c Now, let $Q_1, Q_2 \ldots, Q_r$ be the cliques on an l-star with the minimum alignment score. Then, from Lemma 3, $s(Q) = opt(Q_1) + opt(Q_2) + \cdots + opt(Q_r)$. Clearly, $s(Q)$ can be computed by the following recurrence:

$$s(\phi) = 0$$

$$s(Q) = \min_{Q' \subseteq Q, |Q'| = l-1} \{s(Q \setminus Q') + opt(Q')\}$$

Q is a set of size at most $k-1$. In order to compute $s(Q)$, we need to look at most $\binom{k-1}{l-1} = O(k^l)$ subsets Q'. Therefore, computing $s(Q)$ for each of at most 2^k sets Q takes $O(k^l)$ time, and repeating for each choice of a center vertex, the computation takes $O(k^l 2^k k)$ time. Therefore, if $l-1 | k-1$, we can compute the optimum score in $O(k^l(2^k + kg(l,n)))$ time.

In the general case, when $(l-1)$ does not divide $(k-1)$, we need to consider *hybrid* stars which contain cliques of size l as well as $l+1$. Therefore, we compute $opt(Q)$ for all cliques of size l or $l+1$ in time $O(k^{l+1} kg(l+1,n))$. The new recurrence for $s(Q)$ is as follows:

$$s(\phi) = 0$$

$$s(Q) = \min_{Q' \subseteq Q, |Q'| = l-1 \text{ or } |Q'| = l} \{s(Q \setminus Q') + opt(Q')\}.$$

The net running time increases to $O(k^{l+1}(2^k + kg(l+1,n)))$. □

This approach may be computationally tractable for many problem instances. However, in order to obtain a time bound that is polynomial in n and k, for fixed l, we need to construct balanced sets of l-stars of small size.

Constructing a *small* balanced set of l-stars is not trivial, except for some specific values of l and k. One way of constructing such a set $\mathscr{S}$ for specific values of l and k is to consider a *sharply doubly transitive* set of permutations, and combinatorial block designs [2].

In the following section, we get around the difficulty of constructing small balanced sets *for all* l,k by constructing a balanced set that is exponentially large, but on which we can quickly find a minimum score l-star by solving matching problems.

6. Balanced sets of $(2l-1)$-stars

In this section, we prove the following theorem

Theorem 3. *For all k, l, it is possible to compute an alignment with a performance guarantee of $2 - l/k$, in $O(k^3 g(2l+5,n))$ time.*

Proof. For simplicity, let us first assume that $2(l-1)\,|\,k-1$. For each choice of a center vertex c, let G be an arbitrary l-star with r cliques. Define a *configuration* G' by combining the cliques of G in a pairwise fashion (to form $(2l-1)$-cliques), and assigning weights as follows:

$$c_{ij}=\begin{cases} k-(l-1)-1/2 & i=c \text{ or } j=c,\\ 1 & i,j\neq c,\ i \text{ and } j \text{ are contained in the same clique}\\ & \quad \text{of } G', \text{ but different cliques of } G,\\ 0 & \text{otherwise.}\end{cases}$$

Note that, as in the case of l-stars,

$$C(G)E=(k-1)\left[k-(l-1)-\frac{1}{2}\right]+\frac{k-1}{2(l-1)}(l-1)^2=\binom{k}{2}\left(2-\frac{l}{k}\right).$$

Trivially, Lemmas 2 and 3 hold for a configuration also.

For an arbitrary l-star G with center c, consider the set of all configurations obtained by pairing up cliques in G. Consider an arbitrary edge (i,j) such that $i,j\neq c$, and i,j do not belong to the same clique of G. By symmetry, each such edge will appear an equal number of times, say x, in the set of all configurations.

Now, for each l-star G in a set of k arbitrary l-stars, each with a different center vertex, consider the set of all configurations obtained by pairing up cliques in G. We assert that this set of configurations, along with x copies of each l-stars, forms a balanced set $\mathcal{G}$. For an arbitrary entry in $C(G)$, $C(G)[i,j]=k-(l-1)-1/2$ exactly $(2/k)|\mathcal{G}|$ times (when i or j is the center vertex of G), and $c(G)[i,j]=1$ exactly $(k-2)/kx$ times. Therefore $\sum_{G\in\mathcal{G}}C(G)=pE$, where

$$p=\frac{(\sum_{G\in\mathcal{G}}C(G))E}{\binom{k}{2}}$$

is a scalar. Furthermore, $\sum_{i,j}C(G)[i,j]=(2-l/k)\binom{k}{2}$ is the same for all $G\in\mathcal{G}$, implying that

$$\left(\sum_{G\in\mathcal{G}}C(G)\right)E=\sum_{G\in\mathcal{G}}(C(G)E)=\sum_{G\in\mathcal{G}}(2-l/k)\binom{k}{2}.$$

Therefore, $p=(2-l/k)|\mathcal{G}|$.

Next, we show that we can compute the optimal weighted cost configuration without explicitly generating the set of all configurations. Fix k arbitrary l-stars, one for each choice of a center vertex. For each l-star with center c, form a complete graph of r vertices H_r, with each node corresponding to a clique of the l-star and the weight of an edge being the cost of an optimal weighted alignment on the corresponding $(2l-1)$-clique.

Note that each configuration of G with center c describes a matching in H_r. Further, by Lemma 3 the cost of the configuration is equal to the sum of weights on the matching edges. Therefore, a minimum cost matching on H_r gives the cost of an

optimal weighted configuration of G with center c. In order to find the optimal weighted cost configuration in $\mathscr{G}$, we solve the corresponding matching problem for each choice of a center vertex and pick one with the minimum cost. Finally compare the optimal configuration with each cost of the k l-stars, and return one with the minimum cost. From earlier arguments, the corresponding alignment achieves the desired performance ratio.

For the running time, observe that in computing the graph H_r, we need to solve $\binom{r}{2}$ alignments of $2l-1$ sequences, which takes time $O(r^2 g(2l-1,n)) = O(k^2 g(2l-1,n))$. For typical values of n,k, this dominates the cost of computing a minimum cost matching on a graph of size r. Repeating this for each choice of a center vertex takes time $O(k^3 g(2l-1,n))$. Finally, computing the alignment for each of the k l-stars takes time $O(kg(l,n))$. Therefore, if $2(l-1)|k-1$, it is possible to compute an alignment with performance guarantee of $2-l/k$, in $O(k^3 g(2l-1,n))$ time.

