Sequencing by Enumerative Methods

J.K. Lenstra

SEQUENCING BY ENUMERATIVE METHODS

SEQUENCING BY ENUMERATIVE METHODS

ACADEMISCH PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN
DOCTOR IN DE WISKUNDE EN NATUURWETENSCHAPPEN
AAN DE UNIVERSITEIT VAN AMSTERDAM
OP GEZAG VAN DE RECTOR MAGNIFICUS
DR. G. DEN BOEF
HOOGLERAAR IN DE FACULTEIT
DER WISKUNDE EN NATUURWETENSCHAPPEN
IN HET OPENBAAR TE VERDEDIGEN
IN DE AULA DER UNIVERSITEIT
(TIJDELIJK IN DE LUTHERSE KERK, INGANG SINGEL 411, HOEK SPUI)
OP WOENSDAG 25 FEBRUARI 1976 DES NAMIDDAGS TE 4.00 UUR

DOOR

JAN KAREL LENSTRA

GEBOREN TE ZAANDAM

1976
MATHEMATISCH CENTRUM, AMSTERDAM

PROMOTOR: PROF.DR. G. DE LEVE

COREFERENT: PROF.DR.IR. J.S. FOLKERS

PREFACE

This study was written at the Department of Operations Research of the Mathematisch Centrum in Amsterdam under the supervision of professor G. de Leve. I am very grateful to him for arousing my interest in the area of sequencing and for giving me complete freedom in pursuing my research in this field. Also, I owe great appreciation to professor J.S. Folkers of the Graduate School of Management in Delft, who provided detailed criticism and encouragement.

Large parts of this book evolved from my intensive cooperation with Alexander Rinnooy Kan. I am similarly grateful to Ben Lageweg, who conducted all computational experiments with machine scheduling algorithms. Without their participation, this book would not have been written.

Furthermore, I have benefited from many inspiring discussions with Jack Anthonisse, Gene Lawler, Hendrik Lenstra, Jr., and Tom Wansbeek.

I want to acknowledge the various useful contributions and suggestions from Jack Alanen, Peter Brucker, Bernard Dorhout, Peter van Emde Boas, Mike Florian, Graham McMahon, and Ira Pohl. The applications reported in this study arose from practical problems advanced by J. Berendse, J.H. Galjaard, P. ten Kate, J.S. Knipscheer, J.H. Kuiper, A.W. Roes, and J. Visschers.

With respect to the technical realization of this book, I would like to thank Tona Bays for typing the manuscript, Tobias Baanders for the cover design, Messrs. D. Zwarst, J. Suiker and J. Schipper for the printing, and the Typographical Consulting Group "Oosterpark" for the general supervision.

Finally, I am grateful to my parents for providing a stimulating educational environment, and I owe many thanks to my wife Angeline for her assistance and support.

Jan Karel Lenstra

Amsterdam, January, 1976

CONTENTS

Part III. Sequencing by implicit enumeration

All this is quite genuine mathematics, and has its merits; but it is just that 'proof by enumeration of cases' (and of cases which do not, at bottom, differ at all profoundly) which a real mathematician tends to despise.

G.H. Hardy,
A Mathematician's Apology.

1. INTRODUCTION

The problem of determining an optimal sequence arises under many circumstances. We may wish to schedule jobs on a machine, to route vehicles from depots to customers or to specify a chronological ordering of archeological finds. Each of these situations leads to problems of *combinatorial optimization* in which we seek to find the optimal element within a large but finite set of feasible solutions.

In this study, we shall be especially interested in two classes of problems. One class contains the *quadratic assignment problem* and its various specializations. These include the *acyclic subgraph problem* of finding a total ordering that resembles a set of pairwise preferences as closely as possible, and the *travelling salesman problem* in which a salesman wishes to find the shortest route through a number of cities and back home again. Both problems have many surprising applications.

The larger part of this study is devoted to *machine scheduling problems*. These problems occur whenever *jobs* have to be scheduled on *machines* of limited capacity. More specifically, each job is defined to consist of a sequence of operations, each of which is to be performed on some machine during a given period of time. Given some overall criterion to measure the quality of each possible schedule, we want to find an optimal processing order on each machine.

Both classes of problems are typical examples of combinatorial optimization problems and as such the classical tools of combinatorial programming are available to solve them. On one hand, there exist ingenious algorithms that are *good* or *efficient* in the familiar sense of requiring a predictable number of steps bounded by some polynomial function of problem size; on the other hand, quite often there seems to be no alternative but unpredictable *enumerative methods*.

Recent results in the theory of computational complexity allow a more formal analysis of the question to what extent such methods of explicit or implicit enumeration are really unavoidable. A class of difficult combinatorial problems has been identified with the strong property that a polynomial-bounded algorithm for any of these problems would provide good algorithms for all the others as well. In view of the fact that many notorious problems such as the 0-1 programming problem, the graph coloring problem and the set covering problem are members of this class of so-called NP-complete problems, the existence of such an algorithm is highly unlikely. Therefore, proving that a certain sequencing problem belongs to this class as well can be used as a formal justification to apply enumerative solution methods, since no substantially better method is likely to exist.

Along these lines, we investigate the *complexity* of sequencing problems in Part I. Our results offer a detailed insight into the location of the borderline between "easy" and "hard" sequencing problems. It turns out that most of them indeed require a solution approach based on enumeration of the set of feasible solutions.

By the very nature of enumerative methods, their performance depends on the specific computer implementation adopted. This motivates an in-depth study of an approach that we have found to be particularly attractive, namely a *recursive* approach to the implementation of enumerative methods. In Part II, e demonstrate its properties and virtues on some simply structured schemes of explicit and implicit enumeration.

Part III deals with the solution of sequencing problems by implicit enumeration. With respect to the travelling salesman problem, two one-machine scheduling problems and two m-machine scheduling problems, we survey and extend *branch-and-bound* algorithms and discuss their computational performance. We shall be particularly interested in curtailing the search for an optimal solution as much as possible through the use of sharp bounds on the values of solutions within certain subsets.

We have already alluded to the many practical *applications* of sequencing theory. To illustrate this point in more detail, we shall describe five such applications in Part IV. Each of them arose out of some practical situation and involves a successful solution by means of methods discussed in previous chapters. It is among other things this interplay between theory and practice that makes sequencing problems into such a challenging area within operations research.

We assume the reader of this book to be familiar with the basic principles of mathematical optimization, graph theory, and computer programming. Throughout, graphs will be defined by vertices, (undirected) edges and (directed) arcs; vertices of search trees will be referred to as nodes. Several algorithms will be presented in the form of ALGOL 60 or quasi-ALGOL procedures.

Part I. Sequencing problems

2. COMPLEXITY THEORY

Recent developments in the theory of computational complexity as applied to combinatorial problems have aroused the interest of many researchers. The main credit for this must go to S.A. Cook [Cook 1971] and R.M. Karp [Karp 1972B] who first explored the relation between the classes P and NP of (language recognition) problems solvable by *deterministic* and *non-deterministic* Turing machines respectively, in a number of steps *bounded by a polynomial in the length of the input*. With respect to combinatorial optimization, we do not really require mathematically rigorous definitions of these concepts; for our purposes we may safely identify P with the class of problems for which a *polynomial-bounded*, *good* [Edmonds 1965A] or *efficient algorithm* exists, whereas all problems in NP can be solved by *polynomial-depth backtrack search*.

In this context, all problems are stated in terms of *recognition* problems which require a yes/no answer. In order to deal with the complexity of a combinatorial *minimization* problem, we transform it into the problem of determining the existence of a solution with value at most equal to y, for some *threshold* y.

The class NP is very extensive. All sequencing problems that will be discussed throughout this work can trivially be solved by polynomial-depth backtrack search and thus are members of NP.

It is clear that $P \subset NP$, and the question arises if this inclusion is a proper one or if, on the contrary, $P = NP$. Although this is still an open problem, the equality of P and NP is considered to be highly unlikely and most bets (*e.g.*, in [Knuth 1974]) have been going in the other direction. To examine the consequences of an affirmative answer to the $P = NP$ question, we introduce the following concepts.

- Problem P' is *reducible* to problem P (notation: $P' \propto P$) if for any instance of P' an instance of P can be constructed in polynomial-bounded time such that solving the instance of P will solve the instance of P' as well.
- P' and P are *equivalent* if $P' \propto P$ and $P \propto P'$.
- P is *NP-complete* [Knuth 1974] if $P \in NP$ and $P' \propto P$ for every $P' \in NP$.

Informally, the reducibility of P' to P implies that P' can be considered as a special case of P; the NP-completeness of P indicates that P is, in a sense, the most difficult problem in NP.

In a remarkable paper [Cook 1971], NP-completeness was established with respect to the so-called SATISFIABILITY problem. This problem can be formulated as follows.

Given clauses $C_1,\ldots,C_u$, *each being a disjunction of* literals *from the set* $X = \{x_1,\ldots,x_t,\bar{x}_1,\ldots,\bar{x}_t\}$, *is the conjunction of the clauses* satisfiable, i.e., *does there exist a subset* $S \subset X$ *such that*

- S *does not contain a* complementary *pair of literals* $(x_i,\bar{x}_i)$, *and*
- $S \cap C_j \neq \emptyset$ *for* $j = 1,\ldots,u$?

Cook proves this result by specifying a polynomial-bounded "master reduction" which, given $P \in NP$, constructs for any instance of P an equivalent boolean expression in conjunctive normal form. By means of this reduction, a polynomial-bounded algorithm for the SATISFIABILITY problem could be used to construct a polynomial-bounded algorithm for any problem in NP. It follows that

$$P = NP \text{ if and only if } \text{SATISFIABILITY} \in P.$$

The same argument applies if we replace SATISFIABILITY by any NP-complete problem. A large number of such problems has been identified in [Karp 1972B] (see also [Karp 1975A]). Since they are all notorious combinatorial problems for which typically no good algorithms have been found so far, Karp's results afford strong circumstantial evidence that P is a proper subset of NP.

Theorem 2.1 lists those NP-complete problems that will be used in Chapters 3 and 4 to establish NP-completeness of sequencing problems.

THEOREM 2.1. *The following problems are NP-complete:*

(*a*) 3-SATISFIABILITY

I.e. SATISFIABILITY *with at most three literals per clause.*

(*b*) CLIQUE

Given an undirected graph $G = (V,E)$ *and an integer* k, *does* G *have a* clique *(i.e. a complete subgraph) on* k *vertices?*

(*c*) LINEAR ARRANGEMENT

Given an undirected graph $G = (V,E)$ *and an integer* k, *does there exist a one-to-one function* $\pi: V \rightarrow \{1,\ldots,|V|\}$ *such that* $\sum_{(i,j)\in E}|\pi(i)-\pi(j)| \leq k$?

(*d*) FEEDBACK ARC SET

Given a directed graph $G = (V,A)$ *and an integer* k, *does* G *have a* feedback arc set *(i.e. a set of arcs whose removal breaks are directed cycles) of cardinality* k?

(*e*) DIRECTED HAMILTONIAN CIRCUIT

Given a directed graph $G = (V,A)$, *does* G *have a* hamiltonian circuit *(i.e. a directed cycle passing through each vertex exactly once)?*

(*f*) DIRECTED HAMILTONIAN PATH

Given a directed graph $G' = (V',A')$, *does* G' *have a* hamiltonian path *(i.e. a directed path passing through each vertex exactly once)?*

(*g*) UNDIRECTED HAMILTONIAN CIRCUIT

Given an undirected graph $G = (V,E)$, *does* G *have a* hamiltonian circuit *(i.e. an undirected cycle passing through each vertex exactly once)?*

(*h*) KNAPSACK

Given positive integers $a_1,\ldots,a_t,b$, *does there exist a subset* $S \subset T = \{1,\ldots,t\}$ *such that* $\sum_{i\in S} a_i = b$?

(*i*) PARTITION

Given positive integers $a_1,\ldots,a_t$, *does there exist a subset* $S \subset T = \{1,\ldots,t\}$ *such that* $\sum_{i\in S} a_i = \sum_{i\in T-S} a_i$?

(*j*) 3-PARTITION

Given positive integers $a_1,\ldots,a_{3t},b$, *does there exist a partition* $(T_1,\ldots,T_t)$ *of* $T = \{1,\ldots,3t\}$ *such that* $|T_j| = 3$ *and* $\sum_{i\in T_j} a_i = b$ *for* $j = 1,\ldots,t$?

Proof.

(*a,b*) See [Cook 1971; Karp 1972B].

(*c*) See [Garey *et al.* 1974].

(*d,e,g,h,i*) See [Karp 1972B].

(*f*) NP-completeness of this problem is implied by two observations:

(*A*) DIRECTED HAMILTONIAN PATH $\in NP$;

(*B*) $P \propto$ DIRECTED HAMILTONIAN PATH for some NP-complete problem P.

(*A*) is trivially true, and (*B*) is proved by the following reduction.

DIRECTED HAMILTONIAN CIRCUIT $\propto$ DIRECTED HAMILTONIAN PATH.

Given $G = (V,A)$, we choose $i' \in V$ and construct $G' = (V',A')$ with

$$V' = V \cup \{i''\},$$
$$A' = \{(i,j) \mid (i,j) \in A,\ j \neq i'\} \cup \{(i,i'') \mid (i,i') \in A\}.$$

G has a hamiltonian circuit if and only if G' has a hamiltonian path.

(*j*) See [Garey & Johnson 1974]. □

Karp's work has led to a large amount of research on the location of the borderline separating the "easy" problems (in P) from the "hard" (NP-complete) ones. It turns out that a minor change in the value of a problem parameter (notably - for some as yet mystical reason - an increase from two to three) often transforms an easy problem into a hard one. Not only does knowledge of the borderline lead to fresh insights as to what characteristics of a problem determine its complexity, but there are also important consequences with respect to the solution of these problems. Establishing NP-completeness of a problem can be interpreted as a formal justification to use enumerative methods such as branch-and-bound, since no substantially better method is likely to exist. Conversely, if a problem is known to be in P, then branch-and-bound should certainly not be used. Investigation of these aspects should prevent embarrassing incidents such as the presentation in a standard text-book of an enumerative approach to the undirected Chinese postman problem, for which a good algorithm had already been developed in [Edmonds 1965B] (see also [Edmonds & Johnson 1973]).

It should be emphasized that membership of P versus NP-completeness only yields a very coarse measure of complexity. On one hand, there are significant differences in complexity within the class of NP-complete problems. On the other hand, the question has been raised whether polynomial-bounded algorithms are really good [Anthonisse & Van Emde Boas 1974].

One possible refinement of the complexity measure may be based on the way in which the problem data can be encoded. Taking the KNAPSACK problem as an example and defining $a_* = \max_{i \in T}\{a_i\}$, we observe that the length of the input is $O(t \log a_*)$ in the standard *binary* encoding, and $O(ta_*)$ if a *unary* encoding is allowed. The KNAPSACK problem has been shown to be NP-complete with respect to a binary encoding; however, solution by dynamic programming requires $O(tb)$ steps and thus yields a polynomial-bounded algorithm with respect to a unary encoding. Similar situations exist for several machine scheduling problems, as will be pointed out in Section 4.2. Such "quasi-polynomial" algorithms [Lawler 1975C] need not necessarily be "good" in the practical sense of the word, but it may pay none the less to distinguish between complexity results with respect to unary and binary encodings (*cf.* [Garey *et al.* 1975]). *Unary NP-completeness* or *binary membership of P* would then be the strongest possible result, and it is quite feasible for a problem to be *binary NP-complete* and still to allow a *unary polynomial-bounded* solution. Here, we shall not explore this distinction any further; all our results hold with respect to the standard binary encoding.

Other refinements of the complexity measure may be based on the worst-case analysis of approximate algorithms. For relatively simple problems, there often exist heuristics for which the ratio of the obtained solution value to the optimal value is bounded by a constant, whereas in other cases this worst-case bound depends on the size of the problem (see [Graham 1969; Johnson 1973; Rosenkrantz *et al.* 1974; Garey & Graham 1975; Gonzales & Sahni 1975]). Occasionally, there is no hope to obtain good algorithms even if we settle for approximation, since the problem of finding a feasible solution within any fixed percentage from the optimum has been proved to be NP-complete (see [Sahni & Gonzales 1974; Pohl 1975]).

Altogether, the development of a measure that allows further distinction within the class of NP-complete problems remains a major research challenge.

In the remaining chapters of Part I we will study the complexity of various sequencing problems. The results in Chapter 3 with respect to some types of *quadratic assignment problems* follow from Theorem 2.1 in a fairly straightforward way. Chapter 4 is devoted to *machine scheduling problems*. In this area, a natural problem classification is available and it is particularly challenging to investigate the influence of various parameter values on the complexity of the problems.

3. QUADRATIC ASSIGNMENT PROBLEMS

3.1. The quadratic assignment problem

The quadratic assignment problem (QAP) can be stated as follows.

Given n^4 coefficients a_{ghij} $(g,h,i,j = 1,\ldots,n)$, find a permutation π of $\{1,\ldots,n\}$ minimizing

$$f_{QAP}(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi(i)\pi(j)ij}.$$

We will restrict our attention to the special case where, given two $n\times n$-matrices (c_{ij}) and (d_{ij}), we have $a_{ghij} = c_{gh}d_{ij}$ and therefore

$$f_{QAP}(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{\pi(i)\pi(j)}d_{ij}.$$

This formulation is given in [Koopmans & Beckmann 1957] in the context of the location of economic activities; $f_{QAP}(\pi)$ represents total transportation costs if plants $1,\ldots,n$ are assigned to locations $\pi(1),\ldots,\pi(n)$ respectively, and d_{ij} units are shipped from plant i at location $\pi(i)$ to plant j at location $\pi(j)$ at cost $c_{\pi(i)\pi(j)}$ per unit. The QAP arises in various other situations such as planning a presidential election campaign [Lawler 1963], arranging wedding guests round a table [Müller-Merbach 1970], placing modules on a computer backplane [Hanan & Kurtzberg 1972] (*cf.* Section 14.2.2) and scheduling parallel machines with changeover costs [Geoffrion & Graves 1975].

Some special cases of the QAP can be solved by polynomial-bounded algorithms. For instance, if the locations are situated on a straight line at unit intervals so that $c_{ij} = |i-j|$, and moreover $d_{ij} = e_i e_j$ for some nonnegative $e_1,\ldots,e_n$, then an optimal assignment can be found in $O(n \log n)$ steps [Pratt 1972]. Other special cases are discussed in [Lawler 1975A].

The general QAP, however, is an NP-complete problem. This is implied by its membership of NP, which is obvious, and by the results presented in Theorem 3.1. In this theorem, we formulate three NP-complete problems from Theorem 2.1 as a QAP; any of these reductions suffices to establish NP-completeness of the QAP, and together they illustrate the generality of this sequencing problem.

We note that, in order to state the QAP as a recognition problem, we add a threshold parameter y to the problem specification and investigate the existence of a solution π with value $f_{QAP}(\pi) \le y$.

THEOREM 3.1. *The following problems are reducible to the QAP:*

(*a*) CLIQUE;

(*b*) LINEAR ARRANGEMENT;

(*c*) DIRECTED HAMILTONIAN PATH.

Proof. The problems under (*a*), (*b*) and (*c*) have been formulated in Theorem 2.1 in terms of an undirected or directed graph, $G = (V,E)$ or $G' = (V',A')$ respectively. Let in each case the vertex set be given by $\{1,\ldots,v\}$.

(*a*) CLIQUE $\propto$ QAP:

$$n = v;$$

$$c_{ij} = \begin{cases} 0 & ((i,j) \in E), \\ 1 & (\text{otherwise}); \end{cases}$$

$$d_{ij} = \begin{cases} 1 & (i,j = 1,\ldots,k), \\ 0 & (\text{otherwise}); \end{cases}$$

$$y = 0.$$

For any permutation π of V we have

$$f_{QAP}(\pi) = \sum_{i=1}^{k} \sum_{j=1}^{k} c_{\pi(i)\pi(j)} \geq 0.$$

CLIQUE has a solution if and only if there exists a π such that $(\pi(i),\pi(j)) \in E$ for $i,j = 1,\ldots,k$, *i.e.* $f_{QAP}(\pi) = 0$.

(*b*) LINEAR ARRANGEMENT $\propto$ QAP:

$$n = v;$$

$$c_{ij} = |i-j| \quad (i,j \in V);$$

$$d_{ij} = \begin{cases} 1 & ((i,j) \in E), \\ 0 & (\text{otherwise}); \end{cases}$$

$$y = 2k.$$

For any permutation π of V we have

$$f_{QAP}(\pi) = 2\sum_{(i,j)\in E} |\pi(i)-\pi(j)|.$$

It follows immediately that LINEAR ARRANGEMENT has a solution if and only if there exists a π such that $f_{QAP}(\pi) \leq 2k$.

(*c*) DIRECTED HAMILTONIAN PATH $\propto$ QAP:

$$n = v;$$

$$c_{ij} = \begin{cases} 0 & ((i,j) \in A'), \\ 1 & (\text{otherwise}); \end{cases}$$

$$d_{ij} = \begin{cases} 1 & (i = 1,\ldots,v-1,\ j = i+1), \\ 0 & (\text{otherwise}); \end{cases}$$

$$y = 0.$$

For any permutation π of V' we have

$$f_{QAP}(\pi) = \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \geq 0.$$

DIRECTED HAMILTONIAN PATH has a solution if and only if there exists a π such that $(\pi(i),\pi(i+1)) \in A'$ for $i = 1,\ldots,n-1$, *i.e.* $f_{QAP}(\pi) = 0$. □

It follows that finding an optimal QAP solution is likely to require some form of implicit enumeration. Branch-and-bound algorithms have been proposed in [Gilmore 1962; Lawler 1963; Land 1963; Gavett & Plyter 1966; Burkard 1973; Hansen & Kaufman 1974] and reviewed in [Pierce & Crowston 1971; Kaufman 1975]; they have been moderately successful in solving problems with $n \leq 15$. Suboptimal methods have been extensively tested with varying degrees of success; we refer to the survey in [Hanan & Kurtzberg 1972]. The QAP is clearly very difficult and little progress has been made since its first formulation.

3.2. The acyclic subgraph problem

The acyclic subgraph problem (ASP) can be stated as follows.

Given a directed graph $G = (V,A)$ with a nonnegative weight c_{ij} for each arc $(i,j) \in A$, find an acyclic subgraph of G of maximum total weight.

If $G' = (V,A')$ with $A' \subset A$ is acyclic, then clearly $A-A'$ is a feedback arc set of G, *i.e.* a set of arcs whose removal breaks all directed cycles. Therefore, the ASP is equivalent to the problem of finding a feedback arc set of minimum total weight.

Since all c_{ij} are nonnegative, we may restrict our attention to *maximal* acyclic subgraphs, *i.e.* acyclic subgraphs $G' = (V,A')$ such that no $G'' = (V,A'')$ with $A' \subsetneq A''$ is acyclic; in this case, $A-A'$ is a *minimal* feedback arc set.

Let $V = \{1,\ldots,n\}$. Defining $c_{ij} = 0$ for $(i,j) \notin A$ and taking $A = V\times V$ obviously does not change the problem. Any maximal acyclic subgraph $G' = (V,A')$ is now characterized by a permutation π of V such that $A' = \{(\pi(i),\pi(j)) \mid i < j\}$. Thus, the ASP can be restated as follows. Given an $n\times n$-matrix (c_{ij}), find a permutation π of $\{1,\ldots,n\}$ maximizing

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi(i)\pi(j)}$$

or, equivalently, minimizing

$$f_{ASP}(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{i-1} c_{\pi(i)\pi(j)}.$$

Note that the c_{ij} are allowed to be negative in this formulation.

The ASP turns out to be a special case of the QAP; we obtain a QAP with $f_{QAP}(\pi) = f_{ASP}(\pi)$ by defining (c_{ij}) as above and (d_{ij}) as follows:

$$d_{ij} = \begin{cases} 1 & (i = 1,\dots,n,\ j = 1,\dots,i-1), \\ 0 & (\text{otherwise}). \end{cases}$$

None the less, the ASP is NP-complete. This follows from Theorem 2.1(*d*) and Theorem 3.2.

THEOREM 3.2. FEEDBACK ARC SET $\propto$ ASP.

Proof. Immediate from the above discussion. □

The ASP arises in widely varying situations and there exists a large incoherent body of literature on the problem. For an extensive survey of its history, mathematical aspects, optimal and suboptimal algorithms, we refer to [Lenstra Jr. 1973A]; see Chapter 7 for a brief comment on the construction of *relatively optimal* solutions. We conclude this section by indicating some applications.

(i) *ranking by paired comparisons* [Slater 1961]
A set of n dog foods has to be ordered according to the taste of a particular dog. Let V denote the set of dog foods and let arc (i,j) indicate that the dog prefers food i to food j. A complete set of $\frac{1}{2}n(n-1)$ paired comparisons yields a *tournament* on V, *i.e.* a graph $G = (V,A)$ with $|\{(i,j),(j,i)\} \cap A| = 1$ for each pair $\{i,j\}$ [Moon 1968]. An acyclic subgraph of maximum cardinality corresponds to a total ordering that "resembles the tournament as closely as possible" and minimizes the number of (feed-back) errors of the dog.

(ii) *aggregating individual preferences* [Anthonisse 1972]
A group of persons has to rank n alternatives according to desirability. To this end, each of them determines an individual preference scheme, which need not even be a consistent partial ordering. Choosing c_{ij} to be the number of persons preferring alternative i to alternative j and solving the ASP, we obtain an aggregate total preference ordering that minimizes the number of neglected preferences.

(iii) *determining ancestry relationships* [Glover *et al.* 1974]
At a number of individual gravesites, n pottery types have been found at

various ground depths. Let c_{ij} be some weighted sum over all graves at which type i was found below type j. By solving the ASP we can determine the "most probable" chronological ordering of the pottery types.

(iv) *triangulating input-output matrices* [Korte & Oberhofer 1968]
Let (c_{ij}) be an input-output matrix between n sectors of industries. An optimal ASP solution corresponds to a triangulation of this matrix, *i.e.* a "ranking from raw material to consumer" that maximizes the total supply from higher to lower placed sectors.

3.3. The travelling salesman problem

The travelling salesman problem (TSP) can be stated as follows.

Given a directed graph $G = (V,A)$ with a weight c_{ij} for each arc $(i,j) \in A$, find a hamiltonian circuit on G of minimum total weight.

If $c_{ij} = c_{ji}$ for all $(i,j) \in A$, then we have a *symmetric* TSP (STSP) which corresponds to finding a minimum-weight hamiltonian circuit on an undirected graph $G = (V,E)$. A problem for which the latter equalities need not hold is called an *asymmetric* TSP (ATSP).

The TSP is the problem of a salesman who has to travel through a number of cities with intercity distances c_{ij}, visiting each of them *exactly* once before returning home. If the salesman is allowed to visit each city *at least* once, his problem is equivalent to a TSP with c_{ij} equal to the length of a shortest path from city i to city j; in that case, $c_{ik} \le c_{ij}+c_{jk}$ for all $i,j,k \in V$ and the problem is called *euclidean*.

Let $V = \{1,\dots,n\}$. Defining $c_{ij} = \infty$ for $(i,j) \notin A$ and taking $A = V\times V$ obviously does not change the problem. There is now a one-to-one correspondence between the $(n-1)!$ hamiltonian circuits on G and the $(n-1)!$ *cyclic permutations* of V, *i.e.* permutations μ of V such that for any $i \in V$ we have

$$\mu^k(i) \neq i \ (k = 1,\dots,n-1),\ \mu^n(i) = i;$$

$\mu^k(i)$ is the k-th city reached by the salesman from city i. The TSP may now be restated as follows. Given an $n\times n$-matrix (c_{ij}), find a cyclic permutation μ of V minimizing

$$\textstyle\sum_{i\in V} c_{i\mu(i)}.$$

For each cyclic permutation μ of V we can find n permutations π of V by choosing $i \in V$ and defining $\pi(k) = \mu^k(i)$ for $k = 1,\ldots,n$; $\pi^k(i)$ is the k-th city in a salesman tour. We seek to find a permutation π of V minimizing

$$f_{TSP}(\pi) = \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}.$$

From the latter formulation it is clear that the TSP is a special case of the QAP; we obtain a QAP with $f_{QAP}(\pi) = f_{TSP}(\pi)$ by defining

$$d_{12} = d_{23} = \ldots = d_{n-1,n} = d_{n1} = 1,$$
$$d_{ij} = 0 \quad \text{(otherwise)}.$$

Some special cases of the TSP can be solved by polynomial-bounded algorithms. For instance, if there are real functions f and g with $f(x)+g(x) \geq 0$ and real numbers $a_1,\ldots,a_n,b_1,\ldots,b_n$ such that

$$c_{ij} = \begin{cases} \int_{b_i}^{a_j} f(x)dx & (b_i \leq a_j), \\ \int_{a_j}^{b_i} g(x)dx & (b_i > a_j), \end{cases}$$

then an optimal solution can be found in $O(n^2)$ steps [Gilmore & Gomory 1964]. If, for given $a_2,\ldots,a_n,b_1,\ldots,b_{n-1}$ we have

$$c_{ij} = a_i+b_j \quad (i > j),$$

then the TSP is equivalent to a linear assignment problem [Lawler 1971].

The general ATSP and STSP are easily shown to be NP-complete by Theorem 2.1(*e*,*g*) and Theorem 3.3.

THEOREM 3.3.

(*a*) DIRECTED HAMILTONIAN CIRCUIT $\propto$ ATSP;

(*b*) UNDIRECTED HAMILTONIAN CIRCUIT $\propto$ STSP.

Proof. Immediate. □

Solution methods for the TSP have been surveyed in [Bellmore & Nemhauser 1968; Isaac & Turban 1969; Eilon *et al.* 1971; Christofides 1975]. Branch-and-bound approaches for ATSPs and STSPs will be described in Chapter 9. For suboptimal algorithms we refer to [Lin 1965; Christofides & Eilon 1972; Lin & Kernighan 1973]; see also Chapter 7. Some applications are discussed in Chapter 14.

4. MACHINE SCHEDULING PROBLEMS

4.1. Classification

Machine scheduling problems can be verbally formulated as follows.

A *job* J_i $(i = 1,\ldots,n)$ consists of a sequence of *operations*, each of which corresponds to the uninterrupted processing of J_i on some *machine* M_k $(k = 1,\ldots,m)$ during a given period of time. Each machine can handle at most one job at a time. What is according to some overall *criterion* the optimal *processing order* on each machine?

The following data can be specified for each J_i:

- a *number of operations* n_i;
- a *machine order* ν_i, *i.e.* an ordered n_i-tuple of machines;
- a *processing time* p_{ik} of its k-th operation, $k = 1,\ldots,n_i$ (if $n_i = 1$ for all J_i, we shall usually write p_i instead of p_{i1});
- a *weight* w_i;
- a *release date* or *ready time* r_i, *i.e.* its earliest possible starting time (unless stated otherwise, we assume that $r_i = 0$ for all J_i);
- a *due date* or *deadline* d_i;
- a *cost function* $f_i : \mathbb{N} \to \mathbb{R}$, indicating the costs incurred as a nondecreasing function of the completion time of J_i.

We assume that all data (except ν_i and f_i) are nonnegative integers. Given a processing order on each M_k, we can compute for each J_i:

- the *starting time* S_i;
- the *completion time* C_i;
- the *lateness* $L_i = C_i - d_i$;
- the *tardiness* $T_i = \max\{0, C_i - d_i\}$;
- $U_i = \underline{\text{if}}\ C_i \le d_i\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ 1$.

Machine scheduling problems are traditionally classified by means of four parameters n, m, ℓ, κ. The first two parameters are integer variables, denoting the numbers of jobs and machines respectively; the cases in which m is constant and equal to 1, 2, or 3 will be studied separately. If $m > 1$, the third parameter takes on one of the following values:

- $\ell = F$ in a *flow-shop* where $n_i = m$ and $\nu_i = (M_1,\ldots,M_m)$ for each J_i;
- $\ell = P$ in a *permutation flow-shop*, *i.e.* a flow-shop where passing is not permitted so that each machine has to process the jobs in the same order;
- $\ell = G$ in a *(general) job-shop* where n_i and ν_i may vary per job;
- $\ell = I$ in a *parallel-shop* where each job has to be processed on just

one of m *identical* machines, *i.e.* $n_i = 1$ for all J_i and the ν_i are not defined.

Extensions to the more general situation where *several groups of parallel (possibly non-identical) machines* are available will not be considered.

The fourth parameter indicates the optimality criterion. We will mainly deal with *regular* criteria, *i.e.*, monotone functions κ of the completion times $C_1,\ldots,C_n$ such that

$$C_i \le C_i' \text{ for all } i \Rightarrow \kappa(C_1,\ldots,C_n) \le \kappa(C_1',\ldots,C_n').$$

These functions are usually of one of the following types:

- $\kappa = f_{max} = \max_i\{f_i(C_i)\}$;
- $\kappa = \sum f_i = \sum_{i=1}^n f_i(C_i)$.

The following specific criteria have frequently been chosen to be minimized:

- $\kappa = C_{max} = \max_i\{C_i\}$;
- $\kappa = \sum w_iC_i = \sum_{i=1}^n w_iC_i$;
- $\kappa = L_{max} = \max_i\{L_i\}$;
- $\kappa = \sum w_iT_i = \sum_{i=1}^n w_iT_i$;
- $\kappa = \sum w_iU_i = \sum_{i=1}^n w_iU_i$.

We refer to [Rinnooy Kan 1976] for equivalence relations between these and other objective functions.

Some relevant problem variations are characterized by the presence of one or more elements from a parameter set λ, such as

- *prec* (precedence constraints between the jobs, where "J_i precedes J_j" (notation: $J_i < J_j$) implies $C_i \le S_j$);
- *tree* (precedence constraints between the jobs such that the associated precedence graph can be given as a *branching*, *i.e.* a set of directed trees with either indegree or outdegree at most one for all vertices);
- $r_i \ge 0$ (possibly non-equal release dates for the jobs);
- $r_n \ge 0$ (a possibly non-zero release date for one job, say J_n);
- $C_i \le d_i$ (all jobs have to meet their deadlines);
- $C_n \le d_n$ (one job, say J_n, has to meet its deadline);
- *no wait* (no waiting time for the jobs between their starting and completion times; hence, $C_i = S_i + \sum_k p_{ik}$ for each J_i);
- $n_i \le n_*$ (a constant upper bound on the number of operations per job);
- $p_{ik} \le p_*$ (a constant upper bound on the processing times);
- $p_{ik} = 1$ (unit processing times);
- $w_i = 1$ (equality of the weights; we indicate this case also by writing $\sum C_i$, $\sum T_i$, $\sum U_i$).

In view of the above discussion, we can use the notation $n|m|\ell,\lambda|\kappa$ to indicate specific machine scheduling problems.

The theory of scheduling is surveyed extensively in [Conway *et al.* 1967; Coffman 1976; Rinnooy Kan 1976]. Here, we will deal with the following aspects. In the remaining sections of this chapter we investigate the *complexity* of machine scheduling problems. In Chapters 10 to 13 we present *branch-and-bound algorithms* for various specific types of problems:

- the $n|1|prec,r_i\geq 0|L_{max}$ problem in Chapter 10;
- the $n|1|prec|\sum f_i$ problem, and more especially the $n|1||\sum w_iT_i$ problem, in Chapter 11;
- the $n|m|P|C_{max}$ problem in Chapter 12;
- the $n|m|G|C_{max}$ problem in Chapter 13.

In Chapter 15 we discuss an *application*, involving the $n|1||\sum w_iC_i$ and the $n|1|r_i\geq 0|L_{max}$ problems.

4.2. Complexity

All machine scheduling problems of the type defined in Section 4.1 can be solved by polynomial-depth backtrack search and thus are members of $\mathcal{NP}$. The results on their complexity are summarized in Table 4.1.

The problems which are marked by an asterisk (*) are solvable in polynomial-bounded time. In Table 4.2 we provide for most of these problems references where the algorithm in question can be found; we give also the order of the number of steps in the currently best implementations.

The problems marked by a note of exclamation (!) are NP-complete. The reductions to these problems are listed in Table 4.3. A dagger (†) indicates that NP-completeness is proved only with respect to a special type of encoding of the problem data; we will discuss this question after the proof of Theorem 4.4.

Question-marks (?) indicate open problems. We will return to them in Section 4.3 to motivate our typographical suggestion that these problems are likely to be NP-complete.

Table 4.1 contains the "hardest" problems that are known to be in $\mathcal{P}$ and the "easiest" ones that have been proved to be NP-complete. In this respect, Table 4.1 indicates to the best of our knowledge the location of the borderline between easy and hard machine scheduling problems.

The remaining part of this section will be devoted to the proofs of the theorems, mentioned in Table 4.3. First, we will give a simple example of the interaction between tables and theorems by examining the status of the general job-shop problem, indicated by $n|m|G|C_{max}$.

TABLE 4.1. COMPLEXITY OF MACHINE SCHEDULING PROBLEMS

n jobs	1 machine	2 machines	m machines
C_{max}	* $prec, r_i \geq 0$	* F * F, $no\ wait$! F, $tree$! F, $r_n \geq 0$	! $\underline{m=3}$: F ? $\underline{m=3}$: F, $no\ wait$! F, $no\ wait$
		* G, $n_i \leq 2$! G, $n_i \leq 3$	* $\underline{n=2}$: G ! $\underline{m=3}$: G, $n_i \leq 2$
		! I * I, $prec, r_i \geq 0, C_i \leq d_i, p_i = 1$! I, $prec, p_i \leq 2$	* I, $tree, p_i = 1$? $\underline{m=3}$: I, $prec, p_i = 1$! I, $prec, p_i = 1$
$\sum w_i C_i$	* $tree$! $prec, p_i = 1$! $prec, w_i = 1$! $r_n \geq 0, w_i = 1$ † ! $r_n \geq 0$ * $C_i \leq d_i, w_i = 1$! $C_n \leq d_n$	! F, $w_i = 1$? F, $no\ wait, w_i = 1$	! F, $no\ wait, w_i = 1$
		! I * I, $prec, p_i = 1, w_i = 1$! I, $prec, p_i \leq 2, w_i = 1$	* I, $r_i \geq 0, p_i = 1$ * I, $w_i = 1$! I, $prec, p_i = 1, w_i = 1$
L_{max}	* $prec$ * $prec, r_i \geq 0, p_i = 1$! $r_n \geq 0$	! F	
		! I	
$\sum w_i T_i$	* $r_i \geq 0, p_i = 1$ * $p_i \leq p_*, w_i = 1$? $w_i = 1$! ? $tree, p_i = 1$! $prec, p_i = 1, w_i = 1$! $r_n \geq 0, w_i = 1$	! F, $w_i = 1$	
		! I, $w_i = 1$	
$\sum w_i U_i$	* $r_i \geq 0, p_i = 1$ * $p_i \leq p_*$ * $w_i = 1$! ? $tree, p_i = 1$! $prec, p_i = 1, w_i = 1$! $r_n \geq 0, w_i = 1$	! F, $w_i = 1$	
		! I, $w_i = 1$	

* : problem in $\mathcal{P}$; see Table 4.2.

? : open problem; see Section 4.3.

! : NP-complete problem; see Table 4.3.

† : NP-completeness proof is given with respect to a special type of encoding; see the remark following the proof of Theorem 4.4.

TABLE 4.2. REFERENCES TO POLYNOMIAL-BOUNDED ALGORITHMS

Problem	References	Order
$n\|1\|\mathit{prec},r_i\geq 0\|C_{max}$	–	$O(n^2)$
$n\|1\|\mathit{tree}\|\sum w_iC_i$	[Horn 1972; Sidney 1975] (1)	$O(n \log n)$
$n\|1\|C_i\leq d_i\|\sum C_i$	[Smith 1956]	$O(n \log n)$
$n\|1\|\mathit{prec}\|L_{max}$	[Lawler 1973]	$O(n^2)$
$n\|1\|\mathit{prec},r_i\geq 0,p_i=1\|L_{max}$	[Lageweg *et al.* 1975]; *h.l.*, Section 10.2.1	$O(n^2)$
$n\|1\|r_i\geq 0,p_i=1\|\sum w_iT_i$	[Lawler 1964]	$O(n^3)$
$n\|1\|p_i\leq p_*\|\sum T_i$	[Lawler 1975C] (2)	$O(n^5p_*)$
$n\|1\|r_i\geq 0,p_i=1\|\sum w_iU_i$	[Lageweg & Lawler 1975]	$O(n^2)$
$n\|1\|p_i\leq p_*\|\sum w_iU_i$	[Lawler & Moore 1969] (2)	$O(n^2p_*)$
$n\|1\|\|\sum U_i$	[Moore 1968] (3)	$O(n \log n)$
$n\|2\|F\|C_{max}$	[Johnson 1954]	$O(n \log n)$
$n\|2\|F,\mathit{no\ wait}\|C_{max}$	[Gilmore & Gomory 1964]	$O(n^2)$
$n\|2\|G,n_i\leq 2\|C_{max}$	[Jackson 1956]	$O(n \log n)$
$n\|2\|I,\mathit{prec},r_i\geq 0,C_i\leq d_i,p_i=1\|C_{max}$	[Garey 1975] (4)	
$n\|2\|I,\mathit{prec},p_i=1\|\sum C_i$	[Coffman & Graham 1972; Garey 1975]	$O(n^2)$
$2\|m\|G\|C_{max}$	[Szwarc 1960; Hardgrave & Nemhauser 1963]	$O(m^2)$
$n\|m\|I,\mathit{tree},p_i=1\|C_{max}$	[Hu 1961]	$O(n)$
$n\|m\|I,r_i\geq 0,p_i=1\|\sum w_iC_i$	[Lawler 1964]	$O(n^3)$
$n\|m\|I\|\sum C_i$	[Conway *et al.* 1967] (5)	$O(n \log n)$

(1) An $O(n \log n)$ algorithm for the more general case of *series parallel* precedence constraints is given in [Lawler 1975D].

(2) *I.e.*, there exist *unary polynomial-bounded* algorithms for the $n|1||\sum T_i$ and $n|1||\sum w_iU_i$ problems.

(3) An $O(n \log n)$ algorithm for the more general case of *agreeable weights* (*i.e.* $p_i < p_j \Rightarrow w_i \geq w_j$) is given in [Lawler 1975B].

(4) An $O(n^2)$ algorithm for the $n|2|I,\mathit{prec},p_i=1|C_{max}$ problem is given in [Coffman & Graham 1972]; see also [Garey & Johnson 1975].

(5) Polynomial-bounded algorithms for the more general case of *parallel non-identical* machines are given in [Horn 1973; Bruno *et al.* 1974A; Bruno *et al.* 1974B].

TABLE 4.3. REDUCTIONS TO NP-COMPLETE MACHINE SCHEDULING PROBLEMS

Reduction	Reference
LINEAR ARRANGEMENT $\propto n\mid 1\mid \mathit{prec}, p_i=1\mid \sum w_iC_i$	*h.l.*, Theorem 4.6(*b*)
LINEAR ARRANGEMENT $\propto n\mid 1\mid \mathit{prec}\mid \sum C_i$	*h.l.*, Theorem 4.6(*a*)
KNAPSACK $\propto n\mid 1\mid r_n\geq 0\mid \sum C_i\dagger$	*h.l.*, Theorem 4.4(*l*)
KNAPSACK $\propto n\mid 1\mid r_n\geq 0\mid \sum w_iC_i$	*h.l.*, Theorem 4.4(*k*)
KNAPSACK $\propto n\mid 1\mid C_n\leq d_n\mid \sum w_iC_i$	*h.l.*, Theorem 4.4(*j*)
KNAPSACK $\propto n\mid 1\mid r_n\geq 0\mid L_{max}$	*h.l.*, Theorem 4.4(*c*)
KNAPSACK $\propto n\mid 1\mid \mid \sum w_iT_i$	*h.l.*, Theorem 4.4(*i*)
CLIQUE $\propto n\mid 1\mid \mathit{prec}, p_i=1\mid \sum T_i$	*h.l.*, Theorem 4.5(*b*)
KNAPSACK $\propto n\mid 1\mid r_n\geq 0\mid \sum T_i$	*h.l.*, Theorem 4.4(*d*)
KNAPSACK $\propto n\mid 1\mid \mid \sum w_iU_i$	[Karp 1972B]; *h.l.*, Theorem 4.4(*h*)
CLIQUE $\propto n\mid 1\mid \mathit{prec}, p_i=1\mid \sum U_i$	[Garey & Johnson 1975]; *h.l.*, Theorem 4.5(*a*)
KNAPSACK $\propto n\mid 1\mid r_n\geq 0\mid \sum U_i$	*h.l.*, Theorem 4.4(*e*)
KNAPSACK $\propto n\mid 2\mid F, \mathit{tree}\mid C_{max}$	*h.l.*, Theorem 4.4(*g*)
KNAPSACK $\propto n\mid 2\mid F, r_n\geq 0\mid C_{max}$	*h.l.*, Theorem 4.4(*f*)
KNAPSACK $\propto n\mid 2\mid G, n_i\leq 3\mid C_{max}$	*h.l.*, Theorem 4.4(*a*)
PARTITION $\propto n\mid 2\mid I\mid C_{max}$	[Bruno *et al.* 1974B]; *h.l.*, Theorem 4.2(*a*)
3-SATISFIABILITY $\propto n\mid 2\mid I, \mathit{prec}, p_i\leq 2\mid C_{max}$	[Ullman 1975]
$n\mid 1\mid r_n\geq 0\mid \sum C_i\dagger \propto n\mid 2\mid F\mid \sum C_i\dagger$	*h.l.*, Theorem 4.1(*k*)
3-PARTITION $\propto n\mid 2\mid F\mid \sum C_i$	[Garey *et al.* 1975]
PARTITION $\propto n\mid 2\mid I\mid \sum w_iC_i$	[Bruno *et al.* 1974B]; *h.l.*, Theorem 4.2(*b*)
$n'\mid 2\mid I, \mathit{prec}, p_i\leq 2\mid C_{max} \propto n\mid 2\mid I, \mathit{prec}, p_i\leq 2\mid \sum C_i$	*h.l.*, Theorem 4.1(*l*)
$n\mid 1\mid r_n\geq 0\mid L_{max} \propto n\mid 2\mid F\mid L_{max}$	*h.l.*, Theorem 4.1(*k*)
$n\mid 2\mid I\mid C_{max} \propto n\mid 2\mid I\mid L_{max}$	*h.l.*, Theorem 4.1(*i*)
$n\mid 1\mid r_n\geq 0\mid \sum T_i \propto n\mid 2\mid F\mid \sum T_i$	*h.l.*, Theorem 4.1(*k*)
$n\mid 2\mid I\mid C_{max} \propto n\mid 2\mid I\mid \sum T_i$	*h.l.*, Theorem 4.1(*i*)
$n\mid 1\mid r_n\geq 0\mid \sum U_i \propto n\mid 2\mid F\mid \sum U_i$	*h.l.*, Theorem 4.1(*k*)
$n\mid 2\mid I\mid C_{max} \propto n\mid 2\mid I\mid \sum U_i$	*h.l.*, Theorem 4.1(*i*)
$n\mid 2\mid F, r_n\geq 0\mid C_{max} \propto n\mid 3\mid F\mid C_{max}$	*h.l.*, Theorem 4.1(*k*)
3-PARTITION $\propto n\mid 3\mid F\mid C_{max}$	[Garey *et al.* 1975]
DIRECTED HAMILTONIAN PATH $\propto n\mid m\mid F, \mathit{no\ wait}\mid C_{max}$	*h.l.*, Theorem 4.7(*a*)
KNAPSACK $\propto n\mid 3\mid G, n_i\leq 2\mid C_{max}$	*h.l.*, Theorem 4.4(*b*)
3-SATISFIABILITY $\propto n\mid m\mid I, \mathit{prec}, p_i=1\mid C_{max}$	[Ullman 1975]
DIRECTED HAMILTONIAN PATH $\propto n\mid m\mid F, \mathit{no\ wait}\mid \sum C_i$	*h.l.*, Theorem 4.7(*b*)
$n'\mid m\mid I, \mathit{prec}, p_i=1\mid C_{max} \propto n\mid m\mid I, \mathit{prec}, p_i=1\mid \sum C_i$	*h.l.*, Theorem 4.1(*l*)

In Table 4.1, we see that the $n|2|G,n_i\leq 2|C_{max}$ problem is a member of $\mathcal{P}$ and that two minor extensions, $n|2|G,n_i\leq 3|C_{max}$ and $n|3|G,n_i\leq 2|C_{max}$, are NP-complete. By Theorem 4.1(*c,h*), these problems are special cases of the general job-shop problem, which is thus shown to be NP-complete by Theorem 4.1(*b*). Table 4.2 refers to an $O(n \log n)$ algorithm [Jackson 1956] for the $n|2|G,n_i\leq 2|C_{max}$ problem. Table 4.3 tells us that reductions from KNAPSACK to both NP-complete problems are presented in Theorem 4.4(*a,b*); the NP-completeness of KNAPSACK has been mentioned in Theorem 2.1(*h*).

Theorem 4.1 gives some elementary results on reducibility among machine scheduling problems. It can be used to establish either membership of $\mathcal{P}$ or NP-completeness for problems that are, roughly speaking, either not harder than the polynomially solvable ones or not easier than the NP-complete ones in Table 4.1.

THEOREM 4.1.

(*a*) *If* $n'|m'|\ell',\lambda'|\kappa' \propto n|m|\ell,\lambda|\kappa$ *and* $n|m|\ell,\lambda|\kappa \in \mathcal{P}$, *then* $n'|m'|\ell',\lambda'|\kappa' \in \mathcal{P}$.

(*b*) *If* $n'|m'|\ell',\lambda'|\kappa' \propto n|m|\ell,\lambda|\kappa$ *and* $n'|m'|\ell',\lambda'|\kappa'$ *is NP-complete, then* $n|m|\ell,\lambda|\kappa$ *is NP-complete.*

(*c*) $n|m'|\ell,\lambda|\kappa \propto n|m|\ell,\lambda|\kappa$ *if* $m' \leq m$ *or if* m' *is constant and* m *is variable.*

(*d*) $n|2|F|\kappa$ *and* $n|2|P|\kappa$ *are equivalent.*

(*e*) $n|3|F|C_{max}$ *and* $n|3|P|C_{max}$ *are equivalent.*

(*f*) $n|m|F,\lambda|\kappa \propto n|m|G,\lambda|\kappa$.

(*g*) $n|m|\ell,\lambda|\kappa \propto n|m|\ell,\lambda\cup\lambda'|\kappa$ *if* $\lambda' \subset \{\mathit{prec},\mathit{tree},r_i\geq 0,r_n\geq 0,C_i\leq d_i,C_n\leq d_n\}$.

(*h*) $n|m|\ell,\lambda\cup\lambda'|\kappa \propto n|m|\ell,\lambda|\kappa$ *if* $\lambda' \subset \{n_i\leq n_*,p_{ik}\leq p_*,p_{ik}=1,w_i=1\}$.

(*i*) $n|m|\ell,\lambda|C_{max} \propto n|m|\ell,\lambda|\kappa$ *if* $\kappa \in \{L_{max},\sum T_i,\sum U_i\}$.

(*j*) $n|m|\ell,\lambda|\sum w_iC_i \propto n|m|\ell,\lambda|\sum w_iT_i$.

(*k*) $n|m-1|F,r_n\geq 0,\lambda|\kappa \propto n|m|F,\lambda|\kappa$ *if* $\lambda \subset \{w_i=1\}$.

(*l*) $n'|m|I,\mathit{prec},p_i\leq p_*|C_{max} \propto n|m|I,\mathit{prec},p_i\leq p_*|\sum C_i$.

Proof. Let P' and P denote the problems on the left-hand side and right-hand side respectively.

(*a,b*) Clear from the definition of reducibility.

(*c*) Trivial.

(*d,e*) P' has an optimal solution with the same processing order on each machine (*cf.* Section 12.1).

(*f,g,h*) In each case P' obviously is a special case of P.

