Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

A.W.J. Kolen, J.K. Lenstra

Combinatorics in operations research

Department of Operations Research, Statistics, and System Theory Report BS-R9024 September

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Combinatorics in Operations Research

Antoon W.J. Kolen
University of Limburg, Maastricht

Jan Karel Lenstra
Eindhoven University of Technology, Eindhoven
Centre for Mathematics and Computer Science, Amsterdam

This is a collection of examples of the use of combinatorial techniques in practical decision situations. The emphasis is on the description of real-world problems, the formulation of mathematical models, and the development of algorithms for their solution. We survey related models and applications.

1980 Mathematics Subject Classification (1985 Revision): 90C10, 90C27, 90C35, 90B05, 90B10, 90B35.
Key Words & Phrases: traveling salesman, vehicle routing, multiple postmen, linear ordering, clique partitioning, test cover, bottleneck extrema, minimum cost flow, interval scheduling, job shop scheduling.
Note: This is Chapter 35 of the *Handbook of Combinatorics*, edited by R.L. Graham, M. Grötschel and L. Lovász, and to be published by North-Holland, Amsterdam.

TABLE OF CONTENTS

0. INTRODUCTION

The spectacular growth of combinatorics over the past few decades is to some extent due to the diversity and the importance of its applications. Combinatorial problems occur in other branches of mathematics and in computer science, in the natural sciences and in the humanities, and in all kinds of practical decision situations. In addition, the solution of these combinatorial problems has often yielded significant advances and benefits. There is little doubt that its successful use in other fields has greatly stimulated research in the area of combinatorics.

The sample of applications of combinatorics presented in this chapter all arise in situations

Report BS-R9024
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

of planning and design that are usually dealt with in operations research. This discipline is concerned with the investigation of models and methods that support decision making in practice. We do not intend to give a complete survey. Instead, we have tried to select some typical examples with the hope of conveying the flavor of the subject. Our selection process has been guided by the following biases and principles.

First, it has been our purpose to show how to solve real-world problems, not how to construct applications of combinatorial models. With one or two exceptions, each of our examples finds its origin in practice. At the same time, we have preferred those problems that give rise to clean and elegant models, and we have avoided complications that in the present context would only obscure the essence.

Further, most of the problems we discuss occurred in the Netherlands. While this emphasis reflects the limitations of our experience, we do not feel that it has narrowed the scope of our examples.

Finally, we will concentrate on the design and analysis of models and algorithms. Collecting data, writing computer codes, building user interfaces, and getting solutions implemented are equally important stages in the practice of combinatorics. They do not belong, however, to the subject matter of this chapter.

Each of the sections below follows the same outline: we describe a practical problem, formulate one or more mathematical models, present suitable solution methods, and survey related models and applications. It is assumed that the reader is familiar with the fundamentals of combinatorics and combinatorial optimization. We refer, in particular, to Chapter 2 on connectivity and network flows, Chapter 3 on matching, Chapter 4 on coloring, stable sets and perfect graphs, Chapter 28 on optimization, and Chapter 30 on polyhedral combinatorics.

1. Traveling salesman

1.1. *X-rays and arrays*

To demonstrate the versatility of the traveling salesman problem as a model, we will consider two very different problem situations.

The first situation involves the *sequencing of X-ray measurements* in crystallography. One wishes to analyze the detailed structure of a crystal. To this end, the crystal is mounted in a diffractometer, and the intensity of X-rays is measured for a large number of positions of the crystal and the reading device inside the apparatus. Such an experiment may require many thousands of readings. These readings can be made relatively quickly, but the repositioning time between successive measurements is substantial. The readings can be taken in any order, and the question is how one should sequence them so as to minimize the time to complete the experiment.

Bland and Shallcross [1989] encountered problems of this type at the Cornell High Energy Synchrotron Source. For experiments with up to 14,464 readings, they computed sequences for which the total repositioning time is within 1.7% of a lower bound on the optimum. The standard method used for sequencing the measurements produced solutions that are generally between 55% and 90% above the optimum.

The second situation concerns the *clustering of a data array*. Given are two finite sets R and S and a nonnegative matrix $(a_{rs})_{r\in R, s\in S}$, where a_{rs} measures the strength of the relationship

between elements $r \in R$ and $s \in S$. One would like to permute the rows and columns of the matrix so as to bring its large elements together. The resulting *clustering* should identify strong relationships between subsets of R and S.