This method can be generalized for arbitrary l with a slight increase in running time. Consider a *hybrid* l-star G with an even number of cliques of size l and $l+1$. As before, define a configuration G' by combining cliques of G arbitrarily in a pairwise fashion to form new cliques of sizes $2l-1, 2l$ and $2l+1$. Assign weights exactly as before. Note that Lemmas 2 and 3 still hold. Also, from symmetry, if we take the set of all configurations of l-star G, then each edge that does not belong to a clique of G will appear an equal number of times, say x. Combining this with x copies of G, each edge appears exactly x times. By earlier arguments, this set is also balanced. The only thing that remains is to estimate the value of p. Note that,

$$C(G)E \leqslant (k-1)\left[k-(l-1)-\frac{1}{2}\right] + \frac{k-1}{2(l-1)}\, l^2 \leqslant \binom{k}{2}\left(2-\frac{l-2}{k}\right).$$

Therefore, $p \leqslant (2-(l-2)/k)$. Repeating earlier arguments, we see that the optimal weighted cost configuration for each choice of a center can be computed in time $O(k^2 g(2l+1,n))$. Therefore, in time $O(k^3 g(2l+1,n))$, we can compute an alignment that will guarantee a performance of $2-(l-2)/k$. For $l' = l-2$, this implies an algorithm that runs in time $O(k^3 g(2l'+5,n))$ and guarantees a performance of $2-l'/k$. □

As an aside, a smaller balanced set can be explicitly constructed. Let

$$r = \left\lfloor \frac{k-1}{2(l-1)} \right\rfloor.$$

A *perfect matching* on H_{2r} corresponds to a configuration in the original graph. It is easy to see that a set of configurations corresponding to a 1-*factorization* of H_{2r} (edge-disjoint decomposition of H_{2r} into perfect matchings), for each of the k l-stars, along with a single copy of each l-star, forms a balanced set of size $O(k^2)$.

7. Random sampling of l-stars

What is the performance bound if we choose an l-star at random? Gusfield studied this for 2-stars and gave a bound on the expected score of the alignment [12]. Assuming a uniform distribution on the set of all l-stars, we are interested in the expected value of the random variable $C(G)S(A_G)$. As the set of all l-stars is balanced, $Exp[C(G)S(A_G)] \leqslant (2 - l/k) \min_A ES(A)$. However, it is not clear if we can pick with high probability, an l-star that achieves the $2 - l/k$ performance.

Let $\mathcal{G}_c$ be the set of all l-stars, with a fixed center c. For G in $\mathcal{G}_c$, let $C(G) = C_1(G) + C_2(G)$ be the partition of weight matrix into *Border and Center* weights, with $C_1(G)$ being the same as $C(G)$ except for the cth row and column which are 0. Define $E = E_1 + E_2$ in an identical manner. Observe the balancing property of $C_1(G)$, i.e. $\sum_{G \in \mathcal{G}_c} C_1(G) = p_1 E_1$, where

$$p_1 = \frac{(l-2)}{(k-2)} |\mathcal{G}_c|.$$

We have the following lemma:

Lemma 5. *For G chosen uniformly at random from $\mathcal{G}_c$, and any alignment A,*

$$Prob\left[C_1(G)S(A) > \frac{3}{2}\frac{p_1}{|\mathcal{G}_c|}E_1S(A)\right] < \frac{2}{3}.$$

Proof. Let $BAD = \{G \in \mathcal{G}_c | C_1(G)S(A) > \frac{3}{2}\frac{p_1}{|\mathcal{G}_c|}E_1S(A)\}$. Then,

$$\frac{3}{2}\frac{p_1}{|\mathcal{G}_c|}E_1S(A)|BAD| < \sum_{G \in BAD} C_1(G)S(A) \leqslant \sum_{G \in \mathcal{G}_c} C_1(G)S(A)$$
$$= p_1E_1S(A)$$

which implies that $|BAD| \leqslant \frac{2}{3}|\mathcal{G}_c|$. □

Pick m l-stars randomly from $\mathcal{G}_c$. It follows from the proof of Lemma 5 that the l-star with the minimum weight alignment (among these m stars) is in BAD with probability less than or equal to $(\frac{2}{3})^m$. *Randomized_Alignment* (Fig. 3) uses this fact to construct a set of l-stars which guarantees a good performance with high probability.

Theorem 4. *If $l - 1 | k - 1$, then for an arbitrary $\varepsilon > 0$, Randomized_Alignment runs in time $O(k^2 \lceil \lg(k/\varepsilon) \rceil g(2l, n))$, and returns an alignment that, with probability $1 - \varepsilon$, achieves a performance bound of $2 - l/k$.*

Proof. Consider the set of l-stars in $\mathcal{G} = \{G_c: 1 \leqslant c \leqslant k\}$, constructed by the outer loop. To begin with, assume that none of the l-stars in $\mathcal{G}$ is in BAD. In other words,

$$\text{for all } G \in \mathcal{G}, \quad C_1(G)S(A) \leqslant \frac{3}{2}\frac{p_1}{|\mathcal{G}_c|}E_1S(A).$$

Procedure *Randomized_Alignment*(l, k, ε)
$\mathcal{G} \leftarrow \emptyset$
for $c \in \{1, \ldots, k\}$
repeat $2\lceil \lg(k/\varepsilon) \rceil$ times
choose a random l-star G with center c
compute an alignment A_G with the minimum weighted score $\min_A C(G)S(A)$
$G_c \leftarrow$ an l-star with minimum weighted score among the $2\lceil \lg(k/\varepsilon) \rceil$ l-stars,
$\mathcal{G} \leftarrow \mathcal{G} \cup \{G_c\}$.
$G \leftarrow$ an l-star with the minimum weighted score $\min_{G \in \mathcal{G}} C(G)S(A_G)$
return A_G

Fig. 3. Randomized algorithm for multiple alignment.

Randomized_Alignment returns an l-star G with the minimum weighted score from $\mathcal{G}$. We give a bound on its score by a counting argument. For every alignment A,

$$\min_{G \in \mathcal{G}} C(G)S(A_G) \leqslant \min_{G \in \mathcal{G}} C(G)S(A) \leqslant \frac{1}{k} \sum_{G \in \mathcal{G}} C(G)S(A)$$

$$= \frac{1}{k} \sum_{G \in \mathcal{G}} C_2(G)S(A) + \frac{1}{k} \sum_{G \in \mathcal{G}} C_1(G)S(A)$$

$$\leqslant \frac{2}{k}(k - (l-1))ES(A) + \frac{k-2}{k} \frac{3}{2} \frac{l-2}{k-2} ES(A)$$

$$= \left(2 - \frac{l}{2k}\right) ES(A).$$

Now, recall from Lemma 2 that $ES(A_G) \leqslant C(G)S(A_G)$, which implies that if none of the l-stars in $\mathcal{G}$ is in BAD, the algorithm achieves a performance bound of $(2 - l/2k)$.

Next, we show that none of the l-stars in $\mathcal{G}$ is in BAD with high probability. In each iteration of the inner loop, we consider $2\lceil \lg(k/\varepsilon) \rceil$ random l-stars, and pick a G_c with the minimum weighted score. By definition, this l-star is in BAD only if each l-star picked in that iteration is in BAD. Therefore, for all $1 \leqslant c \leqslant k$, the probability that $G_c \in \mathcal{G}$ is in BAD is less than

$$\left(\frac{2}{3}\right)^{2\lceil \lg(\frac{k}{\varepsilon}) \rceil} < \frac{\varepsilon}{k}.$$

The probabilty that none of the k $G_c \in \mathcal{G}$ are in BAD is greater than or equal to $1 - \varepsilon$.