(*i*) Given any instance of P' and a threshold value y', we construct a corresponding instance of P by defining $d_i = y'$ $(i = 1,\ldots,n)$, $y = 0$. P' has a solution with value $\le y'$ if and only if P has a solution with value $\le y$.

(*j*) Take $d_i = 0$ $(i = 1,\ldots,n)$ in P.

(*k*) Suppose that in P' the machines and the operations of each job are indexed from 2 to m. We specify P by adding a machine M_1 and defining $p_{i1} = r_i$ $(i = 1,\ldots,n)$ (*i.e.*, $p_{11} = \ldots = p_{n-1,1} = 0$, $p_{n1} \ge 0$). Any solution to P' corresponds to a solution to P with the same value for κ, and *vice versa*.

(*l*) Given any instance of P' and an integer y', $0 \le y' \le n'p_*$, we construct a corresponding instance of P by defining

$$n'' = (n'-1)y',$$
$$n = n'+n'',$$
$$y = ny'+\tfrac{1}{2}n''(n''+1),$$

and adding n'' jobs $J_{n'+j}$ $(j = 1,\ldots,n'')$ to P' with

$$p_{n'+j,1} = 1,$$
$$J_i < J_{n'+j} \quad (i = 1,\ldots,n'+j-1).$$

Now P' has a solution with value $\le y'$ if and only if P has a solution with value $\le y$:

$$C_{\max} \le y' \Rightarrow \sum C_i \le n'y' + \sum_{j=1}^{n''}(y'+j) = y;$$
$$C_{\max} > y' \Rightarrow \sum C_i > y' + \sum_{j=1}^{n''}(y'+1+j) = y. \qquad \square$$

Remark. The proofs of Theorem 4.1(*c*,*k*) involve processing times equal to 0, implying that the operations in question require an infinitesimally small amount of time. Whenever these reductions are applied, the processing times can be transformed into strictly positive integers by sufficiently (but polynomially) inflating the problem data. Examples of such constructions can be found in the proofs of Theorem 4.4(*f*,*g*).

In Theorems 4.2 to 4.7 we present a large number of reductions of the form $P \propto n|m|\ell,\lambda|\kappa$ by specifying $n|m|\ell,\lambda|\kappa$ and some y such that P has a solution if and only if $n|m|\ell,\lambda|\kappa$ has a solution with value $\kappa \le y$. This equivalence is proved for some principal reductions; in other cases, it is trivial or clear from the analogy to a reduction given previously. The NP-completeness of $n|m|\ell,\lambda|\kappa$ then follows from the NP-completeness of P as established in Theorem 2.1.

First, we deal with the problems on *identical machines*. Theorem 4.2 presents two reductions which are simplified versions of the reductions given in [Bruno *et al.* 1974B].

THEOREM 4.2. PARTITION *is reducible to the following problems:*

(*a*) $n|2|I|C_{max}$;

(*b*) $n|2|I|\sum w_iC_i$.

Proof. Define $A = \sum_{i\in T} a_i$.

(*a*) PARTITION $\propto$ $n|2|I|C_{max}$:

$n = t$;

$p_i = a_i \quad (i \in T)$;

$y = \frac{1}{2}A$.

(*b*) PARTITION $\propto$ $n|2|I|\sum w_iC_i$:

$n = t$;

$p_i = w_i = a_i \quad (i \in T)$;

$y = \sum_{1\le i\le j\le t} a_ia_j - \frac{1}{4}A^2$.

Suppose that $\{J_i|i \in S\}$ is assigned to M_1 and $\{J_i|i \in T-S\}$ to M_2; let $c = \sum_{i\in S} a_i - \frac{1}{2}A$. Since $p_i = w_i$ for all i, the value of $\sum w_iC_i$ is not influenced by the ordering of the jobs on the machines and only depends on the choice of S [Conway *et al*. 1967]:

$\sum w_iC_i = \kappa(S)$.

It is easily seen (*cf*. Figure 4.1) that

$$\kappa(S) = \kappa(T) - \left(\sum_{i\in S} a_i\right)\left(\sum_{i\in T-S} a_i\right)$$
$$= \sum_{1\le i\le j\le t} a_ia_j - (\tfrac{1}{2}A+c)(\tfrac{1}{2}A-c) = y+c^2,$$

and it follows that PARTITION has a solution if and only if this $n|2|I|\sum w_iC_i$ problem has a solution with value $\le y$. □

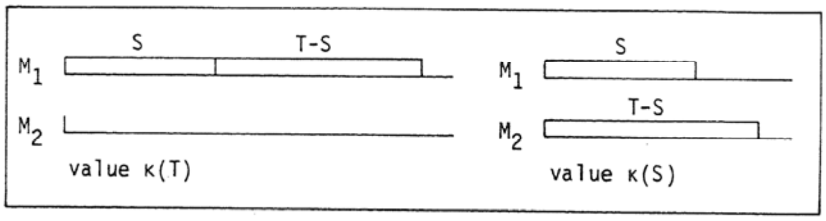


Figure 4.1

Next, we investigate the complexity of the $n|2|I|\kappa$ problem for some irregular choices of κ. The criteria in question have not been mentioned in Section 4.1 and, accordingly, the results presented in Theorem 4.3 have not been included in Tables 4.1 and 4.3.

We will only consider *active schedules*, *i.e.* schedules where we cannot decrease the starting time of any operation without increasing the starting time of at least one other one.

THEOREM 4.3. PARTITION *is reducible to the following problems:*

(*a*) $n|2|I|1/C_{max}$;

(*b*) $n|2|I|\sum C_i \cdot C_{max}$;

(*c*) $n|2|I|\sum C_i/C_{max}$.

Proof. Define $A = \sum_{i \in T} a_i$, $\tau = t(t+1)A+A$, $\sigma = \sum_{i \in T}(t+1-i)(2iA+a_i)$.

(*a*) PARTITION $\propto$ $n|2|I|1/C_{max}$:

$n = t+1$;

$p_i = a_i \quad (i \in T)$;

$p_n = A$;

$y = 2/3A$.

In any active schedule, J_n is the last job on some machine, and $C_{max} = C_n \leq 3A/2 = 1/y$. PARTITION has a solution if and only if this bound can be attained.

(*b*) PARTITION $\propto$ $n|2|I|\sum C_i \cdot C_{max}$:

$n = 2t$;

$p_i = iA+a_i$, $p_{t+i} = iA \quad (i \in T)$;

$y = \frac{1}{2}\sigma\tau$.

A schedule which minimizes $\sum C_i$ is obtained by sequencing J_i and J_{t+i} in the i-th position on both machines [Conway *et al.* 1967] and has a value $\sum C_i = \sigma$. If PARTITION has a solution, then there exists such a schedule with $C_{max} = \frac{1}{2}\sum_{i=1}^{n} p_i = \frac{1}{2}\tau$. If PARTITION has no solution, then we have for any schedule that $C_{max} > \frac{1}{2}\tau$.

(*c*) PARTITION $\propto$ $n|2|I|\sum C_i/C_{max}$:

$n = 2t+1$;

$p_i = iA+a_i$, $p_{t+i} = iA \quad (i \in T)$;

$p_n = \tau$;

$y = 1 + 2\sigma/3\tau$.

Cf. reductions 4.3(*a*,*b*). □

Remarks.

ad (*a*). It follows that the problem of finding the worst active $n|2|I|C_{max}$ schedule is NP-complete.

ad (*b*). Taking $y = \frac{1}{2}\tau$, we can use the same construction to show that the $n|2|I,\min\{\sum C_i\}|C_{max}$ problem (*i.e.*, minimizing C_{max} over all schedules minimizing $\sum C_i$ on two identical machines) is NP-complete (*cf.* [Bruno *et al.* 1974A]).

ad (*c*). The criterion $\sum C_i/C_{max}$ corresponds to the average number of jobs in the shop.

Most of our results on *different machines* involve the KNAPSACK problem, as demonstrated by Theorem 4.4.

THEOREM 4.4. KNAPSACK *is reducible to the following problems:*

(*a*) $n|2|G,n_i \le 3|C_{max}$;
(*b*) $n|3|G,n_i \le 2|C_{max}$;
(*c*) $n|1|r_n \ge 0|L_{max}$;
(*d*) $n|1|r_n \ge 0|\sum T_i$;
(*e*) $n|1|r_n \ge 0|\sum U_i$;
(*f*) $n|2|F,r_n \ge 0|C_{max}$;
(*g*) $n|2|F,\mathit{tree}|C_{max}$;
(*h*) $n|1||\sum w_i U_i$;
(*i*) $n|1||\sum w_i T_i$;
(*j*) $n|1|C_n \le d_n|\sum w_i C_i$;
(*k*) $n|1|r_n \ge 0|\sum w_i C_i$;
(*l*) $n|1|r_n \ge 0|\sum C_i$†.

Proof. Define $A = \sum_{i \in T} a_i$; $a_* = \max_{i \in T}\{a_i\}$. We may assume that $0 < b < A$.

(*a*) KNAPSACK $\propto$ $n|2|G,n_i \le 3|C_{max}$:

$n = t+1$;
$\nu_i = (M_1)$, $p_{i1} = a_i$ $(i \in T)$;
$\nu_n = (M_2,M_1,M_2)$, $p_{n1} = b$, $p_{n2} = 1$, $p_{n3} = A-b$;
$y = A+1$.

If KNAPSACK has a solution, then there exists a schedule with value $C_{max} = y$, as illustrated in Figure 4.2. If KNAPSACK has no solution, then $\sum_{i \in S} a_i - b = c \ne 0$ for each $S \subset T$, and we have for a processing order $(\{J_i|i \in S\}, J_n, \{J_i|i \in T-S\})$ on M_1 that

$$c > 0 \Rightarrow C_{max} \ge \sum_{i \in S} p_{i1} + p_{n2} + p_{n3} = A+c+1 > y;$$
$$c < 0 \Rightarrow C_{max} \ge p_{n1} + p_{n2} + \sum_{i \in T-S} p_{i1} = A-c+1 > y.$$

It follows that KNAPSACK has a solution if and only if this $n|2|G,n_i \le 3|C_{max}$ problem has a solution with value $\le y$.

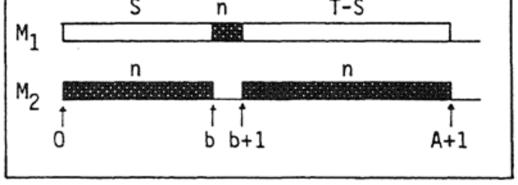


Figure 4.2

(*b*) KNAPSACK $\propto$ $n|3|G,n_i\le 2|C_{max}$:

$n = t+2$;

$\nu_i = (M_1,M_3)$, $p_{i1} = p_{i2} = a_i$ $(i \in T)$;

$\nu_{n-1} = (M_1,M_2)$, $p_{n-1,1} = b$, $p_{n-1,2} = 2(A-b)$;

$\nu_n = (M_2,M_3)$, $p_{n1} = 2b$, $p_{n2} = A-b$;

$y = 2A$.

If KNAPSACK has a solution, then there exists a schedule with value $C_{max} = y$, as illustrated in Figure 4.3. If KNAPSACK has no solution, then $\sum_{i\in S} a_i - b = c \neq 0$ for each $S \subset T$, and we have for a processing order $(\{J_i|i \in S\}, J_{n-1}, \{J_i|i \in T-S\})$ on M_1 that

$$c > 0 \Rightarrow C_{max} \ge \sum_{i\in S} p_{i1} + p_{n-1,1} + p_{n-1,2} = 2A+c > y,$$

$$c < 0 \Rightarrow C_{max} \ge \min\{\sum_{i\in S} p_{i1} + p_{n-1,1} + 1, p_{n1}\} + p_{n2} + \sum_{i\in T-S} p_{i2} = 2A+1 > y,$$

which completes the equivalence proof.

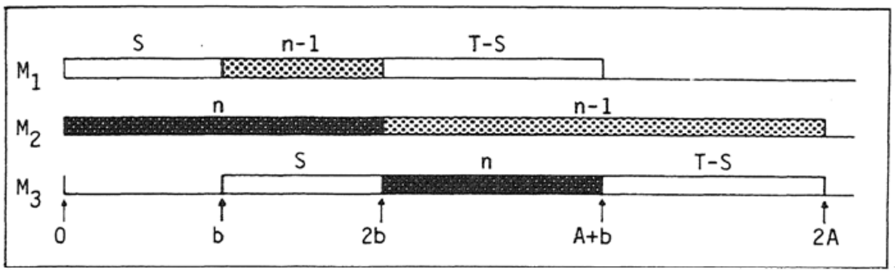


Figure 4.3

(*c*) KNAPSACK $\propto$ $n|1|r_n\ge 0|L_{max}$,

(*d*) KNAPSACK $\propto$ $n|1|r_n\ge 0|\sum T_i$, and

(*e*) KNAPSACK $\propto$ $n|1|r_n\ge 0|\sum U_i$:

$n = t+1$;

$r_i = 0$, $p_i = a_i$, $d_i = A+1$ $(i \in T)$;

$r_n = b$, $p_n = 1$, $d_n = b+1$;

$y = 0$.

Cf. reduction 4.4(*a*) and Figure 4.4.

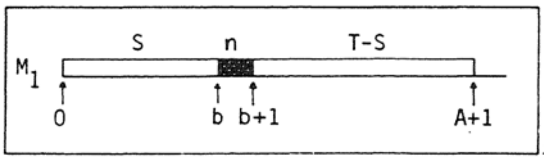


Figure 4.4

(*f*) KNAPSACK $\propto$ $n|2|F,r_n\geq 0|C_{max}$:

$n = t+1$;

$r_i = 0$, $p_{i1} = ta_i$, $p_{i2} = 1$ $(i \in T)$;

$r_n = tb$, $p_{n1} = 1$, $p_{n2} = t(A-b)$;

$y = t(A+1)$.

If KNAPSACK has a solution, then there exists a schedule with value $C_{max} \leq y$, as illustrated in Figure 4.5. If KNAPSACK has no solution, then $\sum_{i\in S} a_i - b = c \neq 0$ for each $S \subset T$, and we have for a processing order $(\{J_i|i \in S\}, J_n, \{J_i|i \in T-S\})$ on M_1 that

$$c > 0 \Rightarrow C_{max} \geq \sum_{i\in S} p_{i1} + p_{n1} + p_{n2} = t(A+c)+1 > y;$$

$$c < 0 \Rightarrow C_{max} \geq r_n + p_{n1} + \sum_{i\in T-S} p_{i1} = t(A-c)+1 > y.$$

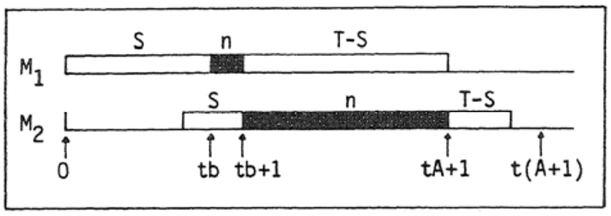


Figure 4.5

(*g*) KNAPSACK $\propto$ $n|2|F,tree|C_{max}$:

$n = t+2$;

$p_{i1} = ta_i$, $p_{i2} = 1$ $(i \in T)$;

$p_{n-1,1} = 1$, $p_{n-1,2} = tb$;

$p_{n1} = 1$, $p_{n2} = t(A-b)$;

$J_{n-1} < J_n$;

$y = t(A+1)+1$.

Cf. Figure 4.6. We have for a processing order $(\{J_i|i \in R\}, J_{n-1}, \{J_i|i \in S\}, J_n, \{J_i|i \in T-S-R\})$ on M_1 that

$$R \neq \emptyset \Rightarrow C_{max} \geq t + p_{n-1,1} + p_{n-1,2} + p_{n1} + p_{n2} = t(A+1)+2 > y.$$

The remainder of the equivalence proof is analogous to that of reduction 4.4(*f*).

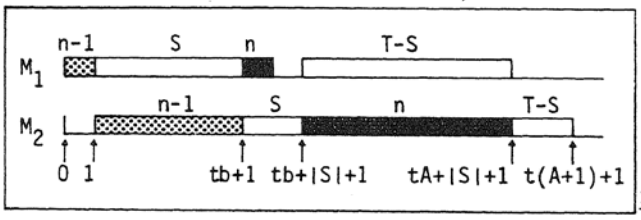


Figure 4.6

(*h*) KNAPSACK $\propto n|1||\sum w_iU_i$:

$n = t$;

$p_i = w_i = a_i$, $d_i = b$ $(i \in T)$;

$y = A-b$.

Cf. [Karp 1972B] and Figure 4.7.

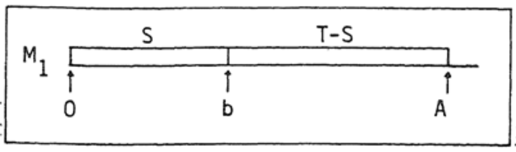


Figure 4.7

(*i*) KNAPSACK $\propto n|1||\sum w_iT_i$:

$n = t+1$;

$p_i = w_i = a_i$, $d_i = 0$ $(i \in T)$;

$p_n = 1$, $w_n = 2$, $d_n = b+1$;

$y = \sum_{1\le i\le j\le t} a_ia_j + A - b$.

Cf. Figure 4.4. We have for a processing order $(\{J_i|i \in S\}, J_n, \{J_i|i \in T-S\})$ that $\sum_{i\in S} a_i - b = L_n$. Since $p_i = w_i$ and $d_i = 0$ for all $i \in T$, the value of $\sum_{i\in T} w_iT_i$ is not influenced by the ordering of S and T-S (*cf.* the proof of Theorem 4.2(*b*)), and we have

$$\begin{aligned}\sum w_iT_i &= \sum_{i\in T} a_iC_i + 2T_n\\ &= \sum_{1\le i\le j\le t} a_ia_j + \sum_{i\in T-S} a_i + 2\max\{0,L_n\}\\ &= y+|L_n| \ge y.\end{aligned}$$

The equivalence follows immediately.

(*j*) KNAPSACK $\propto n|1|C_n\le d_n|\sum w_iC_i$:

$n = t+1$;

$p_i = w_i = a_i$, $d_i = 0$ $(i \in T)$;

$p_n = 1$, $w_n = 0$, $d_n = b+1$;

$y = \sum_{1\le i\le j\le t} a_ia_j + A - b$.

Cf. reduction 4.4(*i*) and Figure 4.4.

(*k*) KNAPSACK $\propto n|1|r_n\ge 0|\sum w_iC_i$:

$n = t+1$;

$r_i = 0$, $p_i = w_i = a_i$ $(i \in T)$;

$r_n = b$, $p_n = 1$, $w_n = 2$;

$y = \sum_{1\le i\le j\le t} a_ia_j + A + b + 2$.

Cf. reduction 4.4(*i*) and Figure 4.4.

(*1*) KNAPSACK $\propto n|1|r_n\geq 0|\sum C_i$†:

$$n = t+t'+u+1;$$
$$r_i = 0, \quad p_i = \tau+a_i \quad (i \in T = \{1,\ldots,t\});$$
$$r_i = 0, \quad p_i = \tau \quad (i \in T' = \{t+1,\ldots,t+t'\});$$
$$r_i = 0, \quad p_i = \upsilon \quad (i \in U = \{t+t'+1,\ldots,t+t'+u\});$$
$$r_n = t\tau+b, \; p_n = 1;$$
$$y = \upsilon+\tfrac{1}{2}u(u+1)\upsilon;$$

where

$$t' = t(t+1)a_*;$$
$$\tau = (t'+1)(b+1)+t';$$
$$u = \tfrac{1}{2}(t+t')(t+t'+1)\tau+(t+1)\tau;$$
$$\upsilon = u(\sigma+1);$$
$$\sigma = \textstyle\sum_{i\notin U} p_i = (t+t')\tau+A+1.$$

If KNAPSACK has a solution, then $\sum_{i\in S} a_i = b$ for some $S \subset T$. Defining $S' = \{t+i|i \in T-S\} \subset T'$, we have for a processing order $(\{J_i|i \in S'\}$, $\{J_i|i \in S\}$, J_n, $\{J_i|i \in T'-S'\}$, $\{J_i|i \in T-S\}$, $\{J_i|i \in U\})$ that

$$\sum_{i\in U} C_i = \sum_{i\in S'\cup S} C_i + C_n + \sum_{i\in(T'-S')\cup(T-S)} C_i$$
$$\leq \sum_{i=1}^{t} i(\tau+a_*) + (t\tau+b+1) + \sum_{i=t+1}^{t+t'}(i\tau+b+1) + \sum_{i=1}^{t} ia_*$$
$$= \tfrac{1}{2}(t+t')(t+t'+1)\tau+t\tau+(t'+1)(b+1)+t(t+1)a_* = u$$

and

$$\sum_{i\in U} C_i = \sum_{i=1}^{u}(\sigma+i\upsilon) = u\sigma+\tfrac{1}{2}u(u+1)\upsilon.$$

Hence,

$$\sum C_i \leq u+u\sigma+\tfrac{1}{2}u(u+1)\upsilon = y.$$

Conversely, if $\sum C_i \leq y$ for some schedule, we claim that

(*A*) $\{J_i|i \notin U\}$ precedes $\{J_i|i \in U\}$;

(*B*) $S_i \leq \sigma$ for some $i \in U$;

(*C*) exactly t jobs precede J_n;

(*D*) $S_n = t\tau+b$.

It follows from (*A*) and (*B*) that $\{J_i|i \notin U\}$ is scheduled without interruption from 0 to σ. By (*C*) and (*D*), exactly t jobs from $\{J_i|i \in T\cup T'\}$ occupy a period of length $t\tau+b$. This implies that KNAPSACK has a solution, as is easily seen.

We now turn to the proofs of (*A*), (*B*), (*C*), and (*D*).

(*A*) If $J_{i'}$, $i' \notin U$, succeeds some J_i, $i \in U$, then we have

$$\sum C_i \geq C_{i'} + \sum_{i\in U} C_i > \upsilon+\tfrac{1}{2}u(u+1)\upsilon = y.$$

(*B*) If $S_i > \sigma$ for all $i \in U$, then we have

$$\sum C_i > \sum_{i \in U} C_i \geq u(\sigma+1)+\tfrac{1}{2}u(u+1)\upsilon = y.$$

Since $\sum C_i \leq y$ and, by (*A*) and (*B*), $\sum_{i \in U} C_i \geq u\sigma+\tfrac{1}{2}u(u+1)\upsilon$, we now know that $\sum_{i \notin U} C_i \leq u$.

(*C*) Let exactly s jobs from $\{J_i | i \in T \cup T'\}$ precede J_n.

If $0 \leq s \leq t-1$, then we have

$$\begin{aligned}\sum_{i \notin U} C_i &\geq \sum_{i=1}^{s} i\tau + (t\tau+b+1) + \sum_{i=s+1}^{t+t'} (i\tau+(t-s)\tau+b+1)\\ &= u-t'+(t+t'-s)(t-s)\tau+(t-s)(b+1)\\ &> u-t'+(t'+1)\tau > u.\end{aligned}$$

If $t+1 \leq s \leq t+t'$, then we have

$$\begin{aligned}\sum_{i \notin U} C_i &\geq \sum_{i=1}^{s} i\tau + (s\tau+1) + \sum_{i=s+1}^{t+t'} (i\tau+1)\\ &= u+(s-t-1)\tau+t+t'-s+1\\ &\geq u+1 > u.\end{aligned}$$

(*D*) Note that $S_n \geq r_n = t\tau+b$. If $S_n > t\tau+b$, then we have

$$\begin{aligned}\sum_{i \notin U} C_i &\geq \sum_{i=1}^{t} i\tau + (t\tau+b+2) + \sum_{i=t+1}^{t+t'} (i\tau+b+2)\\ &= u+1 > u.\end{aligned}$$

This completes the proof of Theorem 4.4. □

Remark. The dagger † indicates that the NP-completeness proof for the $n|1|r_n \geq 0|\sum C_i$ problem is valid only with respect to a binary encoding whereby subsets of identical jobs are represented by a number indicating their cardinality and a single copy of the data. In this encoding, the above transformation is polynomial in the length of the binary KNAPSACK input, which is $O(t \log a_*)$. In the straightforward encoding whereby every job is represented by two entries r_i, p_i, the transformation is exponential. The special encoding would not be necessary if KNAPSACK were unary NP-complete; unfortunately this is not the case, as has been pointed out in Chapter 2. We note that a similar reduction, given in [Garey *et al.* 1975]:

$$\text{3-PARTITION} \propto n|2|F|\sum C_i,$$

can be adapted for application to the $n|1|r_i \geq 0|\sum C_i$ problem. Since 3-PARTITION is unary NP-complete, the unary NP-completeness of these two problems is then proved without the above complications.

The results for *single-machine* scheduling subject to *precedence constraints* are collected in Theorems 4.5 and 4.6. In these reductions, the jobs will not be numbered from 1 up to n. They correspond to the vertices and edges of an undirected graph $G = (V,E)$; therefore, there will be *vertex jobs* J_i or $J_i^{(h)}$ $(i \in V)$ and *edge jobs* $J_{(i,j)}$ or $J_{(i,j)}^{(h)}$ $((i,j) \in E)$.

THEOREM 4.5. CLIQUE *is reducible to the following problems:*

(*a*) $n|1|prec,p_i=1|\sum U_i$;

(*b*) $n|1|prec,p_i=1|\sum T_i$.

Proof. Let $V = \{1,\ldots,v\}$. Define $e = |E|$, $\ell = \frac{1}{2}k(k-1)$, $k' = v-k$, $\ell' = e-\ell$.

(*a*) CLIQUE $\propto n|1|prec,p_i=1|\sum U_i$:

$n = v+e$;

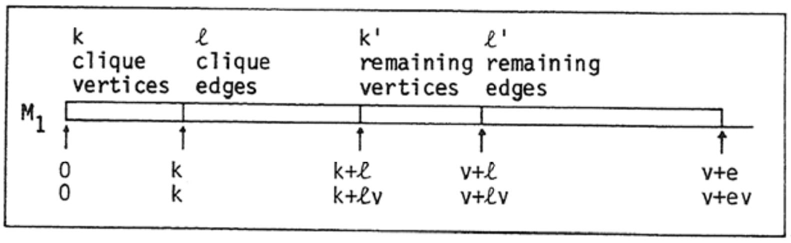
$d_i = v+e \quad (i \in V)$;

$d_{(i,j)} = k+\ell \quad ((i,j) \in E)$;

$J_i < J_{(i,j)} \quad (i \in V,\ (i,j) \in E)$;

$y = \ell'$.

Cf. [Garey & Johnson 1975] and Figure 4.8(*a*).



(*a*)
(*b*)
Figure 4.8

(*b*) CLIQUE $\propto n|1|prec,p_i=1|\sum T_i$:

$n = v+ev$;

$d_i = v+\ell v \quad (i \in V)$;

$d_{(i,j)}^{(h)} = v+ev \ (h = 1,\ldots,v-1),\ d_{(i,j)}^{(v)} = k+\ell v \quad ((i,j) \in E)$;

$J_i < J_{(i,j)}^{(1)} < J_{(i,j)}^{(2)} < \ldots < J_{(i,j)}^{(v)} \quad (i \in V,\ (i,j) \in E)$;

$y = \ell'k'+\frac{1}{2}\ell'(\ell'+1)v$.

It follows from our choice of due dates that $T_{(i,j)}^{(h)} = 0$ for $h = 1,\ldots, v-1$, $(i,j) \in E$ in every feasible active schedule. Hence, we can assume the edge jobs in a set $\{J_{(i,j)}^{(1)},\ldots,J_{(i,j)}^{(v)}\}$ to be scheduled consecutively and we may replace such a chain by one composite edge job $J_{(i,j)}$ with $p_{(i,j)} = v$ and $d_{(i,j)} = k+\ell v$, for each $(i,j) \in E$.

Consider any processing order in which J_h is the first late vertex job. It is easily seen that $S_h > v+\ell v$, and therefore J_h is preceded directly by an edge job $J_{(i,j)}$. Interchanging $J_{(i,j)}$ and J_h will decrease $\sum T_i$, and repeated improvements of this kind will eventually lead to a schedule in which all vertex jobs are on time. Thus, we have for any schedule that

$$\sum T_i \geq \sum_{i=1}^{\ell'}(k'+iv) = y.$$

If CLIQUE has a solution, then this bound can be attained, as illustrated in Figure 4.8(*b*). If CLIQUE has no solution, then at least $\ell'+1$ edge jobs are late and we have

$$\sum T_i \geq 1+y > y. \qquad \square$$

With respect to Theorem 4.6, the suggestion to start from the LINEAR ARRANGEMENT problem is due to E.L. Lawler.

THEOREM 4.6. LINEAR ARRANGEMENT *is reducible to the following problems:*
(*a*) $n|1|prec|\sum C_i$;
(*b*) $n|1|prec,p_i=1|\sum w_i C_i$.

Proof. Let $V = \{1,\ldots,v\}$. Define $e = |E|$, $u_i = |\{(i,j)|(i,j) \in E\}|$, *i.e.* the *degree* of i. We may assume that $e > \max_i\{u_i\}$.
(*a*) LINEAR ARRANGEMENT $\propto$ $n|1|prec|\sum C_i$:

$$n = ve;$$
$$p_i^{(1)} = 1,\ p_i^{(h)} = 0\ (h = 2,\ldots,e-u_i) \qquad (i \in V);$$
$$p_{(i,j)}^{(h)} = 0 \qquad (h = 1,2) \qquad ((i,j) \in E);$$
$$J_i^{(1)} < J_i^{(2)} < \ldots < J_i^{(e-u_i)} < J_{(i,j)}^{(1)} < J_{(i,j)}^{(2)} \qquad (i \in V,\ (i,j) \in E);$$
$$y = \tfrac{1}{2}v(v+1)e+k.$$

Through our choice of processing times we may assume the vertex jobs in a set $\{J_i^{(1)},\ldots,J_i^{(e-u_i)}\}$ and the edge jobs in a set $\{J_{(i,j)}^{(1)},J_{)i,j)}^{(2)}\}$ to be scheduled consecutively. Replacing these chains by composite vertex jobs J_i and composite edge jobs $J_{(i,j)}$, we obtain the following equivalent $n|1|prec|\sum w_i C_i$ problem:

$$n = v+e;$$
$$p_i = 1,\ w_i = e-u_i \qquad (i \in V);$$
$$p_{(i,j)} = 0,\ w_{(i,j)} = 2 \qquad ((i,j) \in E);$$
$$J_i < J_{(i,j)} \qquad (i \in V,\ (i,j) \in E);$$
$$y = \tfrac{1}{2}v(v+1)e+k.$$

Consider a permutation π of V, indicating that J_i ($i \in V$) is scheduled in position $\pi(i)$ among the vertex jobs. We may assume that each $J_{(i,j)}$ ($(i,j) \in E$) is inserted directly after the last one of its corresponding vertex jobs J_i and J_j. Thus we have for this schedule that

$$C_i = \pi(i),$$
$$C_{(i,j)} = \max\{\pi(i),\pi(j)\},$$

and its value is given by

$$\begin{aligned}\sum w_iC_i &= \sum_{i\in V}(e-u_i)\pi(i) + \sum_{(i,j)\in E} 2\max\{\pi(i),\pi(j)\}\\ &= e\sum_{i\in V}\pi(i) + \sum_{(i,j)\in E}(2\max\{\pi(i),\pi(j)\}-\pi(i)-\pi(j))\\ &= \tfrac{1}{2}v(v+1)e + \sum_{(i,j)\in E}|\pi(i)-\pi(j)|.\end{aligned}$$

It follows that LINEAR ARRANGEMENT has a solution if and only if there is a schedule with value $\le y$.

(*b*) LINEAR ARRANGEMENT $\propto$ $n|1|prec,p_i=1|\sum w_iC_i$:

$$n = vt+e;$$
$$w_i^{(1)} = \ldots = w_i^{(t-1)} = 0,\ w_i^{(t)} = e-u_i \quad (i \in V);$$
$$w_{(i,j)} = 2 \quad ((i,j) \in E).$$
$$J_i^{(1)} < J_i^{(2)} < \ldots < J_i^{(t)} < J_{(i,j)} \quad (i \in V,\ (i,j) \in E);$$
$$y = \tfrac{1}{2}v(v+1)et+t(k+1);$$

where

$$t = (v+3)e^2.$$

Replacing the chains $\{J_i^{(1)},\ldots,J_i^{(t)}\}$ by composite J_i's, we obtain an equivalent $n|1|prec|\sum w_iC_i$ problem:

$$n = v+e;$$
$$p_i = t,\ w_i = e-u_i \quad (i \in V);$$
$$p_{(i,j)} = 1,\ w_{(i,j)} = 2 \quad ((i,j) \in E);$$
$$J_i < J_{(i,j)} \quad (i \in V,\ (i,j) \in E);$$
$$y = \tfrac{1}{2}v(v+1)et+t(k+1).$$

Note that this is an inflated version of the $n|1|prec|\sum w_iC_i$ problem under (*a*). Suppose that J_i occupies position $\pi(i)$ among the vertex jobs. It is easily seen that we may again restrict ourselves to schedules in which the last vertex job preceding $J_{(i,j)}$ is either J_i or J_j. For such a schedule we have that

$$t\pi(i) \le C_i \le t\pi(i)+e,$$
$$t|\pi(i)-\pi(j)| \le |C_i-C_j| \le t|\pi(i)-\pi(j)|+e,$$
$$\max\{C_i,C_j\} < C_{(i,j)} < \max\{C_i,C_j\}+e,$$

and hence

$$t|\pi(i)-\pi(j)| < 2C_{(i,j)}-C_i-C_j < t|\pi(i)-\pi(j)|+3e.$$

If $\sum_{(i,j)\in E}|\pi(i)-\pi(j)| \le k$, then we have

$$\begin{aligned}\sum w_iC_i &= e\sum_{i\in V} C_i + \sum_{(i,j)\in E}(2C_{(i,j)}-C_i-C_j)\\ &< e\sum_{i\in V}(t\pi(i)+e) + \sum_{(i,j)\in E}(t|\pi(i)-\pi(j)|+3e)\\ &= \tfrac{1}{2}v(v+1)et + (v+3)e^2 + t\sum_{(i,j)\in E}|\pi(i)-\pi(j)| \le y.\end{aligned}$$

If $\sum_{(i,j)\in E}|\pi(i)-\pi(j)| > k$, then we have

$$\begin{aligned}\sum w_iC_i &= e\sum_{i\in V} C_i + \sum_{(i,j)\in E}(2C_{(i,j)}-C_i-C_j)\\ &> e\sum_{i\in V} t\pi(i) + \sum_{(i,j)\in E} t|\pi(i)-\pi(j)|\\ &= \tfrac{1}{2}v(v+1)et + t\sum_{(i,j)\in E}|\pi(i)-\pi(j)| \ge y.\end{aligned}$$

This proves the equivalence of both problems. □

The NP-completeness proofs for the problems with a *no wait* assumption are based on the well-known relation between these problems and the TSP which has been introduced in Section 3.3.

Given an $n|m|F,$*no wait*$|k$ problem, we define c_{ij} to be the minimum length of the time interval between S_i and S_j if J_j is scheduled directly after J_i. If we define

$$P_{hk} = \sum_{\ell=1}^{k} p_{h\ell}, \tag{4.1}$$

it is easily proved (see Section 14.5.2) that

$$c_{ij} = \max_k\{P_{ik}-P_{j,k-1}\}. \tag{4.2}$$

Finding a schedule that minimizes C_{max} is now equivalent to solving the TSP with $V = \{0,\ldots,n\}$ and weights c_{ij} defined by (4.2) and by $c_{0h} = 0$, $c_{h0} = P_{hm}$ for $h \ne 0$.

THEOREM 4.7. DIRECTED HAMILTONIAN PATH *is reducible to the following problems:*

(*a*) $n|m|F,$*no wait*$|C_{max}$;

(*b*) $n|m|F,$*no wait*$|\sum C_i$.

Proof.

(*a*) DIRECTED HAMILTONIAN PATH $\propto$ $n|m|F,$*no wait*$|C_{max}$.

Given $G' = (V',A')$, we define

$n = |V'|$,

$m = n(n-1)+2$.

All jobs have the same machine order $(M_1,M_2,\ldots,M_{m-1},M_m)$. To each pair of jobs (J_i,J_j) $(i,j = 1,\ldots,n,\ i \neq j)$ there corresponds one machine $M_k = M_{\kappa(i,j)}$ $(k = 2,\ldots,m-1)$, such that for no J_h some $M_{\kappa(i,h)}$ directly follows an $M_{\kappa(h,j)}$. Such an ordering of the pairs (i,j) can easily be constructed. Due to this property of the ordering, partial sums of the processing times can be defined unambiguously by

$$P_{hk} = \begin{cases} k\mu+\lambda & \text{if } k = \kappa(h,j) \text{ and } (h,j) \in A', \\ k\mu+\lambda+1 & \text{if } k = \kappa(h,j) \text{ and } (h,j) \notin A', \\ k\mu-\lambda & \text{if } k+1 = \kappa(i,h) \text{ and } (i,h) \in A', \\ k\mu-\lambda-1 & \text{if } k+1 = \kappa(i,h) \text{ and } (i,h) \notin A', \\ k\mu & \text{otherwise,} \end{cases}$$

for $k = 1,\ldots,m$, $h = 1,\ldots,n$, where

$$\lambda \geq 1,$$
$$\mu \geq 2\lambda+3.$$

The processing times are given by (*cf.* (4.1))

$$p_{h1} = P_{h1},$$
$$p_{hk} = P_{hk}-P_{h,k-1} \quad (k = 2,\ldots,m).$$

Through the choice of μ, these processing times are all strictly positive integers.

We can now compute the c_{ij}, as defined by (4.2). Through the choice of λ, it is immediate that $P_{ik}-P_{j,k-1}$ is maximal for $k = \kappa(i,j)$. Hence,

$$c_{ij} = \begin{cases} \mu+2\lambda & \text{if } (i,j) \in A', \\ \mu+2\lambda+2 & \text{if } (i,j) \notin A'. \end{cases}$$

Since $P_{im} = m\mu$ for all J_i, it now follows that G has a hamiltonian path if and only if this $n|m|F,\textit{no wait}|C_{max}$ problem has a solution with value

$$C_{max} \leq (n-1)(\mu+2\lambda)+m\mu.$$

(*b*) DIRECTED HAMILTONIAN PATH $\propto n|m|F,\textit{no wait}|\sum C_i$.
G' has a hamiltonian path if and only if the $n|m|F,\textit{no wait}|\sum C_i$ problem, constructed as in (*a*), has a solution with value

$$\sum C_i \leq \tfrac{1}{2}n(n-1)(\mu+2\lambda)+nm\mu. \qquad \square$$

4.3. Remarks

The results presented in Section 4.2 offer a valuable insight into the location of the borderline between "easy" and "hard" machine scheduling problems. Computational experience with many problems proved to be NP-complete confirms the impression that a polynomial-bounded algorithm for one and thus for all of them is highly unlikely to exist. As indicated in Chapter 2, NP-completeness thus functions as a formal justification to use enumerative methods of solution such as branch-and-bound.

Many classical machine scheduling problems have now been shown to be efficiently solvable or NP-complete. Some notable exceptions are indicated by question marks in Table 4.1. These open problems are briefly discussed below.

The most notorious one is the $n|1||\sum T_i$ problem. Extensive investigations have failed to uncover either a polynomial-bounded algorithm or a reduction proving its NP-completeness. The existence of an $O(n^5 p_*)$ algorithm [Lawler 1975C] implies that the problem is definitely not *unary* NP-complete. However, we conjecture that it is *binary* NP-complete and that an enumerative approach is unavoidable (see [Fisher 1974] and Chapter 11). This would indicate a major difference between the $\sum w_i T_i$ and $\sum w_i U_i$ problems, as demonstrated by Table 4.1.

With respect to the $n|1|prec|\sum w_i C_i$ problem, the exact location of the borderline has been determined (see Table 4.2 Note (1) and Theorem 4.6); with respect to other criteria of the $\sum f_i$ type the situation is less clear and especially the status of the $n|1|tree,p_i=1|\sum w_i T_i$ and $n|1|tree,p_i=1|\sum w_i U_i$ problems needs investigation.

A conjecture with respect to the $n|3|F,no\ wait|C_{max}$ and $n|2|F,no\ wait|\sum C_i$ problems is not obvious; both problems may well be efficiently solvable. Stimulating prizes have been put up to promote research in this direction (see [Lenstra *et al.* 1975]).

The question of the complexity of the $n|3|I,prec,p_i=1|C_{max}$ problem has been raised already in [Ullman 1975].

Finally, let us stress again that the complexity measure provided by the NP-completeness concept does not capture certain intuitive variations in complexity within the class of NP-complete problems. For instance, in Chapters 12 and 13 we will report on the successful incorporation of an $n|1|r_i \geq 0|L_{max}$ algorithm in lower bound computations for the $n|m|P|C_{max}$ and $n|m|G|C_{max}$ problems; note, however, that these problems are all NP-complete and thus equivalent up to a polynomial-bounded reduction. In order to formalize these differences, a further investigation of the consequences of allowing a unary encoding (*i.e.*, including the $p_{ik} \leq p_*$ condition) seems an interesting research topic.

Part II. Enumerative methods

5. RECURSIVE IMPLEMENTATION

The complexity results presented in Part I indicate that for many sequencing problems a *good algorithm* is highly unlikely to be found. It appears that with respect to these problems we have to settle for some form of *enumeration* of the solution space whereby the feasible solutions are identified and an optimal one is obtained. For all but the smallest problems the number of feasible solutions is so large that the use of a computer for the actual computations is unavoidable. Thus, the computational performance of any enumerative method not only depends on algorithmic details such as those presented in Part III but also on the computer implementation. This latter topic forms the subject of Part II.

More specifically, the following chapters will be devoted to a discussion of a *recursive* approach to the implementation of enumerative methods. We hope to demonstrate that such an approach leads to procedures that are elegant, easy to understand, easily programmed and easily proved. While these positive aspects will probably be recognized by most programmers, a familiar argument against recursive procedures suggests that none the less they require inordinate running times. Thus, ironically, many recursive approaches advocated in the literature are implemented after complicated manipulations in an iterative fashion [Barth 1968; Bitner *et al.* 1975; Gries 1975]! We will demonstrate on a simple example that with respect to efficiency a recursive implementation need certainly not be inferior to an iterative one; this remains true even if we consider a measure of efficiency that is computer and compiler independent.

The example referred to above is closely related to many sequencing problems and involves the *generation of all permutations of a finite set*. In Chapter 6 we discuss various types of recursive permutation generators and present some results concerning their efficiency relative to iterative generators.

Since feasible solutions of many sequencing problems are characterized by permutations, generators of permutations can be used in a straightforward way to solve such problems by *explicit enumeration* of all feasible solutions. We give some examples in Chapter 7, but it should be clear that this approach will solve only relatively small problems.

However, the advantages of a recursive approach carry through to forms of *implicit enumeration* as well. We illustrate this in Chapter 8 by presenting general frameworks for a popular type of implicit enumeration method known as *branch-and-bound*, in which again recursion plays a crucial role.

6. AN EXAMPLE: GENERATION OF PERMUTATIONS

6.1. Introduction

Methods for generating combinatorial configurations can be classified as either *lexicographic* or *minimum-change* methods. The first mentioned type of method generates the configurations in a "dictionary" order, whereas the second type produces a sequence in which successive configurations differ as little as possible. The relative advantages of minimum-change methods have been discussed in the literature: the entire sequence is generated efficiently, each configuration being derived from its predecessor by a simple change; moreover, a minimum-change generator "may permit the value of the current arrangement to be obtained by a small correction to the immediate previous value" [Ord-Smith 1971].

The very "cleanliness" [Lehmer 1964] of these combinatorial methods allows a proper demonstration of what we believe to be the advantages of a recursive approach to the implementation of enumerative methods.

The algorithms which are to be presented in this chapter are defined as ALGOL 60 procedures. They contain no labels and generate the entire sequence of configurations after one call. Each time a new configuration has been obtained, a call of a procedure "problem" is made. Parameters of this procedure are the configuration and, for minimum-change algorithms, the positions in which it differs from its predecessor. The actual procedure corresponding to "problem" has to be defined by the user to handle each configuration in the desired way.

Previously published iterative generators usually have been organized in such a way that each call generates one configuration from its predecessor only. This necessitates continual recomputation of the information that is needed to find the next configuration in the sequence. A mechanism for performing this kind of computations efficiently has been described in [Ehrlich 1973A]. We do feel, however, that much of the clarity of essentially recursive algorithms is lost within any iterative implementation.

For generators of various types of combinatorial configurations such as subsets, combinations and permutations, we refer to [Wells 1971; Ehrlich 1973A; Even 1973; Lenstra 1973; Lenstra & Rinnooy Kan 1975B; Reingold *et al.* 1976]. Permutation generators have been surveyed in [Lehmer 1964; Ord-Smith 1970; Ord-Smith 1971].

In Section 6.2 two minimum-change generators of permutations are presented. The first one produces a sequence in which each permutation is derived from its predecessor by *transposing two adjacent elements*. Its basic principles have been discovered by Steinhaus [Gardner 1974] and were rediscovered independently in [Trotter 1962] and [Johnson 1963]. Trotter's iterative algorithm was for a number of years the fastest permutation generator. A more efficient iterative implementation has been presented in [Ehrlich 1973B]; see also [Gries 1975; Dershowitz 1975]. The second minimum-change generator proceeds by *transposing two (not necessarily adjacent) elements*. Its transposition rules have been developed by Wells [Wells 1961] and simplified by Boothroyd in recursive [Boothroyd 1965] and iterative [Boothroyd 1967A; Boothroyd 1967B] implementations. In [Ord-Smith 1971], the latter algorithm was found to be the fastest of six generators, including those of [Trotter 1962] and [Boothroyd 1967A].

The lexicographic generator of permutations in Section 6.3 produces all permutations π of the set $\{1,\dots,n\}$ in such a way that $\pi(n)\pi(n-1)\dots\pi(1)$ is an *increasing n-ary number*. A slight modification leads to a more efficient pseudolexicographic generator.

In Section 6.4 our recursive generators are compared to previously published procedures.

6.2. Minimum-change generators

Given a set $\{\pi^*(1),\dots,\pi^*(n)\}$, we define an undirected graph $G(n)$ whose vertices are given by the $n!$ n-permutations of this set; (π,ρ) is an edge of $G(n)$ iff π and ρ differ only in two neighbouring components. A hamiltonian path in $G(n)$ corresponds to a sequence of permutations in which each permutation is derived from its predecessor by transposing two adjacent elements.

According to Steinhaus's method, we may construct such a sequence inductively as follows. For $n = 1$, it consists of the 1-permutation. Let the sequence of $(n-1)$-permutations be given. Placing $\pi^*(n)$ at the right of the first $(n-1)$-permutation, we obtain the first n-permutation. The $n-1$ next ones are obtained by successively interchanging $\pi^*(n)$ with its left-hand neighbour. After that, $\pi^*(n)$ is found at the left of the first $(n-1)$-permutation. Replacing this $(n-1)$-permutation by its successor in the $(n-1)$-sequence gives us the $(n+1)$-st n-permutation, and the $n-1$ next ones arise from successive transpositions of $\pi^*(n)$ with its right-hand neighbour. Then

$\pi^*(n)$ is found at the right of the second (n-1)-permutation, which is now replaced by the third one, and the process starts all over again. It is easily seen that the first and last permutations in the sequence are given by $\pi^* = (\pi^*(1),\ldots,\pi^*(n))$ and $\rho^* = (\pi^*(2),\pi^*(1),\pi^*(3),\ldots,\pi^*(n))$ respectively; they are adjacent and thus we have found a hamiltonian circuit in $G(n)$.

Steinhaus's method can be described more formally by a sequence $S(2)$ of $n!-1$ transpositions. Denoting the transposition of $\pi^*(i)$ and the h-th element in the current permutation of $\{\pi^*(1),\ldots,\pi^*(i-1)\}$ by $i\leftrightarrow h$, we define the transposition sequence $S(i)$ recursively by

$$S(i) = S(i+1),i\leftrightarrow h_1,S(i+1),i\leftrightarrow h_2,\ldots,S(i+1),i\leftrightarrow h_{i-1},S(i+1)$$

where

$$h_k = \begin{cases} k & \text{if } \pi^*(i) \text{ moves rightwards,} \\ i-k & \text{if } \pi^*(i) \text{ moves leftwards,} \end{cases}$$

and $S(n+1)$ is empty. Figure 6.1 and Table 6.1(mc1) show the graphs $G(n)$ for $n \le 4$ and the sequence for $n = 4$. Note that $G(4)$ is the edge graph of a solid truncated octahedron, replicas of which fill entire 3-space. Similar

TABLE 6.1. PERMUTATION SEQUENCES

	mc1	mc2	lex	plex
1	1234	1234	4321	4321
2	1243	2134	3421	3421
3	1423	2314	4231	4231
4	4123	3214	2431	2431
5	4132	3124	3241	2341
6	1432	1324	2341	3241
7	1342	1342	4312	4312
8	1324	3142	3412	3412
9	3124	3412	4132	4132
10	3142	4312	1432	1432
11	3412	4132	3142	1342
12	4312	1432	1342	3142
13	4321	1423	4213	4123
14	3421	4123	2413	1423
15	3241	4213	4123	4213
16	3214	2413	1423	2413
17	2314	2143	2143	2143
18	2341	1243	1243	1243
19	2431	3241	3214	1324
20	4231	2341	2314	3124
21	4213	2431	3124	1234
22	2413	4231	1324	2134
23	2143	4321	2134	2314
24	2134	3421	1234	3214

statements of this remarkable property hold for all n [Lenstra Jr. 1973B].

The following *minimum-change generator of permutations* produces the sequence described above.