McCormick, Schweitzer and White [1972] argue that clustering a matrix may be useful for problem decomposition and data reorganization. They illustrate this with three examples. The first one arises in *airport design*. R $(= S)$ is a set of 27 facilities that should be available at the airport and that are under the control of the designer; a_{rs} is fixed at 0, 1, 2 or 3 depending on whether facilities r and s have no, a weak, a moderate or a strong interdependence. The permuted matrix should suggest a decomposition of the design problem into subproblems that interact not at all or only in a limited and well defined way. The second example involves a set R of 53 *aircraft types* and a set S of 37 functions they can perform; $a_{rs} = 1$ if aircraft r is suitable for function s, and $a_{rs} = 0$ otherwise. The rearranged matrix shows which aircraft are able to perform the same functions and which tasks can be performed by the same aircraft. The third example also deals with an object-attribute array. R is a set of 24 *marketing* techniques, S is a set of 17 marketing applications, $a_{rs} = 1$ if technique r has been successfully used for application s, and $a_{rs} = 0$ otherwise. Lenstra and Rinnooy Kan [1975] give a fourth example. It deals with an *input-output matrix*. R $(= S)$ is a set of 50 regions on the Indonesian islands, $a_{rs} = 1$ if at least 50 tons of rice are annually transported from region r to region s, and $a_{rs} = 0$ otherwise.

1.2. *Model formulation*

Both problems can be modeled as a *traveling salesman problem*. This is the problem of a salesman who, starting from his home city, has to find the shortest tour that takes him exactly once through each of a number of other cities and then back home. Suppose there are n cities and c_{ij} is the distance between cities i and j $(i,j = 1, \ldots, n)$. The salesman is interested in a permutation π of $\{1, \ldots, n\}$ that minimizes

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

here, $\pi(i)$ is the ith city visited. The traveling salesman problem is *symmetric* if $c_{ij} = c_{ji}$ for all i,j.

It is straightforward to cast the sequencing problem in these terms. We identify the readings with the cities and the repositioning time between two readings with the distance between the corresponding cities. We then add one more city with equal distances to the others, in order to transform the problem of finding an open sequence into that of finding a closed tour. Note that the distances are symmetric.

As to the clustering problem, we first have to convert it into an optimization problem. McCormick, Schweitzer and White [1975] propose to measure the effectiveness of a clustering by the sum of all products of horizontally or vertically adjacent elements. The reader can easily convince himself that higher sums of these products tend to correspond to better clusterings. The problem is now to permute the rows and columns of the matrix so as to maximize this criterion.

Permuting the rows does not affect the horizontal adjacencies of the elements, and permuting the columns does not affect their vertical adjacencies. The problem therefore decomposes into two separate and similar problems, one for the rows and one for the columns. We consider

the former. The row optimization problem is to find a permutation ρ of R that maximizes

$$\sum_{r=1}^{|R|-1} \sum_{s \in S} a_{\rho(r)s} a_{\rho(r+1)s};$$

here, row $\rho(r)$ of the matrix is put in position r. This is, again, nothing but the symmetric traveling salesman problem in disguise [Lenstra, 1974]. Let $R = \{1, \ldots, |R|\}$, and define

$$n = |R| + 1,$$

$$c_{ij} = -\sum_{s \in S} a_{is} a_{js}, \quad c_{in} = c_{nj} = 0 \quad \text{for } i,j \in R.$$

The rows of the matrix are the cities, the additive inverses of their inner products are the distances, and a dummy city has been added to close the tour.

The symmetric traveling salesman problem is conveniently formulated in terms of undirected graphs. Consider the complete graph $K_n = (V,E)$ on n nodes, where a weight c_e is associated with each edge $e \in E$. The problem is to find a *Hamiltonian circuit* or *tour* in K_n, i.e., a circuit that visits each node exactly once, of minimum total weight. This generalizes the problem of determining whether a given graph contains a Hamiltonian circuit, which is *NP*-complete.

An *integer programming* formulation is as follows. Let $x_e = 1$ indicate that the salesman travels along the edge e. The problem is then to minimize

$$\sum_{e \in E} c_e x_e$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \text{for all } i \in V, \tag{1.1}$$

$$\sum_{e \in \delta(U)} x_e \geqslant 2 \qquad \text{for all } U \subset V,\ U \neq \varnothing,\ U \neq V, \tag{1.2}$$

$$x_e \in \{0,1\} \qquad \text{for all } e \in E. \tag{1.3}$$

Here, $\delta(U)$ is the set of edges with exactly one end in U; we write $\delta(i)$ for $\delta(\{i\})$. Without the constraints (1.2), this is the *2-matching problem*; each node has degree 2, but the selected edges do not necessarily form a single tour. The constraints (1.2) eliminate subtours on any proper subset $U \subset V$.

1.3. *Solution approaches*

Many types of *approximation algorithms* have been developed for the symmetric traveling salesman problem. It is useful to distinguish between *constructive* methods, which build a single tour, and *iterative improvement* methods, which search the neighborhood of the current solution for a better one and continue the search until a local optimum has been obtained. An example of a constructive method is the *nearest neighbor rule*: the salesman starts in a given city and always travels to an unvisited city that is closest to the last chosen city. One of the earliest iterative improvement methods was proposed by Lin [1965]. He defines the *k-exchange neighborhood* of a tour as the set of all tours that can be obtained from it by replacing any set of k edges by another set of k edges, and he calls a tour that is locally optimal with respect to this neighborhood *k-opt*. The values $k = 2$ and $k = 3$ are most often used. The champion among heuristics for the symmetric traveling salesman problem is the *variable-depth search* method of

Lin and Kernighan [1973], where the value of k is not specified in advance.