Now, choose $l' = l/2$. Note from Lemma 3 that computing each alignment A_G takes time $O(kg(2l', n))$. Therefore, *Randomized_Alignment* runs in time $O(k^2 2\lceil \lg(k/\varepsilon) \rceil g(2l', n))$ and returns an alignment that, with probability $1 - \varepsilon$, achieves a performance bound of $2 - l'/k$. □

Acknowledgements

We are grateful to Piotr Berman, Dima Grigoriev, Jeanette Schmidt and Martin Vingron for many useful comments and suggestions.

References

[1] S.F. Altschul, D.J. Lipman, Trees, stars, and multiple biological sequence alignment, SIAM J. Appl. Math. 49 (1989) 197–209.
[2] V. Bafna, E.L. Lawler, P. Pevzner, Approximation Algorithms for Multiple Sequence Alignment, in: Proc. of the 5th Annual Symp. on Combin. Pattern Matching (CPM'94). Lecture Notes in Computer Science, vol. 807, Springer, Berlin, 1994, pp. 43–53.
[3] Z. Baranyai, On the factorization of the complete uniform hypergraph, in: A. Hajnal, T. Rado, V.T. Sós (Eds.), Infinite and Finite Sets, North-Holland, Amsterdam, 1975, pp. 91–108.
[4] J. Bósak, Decompositions of Graphs, Kluwer, Dordrecht, 1990.
[5] J.L. Carter, M.N. Wegman, Universal classes of hash functions, J. Comput. System Sci. 18 (1979) 143–154.
[6] S.C. Chan, A.K.C. Wong, D.K.Y. Chiu, A survey of multiple sequence comparison methods, Bull. Math. Biol. 54 (1992) 563–598.
[7] D. Feng, R. Doolittle, Progressive sequence alignment as a prerequisite to correct phylogenetic trees, J. Molec. Evol. 25 (1987) 351–360.
[8] A.E. Gorbalenya, V.M. Blinov, A.P. Donchenko, E.V. Koonin, An NTP-binding motif is the most conserved sequence in a highly diverged monophyletic group of proteins involved in positive strand RNA viral replication. J. Molec. Evol. 28 (1989) 256–68.
[9] D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, Tech. Report, Computer Science Division, University of California, Davis, CSE-91-4, 1991.
[10] D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, Bulletin of Mathematical Biology 55 (1993) 141–154.
[11] T. Jiang, E.L. Lawler, L. Wang, Aligning sequences via an evolutionary tree: complexity and approximation, in: Proc. ACM STOC'94, 1994, pp. 760–769.
[12] J. Kececioglu, The maximum weight trace alignment problem in multiple sequence alignment, in: A. Apostolico, M. Crochemore, Z. Galil, U. Manber (Eds.), Combinatorial Pattern Matching 93, Padova, Italy, June 1993, Lecture Notes in Computer Science, vol. 684, Springer, Berlin, pp. 106–119.
[13] D.J. Lipman, S.F. Altschul, J.D. Kececioglu, A tool for multiple sequence alignment, Proc. Nat. Acad. Sci. U.S.A 86 (1989) 4412–4415.
[14] P. Lorimer, Finite projective planes and sharply 2-transitive subsets of finite groups, in: Proc. 2nd Internat. Conf. Theory of Groups, Canberra (1973) 432–436.
[15] P. Pevzner, Multiple alignment, communication cost, and graph matching, SIAM J. Appl. Math. 52 (1992) 1763–1779.
[16] D. Sankoff, Minimum mutation tree of sequences, SIAM J. Appl. Math. 28 (1975) 35–42.
[17] D. Sankoff, Simultaneous solution of the RNA folding, alignment and protosequence problems, SIAM J. Appl. Math. 45 (1985) 810–825.
[18] J. Schmidt, A. Siegel, The analysis closed hashing under limited randomness, Proc. 22nd ACM Symp. on Theory of Computing, 1990, pp. 224–234.
[19] W.R. Taylor, Hierarchical method to align large numbers of biological sequences. Methods Enzymol. 183 (1990) 456–474.
[20] L. Wang, T. Jiang, On the complexity of multiple sequence alignment, J. Comp. Biol. 1 (1994) 337–348.
[21] M.S. Waterman, T.F. Smith, W.A. Beyer, Some biological sequence metrics. Adv. in Math. 20 (1976) 367–387.

"If a thing is not worth doing, it is not worth doing right"

Commencement Address to the Computer Science Majors,
College of Letters & Sciences, UC Berkeley, May 1994

Eugene L. Lawler

Greetings to the graduating class of '94, to their families, and to their friends. In a way, I come as a representative of the graduating faculty class of '94, being one of many who accepted an offer from the university that was too good to refuse and are graduating – into retirement. In fact, I think that this will be viewed as a watershed year in the history of the Computer Science Division at Berkeley. If you visit the department in a few years you might hardly recognize it: there will be a lot of new faces on the faculty, and there will be our grand new building.

I personally arrived in Berkeley more than twenty years ago, in another watershed year for computer science at Berkeley. At that time, there were not one, but two computer science departments. After a great deal of anguish, the two departments were merged into one as part of the Department of Electrical Engineering and Computer Science. In fact, the L&S major in computer science is the last remnant of the separate department that existed then in the College of Letters & Sciences.

Over the years, I have had occasion to participate in a number of these commencement exercises. In so doing, I have noticed that two themes recur, over and over. The first is that you have worked very hard for your computer science education. The second is that the education you have received is highly perishable.

I certainly endorse both these views. The Berkeley computer science curriculum *is* arduous and time-consuming. Each of you has demonstrated talent, drive, and fortitude in running this academic gauntlet and, particularly, to have survived CS 150. This is an achievement that you can be proud of, all your lives.

Second, it *is* true that the knowledge that you have gained will soon be obsolete. Each of you will have to renew your technical knowledge several times in the course of your professional careers. I hope that your experience here at Berkeley has helped you to learn how to learn.

Today, I should like to suggest that you have a greater responsibility than simply the renewal of your technical knowledge. Digital technology is having an enormous impact on our society. It does matter what you do in the course of your professional career. You should try to educate yourselves about social issues and public policy questions. You should think very seriously about the consequences of your labors.

At this point, perhaps I might be allowed to reminisce a little. My own college graduation was exactly forty years ago. In the fall of 1954, I entered Harvard Graduate School, in applied mathematics. That year I took my first computer course, at the Harvard Computation Laboratory.

The Computation Laboratory had two machines, the Mark I and Mark IV computers. Needless to say, these machines were not at all like those of today.

Each machine was as big as a small house, and each had a fraction of the capability of one of today's hand-held calculators. The Mark I was a particularly handsome and impressive machine. It had been assembled from standard IBM tabulator components – that is, of electromechanical relays. These relays were synchronized by a thirty-foot long drive shaft, alongside of which ran an foot-thick bundle of thousands of color-coded wires. The entire mechanism was surrounded by large sheets of plate glass, artfully rounded at the corners. One could open a door and enter the machine and take a stroll. I much enjoyed wandering around inside the machine, listening to the hum of the drive shaft, the click of the relays, and admiring the elegant craftsmanship with which it had been constructed.