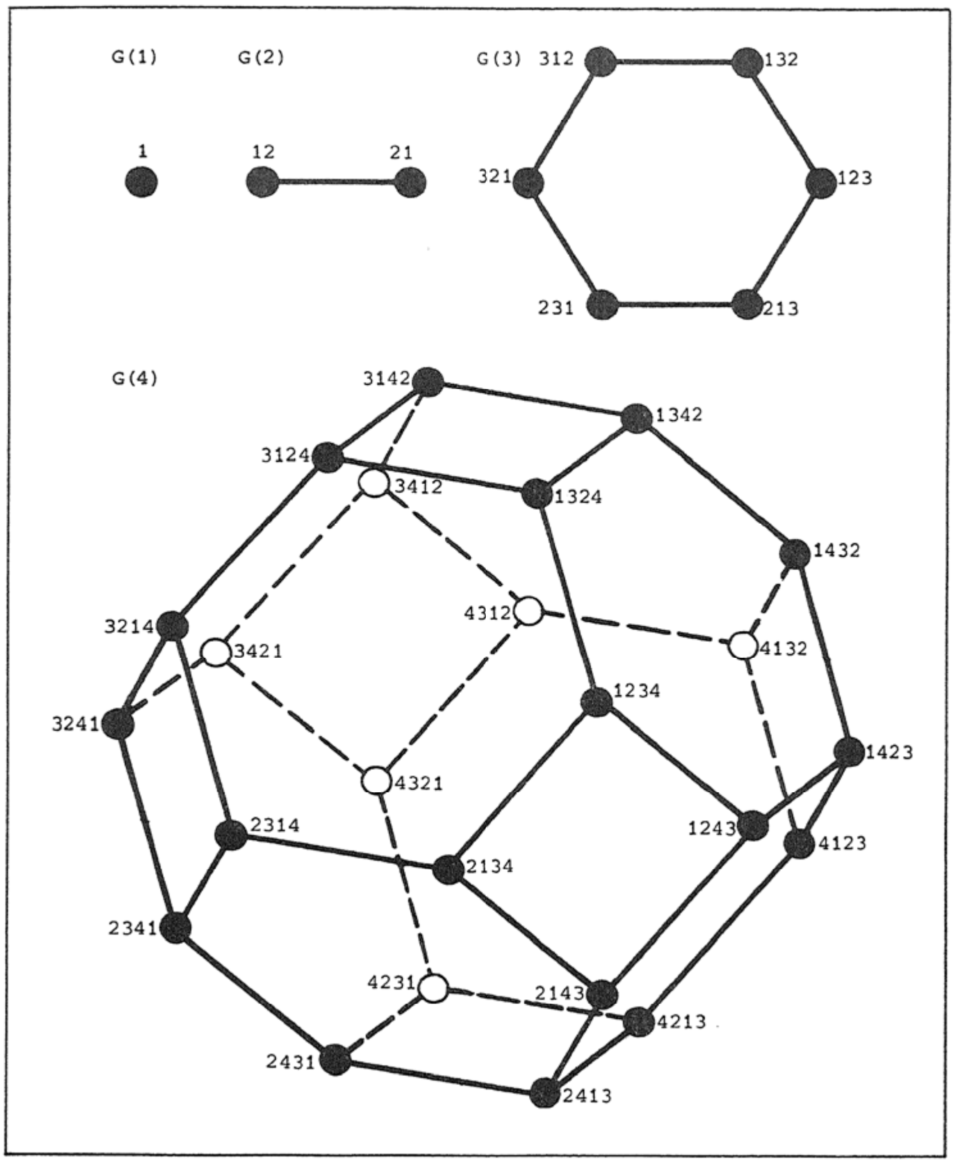


Figure 6.1 Graphs G(n).

```
procedure pm mc1 (problem,n,pi); value n,pi;
integer n; array pi; procedure problem;
begin   real pin; integer k,q; boolean array r[1:n];

        procedure rite(i); value i; integer i;
        if i < n then
        begin   boolean rj; real pii; integer ti,j;
                pii:= pi[q]; j:= i+1;
                q:= q-1;
                rj:= r[j]; if rj then rite(j) else left(j);
                for ti:= 2 step 1 until i do
                begin   k:= q+ti;
                        pi[k-1]:= pi[k]; pi[k]:= pii; problem(pi,k-1);
                        rj:= ¬rj; if rj then rite(j) else left(j)
                end;
                r[j]:= ¬rj
        end      else
        begin   q:= 0;
                for k:= 2 step 1 until n do
                begin   pi[k-1]:= pi[k]; pi[k]:= pin; problem(pi,k-1)
                end
        end;

        procedure left(i); value i; integer i;
        if i < n then
        begin   boolean rj; real pii; integer ti,j;
                pii:= pi[q+i]; j:= i+1;
                rj:= r[j]; if rj then rite(j) else left(j);
                for ti:= i-1 step -1 until 1 do
                begin   k:= q+ti;
                        pi[k+1]:= pi[k]; pi[k]:= pii; problem(pi,k);
                        rj:= ¬rj; if rj then rite(j) else left(j)
                end;
                r[j]:= ¬rj;
                q:= q+1
        end      else
        begin   for k:= n-1 step -1 until 1 do
                begin   pi[k+1]:= pi[k]; pi[k]:= pin; problem(pi,k)
                end;
                q:= 1
        end;

        pin:= pi[n]; q:= 0; for k:= 2 step 1 until n do r[k]:= false;
        problem(pi,0); if n > 1 then left(2)
end pm mc1.
```

A call "pm mc1 (problem,n,π^*)" has the following effect:

if n = 1, then a call "problem(π^*,0)" is made; else

- a hamiltonian path in G(n) from π^* to $\rho^* = (\pi^*(2),\pi^*(1),\pi^*(3),\ldots,\pi^*(n))$ is traversed;
- in vertex π^* a call "problem(π^*,0)" is made;
- in each vertex π, reached by transposition of the elements in positions k and k+1, a call "problem(π,k)" is made.

The latter two assertions are clear from inspection. To prove the first one, we note that a call "rite(i)" ("left(i)") performs a series of i-1 transpositions of $\pi^*(i)$ with its right (left) neighbour, where the predicate r(i) indicates which direction has to be chosen. By induction on i we can show that a call "rite(i)" or "left(i)" generates all permutations in which the current order of $\pi^*(1),\ldots,\pi^*(i-1)$ is preserved, only transposing adjacent elements, whereas just before such a call and immediately after its execution, π and q have the following property:

the indices (i,...,n) can be rearranged as $(j_1,\ldots,j_q,j_{q+i},\ldots,j_n)$ with $j_1 > \ldots > j_q$, $j_{q+i} < \ldots < j_n$, such that $\pi(k) = \pi^*(j_k)$ for $k = 1,\ldots,q,q+i,\ldots,n$.

The first assertion now corresponds to the effect of a call "left(2)", which indeed activates the whole process. This completes the proof.

Using the integer q to determine the place of the transpositions is easier and more efficient than keeping track of the inverse permutation for that purpose, as is done in [Ehrlich 1973A; Ehrlich 1973B].

In order to add to the transparency and efficiency of the procedure, two simple constructions have been applied. First, we have distinguished between the leftward and rightward moves of the elements by means of two procedures calling themselves and one another. Further, the deepest level of the recursion has been written out explicitly. This device clearly reduces the number of checks to see if the bottom of the recursion has been reached already; it enables us also to deal separately with the n-th element, which is involved in (n-1)/n of the transpositions.

Let G'(n) be an extension of G(n) on the same vertex set; (π,ρ) is an edge of G'(n) iff π and ρ differ in only two components. A hamiltonian path in G'(n) corresponds to a sequence of permutations in which each permutation is derived from its predecessor by transposing two elements.

Such a path is defined by a sequence of n!-1 transpositions. Denoting the transposition of the elements in positions k and ℓ by $k \leftrightarrow \ell$, we may define the transposition sequence corresponding to the Wells-Boothroyd method by

$$T(n) = T(n-1), m_{n-1} \leftrightarrow n, T(n-1), m_{n-2} \leftrightarrow n, \ldots, T(n-1), m_1 \leftrightarrow n, T(n-1)$$

where

$$m_k = \begin{cases} k & \text{if } n \text{ is even and } k < n-2, \\ n-1 & \text{if } n \text{ is odd or } k \geq n-2; \end{cases}$$

note that T(1) is empty. Table 6.1(mc2) shows the resulting sequence for $n = 4$.

The above description leads directly to our second *minimum-change generator of permutations*.

```
procedure pm mc2 (problem,n,pi); value n,pi;
integer n; array pi; procedure problem;
begin   real pik,pim;

        procedure even(n); value n; integer n;
        if n > 2 then
        begin   real pin; integer k,m;
                m:= n-1; pin:= pim;
                odd(m);
                for k:= m, m, m-2 step -1 until 1 do
                begin   pi[n]:= pik:= pi[k]; pi[k]:= pin; pin:= pik;
                        problem(pi,k,n); odd(m)
                end
        end     else
        begin   pi[2]:= pi[1]; pi[1]:= pim; problem(pi,1,2)
        end;

        procedure odd(n); value n; integer n;
        begin   real pin; integer k,m;
                m:= n-1; pin:= pi[n]; pim:= pi[m];
                even(m);
                for k:= m step -1 until 1 do
                begin   pi[n]:= pik:= pi[m]; pi[m]:= pim:= pin; pin:= pik;
                        problem(pi,m,n); even(m)
                end
        end;

        problem(pi,0,0); if n > 1 then
        begin   if (n÷2)×2 = n then begin pim:= pi[n]; even(n) end else odd(n)
        end
end pm mc2.
```

A call "pm mc2 (problem,n,π^*)" has the following effect:

if n = 1, then a call "problem(π^*,0,0)" is made; else

- a hamiltonian path in G'(n) from π^* to ρ^* is traversed, where

$$\rho^* = \begin{cases} (\pi^*(2),\ldots,\pi^*(n-3),\pi^*(n-1),\pi^*(n),\pi^*(n-2),\pi^*(1)) & \text{if n is even,} \\ (\pi^*(1),\ldots,\pi^*(n-2),\pi^*(n),\pi^*(n-1)) & \text{if n is odd;} \end{cases}$$

- in vertex π^* a call "problem(π^*,0,0)" is made;
- in each vertex π, reached by transposition of the elements in positions k and ℓ, a call "problem(π,k,ℓ)" is made.

The inductive proof is left to the reader. Again, we have distinguished between two types of changes, *viz.* n even and n odd, and the case n = 2 has been handled separately.

We make one final remark on minimum-change sequences of permutations. Given an undirected graph H(n) on n vertices, we define an undirected graph $G_H(n)$ on the set of n-permutations; (π,ρ) is an edge of $G_H(n)$ iff π can be obtained from ρ by a single transposition of the elements in positions k and ℓ, where (k,ℓ) is an edge of H(n). One [Lenstra Jr. 1973B] can prove that $G_H(n)$ *contains a hamiltonian circuit iff* H(n) *contains a spanning tree*. The "only if"-part is obvious; the "if"-part follows by an inductive argument. The *transposition graph* H(n) of Steinhaus's method is a tree with edge set $\{(k,k+1)\,|\,k = 1,\ldots,n-1\}$; it is properly contained in the transposition graph of the Wells-Boothroyd method.

6.3. Lexicographic generators

A *lexicographic generator of permutations* can be constructed even more simply. At each level of the recursion exactly one component of π is defined and at the bottom a call "problem(π)" is made.

```
procedure pm lex (problem,n); value n;
integer n; procedure problem;
begin   integer h; integer array pi[1:n];

        procedure node(n); value n; integer n;
        if n = 1 then problem(pi) else
        begin   integer k,m,pin;
                m:= n-1; pin:= pi[n];
                node(m);
                for k:= m step -1 until 1 do
                begin   pi[n]:= h:= pi[k]; pi[k]:= pin; pin:= h;
                        node(m)
                end;
                for k:= n step -1 until 2 do pi[k]:= pi[k-1]; pi[1]:= pin
        end;

        for h:= n step -1 until 1 do pi[h]:= n+1-h;
        node(n)
end pm lex.
```

A call "pm lex (problem,n)" has the following effect:

- all permutations π of $\{1,\dots,n\}$ are generated in such a way that $\pi(n)\pi(n-1)\dots\pi(1)$ is an increasing n-ary number;
- for each permutation π a call "problem(π)" is made.

To prove the first assertion, let us assume that, given a permutation π, a call "node(ℓ)" is made. It is easily checked that just before the ℓ calls "node(ℓ-1)" on the next level of the recursion, the then current permutation ρ is given by

$$\begin{aligned}\rho &= (\rho(1),\dots,\rho(k-1),\rho(k),\rho(k+1),\dots,\rho(\ell-1),\rho(\ell),\rho(\ell+1),\dots,\rho(n))\\ &= (\pi(1),\dots,\pi(k-1),\pi(k+1),\pi(k+2),\dots,\pi(\ell),\pi(k),\pi(\ell+1),\dots,\pi(n)),\end{aligned}$$

for $k = \ell,\ell-1,\dots,1$. By induction on ℓ it can be shown that a call "node(ℓ)" generates all permutations π in which $\pi(\ell+1),\dots,\pi(n)$ remain unchanged, in increasing order, whereas just before such a call and immediately after its execution, π satisfies $\pi(1) > \pi(2) > \dots > \pi(\ell)$. The observation that the effect of a call "node(n)" corresponds to the first assertion completes the proof.

Our *pseudolexicographic generator of permutations* is derived from the lexicographic one; their difference can be characterized by the replacement of the above equalities by

$$\begin{aligned}\rho &= (\rho(1),\dots,\rho(k-1),\rho(k),\rho(k+1),\dots,\rho(\ell-1),\rho(\ell),\rho(\ell+1),\dots,\rho(n))\\ &= (\pi(1),\dots,\pi(k-1),\pi(\ell),\pi(k+1),\dots,\pi(\ell-1),\pi(k),\pi(\ell+1),\dots,\pi(n)).\end{aligned}$$

This simplification of the transposition rules leads to a gain in efficiency at the expense of losing the lexicographic ordering.

```
procedure pm plex (problem,n); value n;
integer n; procedure problem;
begin   integer h; integer array pi[1:n];

        procedure node(n); value n; integer n;
        if n = 1 then problem(pi) else
        begin   integer k,m,pik,pin;
                m:= n-1; pin:= pi[n];
                node(m);
                for k:= m step -1 until 1 do
                begin   pi[n]:= pik:= pi[k]; pi[k]:= pin;
                        node(m);
                        pi[k]:= pik
                end;
                pi[n]:= pin
        end;

        for h:= n step -1 until 1 do pi[h]:= n+1-h;
        node(n)
end pm plex.
```

Again, the recursive approach makes the construction and analysis of this generator almost trivial. Table 6.1(lex,plex) shows the lexicographic and pseudolexicographic sequences for n = 4.

6.4. Computational experience

The algorithms presented in Sections 6.2 and 6.3 have been compared to ALGOL 60 versions of several minimum-change generators, mentioned in Section 6.1.

Table 6.2 shows the result of the comparison. The running times have been measured during one uninterrupted run on the Electrologica X8 computer of the Mathematisch Centrum; a procedure with an empty body was chosen for the actual parameter "problem". Our minimum-change algorithms turn out to be faster than corresponding previously published procedures. Although the time differences are not spectacular, a recursive approach should certainly not be rejected on grounds of computational inefficiency *a priori*.

Results like the above ones unavoidably remain computer and compiler dependent. It is of interest to note in this context that some experiments using PASCAL on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam instead of ALGOL 60 on the Electrologica X8 showed a nineteen-fold increase in speed for a recursive subset generator and a fourteen-fold increase for an iterative one. On the other hand, the running times of the iterative generators may be reduced by up to twenty percent by a different transformation of these generators into PASCAL procedures producing all configurations at one call.

In order to develop a computer independent measure of efficiency, let us define

$$a = \lim_{n\to\infty} \frac{\text{number of array subscript evaluations}}{\text{number of generated configurations}} ,$$

array access being a dominant factor in this type of ALGOL 60 procedure [Ord-Smith 1971]. For recursive algorithms, evaluation of a is accomplished by the solution of recursive expressions. For the iterative algorithms except Ehrlich's ones, only lower bounds can be given; it is not clear if finite limits exist.

TABLE 6.2. COMPARISON OF VARIOUS PERMUTATION GENERATORS

generator	restrictions	time	a
pm mc1	$n \geq 1$	42.9	3
[Trotter 1962; Ord-Smith 1971]	$n \geq 2$	91.3	≥ 7
[Ehrlich 1973B]	$n \geq 3,\ n \neq 4$	58.1	3
pm mc2	$n \geq 1$	54.3	3.35
[Boothroyd 1965]	$n \geq 1$	103.3	6.72
[Boothroyd 1967B; Ord-Smith 1971]	$n \geq 5$	83.6	>3.16
pm lex	$n \geq 1$	92.4	6.44
pm plex	$n \geq 1$	82.5	5.44

time : CPU seconds on an Electrologica X8 for $n = 8$.

a : average array access (in the limit).

7. EXPLICIT ENUMERATION

Generators of combinatorial configurations can be used to solve many combinatorial optimization problems through enumeration and evaluation of all feasible solutions. Needless to say, only very small problems can be solved by such a brute force approach, even if the minimum-change property of the generators is exploited. However, they can be applied to validate more complicated solution methods by checking their results on small problems.

As an illustration we will show how generators of permutations can be used to solve sequencing problems P of the form

$$\min_\pi\{f_P(\pi)\}$$

where π runs over all permutations of $\{1,\ldots,n\}$. Several problems of this type have been introduced in Chapter 3. We recall that the criterion function of the *quadratic assignment problem* (QAP) is given by

$$f_{QAP}(\pi) = \sum_{i=1}^n \sum_{j=1}^n c_{\pi(i)\pi(j)}d_{ij}$$

where (c_{ij}) and (d_{ij}) are nonnegative $n\times n$-matrices. If we take $d_{ij} = 1$ for $i > j$, $d_{ij} = 0$ otherwise, we obtain the *acyclic subgraph problem* (ASP). Analogously, the choice $d_{12} = d_{23} = \ldots = d_{n-1,n} = d_{n1} = 1$, $d_{ij} = 0$ otherwise, leads to the *travelling salesman problem* (TSP), that is called *symmetric* if $c_{ij} = c_{ji}$ for all i,j.

If we define the *reflection* of π by $\bar{\pi} = (\pi(n),\ldots,\pi(1))$, it is obvious that $f_{ASP}(\bar{\pi}) = \sum_{i\neq j} c_{ij} - f_{ASP}(\pi)$ for the ASP and $f_{TSP}(\bar{\pi}) = f_{TSP}(\pi)$ for the symmetric TSP. It follows that for these two problems it suffices to enumerate a *reflection-free* set of permutations. Further, since $f_{TSP}((\pi(k+1),\ldots,\pi(n),\pi(1),\ldots,\pi(k))) = f_{TSP}(\pi)$ for any k, we may fix one of the components of π when solving a TSP. The $(n-1)!/2$ solutions to a symmetric TSP are the hamiltonian circuits in a complete undirected graph; they are called *rosary permutations* [Harada 1971; Read 1972; Roy 1973].

In the first minimum-change generator of permutations, discussed in Section 6.2, the elements $\pi^*(1)$ and $\pi^*(2)$ are transposed half-way. If a permutation π is generated before this transposition, then its reflection $\bar{\pi}$ occurs thereafter. Hence the first $n!/2$ permutations form a reflection-free set (*cf.* [Roy 1973]). Generally, the $n!/m!$ permutations preserving the original order of $\pi^*(1),\ldots,\pi^*(m)$ can be generated by a simple adaptation of "pm mc1":

```
procedure pp mc1 (problem,n,m,pi); ...;
begin  ...
        ...; if n > m then left(m+1)
end pp mc1.
```

The above sequencing problems may now be solved by calls

```
     pm mc1 (qap,n,π)
     pp mc1 (asp,n,2,π)
     pp mc1 (tsp,n-1,if symmetric then 2 else 1,π)
```

where "qap", "asp" and "tsp" are procedures which compute the changes occurring in the criterion functions of these problems.

A more sophisticated application of generators of combinatorial configurations arises in the context of a suboptimal approach to combinatorial optimization problems. Several heuristic methods involve the systematic exploration of a neighbourhood of some given solution, starting anew from improved solutions until no further improvement is found and a local optimum has been obtained [Reiter & Sherman 1965]. This exploration can sometimes be described in terms of checking all combinations of m out of n elements, and a minimum-change generator, such as the procedure "cb mc" from [Lenstra & Rinnooy Kan 1975B], might then profitably be applied.

For instance, a solution π to the ASP is called *relatively optimal* [Lenstra Jr. 1973A] if

$$\left.\begin{array}{l} \sum_{i=j+1}^{k} (c_{\pi(j)\pi(i)} - c_{\pi(i)\pi(j)}) \geq 0 \\ \sum_{i=j}^{k-1} (c_{\pi(i)\pi(k)} - c_{\pi(k)\pi(i)}) \geq 0 \end{array}\right\} \text{ for } j,k = 1,\dots,n.$$

Such a solution can be constructed by systematic generation of all pairs (j,k) with $1 \leq j < k \leq n$. This can be done very efficiently with a special version of "cb mc" for $m = 2$; in the phase of verification, when no further improvement is found, this method checks each element of the matrix (c_{ij}) exactly once.

A solution to the symmetric TSP is called m-*optimal* if it is impossible to obtain a better solution by replacing m of its edges by a different set of m edges [Lin 1965]. A 3-optimal method based on "cb mc" proved to be more efficient than the algorithm presented in [Lin 1965].

Analogously, one can obtain an efficient suboptimal algorithm for the QAP. The approach is applicable also to other types of difficult sequencing problems, *e.g.* in the area of machine scheduling.

8. IMPLICIT ENUMERATION

The permutation generators presented in Section 6.3 can easily be adapted to be used for implicit enumeration purposes by adding a lower bound calculation on all possible completions of a partial configuration. In the early fifties, Lehmer used such an approach to solve the linear assignment problem (!) [Tompkins 1956]; similarly, the enumeration scheme of "pm plex" has been applied to the travelling salesman problem [Barth 1968]. The fact that our recursive generators coupled with a simple lower bound may well outperform sophisticated implicit enumeration algorithms that suffer from a large computational overhead (see Section 11.3) underlines the applicability of recursive programming to implicit enumeration methods of the *branch-and-bound* type in general.

In this chapter we present a quasi-ALGOL description of branch-and-bound procedures, indicating in which case a recursive approach might be suitable. For a formal characterization of branch-and-bound procedures, we refer to the axiomatic framework in [Mitten 1970] and its correction in [Rinnooy Kan 1974]; see also [Agin 1966; Balas 1968] for analyses of the case in which the set of feasible solutions is finite and [Kohler & Steiglitz 1974] for the case of permutation problems. Some standard examples of branch-and-bound methods have been surveyed in [Lawler & Wood 1966].

Suppose then, that a *set* X *of feasible solutions* and a *criterion function* $f: X \rightarrow \mathbb{R}$ are given, and define the *set* X^* *of optimal solutions* by

$$X^* = \{x^* | x^* \in X, f(x^*) = \min\{f(x) | x \in X\}\}.$$

A branch-and-bound procedure to find an element of X^* can be characterized as follows.

- Throughout the execution of the procedure, the *best solution* x^* *found so far* provides an *upper bound* $f(x^*)$ on the value of the optimal solution.
- A *branching rule* b associates to $Y \subset X$ a family $b(Y)$ of subsets such that $\cup_{Y' \in b(Y)} Y' \cap X^* = Y \cap X^*$; the subsets Y' are the *descendants* of the *parent* subset Y. This rule only has to be defined on a class $\mathcal{X}$ with $X \in \mathcal{X}$ and $b(Y) \subset \mathcal{X}$ for any $Y \in \mathcal{X}$.
- A *bounding rule* lb: $\mathcal{X} \rightarrow \mathbb{R}$ provides a *lower bound* $lb(Y) \leq f(x)$ for all $x \in Y \in \mathcal{X}$. *Elimination* of Y occurs if $lb(Y) \geq f(x^*)$.
- A *predicate* $\xi: \mathcal{X} \rightarrow \{\underline{\text{true}}, \underline{\text{false}}\}$ indicates if during the examination of Y (*e.g.* during the calculation of lb(Y)) a feasible solution x(Y) is

generated which has to be evaluated. *Improvement* of x^* occurs if $f(x^*) > f(x(Y))$.

- A *search strategy* chooses a subset from the collection of generated subsets which have so far neither been eliminated nor led to branching.

It turns out that, of the three search disciplines that have been used most frequently, two are suitable for recursive implementation. To illustrate this point, we shall now present three general procedures:

- "bb jumptrack" implements a *breadth-first search* where a subset with minimal lower bound is selected for examination; this type of tree search is known as *frontier search;*
- "bb backtrack1" implements a *depth-first search* where the descendants of a parent subset are examined in an arbitrary order; this type is known as *newest active node search;*
- "bb backtrack2" implements a *depth-first search* where the descendants are chosen in order of nondecreasing lower bounds; this type is sometimes called *restricted flooding.*

During the tree search, the parameters na and nb count the numbers of subsets that are eliminated and that lead to branching respectively. We define the operation ":z∈" in the statement "s:z∈ S" to mean that $s := s^*$ with $z(s^*) = \min\{z(s) | s \in S\}$; hence, ":∈" indicates an arbitrary choice.

```
procedure bb jumptrack ($X, f, x^*, \mathcal{b}, lb, \xi, na, nb$);
begin   local $\mathcal{Y}, \mathcal{Y}', \mathcal{B} \subset \mathcal{X}$, $Y, Y' \in \mathcal{X}$, $LB: \mathcal{X} \to \mathbb{R}$;
        na:= nb:= 0; $\mathcal{Y} := \emptyset$;
        $LB(X) := lb(X)$; if $\xi(X)$ then $x^* :f\in \{x^*, x(X)\}$;
        if $LB(X) \geq f(x^*)$ then na:= 1 else $\mathcal{Y} := \{X\}$;
        while $\mathcal{Y} \neq \emptyset$ do
        begin   $Y :LB\in \mathcal{Y}$;
                nb:= nb+1; $\mathcal{B} := \mathcal{b}(Y)$; $\mathcal{Y} := (\mathcal{Y} - \{Y\}) \cup \mathcal{B}$;
                while $\mathcal{B} \neq \emptyset$ do
                begin   $Y' :\in \mathcal{B}$; $\mathcal{B} := \mathcal{B} - \{Y'\}$;
                        $LB(Y') := lb(Y')$; if $\xi(Y')$ then $x^* :f\in \{x^*, x(Y')\}$
                end;
                $\mathcal{Y}' := \{Y' | Y' \in \mathcal{Y}, LB(Y') \geq f(x^*)\}$;
                na:= na+$|\mathcal{Y}'|$; $\mathcal{Y} := \mathcal{Y} - \mathcal{Y}'$
        end
end bb jumptrack.
```

```
procedure bb backtrack1 (X,f,x*,b,lb,ξ,na,nb);
begin   local Y' ∈ X;

        procedure node(Y);
        begin   local B ⊂ X, LB ∈ ℝ;
                LB:= lb(Y); if ξ(Y) then x*:f∈ {x*,x(Y)};
                if LB ≥ f(x*) then na:= na+1 else
                begin   nb:= nb+1; B:= b(Y);
                        while B ≠ ∅ do
                        begin   Y':∈ B; B:= B-{Y'};
                                if LB < f(x*) then node(Y')
                        end
                end
        end;

        na:= nb:= 0;
        node(X)
end bb backtrack1.

procedure bb backtrack2 (X,f,x*,b,lb,ξ,na,nb);
begin   local B ⊂ X, Y' ∈ X, LB: X → ℝ;

        procedure node(Y);
        begin   local Y ⊂ X;
                nb:= nb+1; Y:= B:= b(Y);
                while B ≠ ∅ do
                begin   Y':∈ B; B:= B-{Y'};
                        LB(Y'):= lb(Y'); if ξ(Y') then x*:f∈ {x*,x(Y')}
                end;
                while Y ≠ ∅ do
                begin   Y':LB∈ Y; Y:= Y-{Y'};
                        if LB(Y') ≥ f(x*) then na:= na+1 else node(Y')
                end
        end;

        na:= nb:= 0;
        LB(X):= lb(X); if ξ(X) then x*:f∈ {x*,x(X)};
        if LB(X) ≥ f(x*) then na:= 1 else node(X)
end bb backtrack2.
```

Anyone familiar with branch-and-bound will have noticed that the above descriptions provide only a minimal algorithmic framework. Numerous problem-dependent variations may be included in an actual procedure. For instance, elimination of Y may be possible already during the calculation of lb(Y) or may be due to *elimination criteria* based on dominance rules or feasibility considerations. In a minor (and in our experience quite successful) variation on "bb backtrack1", the descendants Y' of a parent subset Y are not chosen arbitrarily, but according to some heuristic, *e.g.* preliminary lower bounds $lb'(Y')$ with $lb(Y) \le lb'(Y') \le lb(Y')$. Many similar variations are possible but do not affect the basic mechanisms outlined above.

From our experience with the implementation of branch-and-bound algorithms we may conclude that again the recursive approach produces transparent procedures, in which much administrative work is taken over by the compiler without a noticeable negative effect on overall efficiency.

The actual solution of a problem by branch-and-bound can be conveniently represented by means of a *search tree* consisting initially of a single node representing X. If a subset Y leads to branching, $|b(Y)|$ nodes are created representing the subsets $Y' \in b(Y)$; edges are created between the parent node and its descendants. Nodes can be eliminated by lower bounds or elimination criteria.

A main characteristic of many branch-and-bound procedures is the unpredictability of their computational behaviour. Their worst-case performance may be close to explicit enumeration, and no satisfying analyses of average-case behaviour have been presented up to now (see, however, [Karp 1975B]). Extensive computational experience seems to be the only way to test their quality. Branch-and-bound should not be used before one feels sure that the complexity of the problem is such that no better approach can be found (*cf.* Chapter 2). However, this is often the case, and methods of branch-and-bound are widely used for solving combinatorial optimization problems. This will be amply illustrated in Part III.

Part III. Sequencing by implicit enumeration

9. THE TRAVELLING SALESMAN PROBLEM

9.1. Introduction

The travelling salesman problem (TSP) has been formulated in Section 3.3 as follows.

Given a directed graph $G = (V,A)$ with a weight c_{ij} for each arc $(i,j) \in A$, find a hamiltonian circuit on G of minimum total weight.

We shall distinguish between the *asymmetric* TSP (ATSP) and the *symmetric* TSP (STSP) where $c_{ij} = c_{ji}$ for all $(i,j) \in A$. In the latter case we may consider the pair of arcs $\{(i,j),(j,i)\}$ as one edge (i,j) and view the STSP as the problem of finding a minimum-weight hamiltonian circuit on an undirected graph $G = (V,E)$.

It has been pointed out in Section 3.3 that G may be assumed to be a *complete* graph with $V = \{1,\ldots,n\}$, $A = V\times V$ and $c_{ii} = \infty$ for all $i \in V$.

In order to characterize feasible solutions, we note that a hamiltonian circuit or *tour* corresponds to a subgraph on V for which three requirements are satisfied:

(1) every vertex has indegree one;

(2) every vertex has outdegree one;

(3) the subgraph is connected.

This leads to the following formulation of the TSP as a *0-1 linear programming problem* where $x_{ij} = 1$ ($x_{ij} = 0$) denotes inclusion (exclusion) of arc (i,j):

$$\min\{\textstyle\sum_{(i,j)\in A} c_{ij}x_{ij} \mid$$

$$\textstyle\sum_{i\in V} x_{ij} = 1 \qquad (j \in V) \qquad (9.1)$$

$$\textstyle\sum_{j\in V} x_{ij} = 1 \qquad (i \in V) \qquad (9.2)$$

$$\textstyle\sum_{(i,j)\in S\times S} x_{ij} \le |S|-1 \quad (S \subset V,\ S \ne \emptyset,\ S \ne V) \qquad (9.3a)$$

$$x_{ij} \in \{0,1\} \qquad ((i,j) \in A)\}.$$

Conditions (9.1),(9.2),(9.3*a*) correspond to requirements (1),(2),(3) respectively. Alternatively, (9.3*a*) may be replaced by

$$\textstyle\sum_{i\in S} \sum_{j\in V-S} x_{ij} \ge 1 \qquad (S \subset V,\ S \ne \emptyset,\ S \ne V) \qquad (9.3b)$$

The *subtour elimination constraints* (9.3*a*) are equivalent to the *loop constraints* (9.3*b*) since

$$\textstyle\sum_{(i,j)\in S\times S} x_{ij} = \sum_{i\in S} \sum_{j\in S} x_{ij} = |S| - \sum_{i\in S} \sum_{j\in V-S} x_{ij}.$$

The 0-1 linear program contains many redundant constraints. For instance, it is sufficient to impose (9.3*a*) only for $S \subset V$ with $1 \le |S| \le \lceil \frac{1}{2}n \rceil$, and it is easily seen that (9.1) and (9.3*b*) imply (9.2). For linear characterizations of the travelling salesman polytope we refer to [Grötschel & Padberg 1974; Grötschel & Padberg 1975]. Nevertheless, the above formulation will be a useful tool in describing the various solution approaches that will be reviewed in this chapter.

Both the ATSP and the STSP have been proved to be NP-complete in Chapter 3; the only satisfactory solution methods are based on implicit enumeration. The branch-and-bound algorithms developed so far have in common that each node in the search tree is characterized by a set R of *required* arcs and a set F of *forbidden* arcs; the subset of solutions corresponding to this node contains all tours including R and excluding F. We note as a general principle that we may add to F each arc that together with one or more arcs from R could create a nonhamiltonian cycle or *subtour*. Each $(i,j) \in F$ can be removed from the problem, which may be realized by putting $c_{ij} := \infty$. In the case of the ATSP we can view each vertex pair $\{i,j\}$ with $(i,j) \in R$ as a single vertex with $c_{h\{i,j\}} = c_{hi}$ and $c_{\{i,j\}k} = c_{jk}$. In the case of the STSP, this would destroy the symmetry of the problem and the set R has to be taken explicitly into account.

In Section 9.2.1 we shall consider three bounding rules developed for the ATSP and their refinements for the STSP. In Section 9.2.2 we investigate various branching rules that determine the successive augmentations of R and F. In Section 9.2.3 we describe some algorithms that were actually implemented. Our computational experience with these methods is reported in Section 9.3. Concluding remarks are contained in Section 9.4.

9.2. Algorithms

9.2.1. *Lower bounds*

Generally, lower bounds will be obtained by relaxing one of the three requirements (1), (2) and (3) characterizing a feasible tour.

We may view each of these conditions as imposing a matroid structure on A. An optimal tour corresponds to a maximum-weight independent subset in the intersection of three matroids, where (1) and (2) define partition matroids and (3) defines a graphic matroid. By ignoring one of the matroids, we obtain a two-matroid problem which, in general, can be solved by a polynomial-bounded algorithm [Lawler 1976].

(a) the matching approach

Let us first relax the constraints by ignoring the connectivity requirement (3). We thus obtain a *weighted bipartite matching* or *linear assignment problem:*

$$\begin{aligned}\min\{\textstyle\sum_{(i,j)\in A} c_{ij}x_{ij} \mid \; & \textstyle\sum_{i\in V} x_{ij} = 1 \quad (j \in V),\\ & \textstyle\sum_{j\in V} x_{ij} = 1 \quad (i \in V),\\ & x_{ij} \geq 0 \qquad ((i,j) \in A)\}.\end{aligned}$$

This problem can be solved in $\mathcal{O}(n^3)$ steps [Lawler 1976]. Originally, the assignment bound has been proposed in [Eastman 1958]; it can be strengthened by a device due to [Christofides 1972]. For the ATSP, the bound has been used quite successfully in [Shapiro 1966; Bellmore & Malone 1971; Thompson 1975]. For the STSP, it has been less successful; the subgraphs corresponding to the optimal assignment can be expected to contain a large number of 2-cycles (i,j,i).

However, viewing the STSP as the TSP on an undirected graph G = (V,E) we can combine the degree requirements (1) and (2) into the single constraint that every vertex should have degree two. We define

$$V_d = \{i \mid i \in V,\ |R\cap\{(i,j)\mid j\in V\}| = d\} \quad \text{for } d = 0,1,2$$

as the set of vertices incident to exactly d required arcs. Removing the sets V_2 and R from the problem and relaxing (3), we now obtain a *weighted b-matching problem:*

$$\begin{aligned}\min\{\textstyle\sum_{(i,j)\in E} c_{ij}x_{ij} \mid \; & \textstyle\sum_{j\in V} x_{ij} = b_i \quad (i \in V),\\ & x_{ij} \in \{0,1\} \quad ((i,j) \in E)\},\end{aligned} \tag{9.4}$$

where $b_i = 2-d$ for $i \in V_d$ (d = 0,1). This problem can be solved in $\mathcal{O}(n^4)$ steps [Edmonds 1975]. Satisfactory results have been reported in [Bellmore & Malone 1971].

We note in passing that Edmonds' b-matching algorithm employs constraints of the form

$$x_{ij} \in \{0\}\cup\mathbb{N} \quad ((i,j) \in E).$$

It is a well-known secret [Edmonds 1974] that we can enforce $x_{ij} \in \{0,1\}$ by

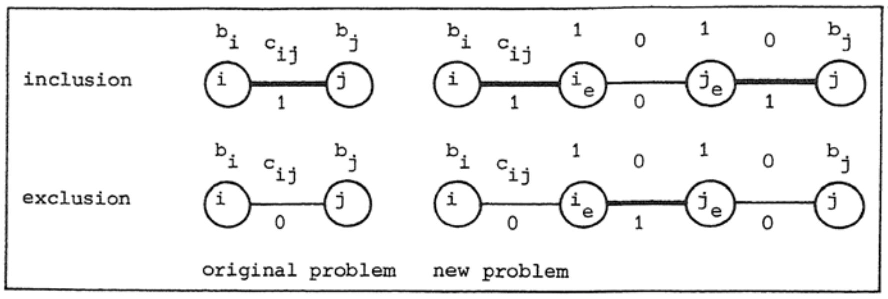


Figure 9.1

replacing each edge $e = (i,j)$ by three edges $(i,i_e),(i_e,j_e),(j_e,j)$ and defining $b_{i_e} = b_{j_e} = 1$, $c_{ii_e} = c_{ij}$, $c_{i_ej_e} = c_{j_ej} = 0$. Each variable in the new problem will be assigned a value from $\{0,1\}$; the combinations representing inclusion and exclusion of e in the original problem are shown in Figure 9.1.

(*b*) *the reduction approach*

A weaker ATSP bound is provided by any lower bound for the linear assignment problem and, more specifically, by the value of any feasible solution to the dual weighted bipartite matching problem:

$$\max\{\textstyle\sum_{i\in V} u_i + \sum_{j\in V} v_j \mid u_i+v_j \le c_{ij} \;((i,j) \in A)\}.$$

In [Little *et al.* 1963] such a feasible dual solution (u_i,v_j) with value lb is obtained by *reduction* of (c_{ij}) according to algorithm LB1 below.

```
procedure algorithm LB1 (V,A,n,c,u,v,lb);
begin  local i,j;
       for i:= 1 to n do $u_i$:= min{$c_{ij}$ | (i,j) $\in$ A};
       for j:= 1 to n do $v_j$:= min{$c_{ij}-u_i$ | (i,j) $\in$ A};
       lb:= sum{$u_i$ | i $\in$ V} + sum{$v_j$ | j $\in$ V}
end algorithm LB1.
```

Similarly, for the STSP we can replace (9.4) by

$$0 \le x_{ij} \le 1 \;((i,j) \in E)$$

and imitate the approach of Little *et al.* by seeking a feasible solution to the dual weighted b-matching problem:

$$\max\{\textstyle\sum_{i\in V} b_iu_i - \sum_{(i,j)\in E} w_{ij} \mid u_i+u_j-w_{ij} \le c_{ij} \;((i,j) \in E),\; w_{ij} \ge 0 \;((i,j) \in E)\},$$

where $b_i = 2-d$ for $i \in V_d$ $(d = 0,1)$. Such a feasible dual solution (u_i,w_{ij}) with value lb can be constructed by algorithm LB2 below (*cf.* [Liesegang 1974]). The sets V_2 and R are assumed to be removed from the problem. The operation ":ϵ" in the statement "s:ϵ S" has been defined to mean that s becomes an arbitrary element selected from the set S (see Chapter 8).

```
procedure algorithm LB2 ($V_0$,$V_1$,E,n,c,u,w,lb);
begin   local $\delta$,i,j;
        for i:= 1 to n do $u_i$:= min{if j < i then $c_{ij}-u_j$ else $c_{ij}$ | (i,j) $\in$ E};
        for i:= 1 to n do for j:= 1 to n do $w_{ij}$:= 0;
        for i:= 1 to n do if i $\in V_0$ then
        begin   j:$\in$ {j | (i,j) $\in$ E, $c_{ij}-u_i-u_j+w_{ij}$ = 0};
                $\delta$:= min{$c_{ik}-u_i-u_k+w_{ik}$ | (i,k) $\in$ E, k $\neq$ j};
                $u_i$:= $u_i+\delta$;
                $w_{ij}$:= $w_{ij}+\delta$
        end;
        lb:= 2 sum{$u_i$ | i $\in V_0$} + sum{$u_i$ | i $\in V_1$} - sum{$w_{ij}$ | (i,j) $\in$ E}
end algorithm LB2.
```

Let us define

$$W_i = \{j \mid (i,j) \in E,\ c_{ij}-u_i-u_j+w_{ij} = 0\}.$$

The initial assignments to u_i and w_{ij} correspond to the reduction of (c_{ij}) in algorithm LB1 and yield a feasible dual solution (u_i,w_{ij}) with $|W_i| \geq 1$ for each $i \in V$. In the case of the STSP, however, we would like to have $|W_i| \geq 2$ for each $i \in V_0$. Therefore, for each $i \in V_0$ we choose a $j \in W_i$ and determine the subminimum $\delta = \min\{c_{ik}-u_i-u_k+w_{ik} \mid (i,k) \in E,\ k \neq j\}$; if $|W_i| = 1$, then $\delta > 0$ and increasing u_i and w_{ij} by δ contributes $2\delta-\delta = \delta$ to the lower bound while maintaining dual feasibility.

Both algorithm LB1 and LB2 operate in $O(n^2)$ steps.

(*c*) *the spanning tree approach*

If we ignore the degree requirement (2) instead of the connectivity constraint (3), the resulting problem is to find a minimum-weight connected subgraph on V with indegree one for every vertex. Such a subgraph consists of a *spanning arborescence*, *i.e.* a directed tree rooted at some vertex r with indegree one for every vertex in V-{r}, and one additional arc directed to the root r; it contains exactly one (directed) cycle, passing through r. We may arbitrarily fix $r = 1$ and thus obtain an ATSP bound by constructing a minimum-weight *spanning 1-arborescence*, consisting of

- a minimum-weight spanning arborescence on V rooted at vertex 1;
- a minimum-weight arc directed to vertex 1.

The two-matroid problem of finding an optimal arborescence can be solved by the algorithm from [Edmonds 1967; Karp 1972A] in $\mathcal{O}(|A| \log n)$ or $\mathcal{O}(n^2)$ steps [Tarjan 1975B].

Similarly, for the STSP we may relax the degree requirements (1) and (2) and look for a minimum-weight connected subgraph on V containing exactly n edges with degree two for vertex 1. Assuming that $c_{ij} = -\infty$ for each $(i,j) \in R$, we now obtain an STSP bound by constructing a minimum-weight *spanning 1-tree*, consisting of

- a minimum-weight spanning tree on $V-\{1\}$;
- a minimum-weight pair of edges incident to vertex 1.

The one-matroid problem of finding an optimal tree can be solved by several efficient algorithms of order $\mathcal{O}(|E| \log n)$ [Kruskal 1956], $\mathcal{O}(n^2)$ [Prim 1957; Dijkstra 1959] and $\mathcal{O}(|E| \log \log n)$ [Tarjan 1975A].

Let $\{u_i | i \in V\}$ be a given set of *penalties* with $\sum_{i \in V} u_i = 0$ and let f^* denote the optimal solution value for the STSP. Replacing (c_{ij}) by $(c_{ij}+u_i+u_j)$ does not change the weight of any tour but may lead to a different weight $w(u)$ of the optimum spanning 1-tree. Thus we have

$$f^* \geq \max_u \{w(u)\}.$$

An *ascent method* for obtaining a lower bound lb by calculating or approximating $\max_u\{w(u)\}$ is given by algorithm LB3 below. We note that

- a call "algorithm S1T (V,E,c,u,d,w)" delivers a spanning 1-tree that is of minimum weight w with respect to $(c_{ij}+u_i+u_j)$ and has degree d_i for vertex $i \in V$;
- if $d_i = 2$ for all $i \in V$ the 1-tree is a tour and the STSP has been solved;
- if $d_i < 2$ $(d_i > 2)$ vertex i is "too expensive" ("too cheap") and u_i is decreased (increased) by $t(d_i-2)$;
- if no improvement of lb occurs during p succeeding iterations, the process is terminated.

```
procedure algorithm LB3 (V,E,n,c,lb,p,t);
begin   local u,d,w,q,i;
        lb:= $-\infty$; q:= p;
        for i:= 1 to n do $u_i$:= 0;
        for q:= q-1 while q $\geq$ 0 do
        begin   algorithm S1T (V,E,c,u,d,w);
                if lb < w then begin lb:= w; q:= p end;
                if sum{$|d_i-2|$ | i $\in$ V} = 0 then q:= 0 else
                for i:= 1 to n do $u_i$:= $u_i+t(d_i-2)$
        end
end algorithm LB3.
```

This bounding approach has been introduced in [Held & Karp 1970; Christofides 1970]. For appropriate choices of p and t as well as for alternative penalizing strategies we refer to [Held & Karp 1971; Helbig Hansen & Krarup 1974; Camerini *et al.* 1974; Christofides 1975; Thompson 1975]. None of these methods does always lead to $\max_u\{w(u)\}$ and, moreover, there may exist a nonremovable *duality gap* $f^*-\max_u\{w(u)\}$. However, excellent results have been obtained with this STSP bound, especially within the framework of the *subgradient optimization* approach [Held *et al.* 1974] that appears in many other contexts as well.

9.2.2. *Enumeration schemes*

Suppose that the current node of the search tree is characterized by a set R of required arcs and a set F of forbidden arcs. If calculation of a lower bound does not lead to elimination of this node, we have to apply a branching rule. Below we shall discuss the branching strategies that have been proposed in combination with the respective bounding approaches.

(*a*) *the matching approach*
If the subgraph corresponding to the optimal assignment is not a tour, it consists of at least two disconnected subtours. Let us select the smallest of those subtours, consisting of, say, vertex set $S = \{i_1,\ldots,i_s\}$ and arc set $\{a_1,\ldots,a_s\}$. We can "break" this subtour by forbidding one of its arcs (*cf.* (9.3*a*)) or, alternatively, by requiring one of its vertices to be adjacent to a vertex not in S (*cf.* (9.3*b*)). We will now formulate four branching

schemes, each of which eliminates the subtour by creating s descendant nodes, characterized by sets R_k and F_k $(k = 1,\dots,s)$.

(*A*) [Eastman 1958; Shapiro 1966]

$$R_k = R,$$
$$F_k = F\cup\{a_k\}.$$

(*A'*) [Bellmore & Malone 1971]

$$R_k = R\cup\{a_j \mid j = 1,\dots,k-1\},$$
$$F_k = F\cup\{a_k\}.$$

(*B*) [Eastman 1959; Bellmore & Malone 1971]

$$R_k = R,$$
$$F_k = F\cup\{(i_k,i) \mid i \in S\}.$$

(*B'*) [Garfinkel 1973]

$$R_k = R,$$
$$F_k = F\cup\{(i_j,i) \mid j = 1,\dots,k-1,\ i \in V-S\}\cup\{(i_k,i) \mid i \in S\}.$$

Strategy (*A*) is based on constraint (9.3*a*); (*A'*) is a variation on (*A*) with the additional feature that the subsets of solutions corresponding to the descendant nodes are disjoint. Strategy (*B*) is based on (9.3*b*); its refinement (*B'*) creates mutually exclusive subsets. We note that the latter two schemes, unlike the first two, cannot make use of the special structure of the STSP.

The choice of a subtour of minimum cardinality is justified by two observations:

- a relatively narrow search tree will be created;
- imposing (9.3*a*) ((9.3*b*)) for S with $|S| = s$ eliminates $[(n-s)!/e+\frac{1}{2}]$ $([(n-s)!/e+\frac{1}{2}][s!/e+\frac{1}{2}])$ feasible assignments, which is maximal for minimal s (*cf.* [Bellmore & Malone 1971]).

In the case of strategy (*A'*), the ordering of $(a_1,\dots,a_s)$ is of importance. The subset of solutions corresponding to $(R_1,F_1) = (R,F\cup\{a_1\})$ is the largest one and should have a large lower bound; the (R_s,F_s) problem is the most constrained one and should preferably contain the optimal tour. Therefore, the arcs should be ordered according to increasing likelihood of their presence in the optimal tour, for instance, according to nonincreasing assignment bounds for the (R,F_k) problem. A recursive search strategy selecting the descendants in the order $(R_s,F_s),\dots,(R_1,F_1)$ seems then most suitable. Similar remarks apply to strategy (*B'*) with respect to the ordering of $(i_1,\dots,i_s)$.

From computational experience reported in [Bellmore & Malone 1971; Thompson 1975] it appears that the "disjoint" enumeration schemes (*A'*) and (*B'*) are superior to (*A*) and (*B*) respectively. For the ATSP, the proper choice between (*A'*) and (*B'*) requires further investigation.

(*b*) *the reduction approach*

The branching approach taken in [Little *et al.* 1963] can be seen as a variation on strategy (*A'*) outlined above. We determine an arc a_1 whose removal leads to a maximum increase in the lower bound and an arc a_2 that forms a subtour together with the longest path consisting of a_1 and arcs from R. Two descendant nodes are then created with

$$R_1 = R,$$
$$F_1 = F\cup\{a_1\};$$
$$R_2 = R\cup\{a_1\},$$
$$F_2 = F\cup\{a_2\}.$$

A depth-first search proceeding along the branch corresponding to (R_2,F_2) appears to be appropriate. Furthermore, a large gain in efficiency is obtained by choosing the first encountered arc a_1 whose removal bridges the gap between the local lower bound and the global upper bound; if such an arc exists, the node corresponding to (R_1,F_1) will never be chosen [Lenstra 1972].

(*c*) *the spanning tree approach*

The enumeration scheme from [Held & Karp 1971] resembles the assignment strategy in that we again start from the structure provided by the lower bound calculation. Restricting ourselves to the penalties for which the maximal 1-tree was obtained, we order its nonrequired edges according to nonincreasing increases of the lower bound caused by their addition to F. We then create s descendant nodes according to strategy (*A'*), where s is the smallest index for which there exists a vertex i such that R does not satisfy its degree requirements but R_s does; the nonrequired edges incident to i can be added to F_s. Also in this case it seems appropriate to select the descendant nodes "from right to left". Computational experience from [Thompson 1975] suggests that Little's variation on the above approach may lead to even better results.

9.2.3. *Implementations*

(i) *algorithm LEA*

Algorithm LEA implements the method of [Little *et al.* 1963] for ATSPs. The bounding approach is given by algorithm LB1. The improved branching rule and the search strategy have been described in Section 9.2.2(*b*).

In the case of the STSP, there exists for every tour an equivalent reverse tour. Algorithm LEA avoids duplication by a simple modification of the branching rule that changes the leftmost nodes of the tree. If such a node corresponds to (R,F), we have $R = \emptyset$, and if $a_1 = (i,j)$ we characterize its first descendant by

$$R_1 = \emptyset,$$
$$F_1 = F\cup\{(i,j),(j,i)\}.$$

(ii) *algorithm HK0*
Algorithm HK0 implements the method of [Held & Karp 1971] for STSPs. The bounding approach is given by algorithm LB3; we used the spanning tree algorithm from [Dijkstra 1959] and choose $p = 20$ in the root node, $p = 10$ elsewhere, and $t = 1$. The branching and search strategies have been described in Section 9.2.2(*c*).

(iii) *algorithm HK1*
Algorithm HK1 is identical to algorithm HK0, except for the fact that in the root node the heuristic method of [Lin 1965] is applied to obtain a good initial upper bound.

9.3. Computational experience

9.3.1. *Test problems*

The approaches sketched in Section 9.2.3 were tested on a set of 25 problems which can be divided into four groups.

(1) *hamiltonian symmetric*
Given an undirected graph $G = (V,E)$, we define a hamiltonian STSP by

$$c_{ij} = \begin{cases} 0 & ((i,j) \in E), \\ 1 & (\text{otherwise}). \end{cases}$$

Clearly, G has a hamiltonian circuit if and only if this STSP has a solution with value 0. Four problems of this type were tested:
- the star graph given in Figure 9.2;
- the Königsberg graph given in Figure 9.3(*b*), representing Euler's famous Königsberg bridge problem (*cf.* Figure 9.3(*a*)) and obtained by a general

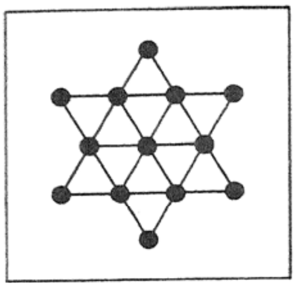
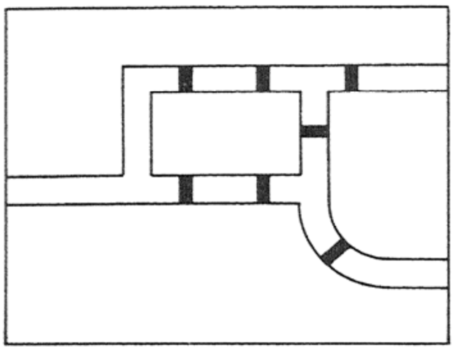


Figure 9.2 Star graph.


(*a*) The seven brigdes of Königsberg.

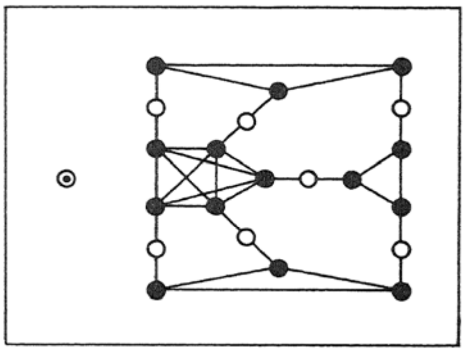

(*b*) Königsberg graph; vertex ⊙ is incident to all vertices ●.

Figure 9.3

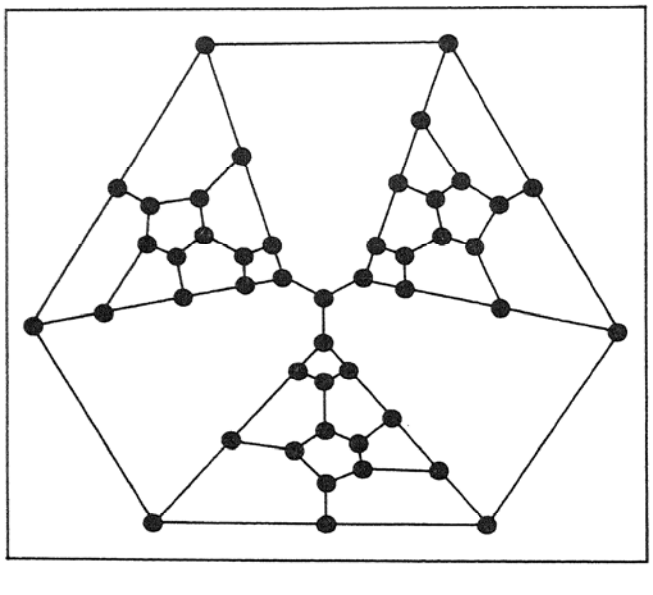


Figure 9.4 Tutte graph.

(and computationally useless) construction transforming a graph H into a graph G such that H has a eulerian path if and only if G has a hamiltonian circuit;

- the Tutte graph given in Figure 9.4;
- the graph on 64 vertices corresponding to the 64 squares of a chessboard, where i and j are adjacent if and only if they are at a knight's move distance.

(2) *euclidean symmetric*

Given 2n coordinates a_i, b_i, we define a euclidean STSP by

$$c_{ij} = \sqrt{(a_i - a_j)^2 + (b_i - b_j)^2}.$$

We obtained six problems of this type by generating 2n integers a_i, b_i from a uniform distribution between 0 and 100. Two problems from [Dantzig *et al.* 1954; Held & Karp 1962], based on a road-map of the U.S.A., were also included.

(3) *random symmetric*

We obtained six problems of this type by generating $\frac{1}{2}n(n-1)$ integers $c_{ij} = c_{ji}$ from a uniform distribution between 0 and 100. The problem from [Croes 1958] was also included.

(4) *random asymmetric*

We obtained six problems of this type by generating n(n-1) integers c_{ij} from a uniform distribution between 0 and 100.

9.3.2. *Results*

Algorithms LEA, HK0 and HK1 were coded in ALGOL 60 and run on the Electrologica X8 of the Mathematisch Centrum in Amsterdam. The text of the procedures can be found in [Lenstra 1972]; Table 9.1 shows the computational results.

Algorithm LEA is quite successful on hamiltonian STSPs and useless on euclidean ones; for algorithms HK0 and HK1, the situation is completely reversed. All methods perform rather well on random problems. For algorithm LEA, ATSPs are easier than STSPs, and algorithm HK1 performs slightly better then algorithm HK0.

TABLE 9.1. RESULTS

problem type	n	solution time			number of nodes		
		alg.LEA	alg.HK0	alg.HK1	alg.LEA	alg.HK0	alg.HK1
hamiltonian symmetric							
star	13	4	>540	>540	55	-	-
Kōnigsberg	22	62	-	>3400	963	-	-
Tutte	46	612	-	-	3693	-	-
knight's tour	64	118	-	-	125	-	-
euclidean symmetric							
	20	>3600	70	53	-	21	1
	20	>3600	63	57	-	26	16
	20	-	47	58	-	20	20
[Held & Karp 1962]	25	>3600	122	146	-	1	1
	25	>3600	198	197	-	41	31
	25	-	194	227	-	50	50
	25	-	248	127	-	55	24
[Dantzig *et al.* 1954]	42	-	3170	1410	-	221	73
random symmetric							
[Croes 1958]	20	23	11	27	235	1	1
	20	42	60	73	473	27	27
	20	59	64	76	642	26	26
	20	77	76	89	587	29	29
	25	288	124	132	2507	33	23
	25	576	1240	647	4695	207	123
	25	92	126	40	631	37	1
random asymmetric							
	20	52			699		
	20	7			37		
	20	52			715		
	30	86			653		
	30	151			1117		
	30	352			2773		

solution time : CPU seconds on an Electrologica X8.

number of nodes : including eliminated nodes.

algorithm LEA, HK0, HK1 : see Section 9.2.3.

Although our experiments involve problems which are, by current standards, of a relatively small size, some valuable conclusions may be drawn. In general, the distinction between the several types of TSPs turns out to be very crucial and the choice of an algorithm to solve a particular TSP should depend on the type of the problem involved. Not surprisingly, it appears useful to obtain a good initial upper bound.

9.4. Remarks

Due to its deceptive simplicity and wide applicability, the TSP occupies a central position in research on problems of combinatorial optimization. The development of optimal TSP algorithms has reached the advanced stage where the specific computer implementation has become crucial. This applies to the selection of a procedure for computing lower bounds as well as to the branching rule and the type of tree search to be chosen.

Typical examples of this phenomenon are provided by the improvements of Held and Karp's algorithm described in [Helbig Hansen & Krarup 1974] and by the computational experiments reported in [Thompson 1975]. From the latter paper it appears that ATSP bounds based on linear assignment are generally stronger than those based on spanning arborescences, while, on the contrary, the b-matching approach to the STSP is completely dominated in efficiency by the spanning tree relaxation. Altogether, we feel that a large scale computational comparison of TSP algorithms, emphasizing the various proposed implementation devices, is justified by the present confusion.

With respect to new algorithmic developments, we note that quite recently a promising and powerful bounding approach has been developed by De Leve (see [Wesseling 1975]). The investigation of elimination criteria has received little attention and seems a worth-while research topic.

10. ONE-MACHINE SCHEDULING I: MINIMIZING MAXIMUM LATENESS

10.1. Introduction

The one-machine problem which will be studied in this chapter is the $n|1|\mathit{prec},r_i\geq 0|L_{max}$ problem. It can be formulated as follows.

> Each of n jobs $J_1,\ldots,J_n$ has to be processed on a single machine which can handle only one job at a time. Job J_i $(i = 1,\ldots,n)$ is available for processing at its release date r_i, requires an uninterrupted processing time p_i and should preferably be completed by its due date d_i. Given precedence constraints define a partial ordering $<$ between the jobs; "$J_i < J_j$" means that J_j cannot start before the completion of J_i. The sets $B_i = \{j|J_j < J_i\}$ and $A_i = \{j|J_i < J_j\}$ indicate the jobs which are constrained to come before and after J_i respectively. Given a feasible processing order of the jobs, we can compute for each J_i a starting time $S_i \geq r_i$ with $S_i \geq C_j$ for all $j \in B_i$, a completion time $C_i = S_i+p_i$ with $C_i \leq S_j$ for all $j \in A_i$, and a lateness $L_i = C_i-d_i$. We want to find a processing order that minimizes the maximum lateness $L_{max} = \max_i\{L_i\}$.

To stress the *symmetry* inherent to the problem, it is useful to describe it in an alternative way. Let M_1 and M_3 be *non-bottleneck* machines of infinite capacity and M_2 a *bottleneck* machine of capacity one, and let K be some constant with $K \geq \max_i\{d_i\}$. J_i $(i = 1,\ldots,n)$ has to visit M_1,M_2,M_3 in that order and has to spend

- a *head* r_i on M_1 from 0 to r_i;
- a *body* p_i on M_2 from S_i to C_i;
- a *tail* q_i on M_3 from C_i to $L_i' = C_i+q_i$.

We want to minimize the maximum completion time $L'_{max} = \max_i\{L_i'\} = L_{max}+K$ on M_3.

The problem, now defined by n triples (r_i,p_i,q_i) and $<$, is clearly equivalent to its *inverse* problem defined by (q_i,p_i,r_i) and $<'$ with $J_i <' J_j$ if $J_j < J_i$; an optimal schedule for one problem can be reversed to obtain an optimal schedule for the other problem with the same solution value.

A number of special cases of this problem in which all r_i, p_i or q_i are equal can be solved by polynomial-bounded algorithms, as will be indicated in Section 10.2.1. Such a good method is unlikely to exist for the case in which r_i, p_i and q_i may assume arbitrary values and $A_i = B_i = \emptyset$ for all J_i.

The NP-completeness of this $n|1|r_i \geq 0|L_{max}$ problem has been established in Chapter 4 and justifies an enumerative approach such as branch-and-bound. Algorithms of this type have been proposed in [Dessouky & Margenthaler 1972; Bratley *et al.* 1973; Baker & Su 1974; McMahon & Florian 1975]. The first of these algorithms is not stated very clearly; the second one is surpassed by the fourth one in elegance and efficiency [McMahon & Florian 1975]. The remaining two algorithms will be described and extended to the general $n|1|\mathit{prec},r_i \geq 0|L_{max}$ case in Sections 10.2.2 and 10.2.3. Extensive computational experience is reported in Section 10.3. Some remarks, notably on the wide range of potential applications of this problem, are contained in Section 10.4.

10.2. Algorithms

10.2.1. *Special cases*

Let us first assume that $A_i = B_i = \emptyset$ for all J_i.

If all r_i are equal, an optimal schedule is provided by *Jackson's rule* [Jackson 1955]: L'_{max} is minimized by ordering the jobs according to nonincreasing q_i.

If all q_i are equal, the problem is similarly solved by ordering the jobs according to nondecreasing r_i. This result can be interpreted as a consequence of the symmetry discussed above.

If all p_i are equal, such a simple solution method is usually not available, unless $p_i = 1$ for all J_i. In the latter situation, algorithm JR below involving repeated application of Jackson's rule produces an optimal schedule [Horn 1974; Baker & Su 1974].

```
procedure algorithm JR (n,r,q,C);
begin   local S,Q,t,i;
        S:= {1,...,n}; t:= 0;
        while S ≠ ∅ do
        begin   t:= max{t,min{$r_j$|j ∈ S}};
                Q:= {j|j ∈ S, $r_j$ ≤ t};
                i:∈ {j|j ∈ Q, $q_j$ = max{$q_k$|k ∈ Q}}; S:= S-{i};
                $C_i$:= t:= t+1
        end
end algorithm JR.
```

The proof of this result is straightforward and depends on the fact that no job can become available during the processing of another one, so that it is never advantageous to postpone processing the selected J_i. This argument does not apply if $p_i = p_*$ for all J_i and p_* does not divide all r_i; *e.g.*, if $n = p_* = 2$, $r_1 = q_1 = 0$, $r_2 = 1$, $q_2 = 2$, postponing J_1 is clearly advantageous. However, algorithm JR does solve the general problem if we allow *job splitting* (*i.e.*, interruptions in the processing of a job); in this case we can interpret J_i as p_i jobs with heads r_i, bodies 1 and tails q_i.

Let us now examine the introduction of *precedence constraints* in the problems discussed so far. As a general principle, note that we may set

$$r_i := \max\{r_i, \max\{r_j+p_j \mid j \in B_i\}\},$$
$$q_i := \max\{q_i, \max\{p_j+q_j \mid j \in A_i\}\},$$

because in every feasible schedule $S_i \geq C_j \geq r_j+p_j$ for all $j \in B_i$ and $L'_j \geq C_i+p_j+q_j$ for all $j \in A_i$. Hence, if $J_i < J_j$, we will assume that

$$r_i < r_i+p_i \leq r_j,$$
$$q_i \geq p_j+q_j > q_j.$$

It follows that the case in which all q_i are equal is again solved by ordering the jobs according to nondecreasing r_i. Such an ordering will respect all precedence constraints in view of the preceding argument.

If we apply this method to the inverse problem to solve the case in which all r_i are equal, the resulting algorithm can be interpreted as a special case of the general $n|1|prec|\sum f_i$ algorithm from [Lawler 1973].

Similarly, if $p_i = 1$ for all J_i, algorithm JR will produce a schedule respecting all precedence constraints.

In the case of the general $n|1|prec,r_i\geq 0|L'_{max}$ problem, however, the precedence constraints are not respected automatically. Consider the $5|1|\{J_4<J_2\},r_i\geq 0|L'_{max}$ example specified by the data in Table 10.1 (*cf.* [Lenstra & Rinnooy Kan 1973]); note that $r_4+p_4 \leq r_2$ and $q_4 \geq p_2+q_2$. If the precedence constraint $J_4 < J_2$ is ignored, the unique optimal $5|1|r_i\geq 0|L'_{max}$

TABLE 10.1. DATA FOR THE EXAMPLE

i	1	2	3	4	5
r_i	0	2	3	0	7
p_i	2	1	2	2	2
q_i	5	2	6	3	2

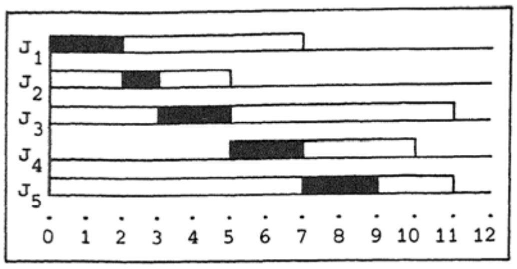


Figure 10.1
Schedule for the example.

schedule is given by (J_1,J_2,J_3,J_4,J_5) with value $L'_{max} = 11$ (*cf.* Figure 10.1). Explicit inclusion of this constraint leads to $L'_{max} = 12$.

10.2.2. *The algorithm of Baker and Su*

The branch-and-bound algorithm to be discussed now has been presented in [Baker & Su 1974] for the problem without precedence constraints. It will be referred to as algorithm BS.

The enumeration scheme is defined by algorithm AS1 below. Algorithm AS1 generates all *active schedules, i.e.* schedules where we cannot decrease the starting time of an operation without increasing the starting time of at least one other one.

```
procedure algorithm AS1 (n,r,p,C);
begin   local i;

        procedure node(S,t);
        if S = ∅ then comment an active schedule has been generated else
        begin   local Q;
                $Q:= \{j|j \in S,\ r_j < \min\{\max\{t,r_k\}+p_k|k \in S\}\}$;
                while Q ≠ ∅ do
                begin   $i:\in Q;\ Q:= Q-\{i\}$;
                        $C_i:= \max\{t,r_i\}+p_i$;
                        $node(S-\{i\},C_i)$
                end
        end;

        node({1,...,n},0)
end algorithm AS1.
```

At the ℓ-th level of the recursion, jobs are scheduled in the ℓ-th position. If the first assignment to Q is replaced by Q:= S, all n! schedules are generated. By means of the current assignment, only active schedules are generated; if $r_j \geq \max\{t,r_k\}+p_k$ for some $j,k \in S$, J_j is no candidate for the next position in the partial schedule since it can be preceded by J_k without postponement of C_j.

The bounding rule is based on the observation that the value of an optimal schedule will not increase if we allow job splitting. A lower bound on all possible completions of a partial schedule $(J_{\pi(1)},\ldots,J_{\pi(\ell)})$ is produced by the use of algorithm JR to schedule the remaining jobs from $C_{\pi(\ell)}$ onwards while allowing job splitting. If no job splitting occurs, this particular completion is an optimal one, and the value of the complete solution is an upper bound on the value of an optimal solution. A partial schedule can be eliminated if its lower bound is not smaller than the global upper bound.

The branch-and-bound algorithm is now completely defined if we specify a search strategy indicating which partial schedule will be chosen for further examination. The strategy used in [Baker & Su 1974] selects a partial schedule with minimum lower bound. We implemented the recursive scheme of algorithm AS1, selecting the unscheduled jobs in the order in which they appear in the solution, produced by algorithm JR. Experiments in which these descendant nodes were chosen in order of nondecreasing lower bounds showed a 50 to 60 per cent increase in solution time. Note that these three strategies correspond to the procedures "bb jumptrack", "bb backtrack1" and "bb backtrack2" respectively (see Chapter 8).

The above algorithm can easily be adjusted to take *precedence constraints* into account. As noted previously, they are automatically respected during the lower bound calculation and the only necessary change is a replacement of the first assignment to Q by

$$Q:= \{j \mid j \in S,\ B_j \cap S = \emptyset,\ r_j < \min\{\max\{t,r_k\}+p_k \mid k \in S,\ B_k \cap S = \emptyset\}\}.$$

Algorithm BS is fairly straightforward and its general principles can be extended to other NP-complete sequencing problems with non-equal release dates.

10.2.3. *The algorithm of McMahon and Florian*

A more sophisticated branch-and-bound algorithm for the problem without precedence constraints has been described in [McMahon & Florian 1975]. Algorithm MF is based on algorithm LS below, a heuristic method suggested in [Schrage 1971] for generating a good solution.

```
procedure algorithm LS (n,r,p,q,C);
begin   local S,Q,t,i;
        S:= {1,...,n}; t:= 0;
        while $S \neq \emptyset$ do
        begin   t:= $\max\{t,\min\{r_j | j \in S\}\}$;
                Q:= $\{j | j \in S,\ r_j \le t\}$;
                i:$\in$ $\{j | j \in Q,\ p_j = \max\{p_k | k \in Q,\ q_k = \max\{q_\ell | \ell \in Q\}\}\}$;
                S:= S-{i};
                $C_i$:= t:= $t+p_i$
        end
end algorithm LS.
```

The schedule $(J_{\pi(1)},\dots,J_{\pi(n)})$ produced by algorithm LS can be decomposed into blocks. $J_{\pi(h)}$ is the *last job in a block* if $C_{\pi(h)} \le r_{\pi(i)}$ for $i = h+1,\dots,n$, *i.e.*, if no job is delayed when $J_{\pi(h)}$ is completed. A set of jobs $\{J_{\pi(g)},\dots,J_{\pi(h)}\}$ forms a *block* if

(*a*) $g = 1$ or $J_{\pi(g-1)}$ is the last job in a block, and

(*b*) $J_{\pi(i)}$ is not the last job in a block, for $i = g,\dots,h-1$, and

(*c*) $J_{\pi(h)}$ is the last job in a block.

It follows that $J_{\pi(g)}$ is the *first job in a block* if $S_{\pi(g)} = r_{\pi(g)} \le r_{\pi(i)}$ for $i = g+1,\dots,n$.

With respect to J_i in block $\{J_{\pi(g)},\dots,J_{\pi(h)}\}$, we define $P_i = \{j | S_{\pi(g)} \le S_j \le S_i\}$, $q_i^* = \min\{q_j | j \in P_i\}$ and $P_i^* = \{j | j \in P_i,\ q_j = q_i^*\}$. We claim that lower bounds on the value of an optimal schedule are given by

$$LB_i' = r_i+p_i+q_i,$$

$$LB_i'' = \begin{cases} C_i+q_i^* & \text{if } i \in P_i^*, \\ C_i+q_i^*+1 & \text{if } i \notin P_i^*. \end{cases}$$

LB_i' requires no comment, but the justification of LB_i'' is actually rather subtle. Defining C_{ji} as the minimum completion time of J_j if this job would be scheduled as the last one of $\{J_k | k \in P_i\}$, we note that $C_{ji} \ge C_{ii} = C_i$ for all $j \in P_i$. A valid lower bound is now given by

$\min\{C_{ji}+q_j \mid j \in P_i\}$.

In the case that $i \in P_i^*$, it is obvious that for all $j \in P_i$

$$C_{ji}+q_j \geq C_{ii}+q_i = C_i+q_i^*. \tag{10.1}$$

Suppose next that $i \notin P_i^*$. If $j \notin P_i^*$, we have

$$C_{ji}+q_j \geq C_i+q_i^*+1. \tag{10.2}$$

Consider finally the case that $i \notin P_i^*$ and $j \in P_i^*$. If we move J_j to the last position of $\{J_k \mid k \in P_i\}$, a gap of at least one unit idle time is unavoidable, unless a J_k with $r_k \leq S_j < S_k$ can be moved forward to start at S_j. From algorithm LS we know that, if such a job exists, then $k \in P_i^*$ and $p_k \leq p_j$. Thus, a gap now threatens to occur between S_k and S_k+1. Repeating this argument as often as necessary, we conclude that $C_{ji} \geq C_i+1$, and therefore

$$C_{ji}+q_j \geq C_i+q_i^*+1. \tag{10.3}$$

Inequalities (10.1), (10.2) and (10.3) establish the validity of LB_i''.

At every node of the search tree, application of algorithm LS yields a complete solution $(J_{\pi(1)},\ldots,J_{\pi(n)})$ with value $L'_{\max}$ and a lower bound $LB = \max_i\{\max\{LB_i', LB_i''\}\}$. We may adjust the upper bound UB on the value of an optimal solution by setting $UB := \min\{UB, L'_{\max}\}$. If $LB \geq UB$, the node is eliminated; else, we apply the branching rule described next.

Let the *critical job* J_i be defined as the first job in the schedule with $C_i+q_i = L'_{\max}$. The schedule can only be improved of C_i can somehow be reduced. The set of solutions corresponding to the current node can now be partitioned into disjoint subsets, each characterized by a particular J_j which is to be scheduled last of $\{J_k \mid k \in P_i\}$. However, jobs J_j with $j \in P_i$, $q_j \geq q_i-L'_{\max}+UB$ need not to be considered, since in that case $C_{ji}+q_j \geq C_i+q_i-L'_{\max}+UB = UB$. Therefore, only for each J_j with $j \in P_i$, $q_j < q_i-L'_{\max}+UB$ a descendant node is actually created.

We can effectively implement the precedence constraints $\{J_k < J_j \mid k \in P_i-\{j\}\}$ by adjusting r_j and q_k $(k \in P_i-\{j\})$ as described in Section 10.2.1. During the next application of algorithm LS, J_j will then be scheduled last of $\{J_k \mid k \in P_i\}$. To maintain disjointness at deeper levels of the tree, we would have to update r_k and q_k for $k \notin P_i$ as well in view of previous choices. This would lead to the time consuming administration of a continually changing precedence graph. Dropping the requirement of disjoint descendants, we will force J_j to follow the critical J_i rather than the whole set $\{J_k \mid k \in P_i-\{j\}\}$. This can be done by putting r_j equal to any lower bound on $C_{ji}-p_j$ not less than r_i, such as $\max\{r_k+p_k \mid k \in P_i-\{j\}\}$, C_i-p_j, or simply r_i, as in [McMahon & Florian 1975]. Computational experiments have shown that the choice of a specific new r_j has only a minor influence on the performance of the algorithm; in our implementation, we put $r_j := \max\{r_i+p_i, C_i-p_j\}$.

The search strategy used in [McMahon & Florian 1975] is of the jump-track type, selecting a node with minimum lower bound. Again, our implementation is of the recursive backtrack type, choosing the descendant nodes in the reverse of the order in which the corresponding jobs J_j appear in the solution produced by algorithm LS.

Algorithm MF is easily adapted to deal with given *precedence constraints*. Since we may assume that $r_i < r_j$ and $q_i > q_j$ if $J_i < J_j$, they are respected by algorithm LS. Obviously, the lower bound remains a valid one. With respect to the branching rule, descendant nodes have to be created only for jobs J_j with $j \in P_i$, $q_j < q_i - L'_{\max} + UB$, $A_j \cap P_i = \emptyset$. We could branch by adding the precedence constraints $\{J_k < J_j | k \in P_i - \{j\}\}$; many heads and tails would then have to be adjusted. If, however, we drop the requirement of disjoint descendants and aim to preserve only the original precedence constraints, we may just as well restrict ourselves to adjust r_j in the way described above and update r_k for all $k \in A_j$. Since the tails still reflect the original precedence constraints, new solutions produced by algorithms LS will respect those constraints. Again, more extensive adjustments turn out to result in additional computing time.

10.3. Computational experience

10.3.1. *Test problems*

For each test problem with n jobs, 3n integer data r_i, p_i, q_i were generated from uniform distributions between 1 and r_*, p_* and q_* respectively. Here, $r_* = R.p_*$ and $q_* = Q.p_*$. In the precedence graph, each arc (J_i, J_j) with $i < j$ was included with probability P. Table 10.2 shows the values of (n, p_*, R, Q, P) during our experiments; the values used in previously reported tests are also given. For each combination of values with $R \le Q$ five problems were generated; inversion of these problems provided test problems with $R \ge Q$ (*cf.* Section 10.1). Significant and systematic differences between the solution times of a problem and its inverse would indicate advantages to be gained from problem inversion.

TABLE 10.2. VALUES OF PARAMETERS OF TEST PROBLEMS

parameter	[Baker & Su 1974]	[McMahon & Florian 1975]	*h.l.*
n	10,20,30	20,50	20,40,80
p_*	2000/n	25	50
R	.5n	.5n,2n	.5,2,.5n,2n
Q	.75n,.875n,n †	.4,1,3	.5,2,.5n,2n
P	0	0	0,.05,.15,.45

† In this case, the q_i are not distributed uniformly.

10.3.2. *Results*

Algorithms BS and MF were coded in ALGOL 60 and run on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam.

Tables 10.3 and 10.4 show the computational results for problems without precedence constraints, *i.e.* with $P = 0$. Algorithm BS solves 294 out of 300 problems with up to 80 jobs within the time limit of ten seconds. The limit is never exceeded for problems of the type for which the method has been tested in [Baker & Su 1974]. Inspection of the results revealed no obvious rule according to which problem inversion might be advantageous and this additional feature was therefore not incorporated into algorithm BS.

Even better results were obtained with algorithm MF. It turns out that this method has been tested in [McMahon & Florian 1975] on the very easiest types of problems. In general, algorithm MF performs especially well on problems with $R > Q$. Accordingly, we also tested algorithm FM, which inverts a problem if $\max_i\{r_i\}-\min_i\{r_i\} < \max_i\{q_i\}-\min_i\{q_i\}$ before applying algorithm MF. The remarkable quality of algorithm FM is clear from Tables 10.3 and 10.4.

Table 10.5 shows the effect of precedence constraints, which was investigated only with respect to algorithms MF and FM. For problems with $P \geq .15$, most of the solution time is spent on adjusting the r_i and q_i in accordance with the precedence constraints, as described in Section 10.2.1; this takes .06 seconds for $n = 20$, $P = .15$ and .70 for $n = 80$, $P = .45$. For each positive value of P which we tested, the median number of generated nodes is equal to one; for $P = .45$ branching never occurs. Inversion according to the rule given above leads to some improvement, albeit not so spectacular as in the case without precedence constraints.

TABLE 10.3. SOLUTION TIMES FOR P = 0: A SURVEY

n	P	median			maximum		
		alg.BS	alg.MF	alg.FM	alg.BS	alg.MF	alg.FM
20	0	.05	.02	.03	>10:2	.99	.11
40	0	.09	.06	.06	1.09	>10:1	.17
80	0	.23	.16	.15	>10:4	>10:3	.57

TABLE 10.4. MAXIMUM SOLUTION TIMES FOR P = 0: THE INFLUENCE OF R AND Q

n = 80	algorithm BS				algorithm MF				algorithm FM			
R↓ Q→	.5	2	.5n	2n	.5	2	.5n	2n	.5	2	.5n	2n
.5	.26 / .25	.25	5.54	5.75	.19 / .19	.21	1.64	>10:1	.19 / .25	.25	.15	.14
2	.24	.25 / .27	>10:1	4.84	.19	.18 / .17	>10:1	>10:1	.20	.22 / .25	.19	.16
.5n	>10:1	>10:2	3.43 / 3.60	3.67	.10	.12	.33 / .52	.47	.08	.11	.57 / .49	.17
2n	.10	.11	2.51	2.54 / 2.55	.09	.07	.13	.11 / .13	.09	.08	.12	.17 / .19

TABLE 10.5. SOLUTION TIMES: THE INFLUENCE OF P

n	P	median		maximum	
		alg.MF	alg.FM	alg.MF	alg.FM
20	0	.02	.03	.99	.11
	.05	.06	.05	.41	.43
	.15	.07	.07	.14	.15
	.45	.07	.08	.12	.11
80	0	.16	.15	>10:3	.57
	.05	.36	.33	>10:6	>10:4
	.15	.47	.42	.85	.57
	.45	.73	.75	.81	.80

LEGEND TO TABLES 10.3,4,5

Each entry in Table 10.3 (Tables 10.4,5) represents 100 (5,100) test problems.
solution times : CPU seconds on a Control Data Cyber 73-28.
$>\ell:k$: the time limit ℓ is exceeded k times.
algorithm BS : see Section 10.2.2.
algorithm MF : see Section 10.2.3.
algorithm FM : algorithm MF with problem inversion if
$\max_i\{r_i\}-\min_i\{r_i\} < \max_i\{q_i\}-\min_i\{q_i\}$.
n : number of jobs.
R : relative range of r_i.
Q : relative range of q_i.
P : expected density of precedence graph.

10.3.3. *Misusing problem reductions*

Let us consider a particular instance of the KNAPSACK problem (see Theorem 2.1(*h*)), defined by $a_i = 90+2i$ $(i = 1,\ldots,9)$, $b = 401$. Clearly, this KNAPSACK problem has no solution.

We applied improved versions of two well-known knapsack optimization algorithms (see [Lageweg & Lenstra 1972]) to this problem. Moreover, we transformed it into two types of machine scheduling problems, according to the reductions given in Chapter 4 and applied three algorithms which are described in the present and following chapter. The results are presented in Table 10.6.

TABLE 10.6. RESULTS FOR A DIFFICULT SCHEDULING PROBLEM

problem formulation	solution method	solution time in seconds on CDC 73-28	number of nodes in search tree
$\max\{\sum a_ix_i \mid \sum a_ix_i \le b, x_i\in\{0,1\}\}$	dynamic programming [Hu 1969]	.26	-
	branch-and-bound [Kolesar 1967]	.09	178
$n\|1\|r_n\ge 0\|L_{max}$ (see Theorem 4.4(*d*))	algorithm BS (see Section 10.2.2)	69.05	14121
	algorithm FM (see Sections 10.2.3, 10.3.2)	8.63	1254
$n\|1\|\|\sum w_iT_i$ (see Theorem 4.4(*i*))	algorithm NA (see Section 11.2)	>300	>97214

10.4. Remarks

The computational experience reported in Section 10.3 leads us to conclude that the $n|1|prec,r_i \geq 0|L_{max}$ problem can be satisfactorily solved by the algorithms described in Sections 10.2.2 and 10.2.3. If solution by implicit enumeration is indeed unavoidable, there seems to be little room for further improvement.

This is a hopeful result, especially in view of the wide applicability of this scheduling model. In Sections 12.2.4 and 13.2.1 the problem arises in the theoretical context of computing lower bounds for flow-shop and job-shop problems. In Chapter 15 we describe a practical scheduling situation in which a processing order on a critical machine is obtained by solving a problem of this type.

It might be worth-while investigating if the ideas underlying algorithms BS and MF could be applied to other machine scheduling problems. An interesting candidate is the $n|2|F,r_i \geq 0|C_{max}$ problem. This problem can be interpreted as a variation on the three-machine model introduced in Section 10.1: a non-bottleneck machine M_1 deals with the release dates and two bottleneck machines M_2 and M_3 constitute the flow-shop. Again, the case in which all r_i are equal can be solved in $O(n \log n)$ steps [Johnson 1954], whereas the general problem is NP-complete (see Chapter 4). Similar remarks apply to the inverse $n|2|F|L_{max}$ problem.

11. ONE-MACHINE SCHEDULING II: MINIMIZING TOTAL COSTS

11.1. Introduction

In this chapter we examine the general $n|1|\mathit{prec}|\sum f_i$ problem. It can be formulated as follows.

> Each of n jobs $J_1,\ldots,J_n$ has to be processed on a single machine which can handle only one job at a time. Job J_i $(i = 1,\ldots,n)$ is available for processing at time $t = 0$ and requires an uninterrupted processing time p_i; costs $f_i(t)$, nondecreasing in t, are incurred if J_i is completed at time t. Given precedence constraints "$J_i < J_j$" indicate that J_j cannot start before the completion of J_i; we use the notations $B_i = \{j|J_j < J_i\}$ and $A_i = \{j|J_i < J_j\}$. We seek to find a processing order with associated completion times C_i $(i = 1,\ldots,n)$ that minimizes the total costs $\sum_{i=1}^n f_i(C_i)$.

Complexity results for various special cases of this problem have been presented in Chapter 4. For instance, there exist $O(n \log n)$ algorithms for the $n|1|\mathit{tree}|\sum w_iC_i$ problem [Horn 1972; Sidney 1975] and for the $n|1||\sum U_i$ problem [Moore 1968]; the $n|1|\mathit{prec}|\sum C_i$, $n|1||\sum w_iT_i$ and $n|1||\sum w_iU_i$ problems have been proved NP-complete. However, the complexity of the $n|1||\sum T_i$ problem remains an open question.

Altogether, it is not surprising that all methods for the general $n|1||\sum f_i$ problem developed so far are based on implicit enumeration. Apart from the work on quadratic and general cost functions in [Schild & Fredman 1962] and dynamic programming formulations for the general criterion in [Held & Karp 1962; Lawler 1964], most researchers have concentrated on the weighted tardiness function $f_i(t) = w_i\max\{0,t-d_i\}$; w_i and d_i stand for weight and due date of J_i respectively.

Especially with respect to the $n|1||\sum T_i$ problem, many elimination criteria have been developed that lead to precedence constraints respected by at least one optimal schedule. These criteria have to be incorporated in some enumeration scheme and combined with a bounding mechanism to yield an enumerative algorithm. For instance, the elimination criteria from [Emmons 1969] were successfully implemented in a dynamic programming algorithm [Srinivasan 1971] that turned out to be superior to other $n|1||\sum T_i$ algorithms surveyed in [Baker & Martin 1974]. For more general cost functions no really powerful elimination criteria have been found so far. A branch-and-bound algorithm for the $n|1||\sum w_iT_i$ problem, using a few simple elimination criteria, was developed in [Shwimer 1972] (see Section 11.3.1). In general, the performance of branch-and-bound algorithms for these types of problems has been rather disappointing. This may be explained by the fact that only one or two jobs out of a subset of jobs on the processing costs of which a lower bound was sought actually contributed to this bound.

In Section 11.2 we shall describe a new and general algorithm for the $n|1|prec|\sum f_i$ problem, incorporating elimination criteria which imply all criteria developed so far for the $n|1||\sum T_i$ and $n|1||\sum w_iT_i$ problems, and a lower bound which at least does not suffer from the defect mentioned above. In Section 11.3 we report on the algorithm's performance on the $n|1||\sum w_iT_i$ problem; our method is compared to Shwimer's algorithm and to a simple brute force approach. Section 11.4 contains concluding remarks.

We will use the notation $P(Q) = \sum_{i\in Q} p_i$ for any $Q \subset \{1,\ldots,n\}$.

11.2. A new algorithm

11.2.1. *Enumeration scheme*

The enumeration scheme generates all feasible schedules according to algorithm BF below. Algorithm BF fills a schedule from back to front. This is possible because there obviously exists an optimal solution without machine idle time; the total time needed to process a set of jobs is therefore independent of the processing order.

```
procedure algorithm BF (n,p,A,C);
begin   local i;

        procedure node(S,t);
        if S = ∅ then comment a feasible schedule has been generated else
        begin   local Q;
                Q:= {j|j ∈ S, S∩A_j = ∅};
                while Q ≠ ∅ do
                begin   i:∈ Q; Q:= Q-{i};
                        C_i:= t;
                        node(S-{i},t-p_i)
                end
        end;

        node({1,...,n},sum{p_i|i ∈ {1,...,n}})
end algorithm BF.
```

Each node in the search tree is characterized by a set $\{J_i | i \in S\}$ of unscheduled jobs, which have to be processed from 0 to $P(S) = \sum_{i \in S} p_i$; $\bar{S} = \{1,\ldots,n\}-S$ will denote the index set of the jobs which have been scheduled from $P(S)$ to $P(\{1,\ldots,n\})$. We enter a descendant node by scheduling a J_i with $i \in S$, $S \cap A_i = \emptyset$ from $P(S)-p_i$ to $P(S)$.

11.2.2. *Elimination criteria*

At each node we can apply the elimination criteria which are to be presented in this section. Throughout, our *theorems* hold for the general $n|1||\sum f_i$ problem; implications for the special case of the $n|1||\sum w_i T_i$ problem are formulated as *corollaries*.

Any relation $J_i < J_j$ which is established by previous application of elimination criteria implies that $i \in B_j$ and $j \in A_i$. We will restrict ourselves to schedules satisfying these precedence constraints.

THEOREM 11.1. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

(*a*) $f_i(t)-f_j(t)$ *is nondecreasing in* t *on the interval* $(P(B_j)+p_j, P(S-A_i))$, *and*

(*b*) $p_i \le p_j$.

Proof. Consider any schedule in which J_j precedes J_i. Denote by D the starting time of J_j and by E the completion time of J_i. Compare this schedule with the schedule obtained by interchanging J_j and J_i (*cf*. Figure 11.1). The contribution to total costs by all jobs except J_j does not increase, because of condition (*b*). As to J_j, it follows from

$$P(B_j)+p_j \le D+p_j \le E \le P(S-A_i)$$

and condition (*a*) that

$$f_i(E)-f_j(E) \ge f_i(D+p_j)-f_j(D+p_j). \qquad (11.1)$$

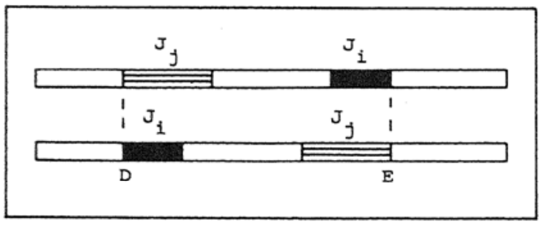


Figure 11.1

Because of condition (*b*), we have

$$f_i(D+p_j) \geq f_i(D+p_i). \qquad (11.2)$$

Together, (11.1) and (11.2) imply

$$f_i(E)+f_j(D+p_j) \geq f_i(D+p_i)+f_j(E),$$

which means that the joint contribution of J_i and J_j to total costs also does not increase. □

COROLLARY 11.1. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

(*a*) $d_i \leq \max\{d_j, P(B_j)+p_j\}$,

(*b*) $w_i \geq w_j$, *and*

(*c*) $p_i \leq p_j$.

Proof. If $d_i \leq d_j$, then condition (*b*) implies that $f_i(t)-f_j(t)$ is nondecreasing on the interval $(0,P(S))$, and we can apply Theorem 11.1 with $B_j = A_i = \emptyset$ (*cf.* Figure 11.2(*a*)). If $d_i \leq P(B_j)+p_j$, then $f_i(t)$ is increasing for $t > P(B_j)+p_j$, and it follows from condition (*b*) that $f_i(t)-f_j(t)$ is nondecreasing on the interval as required in Theorem 11.1 (*cf.* Figure 11.2(*b*)).□

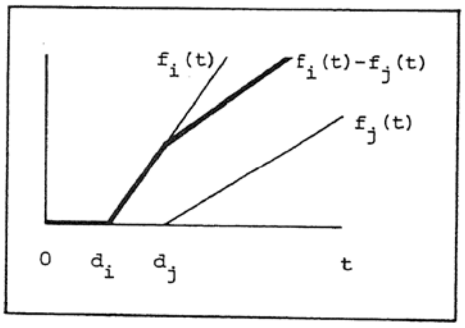


Figure 11.2(*a*)

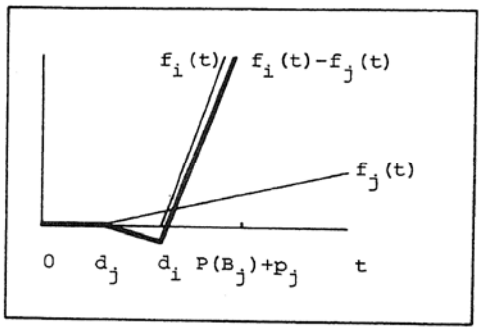


Figure 11.2(*b*)

THEOREM 11.2. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

(*a*) $f_j(P(B_j)+p_j) = f_j(P(S-A_i)-p_j)$, *and*

(*b*) $f_i(t)-f_j(t)$ *is nondecreasing in* t *on the interval* $(P(S-A_i)-p_j, P(S-A_i))$.

Proof. Clearly, conditions (*a*) and (*b*) imply that $f_i(t)-f_j(t)$ is nondecreasing on the interval $(P(B_j)+p_j, P(S-A_i))$, so in the case that $p_i \leq p_j$ we can apply Theorem 11.1. Suppose now that $p_i > p_j$. Again, consider any schedule in which J_j precedes J_i. Denote by D the starting time of J_j and by E the

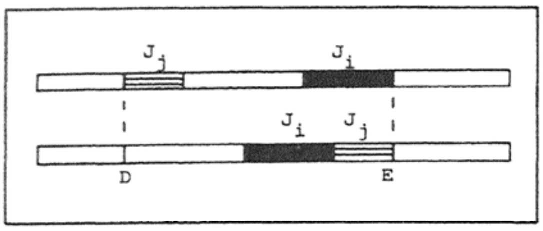


Figure 11.3

completion time of J_i. Compare this schedule with the schedule obtained by putting J_j directly after J_i (*cf.* Figure 11.3). The contribution to total costs by all jobs except J_j does not increase. As to J_j, it follows from

$$P(B_j)+p_j \le D+p_j \le E-p_j \le P(S-A_i)-p_j$$

and condition (*a*) that

$$f_j(E-p_j) = f_j(D+p_j). \tag{11.3}$$

Because of condition (*b*), we have

$$f_i(E)-f_j(E) \ge f_i(E-p_j)-f_j(E-p_j). \tag{11.4}$$

Together, (11.3) and (11.4) imply

$$f_i(E)+f_j(D+p_j) \ge f_i(E-p_j)+f_j(E),$$

which means that the joint contribution of J_i and J_j to total costs also does not increase. □

COROLLARY 11.2. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

(*a*) $d_j \ge P(S-A_i)-p_j$,

(*b*) $d_i \le d_j$, *and*

(*c*) $w_i \ge w_j$.

Proof. Condition (*a*) implies that $f_j(P(B_j)+p_j) = f_j(P(S-A_i)-p_j)$, and it follows from conditions (*b*) and (*c*) that $f_i(t)-f_j(t)$ is nondecreasing on the interval $(0,P(S))$ (*cf.* Figure 11.2(*a*)). □

THEOREM 11.3. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

$$f_j(P(B_j)+p_j) = f_j(P(S-A_i)). \tag{11.5}$$

Proof. We can apply Theorem 11.2, since its conditions (*a,b*) follow from (11.5). □

COROLLARY 11.3. *At least one optimal schedule has* J_i *preceding* J_j $(i,j \in S)$ *if*

$$d_j \geq P(S-A_i).$$

THEOREM 11.4. *In at least one optimal schedule* J_j $(j \in S)$ *comes last among* $\{J_i | i \in S\}$ *if*

$$f_j(p_j) = f_j(P(S)). \tag{11.6}$$

Proof. Since (11.6) implies (11.5) for all $i \in S$, we can apply Theorem 11.3 to each pair (J_i,J_j) with $i \in S-\{j\}$. □

COROLLARY 11.4. *In at least one optimal schedule* J_j $(j \in S)$ *comes last among* $\{J_i | i \in S\}$ *if*

$$d_j \geq P(S).$$

Corollary 11.1 is given in [Shwimer 1972]. Corollaries 11.1, 11.2 and 11.3 are extended versions of Theorems 1, 2 and 3 in [Emmons 1969]. Our proofs, however, are considerably simpler than the original ones. Corollary 11.4 can be found in [Elmaghraby 1968].

11.2.3. *Implementation of the elimination criteria*

The only problem arising with the implementation of the elimination criteria in an $n|1|prec|\sum f_i$ algorithm is the possible creation of precedence cycles; it is perfectly imaginable that two theorems lead to incompatible conditions. The nature of our elimination criteria, however, is such that applying them successively, while guarding against precedence cycles, will always lead to a collection of schedules containing at least one optimal one. We avoid the creation of precedence cycles by immediately constructing the transitive closure of the set of known precedence constraints whenever we find a new relation $J_i < J_j$:

$$A_h := A_h \cup \{j\} \cup A_j \quad \text{for every } h \in \{i\} \cup B_i;$$

$$B_k := B_k \cup \{i\} \cup B_i \quad \text{for every } k \in \{j\} \cup A_j.$$

Furthermore, if we restrict ourselves to examining pairs (k,h) between which no relation has been found so far, we can never create a precedence cycle. For if we found that $J_k < J_h$ and it then turned out that $i \in B_j$ for some $i \in A_h$, $j \in B_k$, then we would have set $k \in A_h$, $h \in B_k$ in a previous stage and therefore would not have examined this pair again.

In the case of general cost functions we can apply Theorems 11.1 to 11.4 at every node; the set S decreases and the sets A_i and B_i increase in size as we progress through the search tree. In the case of weighted tardiness functions we apply Corollary 11.4 at every node, whereas Corollaries 11.1, 11.2 and 11.3 are used only at the root node with $S = \{1,\ldots,n\}$. In principle, all corollaries could be applied at every node, but the advantages of doing so are in this case outweighed by the disadvantages of complicated and time-consuming bookkeeping.

Corollaries 11.1, 11.3 and 11.2 are now implemented by running through them in this order (keeping in mind the above remarks) and repeating this process until no further improvements are possible. If after this process the earliest possible completion time $P(B_j)+p_j$ of J_j is larger than its due date, then we can set $d_j := P(B_j)+p_j$, thereby incurring costs $w_j(P(B_j)+p_j-d_j)$ and increasing the chances of successful application of Corollary 11.4. The latter corollary is checked at every node. To avoid contradictions with the precedence constraints found previously by the other corollaries, we restrict ourselves to the set $\{J_j | j \in S,\ S \cap A_j = \emptyset\}$.

11.2.4. *Lower bound*

The lower bound LB on the value of all possible schedules at a node has the form

$$LB = F(\bar{S})+LB^*.$$

Here $F(\bar{S})$ denotes the known total costs incurred by the set $\{J_i | i \in \bar{S}\}$ of scheduled jobs, and LB^* is a lower bound on the total costs of scheduling the set $\{J_i | i \in S\}$ of remaining jobs from 0 to $P(S)$.

To compute LB^*, we put $s = |S|$ and renumber the jobs in $\{J_i | i \in S\}$ from 1 up to s. Our lower bound is now based on the observation that, if $p_1 = \ldots = p_s = p_*$, the costs f_{ij} of putting J_i in the j-th position are given by

$$f_{ij} = f_i(jp_*),$$

and an optimal schedule corresponds to a solution to the following *linear assignment problem*:

$$\min_\pi\{\textstyle\sum_{j=1}^{s} f_{\pi(j)j}\}, \tag{11.7}$$

where π runs over all permutations of $\{1,\ldots,s\}$ (*cf.* [Lawler 1964]).

If not all p_i are equal, the above idea can be used to compute lower bounds in two ways.

Assuming all p_i are integers, we can find their greatest common divisor g and treat each J_i as a sequence of p_i/g new jobs of equal length g. Problem (11.7) now becomes a $(P(S)/g)\times s$ *linear transportation problem*, that produces a lower bound if we succeed in defining appropriate cost coefficients f_{ij}. For the case that $f_i(t) = w_i'\max\{0,t-d_i\}+w_i''t$, suitable cost coefficients have been developed in [Gelders & Kleindorfer 1974; Gelders & Kleindorfer 1975]. Three problems remain with the transportation approach.

(1) In the optimal solution to the transportation problem job splitting can occur.

(2) Usually g is equal to 1 and the size of the transportation problem tends to be very large. It then becomes practically impossible to solve this problem at every node.

(3) It seems to be difficult to define effective cost coefficients for general cost functions.

For the above reasons, we prefer a different approach, which basically involves redefining the cost coefficients for the $s\times s$ linear assignment problem so that f_{ij} becomes a lower bound on the costs of putting J_i in the j-th position. To accomplish this, we compute the earliest possible completion time C_{ij} of J_i if we put J_i in the j-th position and if we observe the precedence constraints, given by B_i and A_i. Using the notation

$$R_i(q) = \min_Q\{P(Q)\,|\,Q \subset \{1,\ldots,s\}-(B_i\cup\{i\}\cup A_i),\ |Q| = q\},$$

we have

$$C_{ij} = P(B_i)+p_i+R_i(j-|B_i|-1) \quad \text{for } |B_i| < j \le |\{1,\ldots,s\}-A_i|,$$

as can be easily checked. Redefining

$$f_{ij} = \begin{cases} f_i(C_{ij}) & \text{for } |B_i| < j \le |\{1,\ldots,s\}-A_i|, \\ \infty & \text{otherwise,} \end{cases}$$

and using these cost coefficients in problem (11.7) now gives the desired lower bound LB^*. This is easily proved as follows. If an optimal schedule for our original problem is given by a permutation π^* with minimum costs F^*, then we have

$$F^* \ge \textstyle\sum_{j=1}^{s} f_{\pi^*(j)j} \ge LB^*$$

since the C_{ij} underestimate the true completion times, the f_i are nondecreasing, and π^* is a feasible solution to problem (11.7).

11.2.5. *Implementation of the lower bound*

After the computation of the lower bound LB at the current node, the solution to (11.7) can also be evaluated as a schedule, which may lead to a decrease in the value UB of the best schedule found so far. If LB $\geq$ UB the node is eliminated. Otherwise, the jobs in the set $\{J_i \mid i \in S,\ S \cap A_i = \emptyset\}$ are candidates for the s-th position in the schedule. Choosing any of them leads to a new node in the search tree. Fortunately, we can do better than solving *ab initio* the assignment problem at each of these descendant nodes, by exploiting the solution to (11.7) at the current parent node. This problem can be reformulated as

$$\min\{\textstyle\sum_{i=1}^{s} \sum_{j=1}^{s} f_{ij}x_{ij} \mid \sum_{j=1}^{s} x_{ij} = 1 \ (i = 1,\dots,s),$$
$$\textstyle\sum_{i=1}^{s} x_{ij} = 1 \ (j = 1,\dots,s), \qquad (11.8)$$
$$x_{ij} \geq 0 \quad (i,j = 1,\dots,s)\},$$

where $x_{ij} = 1$ corresponds to $\pi(j) = i$ in (11.7). Its dual problem is given by

$$\max\{\textstyle\sum_{i=1}^{s} u_i + \sum_{j=1}^{s} v_j \mid u_i+v_j \leq f_{ij} \ (i,j = 1,\dots,s)\}.$$

An optimal solution to these problems has the value LB^* and is denoted by (x_{ij}^*) and (u_i^*,v_j^*), respectively.

At the parent node, we can with little computational effort obtain a lower bound LB_r on the value of all schedules whereby J_r occupies the s-th position. Observing that $(u_i^*,v_j^*)_{i\neq r, j\neq s}$ is a feasible dual solution to the assignment problem, obtained from (11.8) by deleting row r and column s, we define

$$\begin{aligned} LB_r &= F(\bar{S}\cup\{r\}) + \textstyle\sum_{i\neq r} u_i^* + \sum_{j\neq s} v_j^* \\ &= (F(\bar{S})+f_{rs}) + (LB^*-u_r^*-v_s^*) \\ &= LB + (f_{rs}-u_r^*-v_s^*) \geq LB. \end{aligned}$$

Clearly, any potential descendant node for which $LB_r \geq$ UB can be eliminated.