The Mark I was built during World War II by IBM engineers under the direction of Professor Howard Aiken. It was generally used for compiling endlessly-long numerical tables, particularly tables of Bessel functions. (In those days, we had no idea that there could be non-numerical applications of computers.) It performed multiplication by repeated addition. Sometime after it was built, it was retrofitted with a specialized multiplier unit constructed of relays. At the dedication of the multiplier, Howard Aiken commented on its fast speed. It could multiply two 23-decimal digit numbers in the blindingly fast speed of seven seconds. And Aiken, at its dedication, announced that surely the world would never need a faster multiplier unit.

Howard Aiken also commented once that the world would surely never need more than twenty machines the size of Mark IV. That was because there would not be enough useful problems for them to solve, and not enough technical people to program them.

I mention these stories not to denigrate Howard Aiken, a great computer pioneer, but to point out how difficult it is to predict the consequences of technological development, particularly for those who are innovative pioneers. I ask, rhetorically: Do we today have a better idea of where technology is taking us than Aiken did forty years ago? Is there anyone in this gathering that believes that he or she can accurately predict what the world of digital technology will look like in the year 2034?

Most of us do believe that we are on the threshold of cyberspace. Rapturous accounts have been written about the multimedia information superhighway. In some visions, this highway will seamlessly meld together telephone, broadcast television, cable tv, email, facsimile transmission, home shopping, and other entertainment, educational and information services.

All of which may be true. Or at least, something more or less like this vision will come true within the next forty years. But can we really predict the impact of linking many, many more millions of machines together? When you come to the end of your technical careers in 2034, will that world have been predictable today? I believe that our vision of the future is just as foggy as Howard Aiken's was then. This being the case, it is incumbent upon all of you, as you advance in your technical careers, to think about the impact of your work on society.

I would like to cite one collection of public policy issues as an example of the kinds of things that each of you in this graduating class should be very

concerned with. As we work towards building this information superhighway, it has become easier to link together the massive databases that exist, both private and government-controlled, and so there is an almost inevitable loss of privacy. You would be astonished at the range of information already recorded about each of you here that can be compiled in a relatively short period of time.

Dealing with this issue of privacy seems at times to be almost insurmountable from both a legal or a policy point of view, because of the technological imperative. The technology is there, and is constantly being improved upon, making information even easier to access. We must simply steel ourselves to the inevitable loss of privacy, and try to make the best of it.

And yet there is an area in which computer technology appears to offer us a tremendous opportunity for securing privacy, namely in person-to-person communication. One of the greatest achievements of computer theorists in recent years has been the development of cryptography, and in particular, of public-key encryption systems. This would seem to enable individuals to secure their communications in cyberspace.

Yet law enforcement agencies, not to mention the CIA and National Security Agency, are particularly unhappy with this. They believe that bugging of electronic communications is necessary for effective law enforcement and for "national security." The FBI is currently proposing a Digital Telephony bill requiring that all telephone circuits be designed so as to facilitate wire tapping by law enforcement agencies. The government has banned the export of cryptographic hardware and software, apparently because NSA fears that this will make its task of monitoring the secret communications of friendly governments of less developed nations harder.

Perhaps the most bizarre and misguided exercise that has been undertaken by the Clinton administration, which was in fact initially proposed by the Bush administration, is the clipper chip. The idea is to put this chip into every digital telephone and computer made in the US. This chip would provide encoding or scrambling of all digital messages. In doing so, it uses an encryption algorithm known as Skipjack. Only the mavens of the NSA know the algorithm itself.

But here is the catch. The federal government, specifically the Commerce and Treasury Departments would each hold one part of a two-part key. Upon the rubber-stamping of a warrant by a federal judge, the FBI or other law enforcement agencies will be able to wiretap our communications at will.

As William Safire wrote in a recent column: "The 'clipper chip' – aptly named, as it clips the wings of individual liberty – would encode, for federal perusal whenever a judge rubber-stamped a warrant, everything we say on a phone, everything we write on a computer, every order we give to a shopping network or bank or 800 or 900 number, every electronic note we leave our spouses or dictate to personal-digital-assistant genies.

"Add to that stack of intimate data the medical information derived from the national 'health security card' Mr. Clinton proposes we all carry. Combine it with the travel, shopping and credit data available from all our plastic cards, along with psychological and student test scores. Throw in the confidential tax

returns, sealed divorce proceedings, welfare records, field investigations for job applications, raw files and CIA dossiers available to the Feds and you have the individual citizen standing naked to the nosy bureaucrat.

"... Ah, but wouldn't it be helpful to society to have instant access to the encoded communications of a Mafia capo, or a terrorist ordering the blowup of a skyscraper, or a banker financing a dictator's nuclear development?

"Sure it would. That's why no self-respecting vice-overlord or terrorist or local drug runner would buy or use the clipper-chipped American telecommunications equipment. They would buy non-American hardware with unmonitored Japanese or German or Indian encryption chips and laugh all the way to the plutonium factory.

"The only people tappable by American agents would be honest Americans – or those crooked Americans dopey enough to buy American equipment with the pre-compromised American code ..."

Thank you Mr. Safire. I couldn't have said it better myself.

At this point, I am reminded that one of our own PhD students, now a tenured MIT professor and an authority on encryption, returned to Berkeley a few months ago to give a lecture on, guess what: How the clipper chip system could be technically improved. I told him at the time that I thought the clipper was a prize example of the application of technology for all the wrong purposes.

And so, on this occasion, my points are:

- It is hard to predict the future course of technology.
- Innumerable social issues and public policy issues arise. These require careful, considered thought. They cannot be ignored.
- As technologists, we must keep our heads up, and be as aware as possible of the implications of our actions.
- Think about your work, and what you are doing. Is it important? Is it worth doing?

Perhaps I might better have said to my friend what my colleague Beresford Parlett is fond of saying: "If a thing is not worth doing, it is not worth doing right."

MATHEMATICAL CENTRE TRACTS

1 T. van der Walt. *Fixed and almost fixed points.* 1963.

2 A.R. Bloemena. *Sampling from a graph.* 1964.

3 G. de Leve. *Generalized Markovian decision processes, part I: model and method.* 1964.

4 G. de Leve. *Generalized Markovian decision processes, part II: probabilistic background.* 1964.

5 G. de Leve, H.C. Tijms, P.J. Weeda. *Generalized Markovian decision processes, applications.* 1970.

6 M.A. Maurice. *Compact ordered spaces.* 1964.

7 W.R. van Zwet. *Convex transformations of random variables.* 1964.

8 J.A. Zonneveld. *Automatic numerical integration.* 1964.

9 P.C. Baayen. *Universal morphisms.* 1964.

10 E.M. de Jager. *Applications of distributions in mathematical physics.* 1964.

11 A.B. Paalman-de Miranda. *Topological semigroups.* 1964.

12 J.A.Th.M. van Berckel, H. Brandt Corstius, R.J. Mokken, A. van Wijngaarden. *Formal properties of newspaper Dutch.* 1965.