From the remaining candidates a J_r with minimal LB_r is scheduled in the s-th position, and we start to explore the corresponding descendant node. Application of the elimination criteria at this node may increase LB_r. For example, if in the case of weighted tardiness functions a J_j is scheduled in position s-1 by application of Corollary 11.4, then we have $f_{j,s-1} = 0$ and LB_r can be replaced by $LB_r-u_j^*-v_{s-1}^* \geq LB_r$. However, if this new LB_r does not lead to elimination of the node, we have to solve its assignment problem. Indexing the jobs as at the parent node and considering only indices that correspond to unscheduled jobs or unfilled positions, we can still profit from the optimal solution to the assignment problem at the parent node in the following ways.

(*a*) If we pass from the parent into the descendant node, the earliest completion times C_{ij} will not decrease, nor will the cost coefficients f_{ij}. So (u_i^*,v_j^*) provides a *feasible dual* solution to the new assignment problem.

(*b*) (x_{ij}^*) provides a *partial primal* solution to the new problem and can serve as a starting point for finding an optimal solution. (x_{ij}^*) and (u_i^*,v_j^*) can be made *complementary* by resetting $x_{ij}^* = 0$ if $u_i^*+v_j^* < f_{ij}$.

Remark (*a*) suggests an alternative bounding mechanism whereby the assignment problem is solved only at the root node and provides lower bounds throughout the whole search tree by means of sums of appropriate dual variables. (In fact, this idea has been implemented in [Gelders & Kleindorfer 1974; Gelders & Kleindorfer 1975] since it is not feasible to find a new optimal solution to their large transportation problem at every node again.) Although we obtained reasonable computational results with this approach, we preferred the stronger bound; even then the trees may become quite large for moderate size problems.

In selecting a method for solving the assignment problems, ideally we would like to have a fast algorithm, not requiring an initial basic solution and producing a sequence of nondecreasing feasible dual solutions each of which may lead to early elimination of the current node. The dual method from [Dorhout 1975] turned out to be more suitable than primal methods such as the stepping-stone algorithm or primal-dual ones such as the Hungarian method.

Dorhout's algorithm can be considered as a synthesis of ideas proposed in [Tomizawa 1971; Tabourier 1972]. Essentially, the algorithm works on a complete bipartite graph $G = (S\cup T,E)$ where the vertex sets S and T correspond to the sets of unscheduled jobs and unfilled positions respectively; edge $e_{ij} \in E$ $(i \in S,\ j \in T)$ has a weight $w_{ij} = f_{ij}-u_i-v_j$. The algorithm starts with a feasible dual solution (u_i,v_j) and a partial primal solution (x_{ij}), which is complementary to the dual one and defines a matching on G. The algorithm constructs the shortest augmenting path from any exposed vertex in S to the nearest exposed vertex in T, using the shortest path algorithm from [Dijkstra 1959]. The matching is then augmented and the dual solution is changed in such a way that its feasibility is maintained and complementarity is restored.

11.2.6. *Example*

Consider the $7|1||\sum w_i T_i$ problem specified by the data in Table 11.1.

TABLE 11.1. DATA FOR THE EXAMPLE

i	1	2	3	4	5	6	7
p_i	12	13	14	16	26	31	32
d_i	42	33	51	48	63	88	146
w_i	7	9	5	14	10	11	8

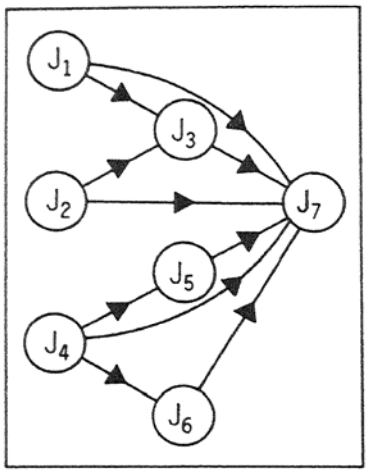


Figure 11.4
Precedence graph for the example.

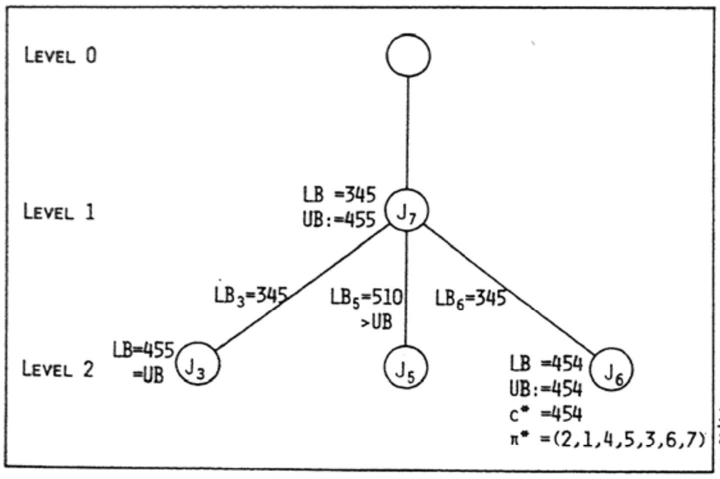


Figure 11.5
Search tree for the example.

Since $d_7 = 146 > 144 = P(\{1,\ldots,7\})$, Corollary 11.4 implies J_7 comes last. Further application of the elimination criteria (here only Corollary 11.1) leads to the precedence graph, given in Figure 11.4. Figure 11.5 represents the search tree. Because J_7 can be scheduled in the last position, there is only one node at the first level, where J_3, J_5 and J_6 are candidates for the sixth position. The assignment problem at this node can be found in Table 11.2; the cells in the optimal primal solution and the optimal values of the dual variables are printed in a different type face. We find $LB = f(\{7\})+LB^* = 0+345 = 345$; when evaluated as a schedule, the assignment solution $(J_2,J_1,J_4,J_5,J_6,J_3,J_7)$ has value $UB = 455$. Since $LB_5 = 345+(490-325) = 510 > UB$, only J_3 and J_6 with $LB_3 = LB_6 = LB$ remain candidates for the sixth position. The two assignment problems at the second level of the tree are given in Table 11.3. If J_3 is scheduled in the sixth position, we have $LB = f(\{3,7\})+LB^* = 305+150 = 455 = UB$, and this node can be eliminated. If J_6 is scheduled in the sixth position, we have $LB = f(\{6,7\})+LB^* = 264+190 = 454$; the schedule $(J_2,J_1,J_4,J_5,J_3,J_6,J_7)$ has value $UB = 454$ and hence must be optimal.

TABLE 11.2. ASSIGNMENT PROBLEM AT THE FIRST LEVEL

	1	2	3	4	5	6	u_i^*
J_1	0	0	0	175	392	∞	0
J_2	0	0	72	306	585	∞	0
J_3	∞	∞	0	20	150	305	-20
J_4	0	0	0	98	∞	∞	0
J_5	∞	0	0	40	180	490	0
J_6	∞	0	0	0	0	264	-61
v_j^*	0	0	0	40	61	325	

TABLE 11.3. ASSIGNMENT PROBLEMS AT THE SECOND LEVEL

	1	2	3	4	5	u_i^*		1	2	3	4	5	u_i^*
J_1	0	0	0	175	392	0	J_1	0	0	0	175	∞	0
J_2	0	0	72	306	585	0	J_2	0	0	72	306	∞	0
J_4	0	0	0	∞	∞	0	J_3	∞	∞	0	20	150	-20
J_5	∞	0	0	40	350	0	J_4	0	0	0	98	∞	0
J_6	∞	0	0	0	110	-41	J_5	∞	0	0	40	180	0
v_j^*	0	0	0	40	151		v_j^*	0	0	0	40	170	

11.3. Computational experience

11.3.1. *Compared algorithms*

Our general algorithm was tested on the case of weighted tardiness functions and compared to the algorithm from [Shwimer 1972] and a simple lexicographic algorithm. These three methods will be referred to as algorithms NA, JS and LE, respectively.

Algorithm JS has been designed specifically for the $n|1||\sum w_i T_i$ problem. The enumeration scheme is equivalent to ours. Shwimer applies only two elimination criteria, formulated here as Corollary 11.4 and the static part of Corollary 11.1 (*i.e.*, $d_i \leq d_j$). His lower bound tries to eliminate potential descendants in the parent node; instead of LB_r he uses

$$LB'_r = F(\bar{S}\cup\{r\})$$
$$+ \min_{i\in S-\{r\}}\left\{w_i \max\{0,P(S-\{r\})-d_i\} + \min_{h\in S-\{r,i\}}\{w_h\}\cdot T_{\max}(S-\{r,i\})\right\},$$

where $T_{\max}(Q)$ denotes the minimal maximal tardiness over all possible schedules of the set $\{J_h | h \in Q\}$, found by ordering these jobs according to nondecreasing d_h (*cf.* Section 10.2.1). It is clear that Shwimer's bound can be computed much more quickly than our bound LB, but that (loosely speaking) only two jobs contribute to its value. Moreover, Shwimer's bound depends explicitly on a property of the tardiness function. It is possible to solve the general $n|1||f_{\max}$ problem [Lawler 1973], but the number of operations then increases from $O(n \log n)$ to $O(n^2)$.

Algorithm LE is a straightforward extension of the lexicographic generator of permutations "pm lex", presented in Section 6.3. The method enumerates schedules according to algorithm BF (see Section 11.2.1), always choosing a J_r with maximal d_r from the remaining candidates in $\{J_i | i \in S\}$. Corollary 11.4 can then easily be applied; no other elimination criteria have been incorporated. Also a simple bounding mechanism is used, with

$$LB''_r = F(\bar{S}\cup\{r\}).$$

For a more general remark on the possible use of such a quick complete enumeration method, we refer to Section 11.4.

11.3.2. *Test problems*

We shall now describe in detail the way in which we generated random data on which to test these three methods. The reasons for this detailed approach will become apparent as we proceed.

Each $n|1||\sum w_i T_i$ problem is completely specified by n integer triples (p_i, d_i, w_i). We regard these triples as a three-dimensional sample from a joint distribution with density function $\phi(x,y,z)$.

In all our tests, the third random variable $\underline{w}$ is independent of $\underline{p}$ and $\underline{d}$ (see [Hemelrijk 1966]). We have

$$\phi(x,y,z) = \phi_{pd}(x,y)\phi_w(z),$$

where $\underline{w}$ is uniformly distributed over the interval (4.5,15.5).

In what follows, we shall introduce four parameters that determine $\phi_{pd}(x,y)$ and that we believed *a priori* to be of possible influence on any algorithm's performance. In fact, three of them are already mentioned as such in [Srinivasan 1971; Baker & Martin 1974]. These papers indicate that the choice of a particular function may have a strong influence on the performance of any tardiness algorithm in a way that may be characteristic for the algorithm in question.

The first parameter measures the *correlation between* $\underline{p}$ *and* $\underline{d}$, $\rho(\underline{p},\underline{d})$. It is intuitively plausible that there may be a significant difference between problems where longer jobs tend also to have later due dates, and problems where there is no correlation whatsoever. If all weights are equal, then a problem with perfect correlation can be trivially solved by ordering the jobs according no nondecreasing d_i [Emmons 1969]. To investigate the influence of correlation, we use two different kinds of functions $\phi_{pd}(x,y)$. Either

$$\phi_{pd}(x,y) = \phi_p(x)\phi_d(y),$$

in which case $\underline{p}$ and $\underline{d}$ are independent random variables and $\rho(p,d) = 0$, or

$$\phi_{pd}(x,y) = \phi_p(x)\phi_{d|p}(y|x),$$

in which case the due date generated depends explicitly on the processing time and $\rho(\underline{p},\underline{d})$ depends on the particular form of the density functions involved.

In both cases, $\underline{p}$ is normally distributed with expectation μ_p and variance σ_p^2. We arbitrarily fix $\mu_p = 100$. With regards to σ_p^2, however, we have to introduce as the second possibly significant parameter the *relative vari-*

ation of processing times $s = \sigma_p/\mu_p$. We introduce s because our lower bound will presumably be sharper when processing times differ relatively little, as will be obvious from Section 11.2.4. Hence, we may expect problems with small s to be relatively easy for our algorithm.

In the case of noncorrelated $\underline{p}$ and $\underline{d}$, $\underline{d}$ is uniformly distributed with expectation μ_d and variance $\sigma_d^2 = \lambda_d^2/12$, where λ_d denotes the length of the interval on which $\phi_d(y) > 0$.

We fix μ_d by introducing as a third parameter the *average tardiness factor* $t = 1-\mu_d/(n\mu_p)$. The value of t roughly indicates the average fraction of jobs that will be late [Baker & Martin 1974]. Problems with $t = 1$ or $t = 0$ tend to be easy - if all jobs are late, then ordering the jobs according to nonincreasing w_i/p_i produces an optimal schedule, and if we find by ordering the jobs according to nondecreasing d_i that no job is late, then clearly this schedule is optimal.

Finally, λ_d is fixed by the fourth parameter, the *relative range of due dates* $r = \lambda_d/(n\mu_p)$. Intuitively, a large r increases the number of times that Corollaries 11.1 and 11.2 can be applied, thereby speeding up computations.

In the case of correlated $\underline{p}$ and $\underline{d}$, $\underline{d}|\underline{p}=p$ is again uniformly distributed, with $\mu_{d|p}$ and $\lambda_{d|p}$ specified analogously by $t = 1-\mu_{d|p}/(np)$ and $r = \lambda_{d|p}/(np)$. Specific values of s, t and r determine the value of $\rho(\underline{p},\underline{d})$. We have

$$\rho(\underline{p},\underline{d}) = (1-t)/\sqrt{(1+1/s^2)r^2/12 + (1-t)^2}$$

as can be established by straightforward calculations.

Choosing for noncorrelated or correlated $\underline{p}$ and $\underline{d}$, and fixing s, t and r, we can generate n triples (p_i,d_i,w_i) to obtain a test problem. Each generated value is rounded off to the nearest integer, and if a negative d_i is generated, we reset $d_i := 0$, which implies adding a constant to $f_i(t)$ and therefore does not influence the final schedule.

11.3.3. *Results*

Algorithms NA, JS and LE were coded in ALGOL 60 and run on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam.

Tables 11.4 and 11.5 show the computational results. They are classified according to the value of the *average tardiness factor* t, this factor having a major influence on the performance of the algorithms. There is a signifi-

TABLE 11.4. SOLUTION TIMES

n	t	number of problems	median			maximum		
			alg.NA	alg.JS	alg.LE	alg.NA	alg.JS	alg.LE
10	.2	24	.1	.0	.0	.3	.1	.0
	.6	24	.6	.8	1.4	3.3	42.4	47.9
15	.2	12	.0	.0	.0	.6	.3	.3
	.4	12	.8	.6	.2	8.2	3.9	14.8
	.6	12	6.3	76.7	>60	121.8	>300:3	>60:10
	.8	12	45.6	>300	>60	85.6	>300:12	>60:12
20	.2	6	.8	.2	.1	1.2	.3	.2
	.4	6	1.1	2.2	1.7	20.3	10.2	21.6
	.6	6	180.8	>300	>60	>300:2	>300:6	>60:6
	.8	6	>300	>300	>60	>300:3	>300:6	>60:6

TABLE 11.5. NUMBERS OF NODES

n	t	number of problems	median			maximum		
			alg.NA	alg.JS	alg.LE	alg.NA	alg.JS	alg.LE
10	.2	24	1	2	6	8	14	64
	.6	24	56	132	3239	456	12284	96328
15	.2	12	1	1	1	28	69	572
	.4	12	44	86	305	541	586	36231
	.6	12	647	13066	–	9564	–	–
	.8	12	4532	–	–	9952	–	–
20	.2	6	9	12	105	29	29	580
	.4	6	25	281	3564	1206	1130	57671
	.6	6	11105	–	–	–	–	–
	.8	6	–	–	–	–	–	–

TABLE 11.6. ALGORITHM NA ON FISHER'S TEST PROBLEMS

n	number of problems	solution time		number of nodes	
		median	maximum	median	maximum
20	25	1.0	35.7	12	19987
30	25	5.6	>300:1	315	–
50	16	41.6	>300:3	6022	–

LEGEND TO TABLES 11.4,5,6

solution times	CPU seconds on a Control Data Cyber 73-28.
numbers of nodes	including eliminated nodes.
$>\ell:k$	the time limit ℓ is exceeded k times.
algorithm NA	see Section 11.2.
algorithm JS	see Section 11.3.1.
algorithm LE	see Section 11.3.1.
n	number of jobs.
t	average tardiness factor

cant difference between "easy" problems with $t = .2$ or $t = .4$ and "difficult" problems with $t = .6$ or $t = .8$.

On the easy problems, algorithm LE is rather successful and runs quickly through large search trees. Algorithm JS also performs well, notably for $n = 15$ and $t = .4$. In fact, Shwimer tested his method only on problems where $t = (n-1)/2n$, *i.e.*, $t = .47$ for $n = 15$. Algorithm NA exhibits a satisfactory and steady behaviour. Both the median and maximum numbers of nodes examined by this method are significantly smaller than the numbers for the other two methods, so our lower bound is indeed more effective in pruning the search tree. For these problems, however, it seems hardly worth-while to spend much time on the computation of sophisticated lower bounds.

Turning to the difficult problems, we see that algorithm NA is by far superior to the other algorithms. This is most clearly shown by the results for the problems with 15 or 20 jobs. Of the latter set of twelve problems, algorithms JS and LE do not finish any problem at all; algorithm NA succeeds in finishing seven of them and finds better solutions to the remaining five. The measures of performance become completely useless in this situation. Our results seem to contradict the remark in [Srinivasan 1971] that problems with $t = .65$ are the most difficult ones; problems with $t = .8$ are clearly the most difficult here.

We will now discuss the influence of the remaining three parameters ρ, s and r on the performance of algorithm NA.

As to the *correlation* ρ, no influence at all could be demonstrated.

The *relative variation of processing times* s has a significant influence for problems with 15 or 20 jobs, as demonstrated by the sign test ($\alpha < .02$). For $n = 20$, eleven out of twelve problems with $s = .05$ were finished with a median solution time of 8 seconds, while only eight out of twelve problems with $s = .25$ were finished with a median of 150 seconds. On the average, 70 per cent of the nodes were eliminated by the underestimate LB_r or by Corollary 11.4 when $s = .05$, and only 40 per cent when $s = .25$. Furthermore, the first schedule found by algorithm NA, corresponding to the assignment in the root node, was at worst 1 per cent from the optimum when $s = .05$ and at worst 20 per cent when $s = .25$. As expected our lower bound depends heavily on s.

The *relative range of due dates* r has a considerable influence. Problems with $r = .95$ are significantly easier than problems with $r = .20$.

Algorithm NA was also tested on a set of problems from [Fisher 1974]. In this paper, a dual algorithm for the $n|1||\sum T_i$ problem is developed, using a subgradient approach to produce strong lower bounds. Table 11.6 shows that algorithm NA performs rather well on Fisher's test problems. However, they are easy ones with equal weights, $\rho = 0$, $s = .54$, $t = .5$ and $r = 1$, and both methods cannot be compared from these data alone.

11.4. Remarks

In view of our computational results, our main conclusion clearly has to be that the $n|1||\sum w_i T_i$ problem remains a very difficult one. The same remark applies *a fortiori* to the general $n|1|prec|\sum f_i$ problem. The results indicate that even stronger elimination criteria and sharper lower bounds are needed to cut down the size of the search tree.

The usefulness of elimination criteria is strongly underlined by our experiments. An easy extension of our algorithm would be to check all of them at every node. Also it may be worth-while to look for more elimination criteria. We feel that we have thoroughly examined the possible effects of interchanging two jobs, but one may look into the effects of moving three or more jobs at a time.

The idea of computing lower bounds by solving linear assignment problems whose coefficients f_{ij} underestimate the costs of putting J_i in position j can be applied to a broader set of problems, *e.g.*, to the $n|m|P|\sum f_i$ problem. In view of the lack of any algorithm in this area, this seems an interesting object for future research.

For the one-machine problem this bounding principle has turned out to be very useful. It could be strengthened by considering only those solutions to the assignment problem that respect known precedence constraints. It is difficult to predict the effectiveness of this approach, since the resulting *linear assignment problem with precedence constraints* is NP-complete (*cf.* Theorem 4.5). Moreover, the precedence constraints are observed already in the computation of the present cost coefficients.

None the less, it seems necessary to develop a fundamentally stronger lower bound. The general $n|1||\sum f_i$ bound from [Fisher 1974], which is based on the use of Lagrangean multipliers, might be a step in the right direction. Especially, very sharp bounds should be used in the upper levels of the search tree, where pruning may lead to large reductions in the number of potentially optimal solutions. As we move down the tree, pruning leads to smaller reductions, and simpler lower bounds combined with more extensive enumeration become more attractive. This observation suggests the use of lower bounds of varying computational complexity throughout the tree: a *gliding lower bound*. We tried to apply this idea in our algorithm by using lexicographic enumeration in the seven deepest levels of the tree. This led to a disappointingly small decrease in computation time, but the idea could become useful in the future.

In spite of all the work done so far, the problem of minimizing total costs in one-machine scheduling is likely to remain a challenge to researchers for a long time to come.

12. PERMUTATION FLOW-SHOP SCHEDULING

12.1. Introduction

The *general flow-shop problem*, indicated by $n|m|F|C_{max}$, can be formulated as follows.

> Each of n jobs $J_1,\ldots,J_n$ has to be processed on m machines $M_1,\ldots,M_m$ in that order. Job J_i $(i = 1,\ldots,n)$ thus consists of a sequence of m operations $O_{i1},\ldots,O_{im}$; O_{ik} corresponds to the processing of J_i on M_k during an uninterrupted processing time p_{ik}. We want to find a processing order on each M_k $(k = 1,\ldots,m)$ such that the time required to complete all J_i $(i = 1,\ldots,n)$ is minimized.

It is well known [Conway *et al.* 1967; Rinnooy Kan 1976] that there exists an optimal $n|m|F|C_{max}$ schedule with the same processing order on M_1 and M_2 and the same processing order on M_{m-1} and M_m. This result cannot be extended any further, as is shown by the $2|4|F|C_{max}$ example with $p_{11} = p_{22} = p_{23} = p_{14} = 1$, $p_{21} = p_{12} = p_{13} = p_{24} = 3$; the unique optimal schedule is illustrated in Figure 12.1. If, none the less, we restrict ourselves to minimization over all

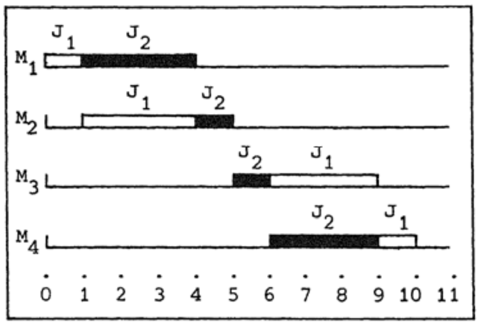


Figure 12.1

permutation schedules (*i.e.* schedules with the same processing order on each machine), the resulting problem is called the *permutation flow-shop problem* ($n|m|P|C_{max}$); it will be studied in the present chapter. To find the true $n|m|F|C_{max}$ optimum, we can apply any of the algorithms for the *general job-shop problem* ($n|m|G|C_{max}$), which is to be discussed in Chapter 13.

In Chapter 4 we have seen that there exists an $O(n \log n)$ algorithm for the $n|2|P|C_{max}$ problem [Johnson 1954], but that the $n|3|P|C_{max}$ problem is already NP-complete. Thus, we shall restrict ourselves to $n|m|P|C_{max}$ algorithms of the branch-and-bound type. A number of these has been developed, each of which is based on the enumeration scheme described in Section 12.2.1 below. Elimination criteria that can be applied within this enumeration scheme are surveyed in Section 12.2.2. In Section 12.2.4 we develop a conceptual framework for generating lower bounds; it leads to two bounds that together dominate all bounds presented previously but not each other. In Sections 12.2.3 and 12.2.5 we consider the implementation of elimination criteria and lower bounds. Section 12.2.6 presents some simple heuristics for obtaining an initial upper bound. In Section 12.3 we report on computational experience. Concluding remarks are contained in Section 12.4.

12.2. Algorithms

12.2.1. *Enumeration scheme*

The enumeration scheme used in all branch-and-bound algorithms developed so far generates all n! permutation schedules according to algorithm PS below.

```
procedure algorithm PS (n,$\sigma$);
begin   local i;

        procedure node(S,$\ell$);
        if S = $\emptyset$ then comment a permutation schedule $\sigma$ has been generated else
        begin   local Q;
                Q:= S;
                while Q $\neq$ $\emptyset$ do
                begin   i:$\in$ Q; Q:= Q-{i};
                        $\sigma(\ell)$ := i;
                        node(S-{i},$\ell$+1)
                end
        end;

        node({1,...,n},1)
end algorithm PS.
```

A node at the ℓ-th level of the search tree is characterized by a *partial schedule* σ = $(\sigma(1),...,\sigma(\ell-1))$, indicating that $J_{\sigma(i)}$ occupies the i-th position on each machine, for $i = 1,\ldots,\ell-1$. Any permutation $\bar{\sigma}$ of the index set S of unscheduled jobs defines a *completion* of σ, *i.e.* a complete permutation schedule $\sigma\bar{\sigma} = (\sigma(1),\ldots,\sigma(\ell-1),\bar{\sigma}(1),\ldots,\bar{\sigma}(s))$, where $s = |S| = n-\ell+1$. By placing any J_i ($i \in S$) in the ℓ-th position, we enter a descendant node, corresponding to a partial schedule $\sigma i = (\sigma(1),\ldots,\sigma(\ell-1),i)$.

12.2.2. *Elimination criteria*

In this section, we shall be interested in finding conditions under which all completions of a partial schedule σ' can be eliminated because a schedule at least as good exists among the completions of another partial schedule σ''. We define S' and S'' as the index sets corresponding to σ' and σ'' respectively, and $C(\sigma,k)$ as the completion time of the last job in the partial schedule σ on M_k. Then σ'' *dominates* σ' if for any completion $\sigma'\bar{\sigma}'$ of σ' there exists a completion $\sigma''\bar{\sigma}''$ of σ'' such that $C(\sigma''\bar{\sigma}'',m) \le C(\sigma'\bar{\sigma}',m)$.

THEOREM 12.1 [Ignall & Schrage 1965; Smith & Dudek 1967; McMahon 1969]. *If* $S' = S''$ *and* $C(\sigma'',k) \le C(\sigma',k)$ *for* $k = 1,\ldots,m$, *then* σ'' *dominates* σ'.

Proof. Trivial. □

For the case that $S' = S''$, the above criterion is the strongest possible one in the following sense: if $C(\sigma'',k) > C(\sigma',k)$ for some k, then processing times for the unscheduled jobs can be chosen in such a way that $C(\sigma''\bar{\sigma},m) > C(\sigma'\bar{\sigma},m)$ for every completion [McMahon 1969].

For the case that $S'\cup\{j\} = S''$, several elimination criteria have been developed that give conditions for the dominance of $\sigma' = \sigma i$ by $\sigma'' = \sigma ji$. Defining $\Delta_k = C(\sigma ji,k)-C(\sigma i,k)$, we can now formulate the following conditions, each of which has been claimed to imply dominance of σi by σji.

(α) [Dudek & Teuton 1964]

$$C(\sigma ji,k) \le C(\sigma ij,k) \qquad (k = 2,\ldots,m);$$

(β) [Smith & Dudek 1967]

$$\Delta_{k-1} \le p_{jk} \qquad (k = 2,\ldots,m);$$

(γ) [Smith & Dudek 1969]

$$\Delta_{k-1} \le p_{jk} \text{ and } C(\sigma j,k-1) \le C(\sigma i,k-1) \qquad (k = 2,\ldots,m);$$

(δ) [Bagga & Chakravarti 1968]

$$\Delta_k \le p_{jk} \qquad (k = 2,\ldots,m);$$

(ε) [McMahon 1969; Szwarc 1973]

$$\max\{\Delta_{k-1},\Delta_k\} \le p_{jk} \qquad (k = 2,\dots,m);$$

(ζ) [Szwarc 1971]

$$\Delta_{k-1} \le \Delta_k \le p_{jk} \qquad (k = 2,\dots,m);$$

(η) [Szwarc 1973]

$$\max\{\Delta_\ell \mid \ell = 1,\dots,k\} \le p_{jk} \qquad (k = 2,\dots,m);$$

(θ) [Gupta 1971]

$$\Delta_k \le \min\{p_{j\ell} \mid \ell = k,\dots,m\} \qquad (k = 2,\dots,m).$$

Elimination criteria (α), (β) and (δ) have been proved incorrect through counterexamples in [Karush 1965; McMahon 1969; Szwarc 1971]. With respect to the remaining ones we have the following theorems.

THEOREM 12.2. *Condition* (γ) *implies condition* (ε).

Proof. If (γ) holds, then $\Delta_{k-1} \le p_{jk}$, and we have only to show that $\Delta_k \le p_{jk}$.

$$\begin{aligned} C(\sigma ji,k-1)+p_{ik} &= C(\sigma i,k-1)+\Delta_{k-1}+p_{ik} \\ &\le C(\sigma i,k-1)+p_{ik}+p_{jk} \qquad (12.1) \\ &\le C(\sigma i,k)+p_{jk}; \\ C(\sigma j,k)+p_{ik} &= \max\{C(\sigma j,k-1),C(\sigma,k)\}+p_{jk}+p_{ik} \\ &\le \max\{C(\sigma i,k-1),C(\sigma,k)\}+p_{ik}+p_{jk} \qquad (12.2) \\ &= C(\sigma i,k)+p_{jk}. \end{aligned}$$

Together, (12.1) and (12.2) imply that $\Delta_k \le p_{jk}$. □

THEOREM 12.3 [Szwarc 1973]. *Conditions* (ε), (ζ), (η), *and* (θ) *are equivalent.*

Proof. See [Szwarc 1973; Szwarc 1975]. □

THEOREM 12.4 [McMahon 1969; Szwarc 1971]. *If condition* (ε), (ζ), (η), *or* (θ) *holds, then* σji *dominates* σi.

Proof. First, we prove by induction on |ρ| and k that (ζ) implies

$$C(\sigma ji\rho,k)-C(\sigma i\rho,k) \le \Delta_k \quad \text{for any } \rho \text{ and } k. \qquad (12.3)$$

For $\rho = \emptyset$ or $k = 1$, (12.3) is trivially true. Assuming that (12.3) has been proved for $\rho = \rho'h$, $k = \ell-1$ and $\rho = \rho'$, $k = \ell$, we have for the case that $\rho = \rho'h$, $k = \ell$ that

$$C(\sigma ji\rho'h,\ell)-C(\sigma i\rho'h,\ell)$$

$$= \max\{C(\sigma ji\rho'h,\ell-1),C(\sigma ji\rho',\ell)\}+p_{h\ell}-\max\{C(\sigma i\rho'h,\ell-1),C(\sigma i\rho',\ell)\}-p_{h\ell}$$

$$\le \max\{C(\sigma ji\rho'h,\ell-1)-C(\sigma i\rho'h,\ell-1),C(\sigma ji\rho',\ell)-C(\sigma i\rho',\ell)\}$$

$$\le \max\{\Delta_{\ell-1},\Delta_{\ell}\} = \Delta_{\ell}.$$

Now, it follows from (12.3) and (ζ) that

$$C(\sigma ji\rho,k) \le C(\sigma i\rho,k)+p_{jk} \le C(\sigma i\rho j,k) \quad \text{for any } \rho \text{ and } k.$$

Thus, by Theorem 12.1, $\sigma ji\rho$ dominates $\sigma i\rho j$ for every ρ. This implies that σji dominates σi. □

We refer to [McMahon 1969] for a systematic example showing that (ε), and, by Theorem 12.3, (ζ), (η), and (θ) as well, are again the strongest possible conditions for elimination in the previously mentioned sense.

The above analysis can be extended to the case that $S' \subset S''$ with $S''-S'$ of arbitrary cardinality [McMahon 1969]. This leads to very stringent conditions and it is questionable if the reduction in the size of the search tree compensates for the additional computational requirements at each node.

Computational experience reported in [McMahon 1971; Baker 1975] indicates that enumerative methods based on the simple elimination criteria above are inferior to those based on lower bounds; inclusion of these criteria in the latter type of algorithm leads to a gain in efficiency only for problems of moderate size ($n \le 15$). Altogether, it seems that the elimination criteria discussed in this section are of little algorithmic value.

12.2.3. *Implementation of the elimination criteria*

The elimination criteria from Theorem 12.4 were combined with some of the more successful lower bounds which will be presented below.

In order to find out if σi is dominated by σji, it is sufficient to check condition (ζ). It follows easily that in that case

$$p_{j1} \le p_{jk} \quad (k = 2,\dots,m). \tag{12.4}$$

The dominance relation is transitive; however, the stronger condition (ζ) need not be transitive and we have to check (ζ) for each pair (i,j) such that (12.4) holds for j. Dominance cycles can occur and have to be avoided. Altogether, application of elimination criterion (ζ) for all $i,j \in S$ requires $O(ms^2)$ calculations.

12.2.4. *Lower bounds*

Given a partial schedule σ, we now want to find lower bounds on the value of all possible completions $\sigma\bar{\sigma}$. We shall be particularly concerned with the trade-off between the sharpness of a lower bound and its computational requirements; a stronger bound eliminates relatively more nodes of the search tree, but if its computational requirements become excessively large it may become advantageous to search through larger parts of the tree, using a weaker but more quickly computable bound.

We shall generally obtain lower bounds by relaxing the capacity constraints on some machines, *i.e.* by treating *bottleneck* machines of capacity one as *non-bottleneck* machines of infinite capacity.

From the complexity results in Chapter 4 we know that any problem involving three or more bottleneck machines is likely to be NP-complete. Let us therefore restrict ourselves to choosing at most two machines M_u and M_v $(1 \le u \le v \le m)$ to be bottleneck machines. Since any remaining sequence of non-bottleneck machines can obviously be treated as one non-bottleneck machine, it follows that each partial schedule σ defines a problem involving at most five machines, of which at most three are non-bottleneck ones. They are indicated by $M_{(0,u-1)}$, $M_{(u+1,v-1)}$ and $M_{(v+1,m)}$, and the processing times on these machines are defined by

$$\begin{aligned} p_{i(0,u-1)} &= \max\{C(\sigma,\ell) + \textstyle\sum_{k=\ell}^{u-1} p_{ik} \mid \ell = 1,\dots,u\} && (i \in S);\\ p_{i(u+1,v-1)} &= \textstyle\sum_{k=u+1}^{v-1} p_{ik} && (i \in S);\\ p_{i(v+1,m)} &= \textstyle\sum_{k=v+1}^{m} p_{ik} && (i \in S).\end{aligned}$$

The $p_{i(0,u-1)}$ may be interpreted as release dates of J_i $(i \in S)$ on M_u. Note that, if $u = v$, at most three machines are involved, including one bottleneck M_u.

Thus, by relaxing capacity constraints, we obtain a problem of scheduling $\{J_i \mid i \in S\}$ on $M_{(0,u-1)}$, M_u, $M_{(u+1,v-1)}$, M_v, $M_{(v+1,m)}$ in that order, where again the maximum completion time is to be minimized. Any lower bound for this problem provides a valid lower bound on all possible completions $\sigma\bar{\sigma}$; in fact, all lower bounds presented in the literature can be interpreted in these terms.

To arrive at a further classification of possible approaches to this lower bound calculation, note that we may eliminate non-bottleneck machines $M_{(g,h)}$ from the problem and compensate for this by adding terms

$$p_{*(g,h)} = \min_{i\in S}\{p_{i(g,h)}\}$$

to a lower bound on the remaining problem (if $u = 1$, $v = u+1$ or $v = m$, we have $p_{*(0,u-1)} = C(\sigma,1)$, $p_{*(u+1,v-1)} = 0$ or $p_{*(v+1,m)} = 0$, respectively). The lower bound that we shall explore here is obtained by finding the optimal schedule with respect to this remaining problem.

Any such approach can be characterized by a string Ω of at most five symbols from {□,○,*} where

- □ indicates a bottleneck machine;
- ○ indicates a non-bottleneck machine on which the various processing times are taken into account;
- * indicates a non-bottleneck machine that is to be eliminated through the device introduced above.

Thus, we obtain a lower bound $LB(u,v,\Omega)$ by finding the optimal value $LB^*(u,v,\Omega)$ of the problem on machines M_u and M_v of type □ and possible machines $M_{(g,h)}$ of type ○, and adding to it terms $p_{*(g,h)}$ for machines $M_{(g,h)}$ of type *. If $u \neq v$, $LB^*(u,v,\Omega)$ can be strengthened by exploiting the fact that M_v is not available before $C(\sigma,v)$.

Defining $Z = \{(u,v) \mid 1 \le u \le v \le m\}$, we conclude that

$$LB(W,\Omega) = \max\{LB(u,v,\Omega) \mid (u,v) \in W\}$$

is a valid lower bound for any $W \subset Z$.

If we do not distinguish between symmetric pairs of strings such as (*□○) and (○□*), we can obtain the nine different strings which together constitute the vertex set of the directed graph drawn in Figure 12.2. An arc (Ω,Ω') in this graph indicates that Ω' *dominates* Ω in the sense that

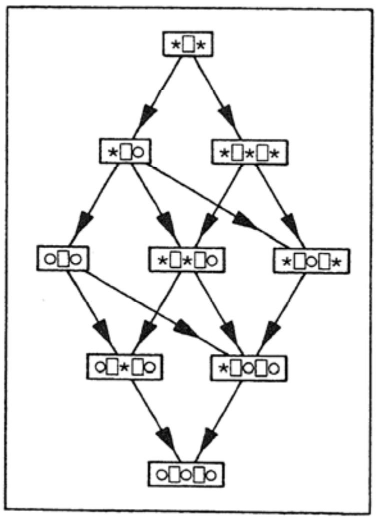


Figure 12.2

for any pair (u,v) we can find a pair (u',v') such that $LB(u',v',\Omega') \geq LB(u,v,\Omega)$. The correctness of the dominance rules expressed by Figure 12.2 is easily proved. Apart from the relations implied by transitivity considerations, no other relations hold. Thus, for example, $LB(Z,(\circ\Box\circ))$ can be larger or smaller than $LB(Z(*\Box\circ\Box*))$, depending on the processing times; an example appears below.

We shall now discuss each lower bound and compare it to bounds presented in the literature, using the following notations:

$$r_{iu} = p_{i(0,u-1)},\ r_{*u} = p_{*(0,u-1)};$$
$$q_{iv} = p_{i(v+1,m)},\ q_{*v} = p_{*(v+1,m)}.$$

(*a*) $(*\Box*)$

Eliminating $M_{(0,u-1)}$ and $M_{(u+1,m)}$, we have to minimize the maximum completion time C_{max} on M_u. Clearly $LB^*(u,u,(*\Box*)) = \sum_{i\in S} p_{iu}$ and

$$LB(u,u,(*\Box*)) = r_{*u} + \sum_{i\in S} p_{iu} + q_{*u}.$$

Note, as a general principle, that we may replace $r_{*u}+q_{*v}$ by $\min\{r_{iu}+q_{jv} \mid i,j \in S,\ i \neq j\}$, leading to a possibly sharper bound.

$LB(\{(u,u) \mid u = 1,\ldots,m\},(*\Box*))$ is the so-called *machine-based bound* used in [Ignall & Schrage 1965; McMahon 1971]; through its use of r_{iu} instead of $C(\sigma,u)$ it is slightly stronger than the bounds used in [Lomnicki 1965; Brown & Lomnicki 1966; McMahon & Burton 1967].

(*b*) $(*\Box\circ)$

Eliminating $M_{(0,u-1)}$, we have to minimize the maximum lateness L_{max} with respect to due dates $K-q_{iu}$ on M_u (*cf.* Section 10.1). $LB^*(u,u,(*\Box\circ))$ is found by ordering the jobs according to nonincreasing q_{iu} (*cf.* Section 10.2.1); adding r_{*u} yields $LB(u,u,(*\Box\circ))$.

Analogously, $LB(u,u,(\circ\Box*))$ is obtained by ordering the jobs according to nondecreasing r_{iu} and adding q_{*u} to the resulting solution value. Through its use of r_{im} instead of $C(\sigma,1) + \sum_{k=1}^{m-1} p_{ik}$, $LB(m,m,(\circ\Box*))$ is stronger than the *noninterference bound* proposed in [Ashour 1970].

(*c*) $(\circ\Box\circ)$

Computation of $LB(u,u,(\circ\Box\circ))$ corresponds to solving an $n|1|r_i{\geq}0|L'_{max}$ problem, defined by $|S|$ triples (r_{iu},p_{iu},q_{iu}), on M_u (*cf.* Section 10.1). We have proved this problem to be NP-complete in Chapter 4. However, the excellent performance of some enumerative $n|1|r_i{\geq}0|L'_{max}$ algorithms, as reported in Section 10.3, justifies serious consideration of this lower bound approach.

(d) (∗□∗□∗)

Eliminating $M_{(0,u-1)}$, $M_{(u+1,v-1)}$ and $M_{(v+1,m)}$, we obtain $LB^*(u,v,(\ast\square\ast\square\ast))$ by solving the $n|2|P|C_{max}$ problem on M_u and M_v by means of Johnson's algorithm [Johnson 1954]. The optimal order of the jobs can be determined in advance; it does not change if some jobs are removed, nor is it influenced by the availability of M_v from $C(\sigma,v)$ onwards. Applying the principle mentioned under *(a)*, we find

$$LB(u,v,(\ast\square\ast\square\ast))$$
$$= LB^*(u,v,(\ast\square\ast\square\ast)) + \min\{r_{hu}+p_{i(u+1,v-1)}+q_{jv} | h,i,j \in S,\ h \neq j\}.$$

(e) (∗□○□∗)

Eliminating $M_{(0,u-1)}$ and $M_{(v+1,m)}$, we have to solve a special $n|3|P|C_{max}$ problem where M_u and M_v are separated by a non-bottleneck machine $M_{(u+1,v-1)}$. $LB^*(u,v,(\ast\square\circ\square\ast))$ is found by applying Johnson's $n|2|P|C_{max}$ algorithm using processing times $p_{iu}+p_{i(u+1,v-1)}$ and $p_{i(u+1,v-1)}+p_{iv}$ $(i \in S)$ [Jackson 1956]; the availability of M_v on $C(\sigma,v)$ again does not change the optimal processing order. Adding $\min\{r_{iu}+q_{jv} | i,j \in S,\ i \neq j\}$ yields $LB(u,v,(\ast\square\circ\square\ast))$. The so-called *job-based bound* from [McMahon 1971]

$$\max_u\{r_{\ast u} + \max_{i\in S}\{p_{i(u,m)} + \textstyle\sum_{h\in S-\{i\}}\min\{p_{hu},p_{hm}\}\}\}$$

and the similar bound from [McMahon & Burton 1967], using $C(\sigma,u)$ instead of r_{iu}, are easily seen to be underestimates of $LB(\{(u,m)|u = 1,\ldots,m-1\},(\ast\square\circ\square\ast))$.

(f) (∗□∗□○), (∗□○□○), (○□∗□○), (○□○□○)

The $n|2|P|L'_{max}$ problem corresponding to $LB^*(u,v,(\ast\square\ast\square\circ))$ has been shown to be NP-complete in Chapter 4, as has the $n|2|P,r_i\geq 0|C_{max}$ problem corresponding to $LB^*(u,v,(\circ\square\ast\square\ast))$. Essentially, we have replaced a non-bottleneck machine in (○□○) by a bottleneck one (*cf.* Section 10.4). No specific algorithms have been developed for these problems as yet. Similar remarks apply to the NP-complete problems corresponding to the remaining lower bound approaches.

In view of the above discussion, the lower bounds under *(c)* and *(e)* are obvious candidates for further investigation. $LB(Z,(\ast\square\circ\square\ast))$ dominates all previously developed bounds. There are, however, situations in which $LB(Z,(\circ\square\circ))$ is stronger then $LB(Z,(\ast\square\circ\square\ast))$, and *vice versa*.

Example. Take $n = m = p_{12} = p_{13} = p_{22} = p_{31} = p_{32} = 3$, $p_{11} = p_{33} = 2$, $p_{21} = p_{23} = 1$. The optimal $3|3|P|C_{\max}$ value is equal to 12, and we find that

$$LB(2,2,(\circ\Box\circ)) = 12 > LB(Z,(*\Box\circ\Box*)) = \max\{11,11,11\} = 11.$$

If we change the p_{i2} to 1 $(i = 1,2,3)$, then the optimal value equals 9, and

$$LB(1,3,(*\Box\circ\Box*)) = 9 > LB(Z,(\circ\Box\circ)) = \max\{8,6,8\} = 8.$$

12.2.5. *Implementation of the lower bounds*

In this section we shall discuss in detail the implementation of each lower bound that was tested. For all lower bounds except $LB(Z,(\circ\Box\circ))$ we replaced r_{iu} by $C(\sigma,u)$; since each of these bounds only involves r_{*u}, very little is gained and, in fact, solution times are increased by using r_{iu} instead of $C(\sigma,u)$.

In each case we applied a recursive search strategy of the type "bb backtrack2" where descendant nodes are chosen in order of nondecreasing lower bounds (see Chapter 8). We can distinguish two types of calculations:

(1) calculations performed once at the root node of the search tree;

(2) calculations performed at the node corresponding to σ in order to obtain lower bounds $LB(W,\Omega)$ for all σi $(i \in S)$.

(*a*) $LB(Z,(*\Box*))$

(1) At the root node q_{iu} is calculated for all (i,u) in $O(mn)$ steps.

(2) For each M_u, $\sum_{i\in S} p_{iu}$ is calculated and indices i_u' and i_u'' with

$$q_{i_u',u} = \min\{q_{iu} | i \in S\},$$
$$q_{i_u'',u} = \min\{q_{iu} | i \in S-\{i_u'\}\}$$

are found in $O(s)$ steps. For each choice $i \in S$, $LB(u,u,(*\Box*))$ is then calculated in $O(1)$ steps as

$$\max\{C(\sigma,u),C(\sigma i,u-1)\} + \textstyle\sum_{i\in S} p_{iu} + \begin{cases} q_{i_u',u} & \text{if } i \neq i_u', \\ q_{i_u'',u} & \text{if } i = i_u'. \end{cases}$$

Altogether, calculation of $LB(Z,(*\Box*))$ for all σi $(i \in S)$ requires $O(ms)$ steps.

(*b*) $LB(Z,(*\Box\circ))$

(1) At the root node the q_{iu} are calculated and, for all u, ordered into a nondecreasing sequence in $\mathcal{O}(mn \log n)$ steps.

(2) For each choice $i \in S$, $LB(u,u,(*\Box\circ))$ is calculated in $\mathcal{O}(s)$ steps by scheduling $\{J_j | j \in S-\{i\}\}$ according to the ordering found in the root node. Calculation of $LB(Z,(*\Box\circ))$ for all σi $(i \in S)$ requires $\mathcal{O}(ms^2)$ steps.

We have not considered $LB(Z,(\circ\Box *))$; this bound performed very poorly in some initial testing.

(*c*) $LB(Z,(\circ\Box\circ))$

(1) At the root node the q_{iu} are calculated and ordered in $\mathcal{O}(mn \log n)$ steps.

(2) Calculation of $LB(Z,(\circ\Box\circ))$ for all σi $(i \in S)$ requires the solution of $\mathcal{O}(ms)$ $n|1|r_i{\ge}0|L'_{max}$ problems.

(*d*) $LB(W,(*\Box *\Box *))$

This bound was not implemented; it is dominated by $LB(W,(*\Box\circ\Box *))$ and requires the same computational effort.

(e0) job-based bound

(1) At the root node $p_{i(u,m)}$ and $\min\{p_{iu},p_{im}\}$ are calculated for all (i,u) in $\mathcal{O}(mn)$ steps.

(2) The job-based bound for σi on M_u can be rewritten as follows:

$$\begin{aligned} &C(\sigma i,u) + \max_{j\in S-\{i\}}\{p_{j(u,m)} + \textstyle\sum_{h\in S-\{i,j\}}\min\{p_{hu},p_{hm}\}\} \\ &= C(\sigma i,u) + \tau_u - \min\{p_{iu},p_{im}\} + \max_{j\in S-\{i\}}\{\upsilon_{ju}\} \end{aligned}$$

where

$$\begin{aligned} \tau_u &= \textstyle\sum_{h\in S}\min\{p_{hu},p_{hm}\}, \\ \upsilon_{ju} &= p_{j(u,m)} - \min\{p_{ju},p_{jm}\}. \end{aligned}$$

Accordingly, for each M_u the τ_u and υ_{ju} $(j \in S)$ are calculated and indices i'_u and i''_u with

$$\begin{aligned} \upsilon_{i'_u,u} &= \max\{\upsilon_{ju} | j \in S\}, \\ \upsilon_{i''_u,u} &= \max\{\upsilon_{ju} | j \in S-\{i'_u\}\} \end{aligned}$$

are found in $\mathcal{O}(s)$ steps. For each choice $i \in S$, the bound on M_u is then calculated in $\mathcal{O}(1)$ steps as

$$C(\sigma i,u) + \tau_u - \min\{p_{iu},p_{im}\} + \begin{cases} \upsilon_{i'_u,u} & \text{if } i \ne i'_u, \\ \upsilon_{i''_u,u} & \text{if } i = i'_u. \end{cases}$$

Altogether, calculation of the job-based bound for all σi ($i \in S$) requires $\mathcal{O}(ms)$ steps.

We note that the *composite bound* LB(McM) = max{LB(Z,$(*\square*)$),job-based bound} from [McMahon 1971] is the strongest lower bound developed so far.

(*e*) LB(W,$(*\square\circ\square*)$)

(*e*1) $W = W_1 = \{(u,m) \mid u = 1,\dots,m-1\}$.

(*e*2) $W = W_2$ consists of m pairs of critical machines for which $\sum_{i=1}^{n} p_{iu}$, $\sum_{i=1}^{n} p_{iv}$ and $\sum_{i=1}^{n} (p_{iu}+p_{iv})$ are relatively high; these pairs are determined in $\mathcal{O}(mn)$ steps.

(1) At the root node the $p_{iu}+p_{i(u+1,v-1)}$ and $p_{i(u+1,v-1)}+p_{iv}$ are calculated and an optimal order of the jobs with respect to $LB^*(u,v,(*\square\circ\square*))$ is found for all $(u,v) \in W$ in $\mathcal{O}(mn \log n)$ steps.

(2) Note that for any subset of unscheduled jobs an optimal order with respect to $LB^*(u,v,(*\square\circ\square*))$ has been determined at the root node. Calculation of LB(W,$(*\square\circ\square*)$) for all σi ($i \in S$) can now easily be seen to require $\mathcal{O}(ms^2)$ steps.

12.2.6. *Upper bounds*

The value of the best solution found during the tree search provides an upper bound on the value of the optimal solution. At the root node a heuristic method is used to obtain an initial upper bound. Two well-known methods are available for this purpose.

(*A*) [Palmer 1965]

Calculate *slope indices*

$$\lambda_i = \sum_{k=1}^{m} \left(k - \frac{m+1}{2}\right) p_{ik} \quad (i = 1,\dots,n),$$

order the jobs according to nondecreasing λ_i and evaluate the resulting $n|m|P|C_{max}$ schedule. This procedure requires $\mathcal{O}(\max\{mn, n \log n\})$ steps.

(*B*) [Campbell *et al.* 1970]

For $\ell = 1,\dots,m-1$, apply Johnson's $n|2|P|C_{max}$ algorithm using processing times $\sum_{k=1}^{\ell} p_{ik}$ and $\sum_{k=m+1-\ell}^{m} p_{ik}$ ($i = 1,\dots,n$) and evaluate the resulting processing order as $n|m|P|C_{max}$ schedule. Choose the best solution value as initial upper bound. This procedure requires $\mathcal{O}(mn \log n)$ steps.

The second method turned out to produce superior results. In the case of $LB(W,(*\square\circ\square*))$, it also outperformed evaluation of the optimal $LB^*(u,v,(*\square\circ\square*))$ schedule for all $(u,v) \in W$. Accordingly, in each implementation heuristic (*B*) was chosen to provide an initial upper bound.

12.3. Computational experience

12.3.1. *Test problems*

For each test problem with n jobs and m machines, mn integer data p_{ik} were generated from uniform distributions between α_{ik} and β_{ik}. The parameters α_{ik} and β_{ik} are characterized by the following two aspects, thought to be of possible influence on an algorithm's performance:

- *correlation* between the processing times of a job, in the sense that the p_{ik} $(k = 1,\ldots,m)$ are consistently relatively large or relatively small; for problems with correlation, n additional integers γ_i were randomly drawn from $\{1,2,3,4,5\}$;
- a *trend* within the processing times p_{ik} as k increases.

For each chosen combination of n and m, four groups of three problems each were generated according to Table 12.1. A second set of twelve problems was obtained by inversion, *i.e.*, by renumbering M_k as M_{m+1-k} for $k = 1,\ldots,m$; thus, problems with a positive trend are transformed into problems with a negative trend.

TABLE 12.1. VALUES OF PARAMETERS OF TEST PROBLEMS

$\alpha_{ik} : \beta_{ik}$	no correlation	correlation
no trend	$1 \quad : 100$	$20\gamma_i+1 \quad : 20\gamma_i+20$
positive trend	$12\frac{1}{2}(k-1)+1 \; : \; 12\frac{1}{2}(k-1)+100$	$2\frac{1}{2}(k-1)+20\gamma_i+1 \; : \; 2\frac{1}{2}(k-1)+20\gamma_i+20$

12.3.2. *Results*

The algorithms were coded in ALGOL 60 and run on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam. Tables 12.2,3,4,5 show the computational results.

First, all lower bounds were tested on three sets of problems with n|m equal to 6|3, 6|5 and 6|8 respectively. These experiments indicate that the "one-machine bounds" LB(Z,(*□*)), LB(Z,(*□○)) and LB(Z,(○□○)) produce inferior results. Furthermore, the job-based bound can be combined quite easily with LB(Z,(*□*)) or LB(Z,(*□○)); the latter combination dominates the former one but leads to increased solution times.

Consequently, only the composite bound LB(McM) and both "two-machine bounds" LB(W_1,(*□○□*)) and LB(W_2,(*□○□*)) were compared on larger problems with n|m equal to 10|3, 10|5, 15|3, 20|3, 20|5, and 50|3. Since LB(W_1,(*□○□*)) dominates LB(McM), the search tree created by the former bound is in most cases smaller than the tree created by the latter bound. However, LB(W_1,(*□○□*)) is computationally more expensive. The behaviour of LB(W_2,(*□○□*)) is rather erratic. Note that quite often these two-machine bounds achieve the minimum number of nodes, which is equal to n+1 in the case that the initial upper bound is optimal and $\frac{1}{2}n(n+1)$ otherwise.

An increase in the number of machines drastically increases the solution times. Less than half of the 20|5 problems could be solved within one minute.

The same three bounds were combined with the elimination criteria. Previous research [McMahon 1971] indicates that these criteria have a positive influence on solution times only for small problems. In our experiments this was confirmed only with respect to LB(McM). An explanation of this phenomenon might be that the computational requirements of the elimination criteria are of a larger order of magnitude than the requirements of this lower bound. In combination with LB(W,(*□○□*)), however, use of elimination criteria leads to significant decreases in solution times and numbers of unsolved problems, especially if the number of machines is small. Apparently, the elimination criteria eliminate nodes that would be eliminated by lower bounds in any case, but do so with less computational effort.

Altogether, the best results were obtained with algorithms incorporating the elimination criteria and LB(W,(*□○□*)). Table 12.4 indicates that if one minute running time is available to solve a particular problem, it should be allocated to such an algorithm.

Finally, Table 12.5 indicates that both correlation and trends influence the computational performance. Problems with correlation are definitely more difficult. Also, problems with a negative trend are more difficult than problems with a positive trend, confirming earlier impressions [McMahon 1971] that it is helpful to invert a problem if that leads to "fuller" machines M_k for $k > \frac{1}{2}m$.

TABLE 12.2. MEDIAN SOLUTION TIMES

EC	–	–	–	–	–	–	–	–	(ζ)	(ζ)	(ζ)
LB	(*a*)	(*b*)	(*c*)	(*e*0)	(*b*,*e*0)	(*a*,*e*0)	(*e*1)	(*e*2)	(*a*,*e*0)	(*e*1)	(*e*2)
6|3	.17	.26	.38	.08	.13	.07	.05	.06	.07	.05	.06
6|5	.63	1.18	2.57	.18	.26	.15	.16	.39	.14	.13	.32
6|8	1.03	1.39	3.17	.18	.28	.19	.22	1.06	.18	.25	.86
10|3						.20	.25	.34	.21	.18	.25
10|5						.91	.40	6.22	1.09	.42	3.62
15|3						.34	.57	.89	.48	.41	.49
20|3						.55	.62	.81	2.99	.36	.54
20|5						–	–	–	–	–	–
50|3						3.73	31.60	28.98	–	13.94	27.29

TABLE 12.3. MEDIAN NUMBERS OF NODES

EC	–	–	–	–	–	–	–	–	(ζ)	(ζ)	(ζ)
LB	(*a*)	(*b*)	(*c*)	(*e*0)	(*b*,*e*0)	(*a*,*e*0)	(*e*1)	(*e*2)	(*a*,*e*0)	(*e*1)	(*e*2)
6|3	69	64	60	46	19	23	7	7	16	7	7
6|5	208	203	179	47	33	35	30	54	29	22	36
6|8	260	212	167	26	23	23	16	136	23	19	97
10|3						55	55	55	55	55	55
10|5						290	55	971	220	55	599
15|3						129	120	120	120	120	120
20|3						219	116	116	978	116	116
20|5						–	–	–	–	–	–
50|3						1681	2356	1275	–	1275	1275

TABLE 12.4. NUMBERS OF UNSOLVED PROBLEMS

EC	–	–	–	(ζ)	(ζ)	(ζ)
LB	(*a*,*e*0)	(*e*1)	(*e*2)	(*a*,*e*0)	(*e*1)	(*e*2)
6|3	0	0	0	0	0	0
6|5	0	0	0	0	0	0
6|8	0	0	0	0	0	0
10|3	1	1	1	1	1	1
10|5	3	0	8	2	0	7
15|3	7	2	1	5	2	1
20|3	10	6	5	10	2	2
20|5	15	14	19	13	13	18
50|3	11	10	9	12	8	10

TABLE 12.5. NUMBERS OF UNSOLVED PROBLEMS FOR ALL n|m

EC	–		–		–	
LB	(*a*,*e*0)		(*e*1)		(*e*2)	
correlation→ trend↓	no	yes	no	yes	no	yes
no	1	9	1	6	1	8
no	4	8	3	5	5	8
positive	0	9	0	7	0	8
negative	7	9	4	7	5	8

LEGEND TO TABLES 12.2,3,4,5

Each entry in Tables 12.2,3,4 (Table 12.5) represents 24 (27) test problems.
solution times : CPU seconds on a Control Data Cyber 73-28.
numbers of nodes : including eliminated nodes.
numbers of unsolved problems : with a time limit of 60 seconds.
EC : elimination criteria; see Section 12.2.3.
LB : lower bound; see Section 12.2.5;

(*a*) : $LB(Z,(*\square *))$;
(*b*) : $LB(Z,(*\square \circ))$;
(*c*) : $LB(Z,(\circ\square \circ))$;
(*e*0) : job-based bound;
(*e*1) : $LB(W_1,(*\square\circ\square *))$;
(*e*2) : $LB(W_2,(*\square\circ\square *))$.

n|m : number of jobs|number of machines.

12.4. Remarks

The computational experiments reported in the preceding section confirm that the new two-machine bound is superior to previous bounds in solving $n|m|P|C_{max}$ problems. It has to be investigated in more detail for which set of machine pairs this bound should be calculated.

As long as the number of machines is small, problems with up to 50 jobs can often be solved reasonably quickly. An increase in the number of machines makes lower bounds less reliable and drastically increases solution times.

For these larger problems a strong lower bound might be obtained
the assignment of processing times to jobs, *i.e.* by ordering p_{1k},
M_k ($k = 1,\ldots,m$) in such a way that the resulting arrangement, wh
as an $n|m|P|C_{max}$ schedule, has an optimal solution; however, it i
how to solve this problem for $m > 2$. Also, a subgradient approach
Lagrangean multipliers seems an interesting topic for future rese

Contrary to expectations, the use of elimination criteria le
ficant improvements when used in combination with the two-machine
view of this empirical result, it may be worth-while to investiga
putational influence of more intricate elimination criteria such
developed in [McMahon 1971].

13. JOB-SHOP SCHEDULING

13.1. Introduction

This final chapter of Part III is devoted to the *general job-shop problem*, indicated by $n|m|G|C_{max}$. The problem can be formulated as follows.

There are n jobs $J_1,\dots,J_n$ that have to be processed on m machines $M_1,\dots,M_m$. Job J_i $(i = 1,\dots,n)$ consists of a sequence of n_i operations O_u; these operations are indexed by $u = N_{i-1}+1,\dots,N_i$, where $N_i = \sum_{j=1}^{i} n_j$. Machine M_k $(k = 1,\dots,m)$ can handle only one job at a time; the set of operations to be performed on M_k is also indicated by M_k. Operation O_u $(u = 1,\dots,N_n)$ corresponds to the processing of job ι_u on machine μ_u during an uninterrupted processing time p_u. We seek to find a processing order on each machine such that the maximum completion time is minimized.

There exists an $O(n \log n)$ algorithm for the $n|2|G,n_i\leq 2|C_{max}$ problem [Jackson 1956], but two minor extensions of this problem, $n|2|G,n_i\leq 3|C_{max}$ and $n|3|G,n_i\leq 2|C_{max}$, have been shown to be NP-complete in Chapter 4. Even within the class of NP-complete problems, the general $n|m|G|C_{max}$ problem appears to be a very difficult one. A classical and by now traditional quotation from [Conway *et al.* 1967] asserts pessimistically that "many proficient people have considered this problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigators who just cannot believe that a problem so simply structured can be so difficult until they have tried it."

An $n|m|G|C_{max}$ problem can be conveniently represented by means of a *disjunctive graph* $G = (V,C\cup D)$ [Roy & Sussmann 1964] where

- V is the set of *vertices*, representing the operations, including fictitious initial and final operations O_0 and O_*:

$$V = \{0,1,\dots,N_n,*\};$$

- C is the set of directed *conjunctive arcs*, representing the given machine orders of the jobs:

$$C = \{(u,u+1)\,|\,\iota_u = \iota_{u+1}\} \cup \{(0,N_{i-1}+1),(N_i,*)\,|\,i = 1,\dots,n\};$$

- D is the set of directed *disjunctive arcs*, representing the possible processing orders on the machines:

$$D = \{(u,v)\,|\,\mu_u = \mu_v,\ u \neq v\};$$

- a *weight* p_u is attached to each vertex u, with $p_0 = p_* = 0$.

The disjunctive graph for a $3|3|G|C_{max}$ example is drawn in Figure 13.1.

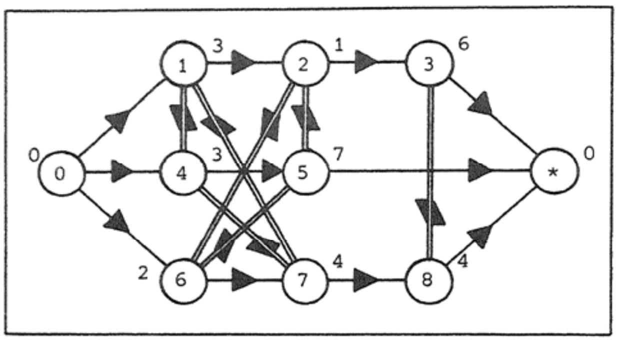


Figure 13.1 Disjunctive graph $G = (\mathcal{V}, \mathcal{C} \cup \mathcal{D})$ for the example.

A pair of disjunctive arcs $\{(u,v),(v,u)\}$ is called *settled* if one of the two arcs has been added to a subset $D \subset \mathcal{D}$ of *chosen* arcs and the other one has been *rejected*; by choosing (u,v), we assign precedence to O_u over O_v on their common machine. A *feasible* schedule is defined by a subset $D^* \subset \mathcal{D}$ such that

- $(u,v) \in D^*$ if and only if $(v,u) \in \mathcal{D}-D^*$;
- the directed graph $G(D^*) = (\mathcal{V}, \mathcal{C} \cup D^*)$ is acyclic.

The value of such a schedule is given by the weight of the maximum-weight path (also called "longest" or "critical") path in $G(D^*)$. The $n|m|G|C_{max}$ problem now consists of finding a *minimaximal* path in G, *i.e.* a maximum-weight path that is minimal over all subsets D^*, satisfying the above requirements. With respect to our example, the graph $G(D^*)$ corresponding to processing orders (O_1,O_4,O_7) on M_1, (O_6,O_2,O_5) on M_2 and (O_3,O_8) on M_3 is drawn in Figure 13.2; the value of the schedule is equal to 14.

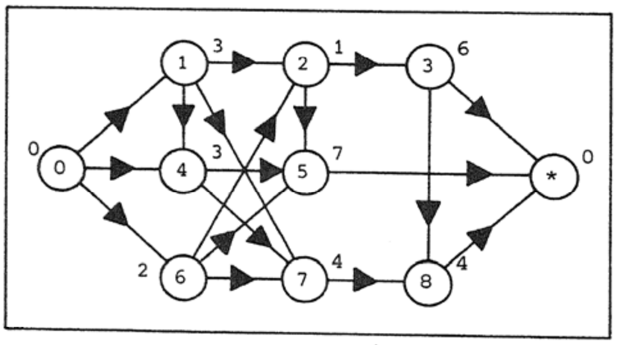


Figure 13.2 Directed graph $G(D^*) = (\mathcal{V}, \mathcal{C} \cup D^*)$ for the example.

The complexity results indicate that our quest for optimal solutions has to involve some form of implicit enumeration. In Section 13.2 several branch-and-bound approaches will be described in terms of the disjunctive graph model above. Accordingly, subsets generated during the tree search will correspond to subsets $D \subset \mathcal{D}$ of chosen disjunctive arcs; the successive augmentations of D are determined by the branching rule. In Section 13.2.1 we discuss how to compute lower bounds on all possible completions $D^* \supset D$ of a partial feasible schedule defined by D. In Section 13.2.2 we examine how a strong bound appearing from this discussion can be combined with several branching rules to yield branch-and-bound algorithms of reasonable quality. In Section 13.2.3 we describe two algorithms that were actually implemented. Section 13.3 reports on some limited computational experience with these methods and Section 13.4 contains concluding remarks.

13.2. Algorithms

13.2.1. *Lower bounds*

Let $D \subset \mathcal{D}$ be a subset of chosen disjunctive arcs such that $G(D) = (\mathcal{V}, C \cup D)$ is acyclic. We seek to find a lower bound on the weight of the critical path in $G(D^*)$ with respect to every $D^* \supset D$ that corresponds to a feasible schedule.

We obtain such a bound $LB_k^*(D)$ $(1 \le k \le m)$ by relaxing the capacity constraints on all machines except M_k. This relaxation corresponds to disregarding all disjunctive arcs in $\mathcal{D}$-D except those on M_k. Accordingly, for each $O_u \in M_k$ we can determine

- a *head* r_u, *i.e.* the maximum weight of a path in G(D) from 0 to u:

$$r_0 = 0,$$
$$r_v = \max\{r_u + p_u \mid (u,v) \in C \cup D\};$$

- a *body* p_u, *i.e.* the given processing time;
- a *tail* q_u, *i.e.* the maximum weight of a path in G(D) from u to $*$ minus p_u:

$$q_* = 0,$$
$$q_u = \max\{p_v + q_v \mid (u,v) \in C \cup D\}.$$

Furthermore, we have a *precedence constraint* $O_u < O_v$ if G(D) contains a path from u to v.

The heads r_u are the earliest possible starting times for O_u. The latest possible starting times R_u can be determined by

$$R_* = r_*,$$
$$R_u = \min\{R_v - p_u \mid (u,v) \in C \cup D\};$$

they are related to the tails q_u by $R_u + p_u + q_u = r_*$.

Clearly, extending D by settling more pairs of disjunctive arcs will never decrease the r_u and q_u. It follows that a valid lower bound $LB_k^*(D)$ is provided by the optimal solution value of the $|M_k||1|prec, r_i \geq 0|L'_{max}$ problem, defined by triples (r_u, p_u, q_u) and precedence constraints (*cf.* LB(k,k,(O□O)) in Section 12.2.4).

A general lower bound $LB^*(D)$ is given by

$$LB^*(D) = \max\{LB_k^*(D) \mid k = 1,\ldots,m\}.$$

In fact, we may take

$$LB^*(D) = \max\{r_*, LB_{k'}^*(D)\}$$

where k' runs over all machine indices such that there are still unsettled pairs of disjunctive arcs on $M_{k'}$.

These observations extend to every lower bound $LB_k(D)$ on $LB_k^*(D)$. It turns out that all bounds presented in the literature correspond to special choices $LB_k(D) \leq LB_k^*(D)$, as indicated by the following survey.

(*a*) (*cf.* [Schrage 1970A; Charlton & Death 1970B; Ashour & Parker 1971; Ashour & Hiremath 1973; Ashour *et al.* 1973; Ashour *et al.* 1974])

$$LB_k(D) = \min_{O_u \in M_k}\{r_u\} + \textstyle\sum_{O_u \in M_k} p_u.$$

(*b*) (*cf.* [Németi 1964; Greenberg 1968; Schrage 1970A; Charlton & Death 1970A; Charlton & Death 1970B; Nabeshima 1971; Ashour & Parker 1971; Sussmann 1972; Ashour *et al.* 1973; Ashour *et al.* 1974])

$$LB_k(D) = \max_{O_u \in M_k}\{r_u + p_u + q_u\}$$

(*cf.* LB_i' in Section 10.2.3).

(*c*) (*cf.* [Brooks & White 1965; Florian *et al.* 1971; Sang & Florian 1970; Ashour *et al.* 1974])

$$LB_k(D) = LB_k'(D) + \min_{O_u \in M_k}\{q_u\}$$

where $LB_k'(D)$ is the value of the optimal $|M_k||1|r_i \geq 0|C_{max}$ schedule on M_k, obtained by ordering the $O_u \in M_k$ according to nondecreasing r_u (*cf.* Section 10.2.1).

(*d*) [Schrage 1970B]

$$LB_k(D) = \min_{O_u \in M_k}\{r_u\} + LB_k''(D)$$

where $LB_k''(D)$ is the value of the optimal $|M_k||1||L'_{max}$ schedule on M_k, obtained by ordering the $O_u \in M_k$ according to nonincreasing q_u (*cf.* Section 10.2.1).

(*e*) [Bratley *et al.* 1973; McMahon & Florian 1975]

$$LB_k(D) = LB_k'''(D)$$

where $LB_k'''(D)$ is the value of the optimal $|M_k||1|r_i{\ge}0|L'_{max}$ schedule on M_k, obtainable by enumerative methods such as algorithms BS and MF (see Sections 10.2.2 and 10.2.3).

Lower bounds (*a,b,c,d*) can be calculated by polynomial-bounded algorithms; it is easy to construct examples in which they are strictly exceeded by $LB_k^*(D)$. The problem of finding bound (*e*) has been shown to be NP-complete in Chapter 4; an example with $LB_k'''(D) < LB_k^*(D)$ can be found in Section 10.2.1.

The relatively small increase in solution times caused by the incorporation of precedence constraints in algorithms for obtaining $LB_k'''(D)$ (see Section 10.2.3) justifies serious consideration of $LB_k^*(D)$ as a lower bound. In the next section we shall see how this bound can be combined with two branching rules to yield $n|m|G|C_{max}$ branch-and-bound algorithms.

Note that we may stop calculating $LB_k^*(D)$ as soon as an upper bound on $LB_k^*(D)$ is not greater than the largest $LB_\ell^*(D)$ ($\ell \neq k$) found so far at the node under examination. Moreover, the node can be eliminated as soon as any lower bound on $LB_k^*(D)$ appearing during its calculation reaches the current upper bound UB.

13.2.2. *Enumeration schemes*

Suppose that at the current node of the $n|m|G|C_{max}$ search tree we have $LB^*(D) < UB$. In that case the node cannot be eliminated and we have to apply some branching rule. In this section we discuss two enumeration schemes; a third one, presented in [Balas 1969] (see also [Agarwal 1975]) has turned out to produce disappointing computational results [Florian *et al.* 1971] and will not be considered here.

(i) *generating active schedules*
(*cf.* [Brooks & White 1965; Florian *et al.* 1971])
A frequently used enumeration scheme generates all active schedules according to algorithm AS2 below (*cf.* algorithm AS1 in Section 10.2.2).

```
procedure algorithm AS2 (G,r);
begin   local k,u,T;

        procedure node(S,T,($t_v | O_v \in T$));
        if T = $\emptyset$ then comment an active schedule has been generated else
        begin   local Q;
                k:$\in$ {$\ell | \min\{t_v+p_v | O_v \in S\cap M_\ell\} = \min\{t_v+p_v | O_v \in S\}$};
                Q:= {$O_u | O_u \in S\cap M_k,\ t_u < \min\{t_v+p_v | O_v \in S\}$};
                while Q $\neq \emptyset$ do
                begin   $O_u$:$\in$ Q; Q:= Q-{$O_u$};
                        $r_u$:= $t_u$;
                        node(if $\iota_u = \iota_{u+1}$ then (S-{$O_u$})$\cup${$O_{u+1}$} else S-{$O_u$},
                             T-{$O_u$},
                             (if $\iota_v = \iota_u$ or $\mu_v = \mu_u$ then $\max\{t_v, t_u+p_u\}$ else $t_v$
                             $| O_v \in$ T-{$O_u$}))
                end
        end;

        T:= {$O_v | v = 1,\ldots,N_n$};
        node({$O_{N_{i-1}+1} | i = 1,\ldots,n$},T,(0|$O_v \in$ T))
end algorithm AS2.
```

THEOREM 13.1. (*cf.* [Giffler & Thompson 1960]). *Algorithm* AS2 *generates every active schedule with respect to a disjunctive graph* G *exactly once.*

Proof. Whenever a call "node(S,T,($t_v|O_v \in$ T))" is made, T contains all unscheduled operations O_u, the t_v indicate their earliest possible starting times, and S $\subset$ T consists of those O_u for which $\{O_v | \iota_v = \iota_u,\ v < u\}$ has been scheduled. Thus, we have only to show that the restriction to Q $\subset$ S is the proper one, *i.e.* that

(1) each generated schedule is active;
(2) a schedule that is not generated is not active;
(3) all generated schedules are different.

We prove (1) and (2), (3) being obvious.

(1) Suppose that in some schedule generated by algorithm AS2 we try to decrease r_v for some O_v. If this is at all possible, it follows that at some earlier stage we must have been able to set $r_v' := t_v$ with $r_v' < r_v$ but have set $r_u := t_u$ instead, where $\mu_u = \mu_v$, $t_u < t_v+p_v$ and $t_u+p_u > t_v$. Hence, r_u would have to be increased.

(2) Suppose that we set $r_u := t_u$ for an $O_u \in S-Q$. If $\iota_u = \iota_{u+1}$, then we have at the next stage $O_{u+1} \in S-Q$ since $t_{u+1} \geq t_u+p_u \geq t_v+p_v$ for some $O_v \in S\cap M_k$, and O_{u+1} should not be scheduled immediately. Thus, if the resulting schedule is at all active (which need not be the case), it can be generated by algorithm AS2. □

Selecting $O_u \in Q$ for the next position on M_k implies that we settle the pair of disjunctive arcs $\{(u,v),(v,u)\}$ for each $O_v \in (T\cap M_k)-\{O_u\}$ by choosing (u,v) and rejecting (v,u). Thus, if a parent node corresponds to a subset $D \subset \mathcal{D}$ of chosen disjunctive arcs, its descendants are characterized by subsets $D\cup\{(u,v)|O_v \in (T\cap M_k)-\{O_u\}\}$ for $O_u \in Q$.

To combine this enumeration scheme with $LB^*(D)$, we consider the structure of the precedence constraints on M_k. Let $M_k' = M_k-T$ and $M_k'' = M_k\cap T$ indicate the sets of scheduled and unscheduled operations on M_k respectively. Then M_k' precedes M_k'' and a processing order for M_k' has been determined, the definitive starting time of $O_u \in M_k'$ being given by r_u. It follows that finding $LB_k^*(D)$ in this case boils down to solving an $|M_k''|\,|1|$*prec*$,r_i\geq 0|L'_{\max}$ problem only with respect to M_k''; we obtain a value $LB_k^{**}(D)$ and take

$$LB_k^*(D) = \max\{\max_{O_u\in M_k'}\{r_u+p_u+q_u\}, LB_k^{**}(D)\}.$$

We finally note that the precedence constraints on M_k'' are of a special type; for $O_u,O_v \in M_k''$ we have $O_u < O_v$ only if $\iota_u = \iota_v$ and $u < v$. Hence, if for a certain problem $\mu_u = \mu_v$ implies that $\iota_u \neq \iota_v$ for all (u,v), then $LB_k'''(D) = LB_k^*(D)$ and thus provides an equally strong bound.

(ii) *settling essential conflicts*

(*cf.* [Németi 1964; Charlton & Death 1970A; Lenstra & Rinnooy Kan 1973])

A second enumeration scheme proceeds by choosing a *branching pair* $\{u,v\}$ with $\{(u,v),(v,u)\} \subset \mathcal{D}-D$ and partitioning the parent subset corresponding to D in two disjoint descendants corresponding to $D_{uv} = D\cup\{(u,v)\}$ and $D_{vu} = D\cup\{(v,u)\}$ respectively. An advantage of such a scheme is that it might allow early settlement of particularly crucial disjunctive pairs, after which all other settlement decisions may follow more or less automatically.

In some respects this branching rule compares unfavourably to the one outlined under (i). The precedence constraints on M_k can now have an arbitrary structure and must be taken into account explicitly during the calculation of $LB_k^*(D)$. Furthermore, the maximum depth of the search tree is $\frac{1}{2}\sum_{k=1}^{m}|M_k|(|M_k|-1)$ as compared to N_n for scheme (i). It seems that this second scheme can be competitive only if

(*a*) we succeed in taking essential branching decisions in the upper levels of the tree;

(*b*) we can choose a branching pair $\{u,v\}$ in such a way that $G(D_{uv})$ and $G(D_{vu})$ are acyclic.

With respect to (*b*) we note that a cycle in $G(D_{uv})$ or $G(D_{vu})$ can only occur if $G(D)$ contains a path from v to u or from u to v. In that case, u and v are linked by an arc in the transitive closure of $G(D)$; construction of this closure would therefore take care of problem (*b*).

It turns out, however, that certain indicators calculated to solve problem (*a*) often allow solution of (*b*) at the same time. In fact, cycles will be avoided altogether if we restrict the choice of a branching pair $\{u,v\}$ to the set $C(D)$ defined by

$$C(D) = \{\{u,v\} | \mu_u = \mu_v,\ r_u+p_u > r_v,\ r_v+p_v > r_u\}.$$

A pair $\{u,v\} \in C(D)$ will be called a *conflict* on machine $\mu_u = \mu_v$.

THEOREM 13.2. *If* $\{u,v\} \in C(D)$, *then* $G(D_{uv})$ *and* $G(D_{vu})$ *are acyclic.*

Proof. If $G(D)$ contains a path from v to u or from u to v, then either $r_v+p_v \le r_u$ or $r_u+p_u \le r_v$, which implies that $\{u,v\} \notin C(D)$. □

THEOREM 13.3. *If* $C(D)\cap\{\{u,v\}|\mu_u = \mu_v = M_k\} = \emptyset$, *then the* r_u *define an optimal one-machine schedule on* M_k *with value at most equal to* r_*. *If* $C(D) = \emptyset$, *then the* r_u *define an optimal* $n|m|G|C_{\max}$ *schedule with value* r_*.

Proof. For each pair $\{u,v\} \notin C(D)$ with $\mu_u = \mu_v = M_k$ we have either $r_u+p_u \le r_v$ or $r_v+p_v \le r_u$. The value of the one-machine schedule is given by

$$\begin{aligned} LB_k^*(D) &= \max_{O_u\in M_k}\{r_u+p_u+q_u\} \\ &= r_* + \max_{O_u\in M_k}\{r_u-R_u\} \le r_*. \end{aligned}$$

For each pair $\{u,u+1\}$ with $\iota_u = \iota_{u+1}$ we have $r_u+p_u \le r_{u+1}$. The value of the overall schedule is given by

$$LB^*(D) = \max_u\{r_u+p_u+q_u\} = r_*. \quad □$$

It follows that, if no conflict exists, there is no need to branch at all. On the other hand, if the r_u do not yield a feasible schedule, we know that $C(D) \neq \emptyset$. Since $LB_k^*(D) > r_*$ indicates the presence of a conflict on M_k, a natural way to find an M_ℓ on which at least one conflict exists is checking the M_k for conflicts in order of nonincreasing $LB_k^*(D)$.

A way to find a suitable branching pair on this M_ℓ is now provided by the introduction of *penalties*

$$P_{uv} = r_u + p_u - R_v.$$

The usefulness of these penalties as branching indicators is illustrated by the following theorem and its corollary.

THEOREM 13.4. *$G(D_{uv})$ contains a path of weight $r_* + P_{uv}$.*

Proof. A path with the required weight is given by the maximum-weight paths from 0 to u and from v to * joined by arc (u,v):

$$r_u + p_u + r_* - R_v = r_* + P_{uv}. \qquad \square$$

Remark. If $P_{uv} > 0$, then $r_* + P_{uv}$ need not be equal to the weight of a maximum-weight path in $G(D_{uv})$. For instance, if $G(D)$ contains a path from v to u of weight $\frac{1}{2}r_*$, then $r_u > \frac{1}{2}r_*$, $R_v < \frac{1}{2}r_*$, $P_{uv} > r_u - R_v > 0$ and $G(D_{uv})$ contains a cycle.

COROLLARY 13.1.

(*a*) *If $P_{uv} \geq 0$, then $LB(D_{uv}) \geq r_* + P_{uv}$.*

(*b*) *If $P_{uv} \geq UB - r_*$, then the node corresponding to D_{uv} can be eliminated.*

Proof. Immediate from Theorem 13.4. $\square$

Corollary 13.1 suggests that a reasonable indicator of the cruciality of a potential branching pair is given by

$$P_{uv}^* = \min\{P_{uv}, P_{vu}\}.$$

A plausible candidate for branching is a pair $\{u^*, v^*\}$ such that

$$P_{u^*v^*}^* = \max\{P_{uv}^* \mid \mu_u = \mu_v = M_\ell,\ \{(u,v),(v,u)\} \subset \mathcal{D} - D\}.$$

Unfortunately, however, the following example shows that, if $\{u^*, v^*\} \notin C(D)$, then $G(D_{u^*v^*})$ or $G(D_{v^*u^*})$ may contain a cycle.

Example. Consider the $4|2|G|C_{max}$ problem specified by the data in Table 13.1. The disjunctive graph $G = (V, C \cup D)$ is drawn in Figure 11.3. At the root node, where $D = \emptyset$, we find $r_* = 2$, $LB_1^*(\emptyset) = LB_2^*(\emptyset) = 3$; note that $C(\emptyset) = \{\{1,2\},\{4,6\}\}$. We may choose M_1 as machine on which to find a branching pair. We have $P_{12}^* = P_{15}^* = P_{25}^* = 0$ and may select $\{u^*,v^*\} = \{2,5\}$ as branching pair. At the descendant node corresponding to $D_{52} = \{(5,2)\}$ we find $r_* = LB_1^*(D_{52}) = LB_2^*(D_{52}) = 4$; now $C(D_{52}) = \{\{4,6\}\}$. Looking for a branching pair on M_2, we have $P_{34}^* = P_{36}^* = P_{46}^* = -2$ and we may select $\{u^*,v^*\} = \{3,4\}$. Choosing (4,3) and rejecting (3,4) would create a cycle (2,3,4,5,2).

TABLE 13.1. DATA FOR THE EXAMPLE

u	1	2	3	4	5	6
ι_u	J_1	J_2	J_2	J_3	J_3	J_4
μ_u	M_1	M_1	M_2	M_2	M_1	M_2
p_u	1	1	1	1	1	1

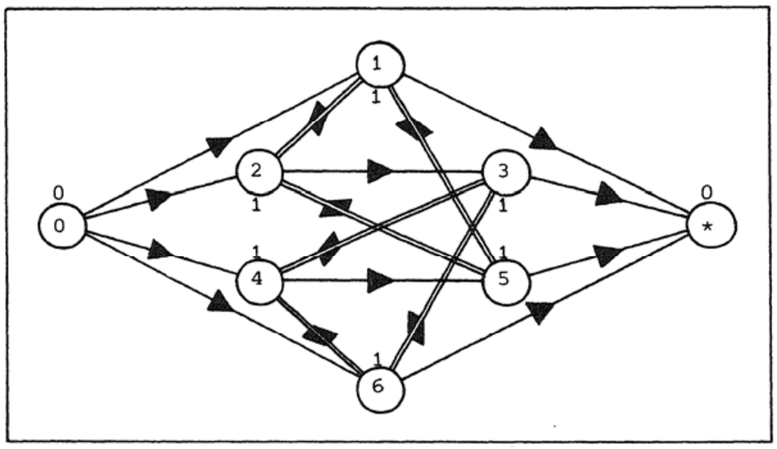


Figure 13.3 Disjunctive graph $G = (V, C \cup D)$ for the example.

The following theorem identifies some cases in which the absence of cycles is guaranteed for both descendants.

THEOREM 13.5. $G(D_{uv})$ *and* $G(D_{vu})$ *are acyclic if any of the following conditions holds.*

(*a*) $P^*_{uv} > 0$;

(*b*) $|P_{uv}-P_{vu}| < p_u+p_v$;

(*c*) $P_{vu} > P_{uv} > r_v-R_v$;

(*d*) $P_{uv} > P_{vu} > r_u-R_u$.

Proof.

(*a*) If $P^*_{uv} > 0$, then $\{u,v\} \in C(D)$ since

$$r_u+p_u-r_v \geq r_u+p_u-R_v = P_{uv} > 0,$$
$$r_v+p_v-r_u \geq r_v+p_v-R_u = P_{vu} > 0,$$

and we can apply Theorem 13.2.

(*b*) If G(D) contains a path from v to u, we have

$$p_u \leq (r_u+p_u)-(r_v+p_v),$$
$$p_v \leq R_u \quad - R_v.$$

Addition of these inequalities yields

$$p_u+p_v \leq P_{uv}-P_{vu} \leq |P_{uv}-P_{vu}| \tag{13.1}$$

which contradicts condition (*b*). The existence of a path from u to v leads to a similar contradiction, and hence both $G(D_{uv})$ and $G(D_{vu})$ are acyclic.

(*c*) If G(D) contains a path from v to u, then it follows from (13.1) that we have

$$P_{vu} < P_{uv}.$$

The existence of a path from u to v implies that

$$P_{uv} = r_u+p_u-R_v \leq r_v-R_v.$$

(*d*) Analogous to (*c*). □

The above discussion leads to various strategies for selecting a branching pair $\{u^*,v^*\}$. These branching rules are outlined below. In all cases, we restrict our attention to the set

$$\{\{u,v\} | \mu_u = \mu_v = M_\ell, \{(u,v),(v,u)\} \subset \mathcal{D}-D\}$$

where the machine M_ℓ on which at least one conflict exists has been found by checking the M_k for conflicts in order of nondecreasing $LB^*_k(D)$.

B1. $\{u^*,v^*\}$ maximizes $\min\{r_u+p_u-r_v, r_v+p_v-r_u\}$ over all $\{u,v\}$.

B2. $\{u^*,v^*\}$ maximizes P^*_{uv} over all $\{u,v\} \in C(D)$.

B3. $\{u^*,v^*\}$ maximizes P^*_{uv} over all $\{u,v\}$ which satisfy any of the following conditions:

(*a*) $\{u,v\} \in C(D)$;

(*b*) $|P_{uv}-P_{vu}| < p_u+p_v$;

(*c*) $P_{vu} > P_{uv} > r_v-R_v$;

(*d*) $P_{uv} > P_{vu} > r_u-R_u$.

B4. $\{u^*,v^*\}$ maximizes P^*_{uv} over all $\{u,v\}$.

Branching rules B1, B2 and B3 guarantee that both $G(D_{u^*v^*})$ and $G(D_{v^*u^*})$ are acyclic; in case of rule B4, possible cycles will be detected during the calculation of r_* in the descendant nodes.

13.2.3. *Implementations*

(i) *algorithm GAS*

Algorithm GAS combines lower bound $LB^*(D)$ with enumeration scheme (i). We implemented a recursive depth-first search, choosing the descendant nodes in order of nondecreasing lower bounds.

It appears useful to find a feasible solution heuristically at some or all nodes of the search tree in order to adjust the upper bound. We tested the following possibilities.

(*A*) Strategy UB0 makes no heuristic attempts to adjust the upper bound.

(*B*) Strategy UB1 evaluates the one-machine schedules obtained during the calculation of $LB^*(D)$ as one overall schedule at every node of the tree.

(*C*) Strategy UB2 applies a priority rule at every node. This rule constructs an active schedule according to algorithm AS2, whereby highest priority is granted to the scheduleable $O_u \in Q$ minimizing

$$t_u + \max\{\textstyle\sum_{\mu_v=\mu_u, O_v\in T} p_v,\ p_u + \max\{\sum_{\iota_w=\iota_v, O_w\in T} p_w \,|\, O_v \in Q-\{O_u\}\}\}.$$

(*D*) Strategy UB3 involves the use of this heuristic at four equidistant levels of the tree.

(ii) *algorithm SEC*

Algorithm SEC combines lower bound $LB^*(D)$ with enumeration scheme (ii) and upper bounding strategy UB1. The branching rules B1, B2, B3 and B4 were implemented using two recursive search strategies S1 and S2, which choose the descendant nodes according to nondecreasing $r_u+p_u-r_v$ and nondecreasing P_{uv} respectively.

13.3. Computational experience

13.3.1. *Test problems*

The two approaches sketched in Section 13.2.3 were tested on three problems, two of which appear in the literature. The data for these problems are presented in Table 13.2.

13.3.2. *Results*

Algorithms GAS and SEC were coded in ALGOL 60 and run on the Control Data Cyber 73-28 of the SARA Computing Centre in Amsterdam.

Table 13.3 shows the results obtained with algorithm GAS. Trying to adjust the upper bound at every node of the search tree appears to be too time-consuming; especially strategy UB1 performed rather badly. The best results were obtained with strategy UB3, applying the priority rule at four levels of the tree.

Table 13.4 shows the results obtained with algorithm SEC. The branching strategies B3 and B4 performed very poorly in some initial testing; the choice of B1 or B2 has only a minor influence on the algorithm's performance. On the other hand, the search strategy S2 based on the penalties P_{uv} is clearly superior to search strategy S1 based on $r_u+p_u-r_v$.

Altogether, algorithm SEC is clearly worse than algorithm GAS. For somewhat larger problems (*e.g.*, the $10|10|G|C_{max}$ and $20|5|G|C_{max}$ problems from [Muth & Thompson 1963, 236-237]) both algorithms failed to produce an optimal schedule within five minutes of running time. In view of the fact that previous experiments [McMahon & Florian 1975] confirm that algorithm GAS is the currently best $n|m|G|C_{max}$ algorithm, this clearly indicates that in spite of some progress a large amount of work remains to be done.

TABLE 13.2. TEST PROBLEMS

problem	130404			130504			360606		
n	4			5			6		
m	4			4			6		
optimum	35			13			55		
source				[Németi 1964]			[Muth & Thompson 1963, 236]		
u	ι'_u	μ'_u	p_u	ι'_u	μ'_u	p_u	ι'_u	μ'_u	p_u
1	1	1	6	1	1	2	1	3	1
2	1	3	9	1	2	3	1	1	3
3	1	4	5	2	1	3	1	2	6
4	2	2	7	2	3	3	1	4	7
5	2	3	6	2	2	2	1	6	3
6	2	4	7	3	1	1	1	5	6
7	3	2	3	3	2	3	2	2	8
8	3	1	7	3	4	2	2	3	5
9	3	4	6	4	1	4	2	5	10
10	3	3	4	4	4	1	2	6	10
11	4	2	9	4	3	3	2	1	10
12	4	1	6	5	4	4	2	4	4
13	4	4	5	5	3	4	3	3	5
14							3	4	4
15							3	6	8
16							3	1	9
17							3	2	1
18							3	5	7
19							4	2	5
20							4	1	5
21							4	3	5
22							4	4	3
23							4	5	8
24							4	6	9
25							5	3	9
26							5	2	3
27							5	5	5
28							5	6	4
29							5	1	3
30							5	4	1
31							6	2	3
32							6	4	3
33							6	6	9
34							6	1	10
35							6	5	4
36							6	3	1

ι'_u and μ'_u denote the indices of ι_u and μ_u respectively, *i.e.*, $\iota_u = J_{\iota'_u}$ and $\mu_u = M_{\mu'_u}$.

TABLE 13.3. RESULTS FOR ALGORITHM GAS

problem	solution time				number of nodes			
	UB0	UB1	UB2	UB3	UB0	UB1	UB2	UB3
130404	.21	.27	.45	.35	19	8	8	8
130504	.30	.28	.28	.21	22	11	5	5
360606	5.39	9.15	6.87	2.83	279	279	62	62

TABLE 13.4. RESULTS FOR ALGORITHM SEC

	problem	solution time		number of nodes	
		S1	S2	S1	S2
B1	130404	.92	.93	23	23
	130504	.40	.32	13	11
	360606	29.88	15.91	347	175
B2	130404	.59	.58	15	15
	130504	.51	.28	17	11
	360606	36.39	15.24	411	181

LEGEND TO TABLES 13.3,4

solution time : CPU seconds on a Control Data Cyber 73-28.
number of nodes : including eliminated nodes.
algorithm GAS : see Section 13.2.3;
     UB : upper bounding strategy.
algorithm SEC : see Section 13.2.3;
     B : branching strategy;
     S : search strategy.

13.4. Remarks

The pessimistic prediction by Conway, Maxwell and Miller, quoted in Section 13.1, seems to have lost little of its validity. Only very small problems can be solved optimally within reasonable time, the main reason being that the lower bound $LB^*(D)$, though the strongest one available, is still too weak to prune large parts of the search tree at an early stage.

In further research on the job-shop problem, the search for stronger bounds deserves priority. One might try to develop two-machine bounds, comparable to those presented in Chapter 12, and, again, a subgradient approach seems worth investigating.

Although our results so far hardly confirm this, we do feel that a flexible branching rule that reveals essential conflicts in the problem under consideration should be more effective than the rigid one used in algorithm GAS. Additional work is needed to provide more accurate indicators than P_{uv} on which to base a branching decision.

Part IV. Some applications

14. APPLICATIONS OF THE TRAVELLING SALESMAN PROBLEM

14.1. Introduction

In this chapter we discuss four apparently unrelated problems that arise in the context of *computer wiring*, *vehicle routing*, *clustering a data array* and *job-shop scheduling with no wait in process*. It turns out that each of these problems can be formulated as a travelling salesman problem (TSP). Three of them originated from real-world situations and were not immediately recognized as TSPs; use of TSP algorithms led to better solutions, as will be illustrated below.

Moreover, not only are the four problems special cases of the TSP, but the TSP can conversely be interpreted as a special case of any of these problems. Formulation as a TSP thus is essentially the simplest way to solve them. The equivalence of the last two problems to the TSP is nontrivial and will be discussed in Sections 14.4.4 and 14.5.4.

The TSP has been introduced in Section 3.3 and solution methods have been surveyed in Chapter 9. Here, we shall be using the following algorithms:

(i) algorithm LIN, *i.e.* a heuristic procedure for generating 3-*optimal* tours for symmetric TSPs, implementing the enumeration scheme given in [Lin 1965] with deletion of some superfluous checks for improvement (see also Chapter 7);

(ii) algorithm LEA, *i.e.* a branch-and-bound procedure based on [Little *et al.* 1963], incorporating an improved branching rule that allows early pruning of a branch through sufficiently large penalties (see Section 9.2.3);

(iii) algorithm HK1, *i.e.* a branch-and-bound procedure for symmetric TSPs, based on [Held & Karp 1971] and algorithm LIN (see Section 9.2.3).

14.2. Computer wiring

14.2.1. *Problem description*

The following problem arises frequently during the design of computer interfaces at the Institute for Nuclear Physical Research in Amsterdam.

An interface consists of a number of modules, and on each module several pins are located. The position of each module has been determined in advance.

A given subset of pins has to be interconnected by wires. In view of possible future changes or corrections and of the small size of the pin, at most two wires are to be attached to any pin. In order to reduce signal cross-talk and to improve ease and neatness of wiring, the total wire length has to be minimized.

14.2.2. *Formulation as a TSP*

Let $W = \{1,\ldots,n\}$ denote the set of pins to be interconnected, c_{ij} the distance between pin i and pin j, and H the complete undirected graph on the vertex set W with weights c_{ij} on the edges.

If any number of wires could be attached to a pin, an optimal wiring would correspond to a minimum *spanning tree* on H, which can be found efficiently by the algorithms in [Kruskal 1956] or [Prim 1957; Dijkstra 1959]. However, the degree restriction implies that we have to find a minimum *hamiltonian path* on H. This problem corresponds to finding a minimum *hamiltonian circuit* on G with $V = \{0,\ldots,n\}$ and $c_{i0} = c_{0i} = 0$ for all $i \in V$. In this way the wiring problem can be converted into a symmetric TSP.

A more difficult problem occurs if the positions of the modules have not been fixed in advance but can be chosen so as to minimize the total wire length for all subsets of pins that have to be interconnected. For a review of this placement problem, which is related to the quadratic assignment problem, we refer to [Hanan & Kurtzberg 1972].

14.2.3. *Results*

The procedure that was used originally produced clearly non-optimal wiring schemes like the example with two subsets of pins in Figure 14.1(*a*). The size and number of the problems was such that algorithm LIN had to be used. The 3-optimal results on the example are given in Figure 14.1(*b*).

More examples and details about the computer implementation can be found in [Visschers & Ten Kate 1973].

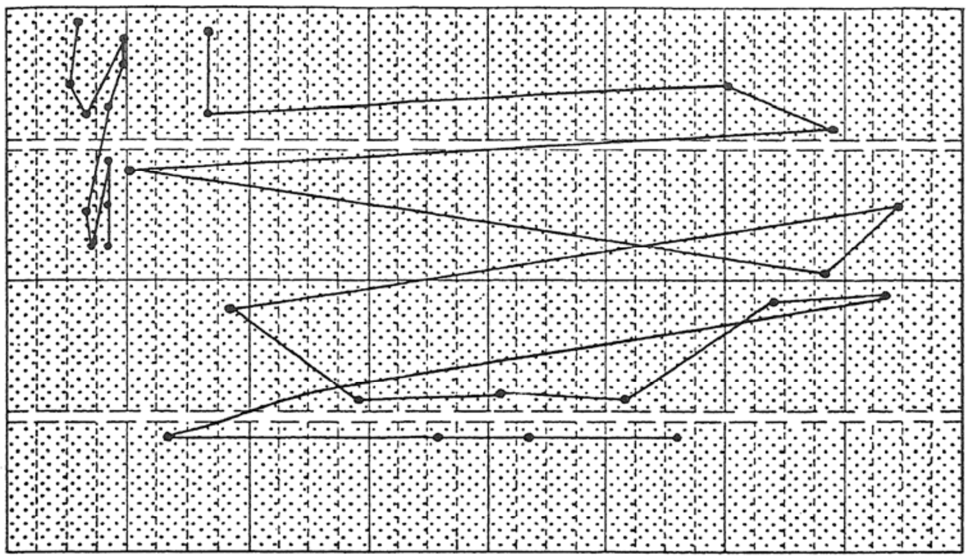


Figure 14.1(*a*) Wiring without optimization.

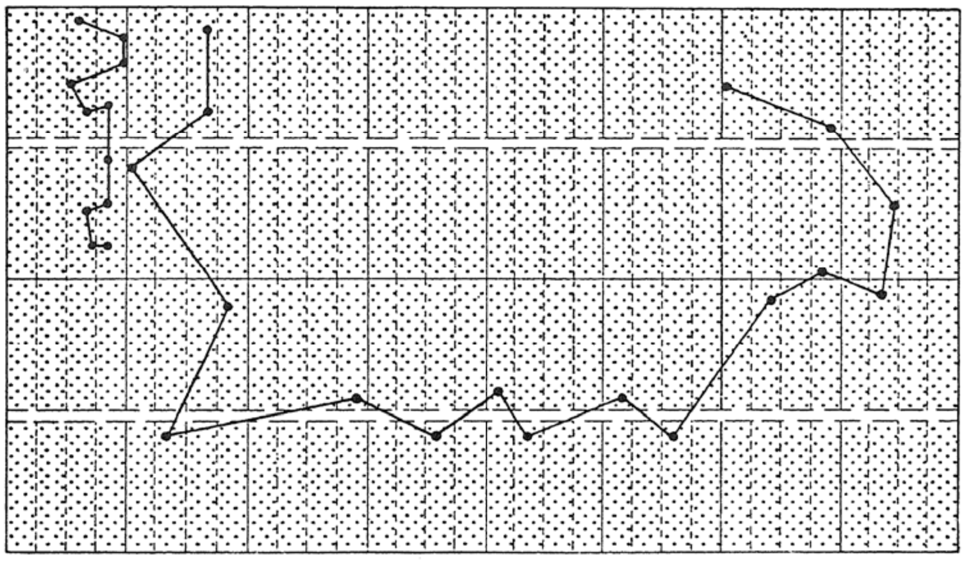


Figure 14.1(*b*) 3-Optimal wiring.

14.3. Vehicle routing

14.3.1. *Problem description*

In 28 towns in the Dutch province of North-Holland telephone boxes have been installed by the national postal service (PTT). A technical crew has to visit each telephone box once or twice a week to empty the coin box and, if necessary, to replace directories and perform minor repairs. Each working day of at most 445 minutes begins and ends in the provincial capital Haarlem. The problem is to minimize the number of days in which all telephone boxes can be visited and the total travelling time.

A similar problem arose in the city of Utrecht. Here about 200 mail boxes have to be emptied each day within a period of one hour by trucks operating from the central railway station. The problem is to find the minimum number of trucks able to do this and the associated minimum travelling time.

14.3.2. *Formulation as a TSP*

Both problems are types of classical *vehicle routing problems* (VRP). They will be denoted by P1 and P2 respectively, and can be characterized more formally as follows.

- n cities i (i = 1,...,n) (the *customers*) are to be visited
  [P1: 28 towns; P2: 200 mail boxes]
- by m *vehicles*
  [P1: m working days; P2: m trucks]
- operating from city 0 (the *depot*)
  [P1: Haarlem; P2: Utrecht, central railway station];
- the travelling time between cities i and j is $d_{ij} = d_{ji}$ minutes, for $i,j \in \{0,\dots,n\}$;
- the time to be spent in city i is e_i minutes, for $i \in \{1,\dots,n\}$
  [P1: 8 × number of telephone boxes in town i; P2: 1];
- there are *global constraints*, imposed by the vehicles, *e.g.*, the maximum allowable time for any vehicle to complete its route is f minutes
  [P1: 445; P2: 60];
- there may be *local constraints*, imposed by the customers
  [P1: one town (nr.28, Den Helder) has to be visited twice on different days];

- *criteria* by which solutions are judged are:

  U , the number of vehicles used;

  T(U), the total time used for U vehicles.

If a city has to be visited twice, it is duplicated, appropriate travelling and visiting times are added, and n is increased by one.

[P1: Den Helder is split up into two cities 28 and 29; $d_{28,29} := \infty$; $n := 29$.]