13 H.A. Lauwerier. *Asymptotic expansions.* 1966, out of print: replaced by MCT 54.

14 H.A. Lauwerier. *Calculus of variations in mathematical physics.* 1966.

15 R. Doornbos. *Slippage tests.* 1966.

16 J.W. de Bakker. *Formal definition of programming languages with an application to the definition of ALGOL 60.* 1967.

17 R.P. van de Riet. *Formula manipulation in ALGOL 60, part 1.* 1968.

18 R.P. van de Riet. *Formula manipulation in ALGOL 60, part 2.* 1968.

19 J. van der Slot. *Some properties related to compactness.* 1968.

20 P.J. van der Houwen. *Finite difference methods for solving partial differential equations.* 1968.

21 E. Wattel. *The compactness operator in set theory and topology.* 1968.

22 T.J. Dekker. *ALGOL 60 procedures in numerical algebra, part 1.* 1968.

23 T.J. Dekker, W. Hoffmann. *ALGOL 60 procedures in numerical algebra, part 2.* 1968.

24 J.W. de Bakker. *Recursive procedures.* 1971.

25 E.R. Paërl. *Representations of the Lorentz group and projective geometry.* 1969.

26 European Meeting 1968. *Selected statistical papers, part I.* 1968.

27 European Meeting 1968. *Selected statistical papers, part II.* 1968.

28 J. Oosterhoff. *Combination of one-sided statistical tests.* 1969.

29 J. Verhoeff. *Error detecting decimal codes.* 1969.

30 H. Brandt Corstius. *Exercises in computational linguistics.* 1970.

31 W. Molenaar. *Approximations to the Poisson, binomial and hypergeometric distribution functions.* 1970.

32 L. de Haan. *On regular variation and its application to the weak convergence of sample extremes.* 1970.

33 F.W. Steutel. *Preservations of infinite divisibility under mixing and related topics.* 1970.

34 I. Juhász, A. Verbeek, N.S. Kroonenberg. *Cardinal functions in topology.* 1971.

35 M.H. van Emden. *An analysis of complexity.* 1971.

36 J. Grasman. *On the birth of boundary layers.* 1971.

37 J.W. de Bakker, G.A. Blaauw, A.J.W. Duijvestijn, E.W. Dijkstra, P.J. van der Houwen, G.A.M. Kamsteeg-Kemper, F.E.J. Kruseman Aretz, W.L. van der Poel, J.P. Schaap-Kruseman, M.V. Wilkes, G. Zoutendijk. *MC-25 Informatica Symposium.* 1971.

38 W.A. Verloren van Themaat. *Automatic analysis of Dutch compound words.* 1972.

39 H. Bavinck. *Jacobi series and approximation.* 1972.

40 H.C. Tijms. *Analysis of (s,S) inventory models.* 1972.

41 A. Verbeek. *Superextensions of topological spaces.* 1972.

42 W. Vervaat. *Success epochs in Bernoulli trials (with applications in number theory).* 1972.

43 F.H. Ruymgaart. *Asymptotic theory of rank tests for independence.* 1973.

44 H. Bart. *Meromorphic operator valued functions.* 1973.

45 A.A. Balkema. *Monotone transformations and limit laws.* 1973.

46 R.P. van de Riet. *ABC ALGOL, a portable language for formula manipulation systems, part 1: the language.* 1973.

47 R.P. van de Riet. *ABC ALGOL, a portable language for formula manipulation systems, part 2: the compiler.* 1973.

48 F.E.J. Kruseman Aretz, P.J.W. ten Hagen, H.L. Oudshoorn. *An ALGOL 60 compiler in ALGOL 60, text of the MC-compiler for the EL-X8.* 1973.

49 H. Kok. *Connected orderable spaces.* 1974.

50 A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.G.L.T. Meertens, R.G. Fisker (eds.). *Revised report on the algorithmic language ALGOL 68.* 1976.

51 A. Hordijk. *Dynamic programming and Markov potential theory.* 1974.

52 P.C. Baayen (ed.). *Topological structures.* 1974.

53 M.J. Faber. *Metrizability in generalized ordered spaces.* 1974.

54 H.A. Lauwerier. *Asymptotic analysis, part 1.* 1974.

55 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 1: theory of designs, finite geometry and coding theory.* 1974.

56 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 2: graph theory, foundations, partitions and combinatorial geometry.* 1974.

57 M. Hall, Jr., J.H. van Lint (eds.). *Combinatorics, part 3: combinatorial group theory.* 1974.

58 W. Albers. *Asymptotic expansions and the deficiency concept in statistics.* 1975.

59 J.L. Mijnheer. *Sample path properties of stable processes.* 1975.

60 F. Göbel. *Queueing models involving buffers.* 1975.

63 J.W. de Bakker (ed.). *Foundations of computer science.* 1975.

64 W.J. de Schipper. *Symmetric closed categories.* 1975.

65 J. de Vries. *Topological transformation groups, 1: a categorical approach.* 1975.

66 H.G.J. Pijls. *Logically convex algebras in spectral theory and eigenfunction expansions.* 1976.

68 P.P.N. de Groen. *Singularly perturbed differential operators of second order.* 1976.

69 J.K. Lenstra. *Sequencing by enumerative methods.* 1977.

70 W.P. de Roever, Jr. *Recursive program schemes: semantics and proof theory.* 1976.

71 J.A.E.E. van Nunen. *Contracting Markov decision processes.* 1976.

72 J.K.M. Jansen. *Simple periodic and non-periodic Lamé functions and their applications in the theory of conical waveguides.* 1977.

73 D.M.R. Leivant. *Absoluteness of intuitionistic logic.* 1979.

74 H.J.J. te Riele. *A theoretical and computational study of generalized aliquot sequences.* 1976.

75 A.E. Brouwer. *Treelike spaces and related connected topological spaces.* 1977.

76 M. Rem. *Associons and the closure statements.* 1976.

77 W.C.M. Kallenberg. *Asymptotic optimality of likelihood ratio tests in exponential families.* 1978.

78 E. de Jonge, A.C.M. van Rooij. *Introduction to Riesz spaces.* 1977.

79 M.C.A. van Zuijlen. *Empirical distributions and rank statistics.* 1977.

80 P.W. Hemker. *A numerical study of stiff two-point boundary problems.* 1977.

81 K.R. Apt, J.W. de Bakker (eds.). *Foundations of computer science II, part 1.* 1976.

82 K.R. Apt, J.W. de Bakker (eds.). *Foundations of computer science II, part 2.* 1976.

83 L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH system.* 1979.

84 H.L.L. Busard. *The translation of the elements of Euclid from the Arabic into Latin by Hermann of Carinthia (?), books vii-xii.* 1977.

85 J. van Mill. *Supercompactness and Wallmann spaces.* 1977.

86 S.G. van der Meulen, M. Veldhorst. *Torrix I, a programming system for operations on vectors and matrices over arbitrary fields and of variable size.* 1978.