We replace the depot (city 0) by m artificial depots (cities n+1,...,n+m) and extend the definition of (d_{ij}) and (e_i) as follows (*cf.* Figure 14.2):

$$\begin{aligned}
d_{i,n+\ell} &= d_{i0} && \text{for } \ell = 1,\dots,m;\\
d_{n+k,j} &= d_{0j} && \text{for } k = 1,\dots,m;\\
d_{n+k,n+\ell} &= \lambda && \text{for } k,\ell = 1,\dots,m;\\
e_{n+k} &= 0 && \text{for } k = 1,\dots,m.
\end{aligned}$$

$$\begin{array}{r|ccccc|ccc|}
 & 1 & \dots & j & \dots & n & n+1 & \dots & n+m \\
\hline
1 & d_{11} & \cdots & d_{1j} & \cdots & d_{1n} & d_{10} & \cdots & d_{10} \\
\vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
i & d_{i1} & \cdots & d_{ij} & \cdots & d_{in} & d_{i0} & \cdots & d_{i0} \\
\vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
n & d_{n1} & \cdots & d_{nj} & \cdots & d_{nn} & d_{n0} & \cdots & d_{n0} \\
\hline
n+1 & d_{01} & \cdots & d_{0j} & \cdots & d_{0n} & \lambda & \cdots & \lambda \\
\vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\
n+m & d_{01} & \cdots & d_{0j} & \cdots & d_{0n} & \lambda & \cdots & \lambda \\
\hline
\end{array}$$

Figure 14.2 The matrix (d_{ij}).

We obtain a symmetric euclidean TSP by defining $V = \{1,\dots,n+m\}$ and $c_{ij} = \frac{1}{2}e_i + d_{ij} + \frac{1}{2}e_j$ for all $i,j \in V$. A salesman's tour is feasible for the VRP provided that the additional global and local constraints are respected. If a TSP solution contains m-U links between artificial depots, then the corresponding VRP solution uses only U vehicles. Adding another vehicle decreases the number of links between artificial depots by one and hence the objective function by λ. Thus, $-\lambda$ may be interpreted as the cost of a vehicle. We may now consider three possible choices of λ:

- $\lambda = +\infty$ will lead to $\min_\pi\{T(m)\}$,

  i.e. the minimum total time for m vehicles (*cf.* [Eilon *et al.* 1971, 188]);
- $\lambda = 0$ will lead to $\min_\pi\{T(U) \mid U = 1,\dots,m\}$,

  i.e. the minimum total time for any number of vehicles (*cf.* [Eilon *et al.* 1971, 188]);

- $\lambda = -\infty$ will lead to $\min_\pi\{T(\min\{U|U = 1,\dots,m\})\}$,

  i.e. the minimum total time for the minimum number of vehicles.

The latter objective is the criterion function for both P1 and P2.

An appropriate method for obtaining good VRP solutions is the following.

- Choose an initial tour which satisfies the VRP constraints.
- Apply an iterative procedure for improving the tour and check the constraints whenever a possible decrease in tour length occurs.

14.3.3. *Results*

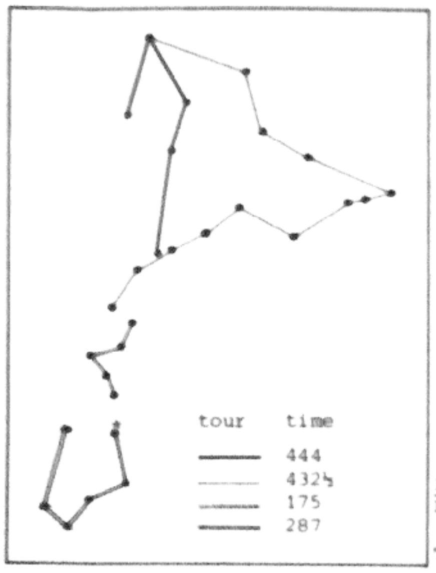
Figures 14.3 and 14.4 illustrate some results, obtained for P1 and P2 by J. Berendse and J.H. Kuiper from PTT. In both figures, the links with the depot, indicated by *, are not shown.

For P1, algorithm LIN was used. All 3-optimal solutions obtained require four days, representing a 50 per cent decrease with respect to the schedule that was previously used. An example is given in Figure 14.3(*a*). Exchanging three links in this solution resulted in the schedule given in Figure 14.3(*b*); it involves only three days, including however one of $449\frac{1}{2}$ minutes. Computational experience revealed that the heuristic procedure converged much faster with $\lambda = -\infty$ than with $\lambda = 0$. More details about this application can be found in [Kuiper 1973].

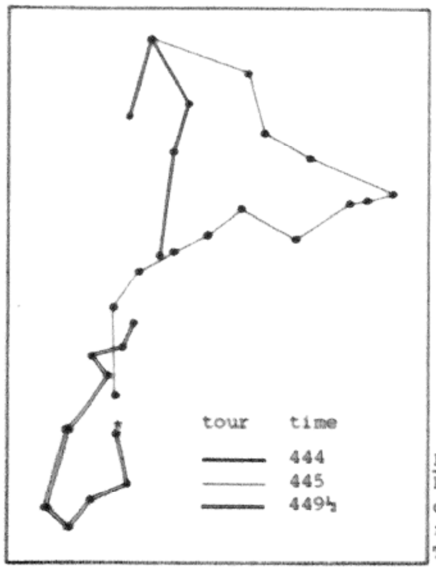
For P2, a variation on algorithm LIN was used, whereby only a limited number of promising potential improvements was checked. The number of trucks needed was reduced from ten (Figure 14.4(*a*)) to eight (Figures 14.4(*b,c,d*)). In view of the size of the problem, both possibilities $\lambda = 0$ and $\lambda = -\infty$ have been run only once; in this incidental case, the convergence with $\lambda = -\infty$ was relatively slow.

14.3.4. *Remarks*

In view of the usual size of practical routing problems, the variety of the additional constraints and the fundamental complexity of the TSP, we depend on approximate algorithms for obtaining satisfactory solutions to the VRP. Many heuristics have been developed, both for constructing a good initial tour and for improving it in a systematic way. For surveys of the literature



<u>Figure 14.3(*a*)</u>
P1: 3-optimal solution;
$\lambda = -\infty$;
T(4) = 1338½.



<u>Figure 14.3(*b*)</u>
P1: infeasible solution,
obtained by hand
from Figure 14.3(*a*);
T(3) = 1338½.

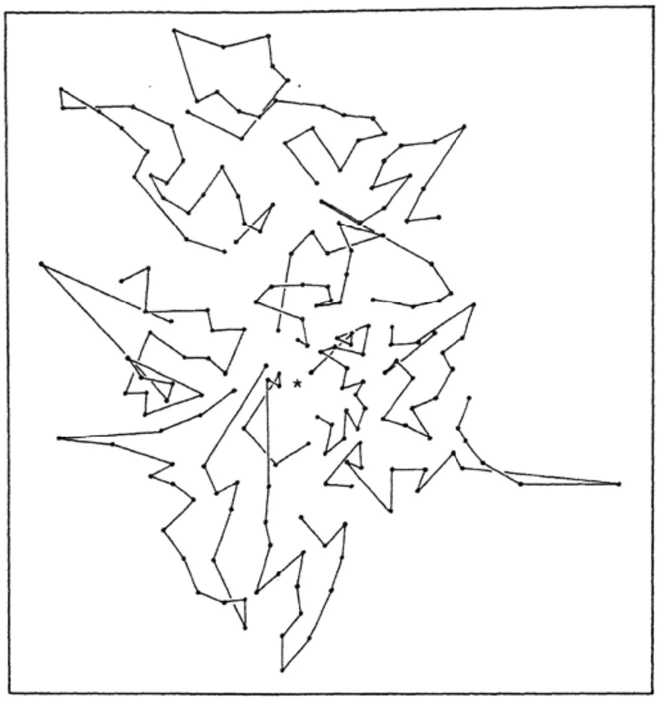


Figure 14.4(*a*)
P2: previously used solution;
T(10) = 442.

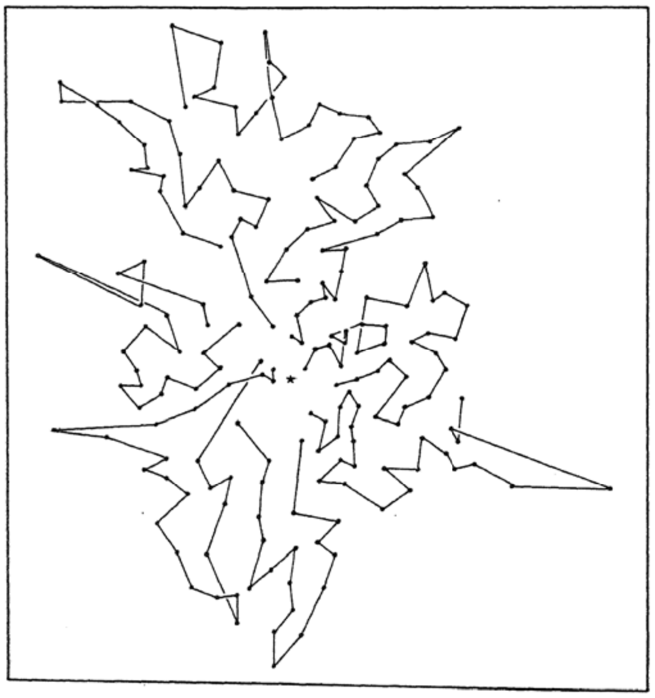


Figure 14.4(*b*)
P2: locally optimal solution, starting from Figure 14.4(*a*);
λ = 0;
T(8) = 404.

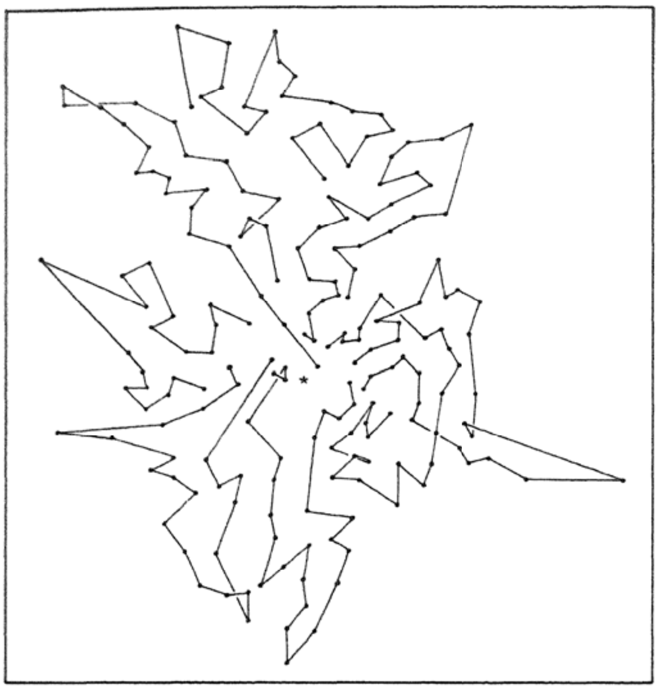


Figure 14.4(*c*)
P2: locally optimal solution, starting from Figure 14.4(*a*);
$\lambda = -\infty$
T(8) = 405.

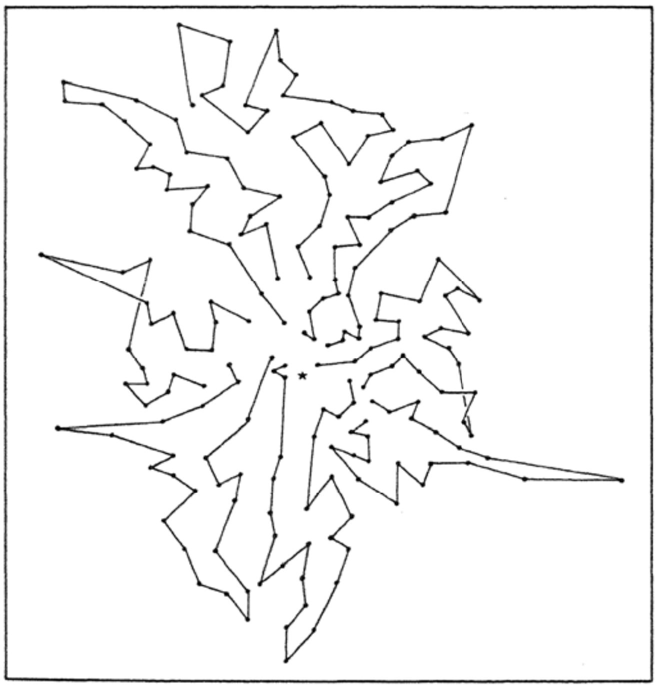


Figure 14.4(*d*)
P2: locally optimal solution, starting from an improvement by hand on Figure 14.4(*c*);
$\lambda = -\infty$;
T(8) = 398.

we refer to [Eilon *et al.* 1971, Ch.9; Christofides 1974]. Without going into detail we remark here that some recently proposed sophisticated methods which are based on the euclidean nature of the VRP [Wren & Holliday 1972; Gillett & Miller 1974; Jonker 1974], perform excellently on standard test problems but seem to fail in handling local constraints appropriately.

An interesting variation on the VRP arises in the context of money collection at post offices. For security reasons, several good routes have to be available. The problem is then equivalent to the *peripatetic salesman problem* where m edge-disjoint hamiltonian circuits of minimum total weight are sought [Krarup 1975]. No algorithms for this problem have been proposed so far.

An important extension of the VRP is the *general routing problem* (GRP), where m vehicles have to be routed on a graph $G = (V,A)$, thereby traversing a subset $W \subset V$ of *required vertices* and a subset $B \subset A$ of *required arcs* [Orloff 1974A; Orloff 1974B; Lenstra & Rinnooy Kan 1974]. The GRP specializes to the TSP for $m = 1$, $W = V$ and $B = \emptyset$, and to the *Chinese postman problem* (CPP) for $m = 1$, $W = \emptyset$ and $B = A$. Since the TSP is NP-complete and the CPP is efficiently solvable, it seems advantageous to convert required vertices to required arcs as far as possible. Such conversions lead to suboptimal but very satisfactory results.

14.4. Clustering a data array

14.4.1. *Problem description*

Suppose that a *data array* (a_{ij}) $(i \in R, j \in S)$ is given, where a_{ij} measures the strength of the relationship between elements $i \in R$ and $j \in S$. A *clustering* of the array is obtained by permuting its rows and columns and should identify subsets of R that are strongly related to subsets of S.

This situation occurs in widely different contexts. Here we will apply a clustering technique to three examples. In the first one [McCormick *et al.* 1972] R is a collection of 24 *marketing* techniques, S is a collection of 17 marketing applications, $a_{ij} = 1$ if technique i has been successfully used for application j, and $a_{ij} = 0$ otherwise. The second example [McCormick *et al.* 1972] arises in *airport design*; R (= S) is a set of 27 control variables and a_{ij} measures their interdependence. The third example [Roes 1973] deals with *import-export analysis*; R (= S) is a set of 50 regions of the

Indonesian islands, $a_{ij} = 1$ if in 1971 a quantity of at least 50 tons of rice was transported from region i to region j, and $a_{ij} = 0$ otherwise.

These three examples indicate that the approach is useful for *problem decomposition* and *data reorganization*. A more elaborate discussion of its applicability and further examples can be found in [McCormick *et al.* 1972].

To convert this problem into an optimization problem, some criterion has to be defined. In [McCormick *et al.* 1972], the proposed *measure of effectiveness* (ME) is the sum of all products of horizontally or vertically adjacent elements in the array. Figure 14.5 shows how this criterion relates to various permutations of a 4×4 array. The problem is to find permutations of rows and columns of (a_{ij}) maximizing ME.

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 1 & 0 \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 0 & 1 & 0 \\ 3 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 3 & 2 & 4 \\ \hline 1 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 0 \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 3 & 2 & 4 \\ \hline 1 & 1 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 1 & 1 \\ \hline \end{array}$$

$$\text{ME} = 0 \qquad \text{ME} = 2 \qquad \text{ME} = 4 \qquad \text{ME} = 6 \qquad \text{ME} = 8$$

Figure 14.5 ME for various permutations of a 4×4 array.

14.4.2. *Formulation as a TSP*

Let $R = \{1,\dots,r\}$ and $S = \{1,\dots,s\}$. With the conventions

$$\rho(0) = \rho(r+1) = \sigma(0) = \sigma(s+1) = 0,$$
$$a_{i0} = a_{0j} = 0 \quad \text{for } i \in R,\ j \in S,$$

the ME, corresponding to permutations ρ of R and σ of S, is given by

$$\begin{aligned} &\text{ME}(\rho,\sigma) \\ &= \tfrac{1}{2} \textstyle\sum_{i\in R} \sum_{j\in S} a_{\rho(i)\sigma(j)} \left(a_{\rho(i)\sigma(j-1)} + a_{\rho(i)\sigma(j+1)} + a_{\rho(i-1)\sigma(j)} + a_{\rho(i+1)\sigma(j)}\right) \\ &= \textstyle\sum_{j=0}^{s} \sum_{i\in R} a_{i\sigma(j)} a_{i\sigma(j+1)} + \sum_{i=0}^{r} \sum_{j\in S} a_{\rho(i)j} a_{\rho(i+1)j} \\ &= \text{ME}(\sigma) + \text{ME}(\rho), \end{aligned}$$

so ME(ρ,σ) decomposes into two parts, and its maximization reduces to two separate and similar optimizations, one of ME(σ) for the columns and the other of ME(ρ) for the rows. It is stated in [McCormick *et al.* 1972] that both subproblems may be rewritten as quadratic assignment problems. More precisely, they are symmetric TSPs:

$$\text{TSP}^{\text{col}}\text{:}\;\; V^{\text{col}} = \{0,\ldots,s\},\;\; c^{\text{col}}_{jk} = -\textstyle\sum_{i\in R} a_{ij}a_{ik} \;\text{ for } j,k \in V^{\text{col}},$$
$$\text{TSP}^{\text{row}}\text{:}\;\; V^{\text{row}} = \{0,\ldots,r\},\;\; c^{\text{row}}_{hi} = -\textstyle\sum_{j\in S} a_{hj}a_{ij} \;\text{ for } h,i \in V^{\text{row}},$$

for ME(σ) and ME(ρ), respectively (*cf.* [Lenstra 1974; Carvajal *et al.* 1974]). In general, the clustering problem for a p-dimensional array can be stated as p TSPs. It may be attacked by any algorithm for the TSP; in fact, the *bond energy algorithm* (BEA), proposed in [McCormick *et al.* 1972], is a simple suboptimal TSP method which constructs a tour by successively inserting the cities (*cf.* [Müller-Merbach 1970, 76]).

If the data array is symmetric (*i.e.* $a_{ij} = a_{ji}$ for all i,j), then TSP^{row} and TSP^{col} are identical and only one optimization needs to be performed (see the airport example).

If the data array is square (*i.e.* r = s) but not necessarily symmetric and we want to have equal permutations of rows and columns (*i.e.* $\rho = \sigma$), then one symmetric TSP results:

$$\text{TSP}^{\text{cow}}\text{:}\;\; V^{\text{cow}} = V^{\text{col}} = V^{\text{row}},\;\; c^{\text{cow}}_{ij} = c^{\text{col}}_{ij} + c^{\text{row}}_{ij} \;\text{ for } i,j \in V^{\text{cow}}$$

(see the import-export example).

The size of the TSPs might be reduced by assigning identical rows or columns to one single city under the assumption that these rows or columns will be adjacent in at least one optimal solution. This assumption is justified under the conditions expressed by the following theorem.

THEOREM 14.1. *If* $a_{ij} \in \{0,1\}$ *for all* $i \in R$, $j \in S$, *and* $c^{\text{row}}_{kk} = c^{\text{row}}_{k\ell} = c^{\text{row}}_{\ell\ell}$ *for some* $k,\ell \in V^{\text{row}}$, *then row* k *and row* ℓ *are identical, and adjacent in at least one optimal solution to* TSP^{row}.

Proof. We define $S_i = \{j | j \in S,\, a_{ij} = 1\}$ for all $i \in V^{\text{row}}$. Since $a_{ij} \in \{0,1\}$ for all $i \in R$, $j \in S$, we have

$$c^{\text{row}}_{ij} = -|S_i \cap S_j| \;\text{ for all } i,j \in V^{\text{row}}, \tag{14.1}$$

and $c^{\text{row}}_{kk} = c^{\text{row}}_{k\ell} = c^{\text{row}}_{\ell\ell}$ implies that $S_k = S_k \cap S_\ell = S$. Hence row k and row ℓ are identical:

$$a_{kj} = a_{\ell j} \;\text{ for all } j \in S. \tag{14.2}$$

Now consider any permutation ρ of R with $\rho(p) = k$, $\rho(q) = \ell$, $|p-q| > 1$. Insert ℓ between k and $\rho(p+1)$. This will not decrease ME(ρ) if

$$c^{\text{row}}_{k\rho(p+1)} + c^{\text{row}}_{\rho(q-1)\ell} + c^{\text{row}}_{\ell\rho(q+1)} \geq c^{\text{row}}_{k\ell} + c^{\text{row}}_{\ell\rho(p+1)} + c^{\text{row}}_{\rho(q-1)\rho(q+1)}.$$

By (14.1) and (14.2), this is equivalent to

$$|S_{\rho(q-1)} \cap S_\ell| + |S_\ell \cap S_{\rho(q+1)}| \le |S_\ell| + |S_{\rho(q-1)} \cap S_{\rho(q+1)}|,$$

which is true, since

$$\begin{aligned} &|S_{\rho(q-1)} \cap S_\ell| + |S_\ell \cap S_{\rho(q+1)}| \\ &= |S_\ell \cap (S_{\rho(q-1)} \cup S_{\rho(q+1)})| + |S_\ell \cap S_{\rho(q-1)} \cap S_{\rho(q+1)}| \\ &\le |S_\ell| + |S_{\rho(q-1)} \cap S_{\rho(q+1)}|. \end{aligned}$$ □

Analogous theorems hold for TSP^{col} and TSP^{cow}. Defining $R_j = \{i | i \in R, a_{ij} = 1\}$ for all $j \in V^{col}$, we have in the latter case

$$c_{ij}^{cow} = -|S_i \cap S_j| - |R_i \cap R_j| \quad \text{for all } i,j \in V^{cow}, \tag{14.3}$$

and we have to show that

$$\begin{aligned} a_{kj} &= a_{\ell j} \quad \text{for all } j \in S, \\ a_{ik} &= a_{i\ell} \quad \text{for all } i \in R. \end{aligned} \tag{14.4}$$

It follows from (14.3) and $c_{kk}^{cow} = c_{k\ell}^{cow} = c_{\ell\ell}^{cow}$ that $|S_k|+|R_k| = |S_k \cap S_\ell|+|R_k \cap R_\ell| = |S_\ell|+|R_\ell|$. If $|S_k| > |S_k \cap S_\ell|$, then $|R_k| < |R_k \cap R_\ell|$, which is impossible; hence $|S_k| = |S_k \cap S_\ell| = |S_\ell|$ and $|R_k| = |R_k \cap R_\ell| = |R_\ell|$, which trivially leads to (14.4).

These results cannot be generalized to cover the case where a_{ij} can take on other values than 0 or 1. For example, if $R = \{1,2,3\}$ and $a_{1j} = a_{2j} = 1$, $a_{3j} = 2$ for $j \in S$, then the identical rows 1 and 2 are separated by row 3 in the optimal solution.

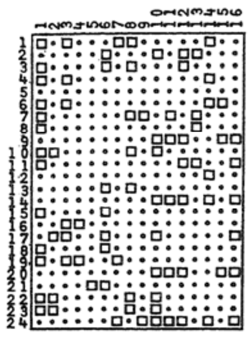
14.4.3. *Results*

The techniques and applications pertaining to the marketing example are given in Table 14.1. Figure 14.6 shows the initial data array, the clustering produced by the BEA as reported in [McCormick *et al.* 1972], and a clustering corresponding to optimal solutions of TSP^{col} and TSP^{row}, found by algorithm LEA after application of Theorem 14.1. It turns out that the BEA clustering is optimal.

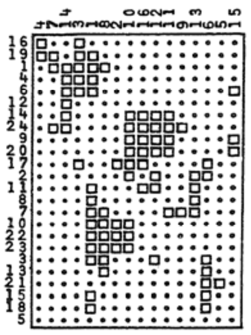
The control variables in the airport example are given in Table 14.2. Figure 14.7 shows the symmetric initial data array, the BEA clustering [McCormick *et al.* 1972], and a clustering corresponding to an optimal solution of TSP^{col} (= TSP^{row}), found by algorithm HK1. The BEA clustering is not

TABLE 14.1. MARKETING EXAMPLE

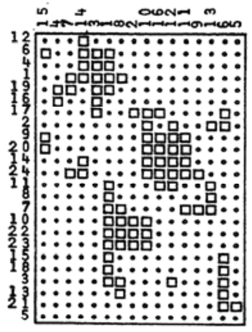
Marketing techniques
1. Regression & correlation analysis
2. Discounted cash flow
3. Incremental analysis
4. Multiple regression/correlation
5. Random sampling
6. Sampling theory
7. Bayesian approach
8. Cost-benefit analysis
9. Critical path method
10. Decision trees
11. Dynamic programming
12. Exponential smoothing
13. Industrial dynamics
14. Input-output analysis
15. Linear programming
16. Markov processes
17. Monte Carlo simulation
18. Nonlinear programming
19. Numerical taxonomy
20. PERT
21. Queueing models
22. Risk analysis
23. Sensitivity analysis
24. Technological forecasting
Marketing applications
1. Advertising research
2. Acquisition screening
3. Brand strategy
4. Customer segmentation
5. Customer service
6. Distribution planning
7. Market segmentation
8. Pricing strategy
9. Product life-cycle analysis
10. Product line analysis
11. Product planning
12. R&D planning
13. ROI analysis
14. Sales forecasting
15. Test marketing
16. Venture planning



(*a*) Initial array; ME = 39.



(*b*) BEA clustering; ME = 97.

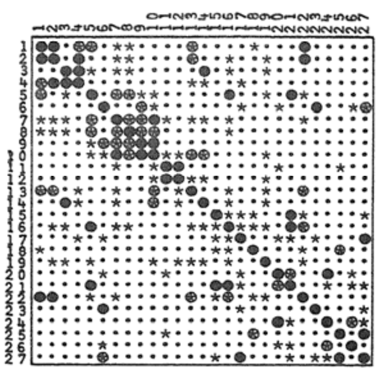


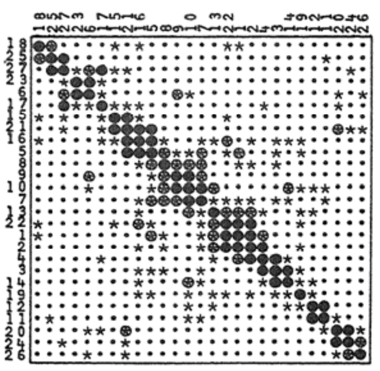
(*c*) Optimal clustering; ME = 97.

Figure 14.6 Marketing example; • = 0, □ = 1.

TABLE 14.2. AIRPORT EXAMPLE

Control variables
1. Passenger check-in
2. Baggage check-in
3. Baggage claim
4. Baggage moving system
5. Intra-airport transportation system
6. Cargo terminal
7. Close-in parking lots
8. Remote parking lots
9. Main access roads to and from airport
10. Circulation roads within airport
11. Service area for rental cars
12. Parking lots for rental cars
13. Curb space for unloading
14. Curb space for loading
15. Waiting areas at gates
16. Stations for intra-airport transportation system
17. Aircraft loading system
18. Concessions
19. Rental car desk
20. Runway capacity
21. Number of gates
22. Passenger information
23. Cargo transfer
24. Air-traffic-control system
25. Refuse removal
26. Flight operations and crew facilities
27. Aircraft service on the apron


(*a*) Initial array; ME = 592.


(*b*) BEA clustering; ME = 1154.

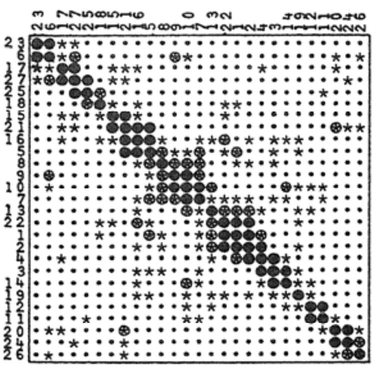
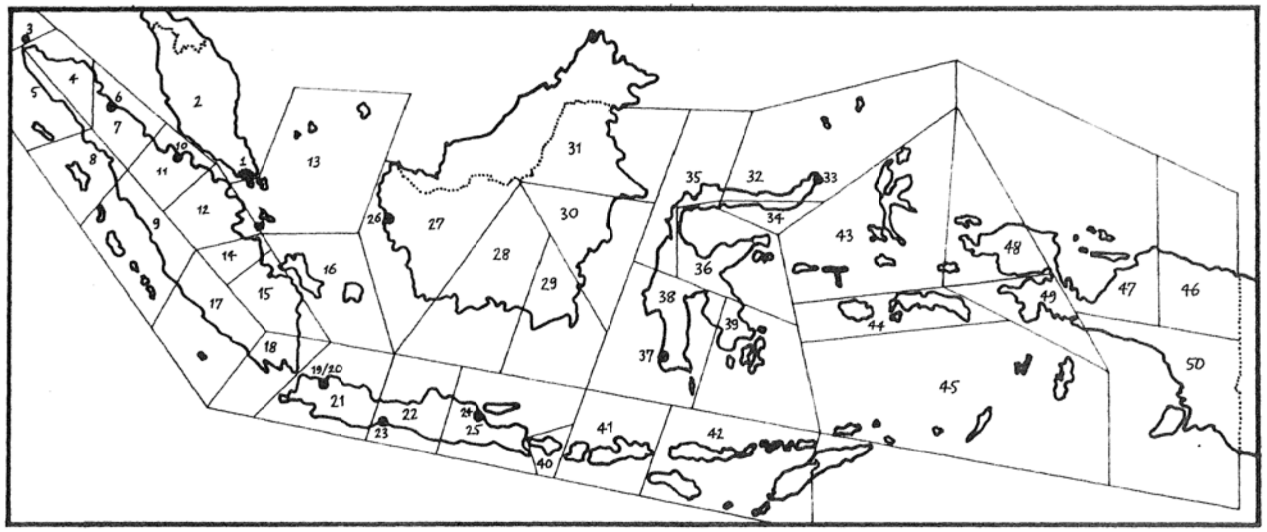
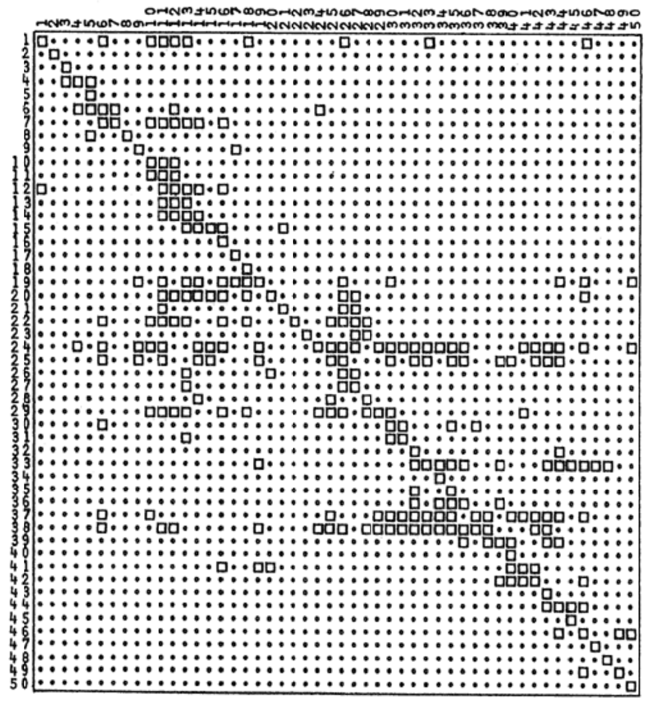

(*c*) Optimal clustering; ME = 1160.

Figure 14.7 Airport example;
• = 0, * = 1, ⊛ = 2, ● = 3.

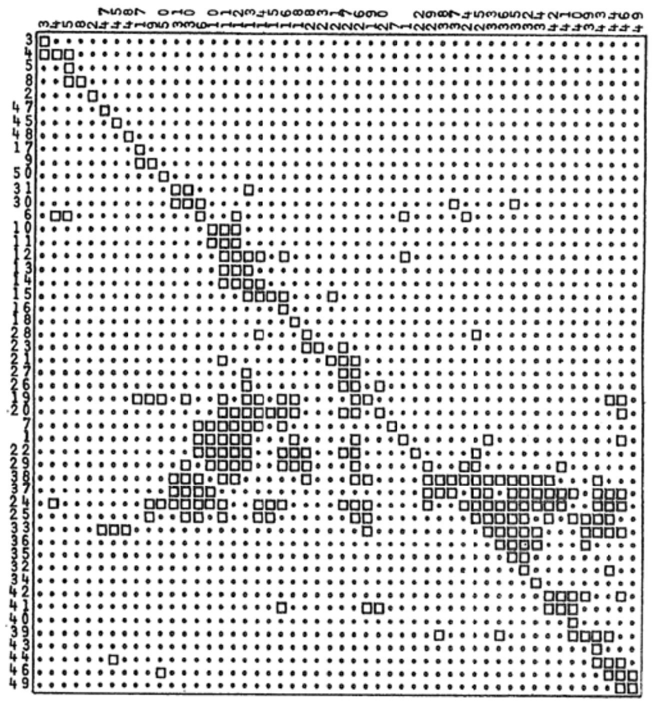


1. Singapore
2. Malay
3. Sabang
4. Aceh I
5. Aceh II
6. Belawan
7. Sumut I
8. Sumut II
9. Sumbar
10. Dumai
11. Ridar I
12. Ridar II
13. Rikep
14. Jambi
15. Sumsel I
16. Sumsel II
17. Bengkulu
18. Lampung
19. Jaya I
20. Jaya II
21. Jabar
22. Jateng I
23. Jateng II
24. Surabaya
25. Jatim
26. Pontianak
27. Kalbar
28. Kalteng
29. Kalsel
30. Kaltim I
31. Kaltim II
32. Sulut I
33. Bitung
34. Sulut II
35. Suteng I
36. Suteng II
37. Makasar
38. Sulsel
39. Sulteng
40. Bail
41. Nusa Tenguara Barat
42. Nusa Tenguara Timur
43. Malut
44. Malteng
45. Malsel
46. Irbaut I
47. Irbaut II
48. Irbaut III
49. Irbasel I
50. Irbasel II

Figure 14.8 Import-export example: regions of the Indonesian islands.



(*a*) Initial array; ME = 223.



(*b*) 3-Optimal clustering subject to ρ = σ; ME = 290.

Figure 14.9 Import-export example; • = 0, □ = 1.

optimal and, in fact, not even 3-optimal, since it can be improved by exchanging three links.

The geographical distribution of the regions of the Indonesian islands in the import-export example is given in Figure 14.8. Figure 14.9 shows the square but asymmetric initial data array and a clustering corresponding to a 3-optimal solution of $\mathrm{TSP}^{\mathrm{cow}}$, found by algorithm LIN.

14.4.4. *Equivalence to the TSP*

Not only can the clustering problem be formulated as one or two symmetric TSPs, but the symmetric TSP can be formulated as a clustering problem as well.

The symmetric TSP corresponds to finding a minimum hamiltonian circuit in the complete undirected graph G on the vertex set $V = \{1,\ldots,n\}$ with a weight c_{ij} for each edge (i,j). This problem is equivalent to finding a minimum hamiltonian path in the complete undirected graph G' on the vertex set $V' = \{0,\ldots,n\}$ with weights c'_{ij}, defined by

$$\begin{aligned}
c'_{01} &= 2\lambda, && \\
c'_{0j} = c'_{1j} &= c_{1j}+\lambda && \text{for } j = 2,\ldots,n, \\
c'_{ij} &= c_{ij} && \text{for } i,j = 2,\ldots,n,
\end{aligned}$$

where λ is greater than the length of any tour. Such a path will have vertices 0 and 1 as extreme points and these vertices can then be joined to arrive at the optimal tour. Now we define a clustering problem with

$$\begin{aligned}
R &= V', && \\
S &= \{(i,j) \mid i,j \in V',\ i < j\}, && \\
a_{i(i,\ell)} &= -c'_{i\ell} && \text{for } i \in R,\ (i,\ell) \in S, \\
a_{i(k,i)} &= 1 && \text{for } i \in R,\ (k,i) \in S, \\
a_{i(k,\ell)} &= 0 && \text{for } i \in R,\ (k,\ell) \in S,\ k,\ell \neq i.
\end{aligned}$$

The contribution of the adjacency of rows i and j with, say, $i < j$ to the ME is equal to

$$\textstyle\sum_{(k,\ell)\in S} a_{i(k,\ell)}a_{j(k,\ell)} = a_{i(i,j)}a_{j(i,j)} = -c'_{ij},$$

and therefore any permutation ρ of R maximizing $\mathrm{ME}(\rho)$ minimizes the weight of the hamiltonian path $(\rho(1),\ldots,\rho(n))$ in G'.

It follows that the clustering problem is NP-complete. Moreover, the symmetric TSP and the clustering problem are of the same difficulty in the sense that the formulations presented in Sections 14.4.2 and 14.4.4 can be interpreted as linear problem reductions with multiplicative coefficients equal to 1.

14.5. Job-shop scheduling with no wait in process

14.5.1. *Problem description*

One of the basic assumptions in most existing theory on machine scheduling is that a job is allowed to wait arbitrarily long before being processed on its next machine. This assumption is highly unrealistic in some real world situations where intermediate storage space is limited or may even be non-existent. The former situation exists for instance in a computer system where buffer space is limited and costly; the latter situation is met in steel or aluminium rolling where the very high temperature of the metal has to be maintained throughout the production process.

We will consider the $n|m|G,\textit{no wait}|C_{\max}$ problem under the following additional assumptions:

(*a*) each job visits each machine at least once;

(*b*) *no passing* is permitted, *i.e.* the processing order is identical on all machines.

Most previous research has been concentrated on the $n|m|F,\textit{no wait}|C_{\max}$ problem [Piehler 1960; Reddi & Ramamoorthy 1972; Wismer 1972; Liesegang & Rüger 1972; Grabowski & Syslo 1973; Syslo 1974]; see [Van Deman & Baker 1974] for the $n|m|F,\textit{no wait}|\sum C_i$ problem. In these cases, (*a*) and (*b*) are redundant conditions.

Extension to a job-shop where different processing orders on the machines are allowed complicates the situation considerably. In the algorithms proposed in [Reddi & Ramamoorthy 1973B; Goyal 1975] the computation of a lower bound is equivalent to solving a TSP and accordingly these methods appear to be time-consuming.

Another extension to the case of non-zero but finite intermediate storage has been considered only for the two-machine flow-shop [Dutta & Cunningham 1975].

14.5.2. *Formulation as a TSP*

The problem under consideration can be formulated as follows.

> Each of n jobs $J_1,\dots,J_n$ has to be processed on each of m machines $M_1,\dots,M_m$. Job J_i consists of a sequence of n_i operations $O_{i1},\dots,O_{in_i}$; operation O_{ik} $(i = 1,\dots,n;\ k = 1,\dots,n_i)$ corresponds to the processing of job J_i on machine $\nu_i(k)$ during an uninterrupted processing time p_{ik}. Under the conditions of *no wait* and *no passing*, we want to find a processing order such that the time required to complete all jobs is minimized.

We define

$$k'_{i\ell} = \min\{k \mid \nu_i(k) = M_\ell,\ k = 1,\dots,n_i\};$$

$$k''_{i\ell} = \max\{k \mid \nu_i(k) = M_\ell,\ k = 1,\dots,n_i\};$$

$$p_i = \sum_{k=1}^{n_i} p_{ik};$$

$$p'_{i\ell} = \sum_{k=k'_{i\ell}}^{n_i} p_{ik};$$

$$p''_{i\ell} = \sum_{k=1}^{k''_{i\ell}} p_{ik}.$$

$O_{ik'_{i\ell}}$ and $O_{ik''_{i\ell}}$ are the first and last operations of J_i on M_ℓ.

For each pair of jobs (J_i,J_j), we will calculate a coefficient c_{ij}, representing the minimum difference between the starting times of O_{i1} and O_{j1} if J_j is scheduled directly after J_i. The *no passing* condition implies that $O_{ik''_{i\ell}}$ has to precede $O_{jk'_{j\ell}}$ on M_ℓ, for $\ell = 1,\dots,m$. We introduce a directed graph G_{ij} with vertex set V_{ij} and arc set A_{ij}, defined by

$$V_{ij} = \{O_{hk} \mid h = i,j;\ k = 1,\dots,n_h\};$$

$$A_{ij} = \{(O_{hk},O_{h,k+1}) \mid h=i,j; k=1,\dots,n_{h-1}\} \cup \{(O_{ik''_{i\ell}},O_{jk'_{j\ell}}) \mid \ell=1,\dots,m\};$$

a weight p_{hk} is attached to each vertex $O_{hk} \in V_{ij}$. For an example with $m = 3$, $\nu_i = (M_2,M_1,M_2,M_3,M_2)$ and $\nu_j = (M_1,M_2,M_3,M_1)$, the graph G_{ij} is given in Figure 14.10. As to the *maximum-weight path* in G_{ij}, it is clear that

$$\text{it starts from } O_{i1} \text{ and ends in } O_{jn_j}; \tag{14.5}$$

$$\text{it contains exactly one arc } (O_{ik''_{i\ell}},O_{jk'_{j\ell}}). \tag{14.6}$$

The *no wait* condition implies that c_{ij} is equal to the latest possible starting time of O_{j1} in G_{ij} if O_{i1} starts at time zero and O_{jn_j} finishes as early as possible. It follows from (14.5) and (14.6) that

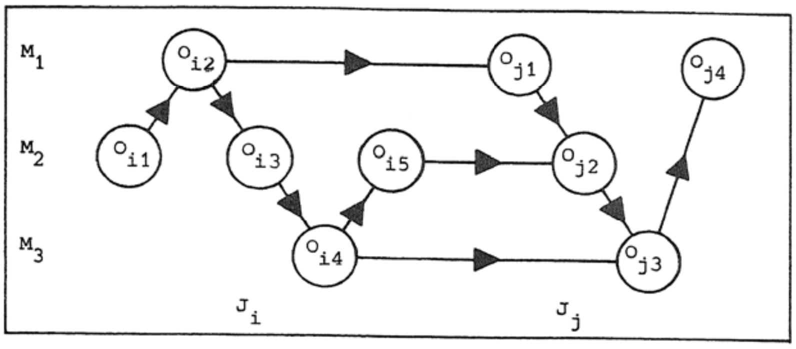


Figure 14.10 Graph G_{ij} for the example.

$$c_{ij} = \max_\ell\{P''_{i\ell}+P'_{j\ell}\} - P_j. \tag{14.7}$$

The minimum time to complete all jobs is now given by

$$\min_\pi\{\textstyle\sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + P_{\pi(n)}\}, \tag{14.8}$$

where π runs over all permutations of $\{1,\dots,n\}$.

We add a job J_0 with $n_0 = m$, $\nu_0(k) = M_k$ and $p_{0k} = 0$ for $k = 1,\dots,m$, representing beginning and end of a schedule. According to (14.7), its coefficients are given by $c_{0i} = 0$, $c_{i0} = P_i$ for $i = 1,\dots,n$. Determination of (14.8) now correspond to solving a TSP with $V = \{0,\dots,n\}$ and (c_{ij}) defined by (14.7).

This asymmetric TSP is *euclidean, i.e.* $c_{ij}+c_{jk} \geq c_{ik}$ for all $i,j,k \in V$:

$$\max_\ell\{P''_{i\ell}+P'_{j\ell}\} + \max_\ell\{P''_{j\ell}+P'_{k\ell}\} \geq \max_\ell\{P''_{i\ell}+P'_{k\ell}\} + P_j.$$

This is true, since for any $\ell \in \{1,\dots,m\}$

$$(P''_{i\ell}+P'_{j\ell}) + (P''_{j\ell}+P'_{k\ell}) \geq (P''_{i\ell}+P'_{k\ell}) + P_j.$$

Remark 1. In a flow-shop we know that $\nu_i = (M_1,M_2,\dots,M_m)$ for $i = 1,\dots,n$, and (14.7) simplifies to $c_{ij} = \max_\ell\{P''_{i\ell}-P''_{j,\ell-1}\}$, which corresponds to the results given in [Piehler 1960; Reddi & Ramamoorthy 1972; Grabowski & Syslo 1973] (*cf.* Section 4.2 Formula (4.2)).

Remark 2. So far, distances have been defined as differences between the starting times of the first operations of jobs. More generally, one might arbitrarily select any two operations $O_{ik_i^*}$ and $O_{ik_i^{**}}$ for each J_i and define c_{ij} as the minimum difference between the starting times of $O_{ik_i^*}$ and $O_{jk_j^{**}}$ if J_i precedes J_j directly. This will lead to modifications in (14.7) and (14.8), but to an equivalent TSP (*cf.* [Goyal 1973; Reddi & Ramamoorthy 1973A]).

14.5.3. *Results*

To illustrate the consequences of the *no wait* condition, we solved the three job-shop problems from [Muth & Thompson 1963, 236-237] under this restriction, using algorithm LEA. In Table 14.3 the solution values are compared with the lengths of the schedules when arbitrary waiting times are allowed. Figure 14.11 illustrates the optimal schedules for one of these problems; the unrestricted schedule was found by the method from [Florian *et al.* 1971]. In general, the *no wait* and *no passing* conditions can be expected to lead to large amounts of idle time on the machines.

TABLE 14.3. EFFECT OF THE *no wait* CONDITION

number of jobs	number of machines	value of *no wait* schedule	value of unrestricted schedule
6	6	120	55
10	10	2433	972*
20	5	2132	1165

* indicates that the optimality has not been proved

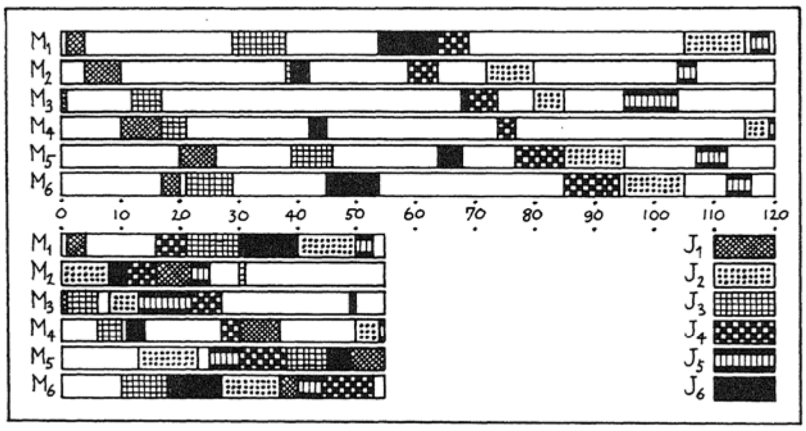


Figure 14.11 Optimal schedules for a 6×6 problem.

14.5.4. *Equivalence to the TSP*

In Chapter 4 we have seen that the $n|2|F$,*no wait*$|C_{max}$ problem can be solved in $O(n^2)$ steps [Gilmore & Gomory 1964] and that the $n|m|F$,*no wait*$|C_{max}$ problem is NP-complete. The reduction given in Theorem 4.7(*a*) can easily be adapted to formulate any TSP in terms of an $n|m|F$,*no wait*$|C_{max}$ problem. Together with the formulation presented in Section 14.5.2 this establishes the complete equivalence of the TSP and the *no wait* problem.

15. AN APPLICATION OF MACHINE SCHEDULING THEORY

15.1. Problem description

The practical scheduling situation that we shall describe arises in the context of the production of aluminium airplane parts. In a certain section of the factory in question, the production is centered around a rubber press. The metal pieces are first processed either by a *cutting* or by a *milling machine*. They next have to pass a *fitting shop* and subsequently have to spend a full working day in an *annealing furnace* before being pressed into their proper shape by the *rubber press*. After passing the fitting shop for a second time they are completely finished. The processing time of each operation is known in advance.

There are nine operators available to process the jobs. One of them operates the cutting and milling machines, six are working in the fitting shop and two handle the rubber press; the annealing furnace requires no attention and can be assumed to have an infinite capacity, *i.e.*, it can handle any number of jobs at the same time.

Since the rubber press is a relatively costly machine, the objective is to choose processing orders in such a way that the total completion time is minimized while idle time on the rubber press is avoided as much as possible.

If we denote the operations of J_i by O_{ik} with processing times p_{ik} $(k = 1,\dots,5)$, typical data for a week's production of 35 jobs look like those presented in the left-hand part of Table 15.1. Note that some jobs, which are left over from last week, have completed some of their intial operations.

15.2. A heuristic approach

We can model the above situation as a job-shop with four machines:

- M_1 represents the cutting and milling machines and has capacity 1;
- M_2 represents the fitting shop and has capacity 6;
- M_3 represents the annealing furnace and has capacity ∞;
- M_4 represents the rubber press and has capacity 1.

Each job has the same machine order (M_1,M_2,M_3,M_4,M_2).

Approaching the problem in a heuristic way, we note that

$\sum_{i=1}^{35} p_{i1} = 56$, $\sum_{i=1}^{35} p_{i2} = 70$, $\sum_{i=1}^{35} p_{i4} = 48.5$, $\sum_{i=1}^{35} p_{i5} = 202$.

Clearly, not all jobs can be processed on M_1 and M_4 within one week of 40 hours and some overflow will result. It seems quite possible to schedule O_{i2} and O_{i3} directly after the completion of O_{i1}, but some waiting time for the jobs before the processing of O_{i4} and O_{i5} seems unavoidable. It is expedient to schedule O_{i1} in such a way that many jobs are quickly available for further processing, thereby taking p_{i4} and p_{i5} into account.

These intuitive considerations led to the following heuristic method, in which C_{ik} stands for the completion time of O_{ik}.

1. Schedule O_{i1} on M_1 according to nonincreasing $(p_{i4}+p_{i5})/p_{i1}$, thereby minimizing the total weighted completion time $\sum_{i=1}^{35}(p_{i4}+p_{i5})C_{i1}$ (*cf.* [Smith 1956]).
2. Schedule O_{i2} as early as possible on M_2 according to nondecreasing C_{i1}.
3. Schedule O_{i3} on M_3 according to $C_{i3} := 8\lceil C_{i2}/8 \rceil + 8$ ($\lceil x \rceil$ is the smallest integer not less than x).
4. Schedule O_{i4} on M_4 by solving the $n|1|r_i \geq 0|L'_{\max}$ problem as discussed in Chapter 10, defined by heads C_{i3}, bodies p_{i4} and tails p_{i5}.
5. Schedule O_{i5} as early as possible on M_2 according to nondecreasing C_{i4}.

15.3. Results

The above heuristic was applied to the problem data in Table 15.1. The one-machine problem on M_4 was solved by algorithm MF (see Section 10.2.3); the first application of algorithm LS yielded an optimal solution. The resulting schedule is given by the completion times in Table 15.1; the corresponding *Gantt-chart* is shown in Figure 15.1. This schedule compares favourably with several schedules obtained by trial-and-error and rules of thumb.

15.4. Remarks

The approach described above seems to be more generally applicable. Basically, it involves the determination of *critical machines* in the production process, *i.e.*, the machines that are important from a cost minimizing point of view and on which the processing orders have a crucial influence on the quality of the schedule as a whole. The problem is then *decomposed* into problems involving one or more of those critical machines; these problems

TABLE 15.1. A PRACTICAL SCHEDULING PROBLEM: DATA AND RESULTS

i	p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	$\frac{p_{i1}}{p_{i4}+p_{i5}}$	C_{i1}	C_{i2}	C_{i3}	C_{i4}	C_{i5}
1	4	4	8	2	6	.50	32	36	48	50	56
2	2	2	8	1	4	.40	22	24	32	33	38
3	1	1	8	1	2	.33	6	7	16	27.5	32
4	–	2	8	.5	6	–	0	2	16	22	32
5	6	2	8	2	7	.67	50	52	64	66	73
6	1	1	8	1	3	.25	4	5	16	24	33
7	1	1	8	.5	4	.22	2	3	16	22.5	31
8	–	5	8	.5	7	–	0	5	16	18.5	27
9	1	1	8	.5	4	.22	3	4	16	23	34
10	–	–	8	2	6	–	–	0	8	14	20
11	1	1	8	.5	2	.40	23	24	32	33.5	36
12	1	1	8	.5	2	.40	24	25	40	45	47
13	1	1	8	1	2	.33	7	8	16	28.5	34
14	2	1	8	1	4	.40	26	27	40	43	47
15	–	5	8	1	9	–	0	5	16	17	26
16	7	10	8	2	16	.39	20	30	40	42	58
17	–	–	–	1.5	6	–	–	–	0	6.5	12.5
18	1	1	8	1.5	6	.13	1	2	16	20	30
19	–	–	8	2	8	–	–	0	8	10	18
20	6	6	8	3	6	.67	56	62	72	75	81
21	–	–	–	2.5	12	–	–	–	0	2.5	15
22	–	–	8	2	7	–	–	0	8	12	19.5
23	–	–	–	2.5	11	–	–	–	0	5	16
24	2	2	8	1.5	3	.44	28	30	40	44.5	47.5
25	–	–	–	–	4	–	–	–	–	0	6
26	6	3	8	2.5	7	.63	44	47	56	61.5	68.5
27	–	–	8	1	5	–	–	0	8	15	20
28	1	1	8	1	3	.25	5	6	16	26.5	34
29	–	–	–	1.5	6	–	–	–	0	8	14
30	1	1	8	1	2	.33	8	9	24	29.5	35
31	1	1	8	1	2	.33	9	10	24	30.5	36
32	–	3	8	1	7	–	0	3	16	18	25
33	6	6	8	3	7	.60	38	44	56	59	66
34	–	6	8	1.5	6	–	0	6	16	21.5	31
35	4	2	8	1.5	10	.35	13	15	24	25.5	41

may be solved by methods inspired by sequencing theory. The resulting schedules are *concatenated* by suitable processing orders on the other machines leading to an overall schedule of reasonable quality.

Our experience with this heuristic approach has been limited to the small example above and our only conclusion would be that it merits further experimentation. We feel that through this approach the models from sequencing theory, which may well correspond to an oversimplified picture of reality, can find application in varying situations that do not fit the standard models. In view of the frequent complaint about the lack of successful practical applications of machine scheduling theory, this seems a worth-while area for future research.

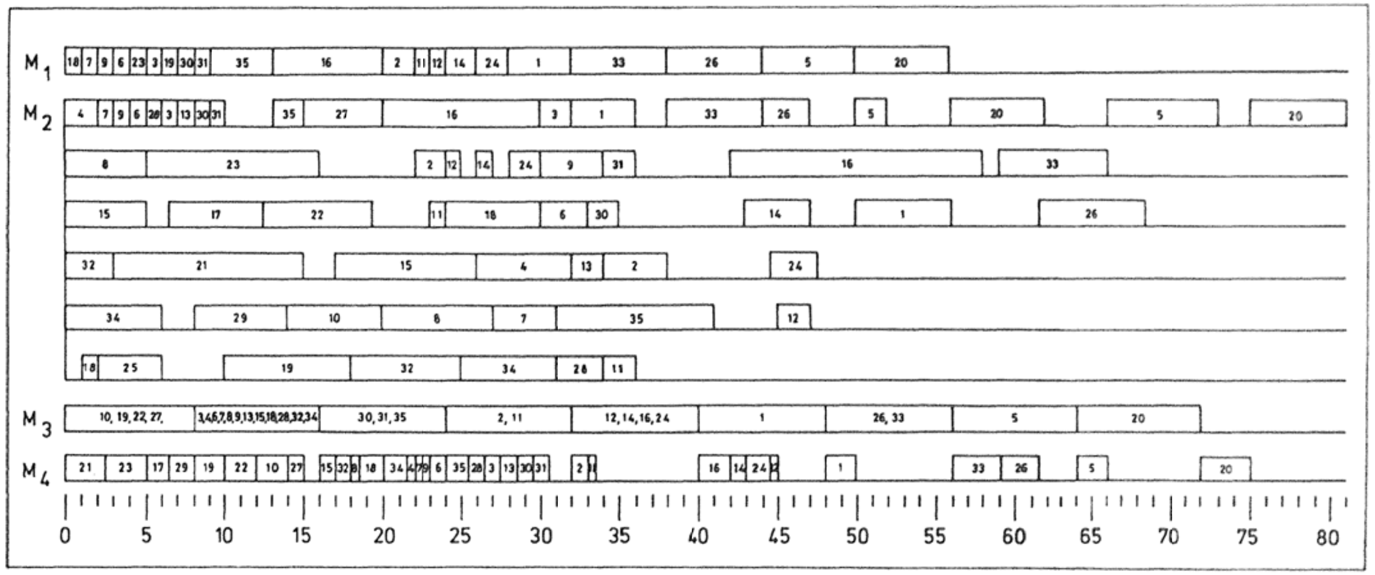


Figure 15.1 A practical scheduling problem: Gantt-chart.

"What about your readers - my readers! What shall I tell them?"

"Anything you like," was the bland reply. "Tell them I was murdered by my mathematics tutor, if you like."

The Seven Per Cent Solution,
Being a Reprint from the Reminiscences of John H. Watson, M.D.,
as Edited by Nicholas Meyer.

16. CONCLUSION

In this final chapter we intend to indicate briefly some promising areas for future sequencing research. Here we can fruitfully distinguish between questions concerning problems in P, for which a good algorithm exists, and problems that are known to be NP-complete.

With respect to the former category there are two important directions for further investigations. First, we can try to find a better polynomial-bounded algorithm or improve the quality of the currently best implementation. Secondly, we may derive sharp lower bounds on the number of steps minimally required to solve the problem. Whenever these two approaches meet, optimality of a certain algorithm has been proved.

With respect to the latter category, we have already advocated several times the need for further refinement of the complexity measure provided by the NP-completeness concept. Two possible ways of doing this have been indicated and proved useful in preliminary research. We can study the various possibilities of encoding the problem data and distinguish between complexity results with respect to unary and binary encodings. Furthermore, we may attempt to investigate the existence of an algorithm that finds a solution within a constant percentage from the optimum. For some problems, such an approximate algorithm has been identified; in other cases, even this approximation problem turns out to be NP-complete.

Given NP-completeness of a problem, the use of enumerative methods seems inevitable and in fact still more sophisticated ones may be required for its solution. Problems of reasonable size may be solved only if we apply branching schemes, bounding rules and elimination criteria that exploit the characteristic features of the specific type of problem under consideration. Sharper bounds might for instance be based on relaxations to "easier" NP-complete problems or on a form of Lagrangean relaxation. A recursively implemented depth-first search appears to be an attractive approach for many branch-and-bound algorithms, provided that programming languages and compilers are available that are well suited for recursive procedures.

None the less, approximate methods turn out to be unavoidable for many problems. An investigation of the worst-case behaviour of such methods and probabilistic analyses of their average-case or "almost everywhere" behaviour still leads to a host of challenging mathematical problems. Generally though, the development and testing of very general heuristics does not seem the most appropriate way to attack practical problems, where often special structural properties allow a more tailor-made approach.

With respect to those real-world problems, we finally feel that the potential applicability of machine scheduling theory has been sorely underestimated, especially when compared to the many varied applications of quadratic assignment problems. The area of sequencing research should be one of the prime examples of a specialization within operations research where the artificial distinction between theoretical and practical work is minimized to the benefit of all.

BIBLIOGRAPHY

A.K. AGARWAL (1975) Multiple reversal procedure in implicit enumeration algorithm for machine sequencing via disjunctive graphs. Presented at ORSA/TIMS meeting, Chicago.

N. AGIN (1966) Optimum seeking with branch-and-bound. *Management Sci.* 13, B176-185.

J.M. ANTHONISSE (1972) Private communication.

J.M. ANTHONISSE, P. VAN EMDE BOAS (1974) Are polynomial algorithms really good? Report BW 40, Mathematisch Centrum, Amsterdam.

S. ASHOUR (1970) A branch-and-bound algorithm for flow-shop scheduling problems. *AIIE Trans.* 2,172-176.

S. ASHOUR, K.-Y. CHIU, T.E. MOORE (1973) An optimal schedule time of a job shop-like disjunctive graph. *Networks* 3,333-349.

S. ASHOUR, S.R. HIREMATH (1973) A branch-and-bound approach to the job-shop scheduling problem. *Internat. J. Production Res.* 11,47-58.

S. ASHOUR, T.E. MOORE, K.-Y. CHIU (1974) An implicit enumeration algorithm for the nonpreemptive shop scheduling problem. *AIIE Trans.* 6,62-72.

S. ASHOUR, R.G. PARKER (1971) A precedence graph algorithm for the shop scheduling problem. *Operational Res. Quart.* 22,165-175,379.

P.C. BAGGA, N.K. CHAKRAVARTI (1968) Optimal *m*-stage production schedule. *Canad. Operational Res. Soc. J.* 6,71-78.

K.R. BAKER (1975) A comparative study of flow-shop algorithms. *Operations Res.* 23,62-73.

K.R. BAKER, J.B. MARTIN (1974) An experimental comparison of solution algorithms for the single-machine tardiness problem. *Naval Res. Logist. Quart.* 21,187-199.

K.R. BAKER, Z.-S. SU (1974) Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Res. Logist. Quart.* 21,171-176.

E. BALAS (1968) A note on the branch-and-bound principle. *Operations Res.* 16,442-445,886.

E. BALAS (1969) Machine-sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Res.* 17,941-957.

W. BARTH (1968) Ein ALGOL 60 Programm zur Lösung des traveling Salesman Problems. *Ablauf- und Planungsforschung* 9,99-105.

E.F. BECKENBACH (ed.) (1964) *Applied Combinatorial Mathematics*. Wiley, New York.

M. BELLMORE, J.C. MALONE (1971) Pathology of traveling-salesman subtour elimination algorithms. *Operations Res.* 19,278-307,1766.

M. BELLMORE, G.L. NEMHAUSER (1968) The traveling salesman problem: a survey. *Operations Res.* 16,538-558.

J.R. BITNER, G. EHRLICH, E.M. REINGOLD (1975) Efficient generation of the binary reflected Gray code and its applications. Department of Computer Science, University of Illinois at Urbana-Champaign.

J. BOOTHROYD (1965) Algorithm 6, Perm. *Comput. Bull.* 9,104.

J. BOOTHROYD (1967A) Algorithm 29, Permutation of the elements of a vector. *Comput. J.* 10,311.

J. BOOTHROYD (1967B) Algorithm 30, Fast permutation of the elements of a vector. *Comput. J.* 10,311-312.

P. BRATLEY, M. FLORIAN, P. ROBILLARD (1973) On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{\max})$ problem. *Naval Res. Logist. Quart.* 20,57-67.

G.H. BROOKS, C.R. WHITE (1965) An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *J. Indust. Eng.* 16,34-40.

A.P.G. BROWN, Z.A. LOMNICKI (1966) Some applications of the "branch-and-bound" algorithm to the machine scheduling problem. *Operational Res. Quart.* 17,173-186.

J. BRUNO, E.G. COFFMAN, Jr., R. SEHTI (1974A) Algorithms for minimizing mean flow time. [Rosenfeld 1974, 504-510].

J. BRUNO, E.G. COFFMAN, Jr., R. SEHTI (1974B) Scheduling independent tasks to reduce mean finishing time. *Comm. ACM* 17,382-387.

R.E. BURKARD (1973) A perturbation method for solving quadratic assignment problems. Presented at 8th Mathematical Programming Symposium, Stanford.

P.M. CAMERINI, L. FRATTA, F. MAFFIOLI (1974) Traveling salesman problem: heuristically guided search and modified gradient techniques. Politecnico di Milano.

H.G. CAMPBELL, R.A. DUDEK, M.L. SMITH (1970) A heuristic algorithm for the *n* job, *m* machine sequencing problem. *Management Sci.* 16,B630-637.

R. CARVAJAL, G. ESPINOSA, A. LÓPEZ (1974) The traveling salesman problem and the bond energy algorithm used in cluster analysis. Comunicaciones Tecnicas 5B No.79, Centro de Investigacion en Matematicas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México.

J.M. CHARLTON, C.C. DEATH (1970A) A generalized machine scheduling algorithm. *Operational Res. Quart.* 21,127-134.

J.M. CHARLTON, C.C. DEATH (1970B) A method of solution for general machine scheduling problems. *Operations Res.* 18,689-707.

N. CHRISTOFIDES (1970) The shortest Hamiltonian chain of a graph. *SIAM J. Appl. Math.* 19,689-696.

N. CHRISTOFIDES (1972) Bounds for the travelling-salesman problem. *Operations Res.* 20,1044-1056.

N. CHRISTOFIDES (1974) The vehicle routing problem. *Rev. Française Automat. Informat. Recherche Opêrationelle*, to appear.

N. CHRISTOFIDES (1975) *Graph Theory: an Algorithmic Approach*. Academic Press, New York.

N. CHRISTOFIDES, S. EILON (1972) Algorithms for large-scale travelling salesman problems. *Operational Res. Quart.* 23,511-518.

E.G. COFFMAN, Jr. (ed.) (1976) *Computer and Job-shop Scheduling Theory*. Wiley, New York.

E.G. COFFMAN, Jr., R.L. GRAHAM (1972) Optimal scheduling for two-processor systems. *Acta Informat.* 1,200-213.

R.W. CONWAY, W.L. MAXWELL, L.W. MILLER (1967) *Theory of Scheduling*. Addison-Wesley, Reading, Mass.

S.A. COOK (1971) The complexity of theorem-proving procedures. *Proc. 3rd Annual ACM Symp. Theory Comput.*, 151-158.

G.A. CROES (1958) A method for solving traveling salesman problems. *Operations Res.* 6,791-814.

G.B. DANTZIG, D.R. FULKERSON, S.M. JOHNSON (1954) Solution of a large scale traveling salesman problem. *Operations Res.* 2,393-410.

N. DERSHOWITZ (1975) A simplified loop-free algorithm for generating permutations. *BIT* 15,158-164.

M.I. DESSOUKY, C.R. MARGENTHALER (1972) The one-machine sequencing problem with early starts and due dates. *AIIE Trans.* 4,214-222.

E.W. DIJKSTRA (1959) A note on two problems in connexion with graphs. *Numer. Math.* 1,269-271.

B. DORHOUT (1975) Experiments with some algorithms for the linear assignment problem. Report BW 39, Mathematisch Centrum, Amsterdam.

R.A. DUDEK, O.F. TEUTON, Jr. (1964) Development of *M*-stage decision rule for scheduling *n* jobs through *M* machines. *Operations Res.* 12,471-497.

S.K. DUTTA, A.A. CUNNINGHAM (1975) Sequencing two-machine flow-shops with finite intermediate storage. *Management Sci.* 21,989-996.

W.L. EASTMAN (1958) *Linear Programming with Pattern Constraints*. Thesis, Harvard University, Cambridge, Mass.

W.L. EASTMAN (1959) A solution to the traveling-salesman problem. *Econometrica* 27,282.

J. EDMONDS (1965A) Paths, trees, and flowers. *Canad. J. Math.* 17,449-467.

J. EDMONDS (1965B) The Chinese postman's problem. *Operations Res.* 13 Suppl.1, B73.

J. EDMONDS (1967) Optimum branchings. *J. Res. Nat. Bur. Standards* 71B,233-240.

J. EDMONDS (1974) Private Communication. Presented at the NATO Advanced Study Institute on Combinatorial Programming: Methods and Applications, Versailles, September 2-13.

J. EDMONDS (1975) Some well-solved problems in combinatorial optimization. [Roy 1975, 285-301].

J. EDMONDS, E.L. JOHNSON (1973) Matching, Euler tours and the Chinese postman. *Math. Programming* 5,88-124.

G. EHRLICH (1973A) Loopless algorithms for generating permutations, combinations and other combinatorial configurations. *J. Assoc. Comput. Mach.* 20,500-513.

G. EHRLICH (1973B) Algorithm 466, Four combinatorial algorithms. *Comm. ACM* 16,690-691.

S. EILON, C.D.T. WATSON-GANDY, N. CHRISTOFIDES (1971) *Distribution Management: Mathematical Modelling and Practical Analysis*. Griffin, London.

S.E. ELMAGHRABY (1968) The one-machine sequencing problem with delay costs. *J. Indust. Engrg.* 19,105-108.

H. EMMONS (1969) One-machine sequencing to minimize certain functions of job tardiness. *Operations Res.* 17,701-715.

S. EVEN (1973) *Algorithmic Combinatorics*. Macmillan, London.

M.L. FISHER (1974) A dual algorithm for the one-machine scheduling problem. *Math. Programming*, to appear.

M. FLORIAN, P. TRÉPANT, G. McMAHON (1971) An implicit enumeration algorithm for the machine sequencing problem. *Management Sci.* 17,B782-792.

M. GARDNER (1974) Some new and dramatic demonstrations of number theorems with playing cards. *Sci. Amer.* 231,122-125.

M.R. GAREY (1975) Private communications.

M.R. GAREY, R.L. GRAHAM (1975) Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.* 4,187-200.

M.R. GAREY, D.S. JOHNSON (1974) Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, to appear.

M.R. GAREY, D.S. JOHNSON (1975) Scheduling tasks with non-uniform deadlines on two processors. Bell Laboratories, Murray Hill, N.J.

M.R. GAREY, D.S. JOHNSON, R. SEHTI (1975) The complexity of flowshop and jobshop scheduling. Technical Report 168, Computer Science Department, The Pennsylvania State University.

M.R. GAREY, D.S. JOHNSON, L. STOCKMEYER (1974) Some simplified NP-complete problems. *Theoret. Comput. Sci.*, to appear.

R.S. GARFINKEL (1973) On partitioning the feasible set in a branch-and-bound algorithm for the asymmetric traveling-salesman problem. *Operations Res.* 21,340-343.

J.W. GAVETT, N.V. PLYTER (1966) The optimal assignment of facilities to locations by branch and bound. *Operations Res.* 14,210-232.

L. GELDERS, P.R. KLEINDORFER (1974) Coordinating aggregate and detailed scheduling decisions in the one-machine job shop: part I. Theory. *Operations Res.* 22,46-60.

L. GELDERS, P.R. KLEINDORFER (1975) Coordinating aggregate and detailed scheduling in the one-machine job shop: II - computation and structure. *Operations Res.* 23,312-324.

A.M. GEOFFRION, G.W. GRAVES (1975) Scheduling parallel production lines with changeover costs: practical application of a quadratic assignment/LP approach. Working Paper No.231, Western Management Science Institute, University of California, Los Angeles.

B. GIFFLER, G.L. THOMPSON (1960) Algorithms for solving production-scheduling problems. *Operations Res.* 8,487-503.

B.E. GILLETT, L.R. MILLER (1974) A heuristic algorithm for the vehicle-dispatch problem. *Operations Res.* 22,340-349.

P.C. GILMORE (1962) Optimal and suboptimal algorithms for quadratic assignment problems. *J. SIAM* 10,305-313.

P.C. GILMORE, R.E. GOMORY (1964) Sequencing a one-state variable machine: a solvable case of the traveling salesman problem. *Operations Res.* 12, 655-679.

F. GLOVER, T. KLASTORIN, D. KLINGMAN (1974) Optimal weighted ancestry relationships. *Management Sci.* 20,B1190-1193.

T. GONZALES, S. SAHNI (1975) Flow shop and job shop schedules. Technical Report 75-14, Department of Computer Sciences, University of Minnesota, Minneapolis.

S.K. GOYAL (1973) A note on the paper: On the flow-shop sequencing problem with no wait in process. *Operational Res. Quart.* 24,130-133.

S.K. GOYAL (1975) Job-shop sequencing problems with no wait in process. *Internat. J. Production Res.* 13,197-206.

J. GRABOWSKI, M.M. SYSLO (1973) On some machine sequencing problems (I). *Zastos. Mat.* 13,339-345.

R.L. GRAHAM (1969) Bounds on multiprocessing time anomalies. *SIAM J. Appl. Math.* 17,263-269.

H.H. GREENBERG (1968) A branch-and-bound solution to the general scheduling problem. *Operations Res.* 16,353-361.

D. GRIES (1975) Recursion as a programming tool. Technical Report 234, Department of Computer Science, Cornell University, Ithaca.

M. GRÖTSCHEL, M.W. PADBERG (1974) Lineare Charakterisierungen von travelling Salesman Problemen. *Z. Operations Res.*, to appear.

M. GRÖTSCHEL, M.W. PADBERG (1975) Partial linear characterizations of the travelling salesman polytope. *Math. Programming* 8,378-381.

J.N.D. GUPTA (1971) An improved combinatorial algorithm for the flowshop-scheduling problem. *Operations. Res.* 19,1753-1758.

M. HANAN, J.M. KURTZBERG (1972) A review of the placement and quadratic assignment problems. *SIAM Rev.* 14,324-342.

P. HANSEN, L. KAUFMAN (1974) A note on the quadratic assignment problem. *Cahiers Centre Études Recherche Opér.* 16,441-446.

K. HARADA (1971) Generation of rosary permutations expressed in hamiltonian circuits. *Comm. ACM* 14,373-379.

W.W. HARDGRAVE, G.L. NEMHAUSER (1963) A geometric model and a graphical algorithm for a sequencing problem. *Operations Res.* 11,889-900.

G.H. HARDY (1940) *A Mathematician's Apology*. Cambridge University Press, Cambridge.

K. HELBIG HANSEN, J. KRARUP (1974) Improvements of the Held-Karp algorithm for the symmetric traveling-salesman problem. *Math. Programming* 7,87-96.

M. HELD, R.M. KARP (1962) A dynamic programming approach to sequencing problems. *J. SIAM* 10,196-210.

M. HELD, R.M. KARP (1970) The traveling-salesman problem and minimum spanning trees. *Operations Res.* 18,1138-1162.

M. HELD, R.M. KARP (1971) The traveling-salesman problem and minimum spanning trees: part II. *Math. Programming* 1,6-25.

M. HELD, P. WOLFE, H.P. CROWDER (1974) Validation of subgradient optimization. *Math. Programming* 6,62-88.

J. HEMELRIJK (1966) Underlining random variables. *Statistica Neerlandica* 20,1-7.

W.A. HORN (1972) Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM J. Appl. Math.* 23,189-202.

W.A. HORN (1973) Minimizing average flow time with parallel machines. *Operations Res.* 21,846-847.

W.A. HORN (1974) Some simple scheduling algorithms. *Naval Res. Logist. Quart.* 21,177-185.

T.C. HU (1961) Parallel sequencing and assembly line problems. *Operations Res.* 9,841-848.

T.C. HU (1969) *Integer Programming and Network Flows.* Addison-Wesley, Reading, Mass.

E. IGNALL, L. SCHRAGE (1965) Application of the branch-and-bound technique to some flow-shop scheduling problems. *Operations Res.* 13,400-412.

A.M. ISAAC, E. TURBAN (1969) Some comments on the traveling salesman problem. *Operations Res.* 17,543-546.

J.R. JACKSON (1955) Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

J.R. JACKSON (1956) An extension of Johnson's results on job lot scheduling. *Naval Res. Logist. Quart.* 3,201-203.

D.S. JOHNSON (1973) Approximation algorithms for combinatorial problems. *Proc. 5th Annual ACM Symp. Theory Comput.*, 38-49.

S.M. JOHNSON (1954) Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Quart.* 1,61-68.

S.M. JOHNSON (1963) Generation of permutations by adjacent transposition. *Math. Comp.* 17,282-285.

R. JONKER (1974) The one depot vehicle scheduling problem: an improved heuristic approach. Instituut voor Actuariaat en Econometrie, Universiteit van Amsterdam.

R.M. KARP (1972A) A simple derivation of Edmonds' algorithm for optimum branchings. *Networks* 1,265-272.

R.M. KARP (1972B) Reducibility among combinatorial problems. [Miller & Thatcher 1972, 85-103].

R.M. KARP (1975A) On the computational complexity of combinatorial problems. *Networks* 5,45-68.

R.M. KARP (1975B) Non-heuristic analysis of heuristic search methods. Presented at Workshop on Integer Programming, Bonn, September 8-12.

W. KARUSH (1965) A counterexample to a proposed algorithm for optimal sequencing of jobs. *Operations Res.* 13,323-325.

L. KAUFMAN (1975) *The Location of Economic Activities by 0-1 Programming.* Thesis, Vrije Universiteit, Brussels.

D.E. KNUTH (1974) A terminological proposal. *SIGACT News* 6.1,12-18.

W.H. KOHLER, K. STEIGLITZ (1974) Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *J. Assoc. Comput. Mach.* 21,140-156.

P.J. KOLESAR (1967) A branch and bound algorithm for the knapsack problem. *Management Sci.* 13,723-735.

T.C. KOOPMANS, M. BECKMANN (1957) Assignment problems and the location of economic activities. *Econometrica* 25,53-76.

B. KORTE, W. OBERHOFER (1968) Zwei Algorithmen zur Lösung eines komplexen Reihenfolgeproblems. *Unternehmensforschung* 12,217-231.

J. KRARUP (1975) The peripatetic salesman and some related unsolved problems. [Roy 1975, 173-178].

J.B. KRUSKAL (1956) On the shortest spanning subtree of a graph and the traveling-salesman problem. *Proc. Amer. Math. Soc.* 2,48-50.

J.H. KUIPER (1973) "Hoe een PTT-er handelsreiziger werd" - een routing probleem. Unpublished Report.

B.J. LAGEWEG, E.L. LAWLER (1975) Private communication.

B.J. LAGEWEG, J.K. LENSTRA (1972) Algoritmen voor knapzakproblemen. Report BN 14, Mathematisch Centrum, Amsterdam.

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1975) Minimizing maximum lateness on one machine: computational experience and some applications. *Statistica Neerlandica*, to appear.

A.H. LAND (1963) A problem of assignment with inter-related costs. *Operational Res. Quart.* 14,185-199.

E.L. LAWLER (1963) The quadratic assignment problem. *Management Sci.* 9,586-599.

E.L. LAWLER (1964) On scheduling problems with deferral costs. *Management Sci.* 11,280-288.

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

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

E.L. LAWLER (1975A) The quadratic assignment problem: a brief review. [Roy 1975, 351-360].

E.L. LAWLER (1975B) Sequencing the weighted number of tardy jobs. *Rev. Française Automat. Informat. Recherche Opérationelle*, to appear.

E.L. LAWLER (1975C) A "quasi-polynomial" algorithm for sequencing jobs to minimize total tardiness. To be published.

E.L. LAWLER (1975D) Optimal sequencing of jobs subject to series parallel precedence constraints. Report BW 54, Mathematisch Centrum, Amsterdam.

E.L. LAWLER (1976) *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, New York, to appear.

E.L. LAWLER, J.M. MOORE (1969) A functional equation and its application to resource allocation and sequencing problems. *Management Sci.* 16,77-84.

E.L. LAWLER, D.E. WOOD (1966) Branch-and-bound methods: a survey. *Operations Res.* 14,699-719.

D.H. LEHMER (1964) The machine tool of combinatorics. [Beckenbach 1964, 5-31].

H.W. LENSTRA, Jr. (1973A) The acyclic subgraph problem. Report BW 26, Mathematisch Centrum, Amsterdam.

H.W. LENSTRA, Jr. (1973B) Private communications.

J.K. LENSTRA (1972) Branch-and-bound algorithmen voor het handelsreizigersprobleem. Report BN 16, Mathematisch Centrum, Amsterdam.

J.K. LENSTRA (1973) Recursive algorithms for enumerating subsets, latticepoints, combinations and permutations. Report BW 28, Mathematisch Centrum, Amsterdam.

J.K. LENSTRA (1974) Clustering a data array and the traveling-salesman problem. *Operations Res.* 22,413-414.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1973) Towards a better algorithm for the job-shop scheduling problem - I. Report BN 22, Mathematisch Centrum, Amsterdam.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1974) On general routing problems. *Networks*, to appear.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1975A) Some simple applications of the travelling salesman problem. *Operational Res. Quart.* 26,717-733.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1975B) A recursive approach to the generation of combinatorial configurations. Report BW 50, Mathematisch Centrum, Amsterdam.

J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER (1975) Complexity of machine scheduling problems. *Ann. Discrete Math.* 1, to appear.

D.G. LIESEGANG (1974) *Möglichkeiten zur wirkungsvollen Gestaltung von Branch and Bound-Verfahren dargestellt an ausgewählten Problemen der Reihenfolgeplanung*. Thesis, Universität zu Köln, Cologne.

D.G. LIESEGANG, M. RÜGER (1972) Letter. *Operational Res. Quart.* 23,591.

S. LIN (1965) Computer solutions of the traveling salesman problem. *Bell System Tech. J.* 44,2245-2269.

S. LIN, B.W. KERNIGHAN (1973) An effective heuristic algorithm for the traveling-salesman problem. *Operations Res.* 21,498-516.

J.D.C. LITTLE, K.G. MURTY, D.W. SWEENEY, C. KAREL (1963) An algorithm for the traveling salesman problem. *Operations Res.* 11,972-989.

Z.A. LOMNICKI (1965) A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. *Operational Res. Quart.* 16,89-100.

W.T. McCORMICK, Jr., P.J. SCHWEITZER, T.W. WHITE (1972) Problem decomposition and data reorganization by a clustering technique. *Operations Res.* 20, 993-1009.

G.B. McMAHON (1969) Optimal production schedules for flow shops. *Canad. Operational Res. Soc. J.* 7,141-151.

G.B. McMAHON (1971) *A Study of Algorithms for Industrial Scheduling Problems*. Thesis, University of New South Wales, Kensington.

G.B. McMAHON, P.G. BURTON (1967) Flow-shop scheduling with the branch-and-bound method. *Operations Res.* 15,473-481.

G.B. McMAHON, M. FLORIAN (1975) On scheduling with ready times and due dates to minimize maximum lateness. *Operations Res.* 23,475-482.

N. MEYER (1975) *The Seven Per Cent Solution, Being a Reprint from the Reminiscences of John H. Watson, M.D.* Hodder and Stoughton, London.

R.E. MILLER, J.W. THATCHER (eds.) (1972) *Complexity of Computer Computations*. Plenum Press, New York.

L.G. MITTEN (1970) Branch-and-bound methods: general formulation and properties. *Operations Res.* 18,24-34.

J.W. MOON (1968) *Topics on Tournaments*. Holt, Rinehart, and Winston, New York.

J.M. MOORE (1968) An *n* job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Sci.* 15,102-109.

H. MÜLLER-MERBACH (1970) *Optimale Reihenfolgen*. Springer, Berlin.

J.F. MUTH, G.L. THOMPSON (eds.) (1963) *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs, N.J.

I. NABESHIMA (1971) General scheduling algorithms with applications to parallel scheduling and multiprogramming scheduling. *J. Operations Res. Soc. Japan* 14,72-99.

L. NÉMETI (1964) Das Reihenfolgeproblem in der Fertigungsprogrammierung und Linearplanung mit logischen Bedingungen. *Mathematica (Cluj)* 6,87-99.

R.J. ORD-SMITH (1970) Generation of permutation sequences: part 1. *Comput. J.* 13,152-155.

R.J. ORD-SMITH (1971) Generation of permutation sequences: part 2. *Comput. J.* 14,136-139.

C.S. ORLOFF (1974A) A fundamental problem in vehicle routing. *Networks* 4,35-64.

C.S. ORLOFF (1974B) Routing a fleet of M vehicles to/from a central facility. *Networks* 4,147-162.

D.S. PALMER (1965) Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum. *Operational Res. Quart.* 16,101-107.

J. PIEHLER (1960) Ein Beitrag zum Reihenfolgeproblem. *Unternehmensforschung* 4,138-142.

J.F. PIERCE, W.B. CROWSTON (1971) Tree-search algorithms for quadratic assignment problems. *Naval Res. Logist. Quart.* 18,1-36.

I. POHL (1975) Practical and theoretical considerations in heuristic search algorithms. Memo HP-75-1, Department of Information Sciences, University of California, Santa Cruz.

V.R. PRATT (1972) An n log n algorithm to distribute n records optimally in a sequential access file. [Miller & Thatcher 1972, 111-118].

R.C. PRIM (1957) Shortest connection networks and some generalizations. *Bell System Tech. J.* 36,1389-1401.

R.C. READ (1972) A note on the generation of rosary permutations. *Comm. ACM* 15,775.

S.S. REDDI, C.V. RAMAMOORTHY (1972) On the flow-shop sequencing problem with no wait in process. *Operational Res. Quart.* 23,323-331.

S.S. REDDI, C.V. RAMAMOORTHY (1973A) Reply to Dr. Goyal's comments. *Operational Res. Quart.* 24,133-134.

S.S. REDDI, C.V. RAMAMOORTHY (1973B) A scheduling problem. *Operational Res. Quart.* 24,441-446.

E.M. REINGOLD, J. NIEVERGELT, N. DEO (1976) *Combinatorial Computing.* To appear.

S. REITER, G. SHERMAN (1965) Discrete optimizing. *J. SIAM* 13,864-889.

A.H.G. RINNOOY KAN (1974) On Mitten's axioms for branch-and-bound. *Operations Res.*, to appear.

A.H.G. RINNOOY KAN (1976) *Machine Scheduling Problems: Classification, Complexity and Computations*. Nijhoff, The Hague.

A.H.G. RINNOOY KAN, B.J. LAGEWEG, J.K. LENSTRA (1975) Minimizing total costs in one-machine scheduling. *Operations Res.* 23,908-927; [Roy 1975, 343-350].

A.W. ROES (1973) Enige methoden ter verkrijging van een routeschema voor de interinsulaire scheepvaart in Indonesië. Unpublished Report, Universiteit van Amsterdam.

J.L. ROSENFELD (ed.) (1974) *Information Processing 74*. North-Holland, Amsterdam.

D.J. ROSENKRANTZ, R.E. STEARNS, P.M. LEWIS (1974) Approximate algorithms for the traveling salesperson problem. *Proc. 15th Annual Symp. Switching & Automata Theory*, 33-42.

B. ROY (ed.) (1975) *Combinatorial Programming: Methods and Applications*. Reidel, Dordrecht.

B. ROY, B. SUSSMANN (1964) Les problèmes d'ordonnancement avec contraintes disjonctives. Note DS No.9 bis, SEMA, Montrouge.

M.K. ROY (1973) Reflection-free permutations, rosary permutations, and adjacent transposition algorithms. *Comm. ACM* 16,312-313.

S. SAHNI, T. GONZALES (1974) P-complete problems and approximate solutions. *Proc. 15th Annual Symp. Switching & Automata Theory*, 28-32.

N. SANG, M. FLORIAN (1970) A note on lower bounds for the machine scheduling problem. Publication #49, Département d'Informatique, Université de Montréal.

A. SCHILD, I.J. FREDMAN (1962) Scheduling tasks with deadlines and nonlinear loss functions. *Management Sci.* 9,73-81.

L. SCHRAGE (1970A) Solving resource-constrained network problems by implicit enumeration - nonpreemptive case. *Operations Res.* 18,263-278.

L. SCHRAGE (1970B) A bound based on the equivalence of min-max-completion time and min-max-lateness scheduling objectives. Report 7042, Department of Economics and Graduate School of Business, University of Chicago.

L. SCHRAGE (1971) Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript.

D. SHAPIRO (1966) *Algorithms for the Solution of the Optimal Cost and Bottleneck Traveling Salesman Problem.* Thesis, Washington University, St. Louis.

J. SHWIMER (1972) On the *N*-job, one-machine, sequence-independent scheduling problem with tardiness penalties: a branch-and-bound solution. *Management Sci.* 18,B301-313.

J.B. SIDNEY (1975) Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Res.* 23,283-298.

P. SLATER (1961) Inconsistencies in a schedule of paired comparisons. *Biometrika* 48,303-312.

R.D. SMITH, R.A. DUDEK (1967) A general algorithm for the solution of the *n*-job, *m*-machine sequencing problem of the flowshop. *Operations Res.* 15,71-82.

R.D. SMITH, R.A. DUDEK (1969) Errata. *Operations Res.* 17,756.

W.E. SMITH (1956) Various optimizers for single-state production. *Naval Res. Logist. Quart.* 3,59-66.

V. SRINIVASAN (1971) A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness. *Naval. Res. Logist. Quart.* 18,317-327.

B.G. SUSSMANN (1972) Scheduling problems with interval disjunctions. *Z. Operations Res.* 16,165-178.

M.M. SYSLO (1974) On some machine sequencing problems (II). *Zastos. Mat.* 14, 93-97.

W. SZWARC (1960) Solution of the Akers-Friedman scheduling problem. *Operations Res.* 8,782-788.

W. SZWARC (1971) Elimination methods in the $m \times n$ sequencing problem. *Naval Res. Logist. Quart.* 18,295-305.

W. SZWARC (1973) Optimal elimination methods in the $m \times n$ sequencing problem. *Operations Res.* 21,1250-1259.

W. SZWARC (1975) Remarks. *Operations Res.* 23, 1043.

Y. TABOURIER (1972) Un algorithme pour le problème d'affectation. *Rev. Française Automat. Informat. Recherche Opérationelle* 6.V3,3-15.

R.E. TARJAN (1975A) Finding minimum spanning trees. Memorandum No.ERL-M501, Electronics Research Laboratory, University of California, Berkeley.

R.E. TARJAN (1975B) Finding optimum branchings. Memorandum No.ERL-M506, Electronics Research Laboratory, University of California, Berkeley.

G.L. THOMPSON (1975) Algorithmic and computational methods for solving symmetric and asymmetric travelling salesman problems. Presented at Workshop on Integer Programming, Bonn, September 8-12.

N. TOMIZAWA (1971) On some techniques useful for solution of transportation network problems. *Networks* 1,173-194.

C. TOMPKINS (1956) Machine attacks on problems whose variables are permutations. *Proc. Sympos. Appl. Math.* 6, Amer. Math. Soc., Providence, 195-211.

H.F. TROTTER (1962) Algorithm 115, Perm. *Comm. ACM* 5,434-435.

J.D. ULLMAN (1975) *NP*-complete scheduling problems. *J. Comput. System Sci.* 10,384-393.

J.M. VAN DEMAN, K.R. BAKER (1974) Minimizing mean flowtime in the flow shop with no intermediate queues. *AIIE Trans.* 6,28-34.

J. VISSCHERS, P. TEN KATE (1973) BAKALG, een bedradingsprogramma met optimalisatie van draadlengte. Report SO-1, Instituut voor Kernphysisch Onderzoek, Amsterdam.

M.B. WELLS (1961) Generation of permutations by transposition. *Math. Comp.* 15,192-195.

M.B. WELLS (1971) *Elements of Combinatorial Computing*. Pergamon, Oxford.

J.A.M. WESSELING (1975) Nieuwe ondergrenzen voor handelsreizigersalgorithmen. Instituut voor Actuariaat en Econometrie, Universiteit van Amsterdam.

D.A. WISMER (1972) Solution of the flowshop-scheduling problem with no intermediate queues. *Operations Res.* 20,689-697.

A. WREN, A. HOLLIDAY (1972) Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Res. Quart.* 23,333-344.

AUTHOR INDEX

SUBJECT INDEX

SAMENVATTING

Volgordeproblemen doen zich onder vele omstandigheden voor. Men kan zich bijvoorbeeld gesteld zien voor de opgave een optimale produktievolgorde van jobs op machines te kiezen, de kortste route voor een vrachtwagen langs een aantal klanten te bepalen of de meest waarschijnlijke chronologische ordening van archeologische vondsten te specificeren. Elk van deze situaties leidt tot *combinatorische optimaliseringsproblemen*, waarbij we het optimale element uit een grote maar eindige verzameling toegelaten oplossingen proberen te vinden.

In dit proefschrift houden we ons voornamelijk met twee klassen van problemen bezig. De eerste klasse bevat het *kwadratische toewijzingsprobleem* en de problemen die als speciale gevallen hiervan geïnterpreteerd kunnen worden. Hieronder vallen het *acyclische-deelgraafprobleem*, waar gezocht wordt naar een rangschikking van alternatieven die aan zo weinig mogelijk uitgesproken voorkeuren voorbijgaat, en het *handelsreizigersprobleem*, waar een handelsreiziger zich afvraagt in welke volgorde hij een aantal steden moet bezoeken opdat hij zo snel mogelijk weer thuis is. Beide problemen kennen vele verrassende toepassingen.

Het grootste gedeelte van dit proefschrift is gewijd aan *machinevolgordeproblemen*. Deze problemen treden op wanneer een aantal jobs dient te worden uitgevoerd door een aantal machines die elk slechts één job tegelijk kunnen behandelen. Een job bestaat uit een serie bewerkingen die elk een gegeven tijd vergen op een machine; voor elke job ligt de bewerkingsvolgorde vast. Gevraagd wordt naar een produktievolgorde op elke machine zodanig dat het resulterende produktieschema optimaal is met betrekking tot een gegeven globale kostenfunctie.

De methoden die ontwikkeld zijn om volgordeproblemen optimaal op te lossen vallen in twee klassen uiteen. Enerzijds bestaan er voor sommige problemen *goede* of *efficiënte* algoritmen die hoogstens een voorspelbaar aantal stappen vergen dat polynomiaal afhangt van de omvang van het probleem. Anderzijds kunnen we vele problemen slechts oplossen door alle toegelaten oplossingen expliciet of impliciet af te tellen. Expliciete aftelling vereist vaak een aantal stappen dat superexponentieel toeneemt met de probleemomvang en is alleen voor zeer kleine problemen bruikbaar; impliciete-aftellingsmethoden zoals "branch-and-bound" vertonen gewoonlijk een grillig en moeilijk voorspelbaar gedrag.

Recente resultaten uit de concrete complexiteitstheorie maken het mogelijk te bepalen in welke gevallen het gebruik van dergelijke *aftellings-*

methoden onvermijdelijk lijkt. Er bestaat een klasse van zogenaamd NP-complete problemen met de eigenschap dat een polynomiaal begrensde algoritme voor enig NP-compleet probleem via probleemtransformaties leidt tot goede algoritmen voor alle NP-complete problemen. Aangezien nu vele beruchte combinatorische problemen zoals het handelsreizigersprobleemn, het knapzakprobleem en het lineaire 0-1 programmeringsprobleem NP-compleet blijken te zijn, is het bestaan van een goede algoritme voor deze klasse problemen hoogst onwaarschijnlijk. Het bewijs dat een bepaald volgordeprobleem eveneens NP-compleet is kan daarom worden gebruikt als een formele rechtvaardiging voor het gebruik van aftellingsmethoden.

Met behulp van de hierboven aangeduide methodiek wordt in Deel I de *complexiteit* van volgordeproblemen onderzocht. We proberen de grens tussen "gemakkelijke" en "moeilijke" problemen zo scherp mogelijk te trekken. Het blijkt dat voor het merendeel inderdaad een beroep moet worden gedaan op aftellingsmethoden.

De aard van dergelijke methoden brengt met zich mee dat hun gedrag sterk afhankelijk is van de wijze waarop zij worden geprogrammeerd. Dit rechtvaardigt een diepgaande bestudering van een aanpak die zeer aantrekkelijk is gebleken, namelijk een *recursieve* implementatie van aftellingsmethoden. In Deel II worden de voordelen van een dergelijke aanpak gedemonstreerd aan de hand van enige eenvoudig gestructureerde voorbeelden van expliciete en impliciete aftelling.

Deel III handelt over de oplossing van volgordeproblemen door middel van impliciete aftelling. Voor een vijftal belangrijke NP-complete problemen worden overzichten en uitbreidingen van branch-and-bound algoritmen gepresenteerd. Er is naar gestreefd het zoeken naar een optimale oplossing zoveel mogelijk te beperken door het gebruik van scherpe grenzen op de waarden van oplossingen binnen bepaalde deelverzamelingen. In sommige gevallen wordt een algemeen begrenzingsvoorschrift geformuleerd dat alle eerder ontwikkelde ondergrenzen voortbrengt. Onze rekenervaring met diverse methoden wordt in detail besproken.

Het grote aantal praktische *toepassingen* van volgordeproblemen is al ter sprake gebracht. In Deel IV beschrijven we vijf problemen die uit praktijksituaties naar voren zijn gekomen; zij worden op succesvolle wijze opgelost met behulp van modellen en methoden uit voorgaande hoofdstukken.

Tenslotte worden er enige veelbelovende gebieden voor verder onderzoek aangegeven. Zo bestaat er behoefte aan een verfijning van de complexiteitsmaat waarmee met name de verschillen in complexiteit binnen de klasse der

NP-complete problemen een verklaring kunnen vinden. In deze context kan het van nut zijn een onderzoek in te stellen naar het bestaan van benaderende algoritmen die een oplossing binnen een vast percentage van het optimum leveren.

Gezien de gebruikelijke omvang van praktijkproblemen en hun fundamentele complexiteit zullen we in de praktijk veelal genoegen moeten nemen met een heuristische methode die een bevredigende maar niet *per se* optimale oplossing vindt. De theorie kan een wezenlijke bijdrage leveren tot het ontwikkelen van heuristieken die probleemgerichter zijn dan de tot nu toe gepresenteerde krachteloze gis-en-mis-methoden.

STELLINGEN

1

In een ongerichte graaf $G = (V,E)$ met een niet-negatief gewicht c_{ij} voor iedere kant (i,j) moet aan ieder punt precies één van m kleuren worden toegewezen zodanig dat het totale gewicht van de kanten waarvan de eindpunten dezelfde kleur bezitten minimaal is. Iedere kleuring correspondeert met een toegelaten geheeltallige oplossing van onderstaand kwadratisch programmeringsprobleem, waarbij x_{hi} te interpreteren is als de fractie van punt i die kleur h bezit:

$$\min\{\textstyle\sum_{h=1}^{m} \sum_{(i,j)\in E} c_{ij}x_{hi}x_{hj} \mid \sum_{h=1}^{m} x_{hi} = 1 \ (i \in V),$$
$$x_{hi} \geq 0 \ (h = 1,\ldots,m,\ i \in V)\}.$$

Dit probleem heeft een geheeltallige optimale oplossing. Een geheeltallige toegelaten oplossing voldoet aan de voorwaarden van Kuhn en Tucker dan en slechts dan als zij overeenkomt met een kleuring die locaal optimaal is in die zin dat het onvoordelig is één enkel punt i van kleur te veranderen, voor elke $i \in V$.

2

In een netwerk wordt gezocht naar de kortste gesloten route die bepaalde punten en kanten doorloopt. De transformatie waarmee C.S. Orloff verplichte punten vervangt door verplichte kanten kan leiden tot niet-optimale oplossingen en laat bovendien de NP-compleetheid van het probleem onverlet.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1974) On general routing problems. Zal verschijnen in *Networks*.

C.S. ORLOFF (1974) A fundamental problem in vehicle routing. *Networks* 4, 35-64.

3

Gegeven zij een familie die drie generaties omvat en onder meer bestaat uit m grootvaders en n-m kleinzonen. In het familiebedrijf zijn n functies vrijgekomen die moeten worden toegewezen aan grootvaders en kleinzonen. Deze functies zijn totaal geordend; iedere kleinzoon dient een lagere positie in de hiërarchie in te nemen dan zijn twee grootvaders. Voorts is van elk dezer personen bekend voor welke functies hij al dan niet geschikt is. Er wordt gezocht naar een toewijzing waarbij het aantal competente functionarissen zo groot mogelijk is. Dit probleem is NP-compleet; indien aan de familierelaties wordt voorbijgegaan is het in $\mathcal{O}(n^{2\frac{1}{2}})$ stappen oplosbaar.

4

Aan de polemiek betreffende de chronologische ordening van de avonturen van Sherlock Holmes en Dr. John H. Watson kan een verhelderende bijdrage worden geleverd door oplossing van een acyclische-deelgraafprobleem. Wenst men uitsluitend het aantal huwelijken van Dr. Watson te minimaliseren, dan dient men een handelsreizigersprobleem op te lossen.

W.S. BARING-GOULD (1968) *The Annotated Sherlock Holmes: The Four Novels and the Fifty-six Short Stories Complete, by Sir Arthur Conan Doyle; Edited, with an Introduction, Notes, and Bibliography*. Murray, London.

5

Een sultan wiens harem uit veertien vrouwen bestaat maar wiens divan aan slechts vier van hen plaats biedt wenst alle verschillende combinaties af te tellen in een zodanige volgorde dat telkens precies één dame haar plaats aan een collega dient af te staan. De deelrij van de binaire gespiegelde Gray-code bestaande uit de configuraties die precies vier elementen bevatten levert hem een reeks van 1001 nachten met de gewenste eigenschap.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1975) A recursive approach to the generation of combinatorial configurations. Report BW 50, Mathematisch Centrum, Amsterdam.

6

Gegeven twee positieve gehele getallen c en d definiëren we de rij $(a(n))_{n=1}^{\infty}$ door $a(1) = d$, $a(n+1) = c^{a(n)}$. Laat nu $(b_i)_{i=0}^{\infty}$ een rij gehele getallen groter dan 1 zijn. We schrijven de a(n) in het gemengde talstelsel behorende bij de b_i:

$$a(n) = a(n)_0 + a(n)_1 b_0 + a(n)_2 b_1 b_0 + a(n)_3 b_2 b_1 b_0 + \dots$$

waarbij elk cijfer $a(n)_i$ een geheel getal is met $0 \le a(n)_i < b_i$. Dan is voor iedere i de rij $(a(n)_i)_{n=1}^{\infty}$ uiteindelijk constant:

$$\forall\ i : \exists\ n_0 : \forall\ n \ge n_0,\ m \ge n_0 : a(n)_i = a(m)_i .$$

Voorts hangt deze uiteindelijke waarde niet van d af.

Nemen we bijvoorbeeld c = 2 en kiezen we het factoriële talstelsel (d.w.z. $b_i = i+2$, $i = 0,1,2,\dots$), dan zijn de cijfers $a(n)_0, a(n)_1, \dots, a(n)_{24}$ voor alle d en alle voldoend grote n gelijk aan respectievelijk:

0,2,2,0,0,0,5,4,3,7,2,7,1,12,5,15,16,1,19,3,9,20,18,13,1.

7

Bij het berekenen van kleinste-kwadratenschatters voor de parameters $\beta_{31}, \dots, \beta_{3r}$ in het model

$$\underline{y}_{ij} = \textstyle\sum_{k=1}^{p} \beta_{1jk} x_{1ik} + \sum_{k=1}^{q} \beta_{2ik} x_{2jk} + \sum_{k=1}^{r} \beta_{3k} x_{3ijk} + \underline{u}_{ij}$$

waarin $1 \le i \le m$ en $1 \le j \le n$, kan de gewoonlijk gebruikte methode, die inversie vereist van een s×s-matrix met $s = np+mq+r-pq$, worden vervangen door een methode waarbij slechts een p×p-, een q×q- en een r×r-matrix behoeven te worden geïnverteerd.

8

Bij informatici binnen de Verenigde Staten van Amerika ontmoet men onwil om in de discussie aangaande de relatieve merites van recursieve en iteratieve beschrijvingen van algoritmen naast het aspect van efficiëntie ook de aspecten van elegantie en inzichtelijkheid te betrekken. Dit is vermoedelijk een cultuurpsychologisch verschijnsel.

9

Tenminste de helft van alle verwijzingen naar het artikel van het handelsreizigerskwartet Little, Murty, Sweeney en Karel bevat één of meer onjuistheden.

J.D.C. LITTLE, K.G. MURTY, D.W. SWEENEY, J. KAREL (1963) An algorithm for the traveling salesman problem. *Operations Res.* 11,972-989.

10

De coëxistentietheorie van Leibniz impliceert dat de schepping van de wereld kan worden geïnterpreteerd als de oplossing van een wezenloos grote instantie van het NP-complete kliekprobleem.

B. RUSSELL (1946) *History of Western Philosophy and its Connection with Political and Social Circumstances from the Earliest Times to the Present Day*. Allen & Unwin, London.