88 A. Schrijver. *Matroids and linking systems.* 1977.

89 J.W. de Roever. *Complex Fourier transformation and analytic functionals with unbounded carriers.* 1978.

90 L.P.J. Groenewegen. *Characterization of optimal strategies in dynamic games.* 1981.

91 J.M. Geysel. *Transcendence in fields of positive characteristic.* 1979.

92 P.J. Weeda. *Finite generalized Markov programming.* 1979.

93 H.C. Tijms, J. Wessels (eds.). *Markov decision theory.* 1977.

94 A. Bijlsma. *Simultaneous approximations in transcendental number theory.* 1978.

95 K.M. van Hee. *Bayesian control of Markov chains.* 1978.

96 P.M.B. Vitányi. *Lindenmayer systems: structure, languages, and growth functions.* 1980.

97 A. Federgruen. *Markovian control problems; functional equations and algorithms.* 1984.

98 R. Geel. *Singular perturbations of hyperbolic type.* 1978.

99 J.K. Lenstra, A.H.G. Rinnooy Kan, P. van Emde Boas (eds.). *Interfaces between computer science and operations research.* 1978.

100 P.C. Baayen, D. van Dulst, J. Oosterhoff (eds.). *Proceedings bicentennial congress of the Wiskundig Genootschap, part 1.* 1979.

101 P.C. Baayen, D. van Dulst, J. Oosterhoff (eds.). *Proceedings bicentennial congress of the Wiskundig Genootschap, part 2.* 1979.

102 D. van Dulst. *Reflexive and superreflexive Banach spaces.* 1978.

103 K. van Harn. *Classifying infinitely divisible distributions by functional equations.* 1978.

104 J.M. van Wouwe. *GO-spaces and generalizations of metrizability.* 1979.

105 R. Helmers. *Edgeworth expansions for linear combinations of order statistics.* 1982.

106 A. Schrijver (ed.). *Packing and covering in combinatorics.* 1979.

107 C. den Heijer. *The numerical solution of nonlinear operator equations by imbedding methods.* 1979.

108 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science III, part 1.* 1979.

109 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science III, part 2.* 1979.

110 J.C. van Vliet. *ALGOL 68 transput, part I: historical review and discussion of the implementation model.* 1979.

111 J.C. van Vliet. *ALGOL 68 transput, part II: an implementation model.* 1979.

112 H.C.P. Berbee. *Random walks with stationary increments and renewal theory.* 1979.

113 T.A.B. Snijders. *Asymptotic optimality theory for testing problems with restricted alternatives.* 1979.

114 A.J.E.M. Janssen. *Application of the Wigner distribution to harmonic analysis of generalized stochastic processes.* 1979.

115 P.C. Baayen, J. van Mill (eds.). *Topological structures II, part 1.* 1979.

116 P.C. Baayen, J. van Mill (eds.). *Topological structures II, part 2.* 1979.

117 P.J.M. Kallenberg. *Branching processes with continuous state space.* 1979.

118 P. Groeneboom. *Large deviations and asymptotic efficiencies.* 1980.

119 F.J. Peters. *Sparse matrices and substructures, with a novel implementation of finite element algorithms.* 1980.

120 W.P.M. de Ruyter. *On the asymptotic analysis of large-scale ocean circulation.* 1980.

121 W.H. Haemers. *Eigenvalue techniques in design and graph theory.* 1980.

122 J.C.P. Bus. *Numerical solution of systems of nonlinear equations.* 1980.

123 I. Yuhász. *Cardinal functions in topology - ten years later.* 1980.

124 R.D. Gill. *Censoring and stochastic integrals.* 1980.

125 R. Eising. *2-D systems, an algebraic approach.* 1980.

126 G. van der Hoek. *Reduction methods in nonlinear programming.* 1980.

127 J.W. Klop. *Combinatory reduction systems.* 1980.

128 A.J.J. Talman. *Variable dimension fixed point algorithms and triangulations.* 1980.

129 G. van der Laan. *Simplicial fixed point algorithms.* 1980.

130 P.J.W. ten Hagen, T. Hagen, P. Klint, H. Noot, H.J. Sint, A.H. Veen. *ILP: intermediate language for pictures.* 1980.

131 R.J.R. Back. *Correctness preserving program refinements: proof theory and applications.* 1980.

132 H.M. Mulder. *The interval function of a graph.* 1980.

133 C.A.J. Klaassen. *Statistical performance of location estimators.* 1981.

134 J.C. van Vliet, H. Wupper (eds.). *Proceedings international conference on ALGOL 68.* 1981.

135 J.A.G. Groenendijk, T.M.V. Janssen, M.J.B. Stokhof (eds.). *Formal methods in the study of language, part I.* 1981.

136 J.A.G. Groenendijk, T.M.V. Janssen, M.J.B. Stokhof (eds.). *Formal methods in the study of language, part II.* 1981.

137 J. Telgen. *Redundancy and linear programs.* 1981.

138 H.A. Lauwerier. *Mathematical models of epidemics.* 1981.

139 J. van der Wal. *Stochastic dynamic programming, successive approximations and nearly optimal strategies for Markov decision processes and Markov games.* 1981.

140 J.H. van Geldrop. *A mathematical theory of pure exchange economies without the no-critical-point hypothesis.* 1981.

141 G.E. Welters. *Abel-Jacobi isogenies for certain types of Fano threefolds.* 1981.

142 H.R. Bennett, D.J. Lutzer (eds.). *Topology and order structures, part 1.* 1981.

143 J.M. Schumacher. *Dynamic feedback in finite- and infinite-dimensional linear systems.* 1981.

144 P. Eijgenraam. *The solution of initial value problems using interval arithmetic; formulation and analysis of an algorithm.* 1981.

145 A.J. Brentjes. *Multi-dimensional continued fraction algorithms.* 1981.

146 C.V.M. van der Mee. *Semigroup and factorization methods in transport theory.* 1981.

147 H.H. Tigelaar. *Identification and informative sample size.* 1982.

148 L.C.M. Kallenberg. *Linear programming and finite Markovian control problems.* 1983.

149 C.B. Huijsmans, M.A. Kaashoek, W.A.J. Luxemburg, W.K. Vietsch (eds.). *From A to Z, proceedings of a symposium in honour of A.C. Zaanen.* 1982.

150 M. Veldhorst. *An analysis of sparse matrix storage schemes.* 1982.

151 R.J.M.M. Does. *Higher order asymptotics for simple linear rank statistics.* 1982.

152 G.F. van der Hoeven. *Projections of lawless sequencies.* 1982.

153 J.P.C. Blanc. *Application of the theory of boundary value problems in the analysis of a queueing model with paired services.* 1982.

154 H.W. Lenstra, Jr., R. Tijdeman (eds.). *Computational methods in number theory, part I.* 1982.

155 H.W. Lenstra, Jr., R. Tijdeman (eds.). *Computational methods in number theory, part II.* 1982.

156 P.M.G. Apers. *Query processing and data allocation in distributed database systems.* 1983.

157 H.A.W.M. Kneppers. *The covariant classification of two-dimensional smooth commutative formal groups over an algebraically closed field of positive characteristic.* 1983.

158 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science IV, distributed systems, part 1.* 1983.

159 J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of computer science IV, distributed systems, part 2.* 1983.

160 A. Rezus. *Abstract AUTOMATH.* 1983.

161 G.F. Helminck. *Eisenstein series on the metaplectic group, an algebraic approach.* 1983.

162 J.J. Dik. *Tests for preference.* 1983.

163 H. Schippers. *Multiple grid methods for equations of the second kind with applications in fluid mechanics.* 1983.

164 F.A. van der Duyn Schouten. *Markov decision processes with continuous time parameter.* 1983.

165 P.C.T. van der Hoeven. *On point processes.* 1983.

166 H.B.M. Jonkers. *Abstraction, specification and implementation techniques, with an application to garbage collection.* 1983.

167 W.H.M. Zijm. *Nonnegative matrices in dynamic programming.* 1983.

168 J.H. Evertse. *Upper bounds for the numbers of solutions of diophantine equations.* 1983.

169 H.R. Bennett, D.J. Lutzer (eds.). *Topology and order structures, part 2.* 1983.

CWI TRACTS

1 D.H.J. Epema. *Surfaces with canonical hyperplane sections.* 1984.

2 J.J. Dijkstra. *Fake topological Hilbert spaces and characterizations of dimension in terms of negligibility.* 1984.

3 A.J. van der Schaft. *System theoretic descriptions of physical systems.* 1984.

4 J. Koene. *Minimal cost flow in processing networks, a primal approach.* 1984.

5 B. Hoogenboom. *Intertwining functions on compact Lie groups.* 1984.

6 A.P.W. Böhm. *Dataflow computation.* 1984.

7 A. Blokhuis. *Few-distance sets.* 1984.

8 M.H. van Hoorn. *Algorithms and approximations for queueing systems.* 1984.

9 C.P.J. Koymans. *Models of the lambda calculus.* 1984.

10 C.G. van der Laan, N.M. Temme. *Calculation of special functions: the gamma function, the exponential integrals and error-like functions.* 1984.

11 N.M. van Dijk. *Controlled Markov processes; time-discretization.* 1984.

12 W.H. Hundsdorfer. *The numerical solution of nonlinear stiff initial value problems: an analysis of one step methods.* 1985.

13 D. Grune. *On the design of ALEPH.* 1985.

14 J.G.F. Thiemann. *Analytic spaces and dynamic programming: a measure theoretic approach.* 1985.

15 F.J. van der Linden. *Euclidean rings with two infinite primes.* 1985.

16 R.J.P. Groothuizen. *Mixed elliptic-hyperbolic partial differential operators: a case-study in Fourier integral operators.* 1985.

17 H.M.M. ten Eikelder. *Symmetries for dynamical and Hamiltonian systems.* 1985.

18 A.D.M. Kester. *Some large deviation results in statistics.* 1985.

19 T.M.V. Janssen. *Foundations and applications of Montague grammar, part 1: Philosophy, framework, computer science.* 1986.

20 B.F. Schriever. *Order dependence.* 1986.

21 D.P. van der Vecht. *Inequalities for stopped Brownian motion.* 1986.

22 J.C.S.P. van der Woude. *Topological dynamix.* 1986.

23 A.F. Monna. *Methods, concepts and ideas in mathematics: aspects of an evolution.* 1986.

24 J.C.M. Baeten. *Filters and ultrafilters over definable subsets of admissible ordinals.* 1986.

25 A.W.J. Kolen. *Tree network and planar rectilinear location theory.* 1986.

26 A.H. Veen. *The misconstrued semicolon: Reconciling imperative languages and dataflow machines.* 1986.

27 A.J.M. van Engelen. *Homogeneous zero-dimensional absolute Borel sets.* 1986.

28 T.M.V. Janssen. *Foundations and applications of Montague grammar, part 2: Applications to natural language.* 1986.

29 H.L. Trentelman. *Almost invariant subspaces and high gain feedback.* 1986.

30 A.G. de Kok. *Production-inventory control models: approximations and algorithms.* 1987.

31 E.E.M. van Berkum. *Optimal paired comparison designs for factorial experiments.* 1987.

32 J.H.J. Einmahl. *Multivariate empirical processes.* 1987.

33 O.J. Vrieze. *Stochastic games with finite state and action spaces.* 1987.

34 P.H.M. Kersten. *Infinitesimal symmetries: a computational approach.* 1987.

35 M.L. Eaton. *Lectures on topics in probability inequalities.* 1987.

36 A.H.P. van der Burgh, R.M.M. Mattheij (editors). *Proceedings of the first international conference on industrial and applied mathematics (ICIAM 87).* 1987.

37 L. Stougie. *Design and analysis of algorithms for stochastic integer programming.* 1987.

38 J.B.G. Frenk. *On Banach algebras, renewal measures and regenerative processes.* 1987.

39 H.J.M. Peters, O.J. Vrieze (eds.). *Surveys in game theory and related topics.* 1987.

40 J.L. Geluk, L. de Haan. *Regular variation, extensions and Tauberian theorems.* 1987.

41 Sape J. Mullender (ed.). *The Amoeba distributed operating system: Selected papers 1984-1987.* 1987.

42 P.R.J. Asveld, A. Nijholt (eds.). *Essays on concepts, formalisms, and tools.* 1987.

43 H.L. Bodlaender. *Distributed computing: structure and complexity.* 1987.

44 A.W. van der Vaart. *Statistical estimation in large parameter spaces.* 1988.

45 S.A. van de Geer. *Regression analysis and empirical processes.* 1988.

46 S.P. Spekreijse. *Multigrid solution of the steady Euler equations.* 1988.

47 J.B. Dijkstra. *Analysis of means in some non-standard situations.* 1988.

48 F.C. Drost. *Asymptotics for generalized chi-square goodness-of-fit tests.* 1988.

49 F.W. Wubs. *Numerical solution of the shallow-water equations.* 1988.

50 F. de Kerf. *Asymptotic analysis of a class of perturbed Korteweg-de Vries initial value problems.* 1988.

51 P.J.M. van Laarhoven. *Theoretical and computational aspects of simulated annealing.* 1988.

52 P.M. van Loon. *Continuous decoupling transformations for linear boundary value problems.* 1988.

53 K.C.P. Machielsen. *Numerical solution of optimal control problems with state constraints by sequential quadratic programming in function space.* 1988.

54 L.C.R.J. Willenborg. *Computational aspects of survey data processing.* 1988.

55 G.J. van der Steen. *A program generator for recognition, parsing and transduction with syntactic patterns.* 1988.

56 J.C. Ebergen. *Translating programs into delay-insensitive circuits.* 1989.

57 S.M. Verduyn Lunel. *Exponential type calculus for linear delay equations.* 1989.

58 M.C.M. de Gunst. *A random model for plant cell population growth.* 1989.

59 D. van Dulst. *Characterizations of Banach spaces not containing l^1.* 1989.

60 H.E. de Swart. *Vacillation and predictability properties of low-order atmospheric spectral models.* 1989.

61 P. de Jong. *Central limit theorems for generalized multilinear forms.* 1989.
62 V.J. de Jong. *A specification system for statistical software.* 1989.
63 B. Hanzon. *Identifiability, recursive identification and spaces of linear dynamical systems, part I.* 1989.
64 B. Hanzon. *Identifiability, recursive identification and spaces of linear dynamical systems, part II.* 1989.
65 B.M.M. de Weger. *Algorithms for diophantine equations.* 1989.
66 A. Jung. *Cartesian closed categories of domains.* 1989.
67 J.W. Polderman. *Adaptive control & identification: Conflict or conflux?.* 1989.
68 H.J. Woerdeman. *Matrix and operator extensions.* 1989.
69 B.G. Hansen. *Monotonicity properties of infinitely divisible distributions.* 1989.
70 J.K. Lenstra, H.C. Tijms, A. Volgenant (eds.). *Twenty-five years of operations research in the Netherlands: Papers dedicated to Gijs de Leve.* 1990.
71 P.J.C. Spreij. *Counting process systems. Identification and stochastic realization.* 1990.
72 J.F. Kaashoek. *Modeling one dimensional pattern formation by anti-diffusion.* 1990.
73 A.M.H. Gerards. *Graphs and polyhedra. Binary spaces and cutting planes.* 1990.
74 B. Koren. *Multigrid and defect correction for the steady Navier-Stokes equations. Application to aerodynamics.* 1991.
75 M.W.P. Savelsbergh. *Computer aided routing.* 1992.
76 O.E. Flippo. *Stability, duality and decomposition in general mathematical programming.* 1991.
77 A.J. van Es. *Aspects of nonparametric density estimation.* 1991.
78 G.A.P. Kindervater. *Exercises in parallel combinatorial computing.* 1992.
79 J.J. Lodder. *Towards a symmetrical theory of generalized functions.* 1991.
80 S.A. Smulders. *Control of freeway traffic flow.* 1996.
81 P.H.M. America, J.J.M.M. Rutten. *A parallel object-oriented language: design and semantic foundations.* 1992.
82 F. Thuijsman. *Optimality and equilibria in stochastic games.* 1992.
83 R.J. Kooman. *Convergence properties of recurrence sequences.* 1992.
84 A.M. Cohen (ed.). *Computational aspects of Lie group representations and related topics. Proceedings of the 1990 Computational Algebra Seminar at CWI, Amsterdam.* 1991.
85 V. de Valk. *One-dependent processes.* 1994.
86 J.A. Baars, J.A.M. de Groot. *On topological and linear equivalence of certain function spaces.* 1992.
87 A.F. Monna. *The way of mathematics and mathematicians.* 1992.
88 E.D. de Goede. *Numerical methods for the three-dimensional shallow water equations.* 1993.
89 M. Zwaan. *Moment problems in Hilbert space with applications to magnetic resonance imaging.* 1993.
90 C. Vuik. *The solution of a one-dimensional Stefan problem.* 1993.
91 E.R. Verheul. *Multimedians in metric and normed spaces.* 1993.
92 J.L.M. Maubach. *Iterative methods for non-linear partial differential equations.* 1994.
93 A.W. Ambergen. *Statistical uncertainties in posterior probabilities.* 1993.
94 P.A. Zegeling. *Moving-grid methods for time-dependent partial differential equations.* 1993.
95 M.J.C. van Pul. *Statistical analysis of software reliability models.* 1993.
96 J.K. Scholma. *A Lie algebraic study of some integrable systems associated with root systems.* 1993.
97 J.L. van den Berg. *Sojourn times in feedback and processor sharing queues.* 1993.
98 A.J. Koning. *Stochastic integrals and goodness-of-fit tests.* 1993.
99 B.P. Sommeijer. *Parallelism in the numerical integration of initial value problems.* 1993.
100 J. Molenaar. *Multigrid methods for semiconductor device simulation.* 1993.
101 H.J.C. Huijberts. *Dynamic feedback in nonlinear synthesis problems.* 1994.
102 J.A.M. van der Weide. *Stochastic processes and point processes of excursions.* 1994.
103 P.W. Hemker, P. Wesseling (eds.). *Contributions to multigrid.* 1994.
104 I.J.B.F. Adan. *A compensation approach for queueing problems.* 1994.
105 O.J. Boxma, G.M. Koole (eds.). *Performance evaluation of parallel and distributed systems - solution methods. Part 1.* 1994.
106 O.J. Boxma, G.M. Koole (eds.). *Performance evaluation of parallel and distributed systems - solution methods. Part 2.* 1994.
107 R.A. Trompert. *Local uniform grid refinement for time-dependent partial differential equations.* 1995.
108 M.N.M. van Lieshout. *Stochastic geometry models in image analysis and spatial statistics.* 1995.
109 R.J. van Glabbeek. *Comparative concurrency semantics and refinement of actions.* 1996.
110 W. Vervaat, H. Holwerda (ed.). *Probability and lattices.* 1997.
111 I. Helsloot. *Covariant formal group theory and some applications.* 1995.
112 R.N. Bol. *Loop checking in logic programming.* 1995.
113 G.J.M. Koole. *Stochastic scheduling and dynamic programming.* 1995.
114 M.J. van der Laan. *Efficient and inefficient estimation in semiparametric models.* 1995.
115 S.C. Borst. *Polling models.* 1996.
116 G.D. Otten. *Statistical test limits in quality control.* 1996.
117 K.G. Langendoen. *Graph reduction on shared-memory multiprocessors.* 1996.
118 W.C.A. Maas. *Nonlinear $\mathcal{H}_\infty$ control: the singular case.* 1996.
119 A. Di Bucchianico. *Probabilistic and analytical aspects of the umbral calculus.* 1997.
120 M. van Loon. *Numerical methods in smog prediction.* 1997.
121 B.J. Wijers. *Nonparametric estimation for a windowed line-segment process.* 1997.
122 W.K. Klein Haneveld, O.J. Vrieze, L.C.M. Kallenberg (editors). *Ten years LNMB – Ph.D. research and graduate courses of the Dutch Network of Operations Research.* 1997.
123 R.W. van der Hofstad. *One-dimensional random polymers.* 1998.

124 W.J.H. Stortelder. *Parameter estimation in nonlinear dynamical systems.* 1998.

125 M.H. Wegkamp. *Entropy methods in statistical estimation.* 1998.

126 K. Aardal, J.K. Lenstra, F. Maffioli, D.B. Shmoys (eds.) *Selected publications of Eugene L. Lawler.* 1999.

Selected publications of Eugene L. Lawler

edited by

Karen Aardal, Jan Karel Lenstra, Francesco Maffioli, David B. Shmoys

Summary. Eugene L. Lawler (1933–1994) was one of the earliest researchers who concentrated on combinatorial optimization as a field of investigation. After his work in switching theory in the 1960's, he became interested in networks and matroids and in sequencing and scheduling. In the 1990's he investigated combinatorial problems in computational biology. As for algorithmic approaches, he started by studying enumerative methods, first branch-and-bound and then dynamic programming. Under the influence of complexity theory, he turned to polynomial-time optimization algorithms and to approximation algorithms with performance guarantees.

This volume contains a selection of twenty-six of his publications and gives a representative picture of his work. It illustrates the development of his research interests, and also includes a fair amount of his writings for a more general audience.