In order to enhance the computational efficiency of edge exchange procedures on large problem instances, one usually replaces K_n by a sparse subgraph. Bland and Shallcross [1989], for example, included for each node only the ten shortest incident edges. Their upper bounds were computed by the Lin-Kernighan algorithm, and their lower bounds by the Held-Karp algorithm mentioned below.

Optimization algorithms for the symmetric traveling salesman problem usually proceed by branch and bound, with lower bounds based on spanning 1-trees combined with Lagrangean relaxation, or on fractional 2-matchings combined with cutting planes.

A *spanning 1-tree* consists of a spanning tree on the node set $V \setminus \{1\}$ and two edges incident to node 1. A minimum-weight spanning 1-tree can be found in polynomial time. In comparison with a tour, it is still a connected graph with n edges, but the requirement that all node degrees should be equal to 2 has been relaxed. Its weight is therefore a lower bound on the length of a shortest tour. The lower bound may be improved by calculating a penalty d_i for each node i (positive if the degree of i is larger than 2, negative if it is smaller) and to replace the weight c_e by $c_e + d_i + d_j$ for each edge $e = \{i,j\}$. The ordering of tours according to length is invariant under this transformation, but the optimal spanning 1-tree may change. The approach is due to Held and Karp [1970, 1971] and signified the beginning of the use of *Lagrangean relaxation* in combinatorial optimization. The penalties or *Lagrangean multipliers* d_i are calculated by general subgradient optimization techniques or by special multiplier adjustment schemes.

A *fractional 2-matching* is a feasible solution to (1.1) and $0 \leqslant x_e \leqslant 1$ ($e \in E$). In comparison with a tour, the subtour elimination constraints (1.2) and the integrality requirements in (1.3) have been relaxed. An optimal fractional 2-matching can be obtained in polynomial time, e.g., by linear programming. The resulting lower bound can be improved by the addition of *cutting planes* that correspond to facets of the symmetric traveling salesman polytope, i.e., the convex hull of all solutions to (1.1-3). The subtour elimination constraints (1.2) define facets, but many more classes of facets have been identified. Given the solution corresponding to such a lower bound, one tries to find a violated facet and adds the corresponding constraint to the linear program. If the sequence of linear programs does not yield a feasible solution to (1.1-3) (and hence an optimal tour), then some form of tree search is applied. There are, in general, two difficulties with this *polyhedral approach.* First, it is very unlikely that one will ever be able to characterize all of the facets. Secondly, the so-called *separation problem* of finding a facet that is violated by the solution to the linear program is often far from trivial; usually, fast heuristics are used. However, Padberg and Rinaldi [1987] and Grötschel and Holland [1990] have obtained impressive computational results with this approach.

For more detailed information on branch and bound, Lagrangean relaxation, polyhedral techniques, and the relation between the optimization problem and the separation problem, see Chapters 28 and 30.

1.4. *Related models and applications*

As has already been observed, the problem of determining whether a given graph contains a Hamiltonian circuit is a special case of the traveling salesman problem. This, in turn, generalizes to the *vehicle routing problem*, which is the subject of Section 2.

A quite different class of routing problems emerges if one wishes to visit all of the *edges* (or

arcs) of a graph rather than all of the nodes. The basic problem is then to determine if a given graph contains an *Eulerian tour*, i.e, a closed walk that traverses each edge (or arc) exactly once. This generalizes to the *Chinese postman problem*, where one has to find the shortest closed walk that traverses each edge (or arc) at least once. It will arise in the solution of a practical multi-postman problem in Section 3.

The traveling salesman problem occurs in many more practical situations than sequencing measurements, clustering matrices, or routing vehicles. Ratliff and Rosenthal [1983] describe the problem of order picking in a rectangular warehouse. This is a traveling salesman problem that, due to the structure of the underlying network, can be solved in polynomial time by dynamic programming techniques. Other applications are discussed by Lenstra and Rinnooy Kan [1975].

The traveling salesman problem has become the prototypical problem of combinatorial optimization. This is partly because its simplicity of statement and difficulty of solution are even more apparent than for most other problems in the area. In addition, many of the solution approaches that have become standard in combinatorial optimization were first developed and tested in the context of the traveling salesman problem. Our presentation of mathematical formulations, solution approaches and applications is only meant to be illustrative. A full treatment of the problem justifies a book of its own [Lawler, Lenstra, Rinnooy Kan and Shmoys, 1985].

2. Vehicle routing

2.1. *CAR*

In the period 1983-1986, the Centre for Mathematics and Computer Science (CWI) in Amsterdam was involved in the development of a computer system for vehicle routing. The resulting system is called CAR, which stands for 'Computer Aided Routing'. Before we discuss the models and the algorithms that form the mathematical basis of CAR, we review some aspects of the practical background and the computer implementation in this section. The reader is referred to Savelsbergh [1990] for details.

CAR has been designed for the solution of the *single-depot vehicle routing problem with time windows*. A problem situation of this type that occurred at the hanging garment division of a Dutch road transportation firm was our main source of information and motivation from practice. The situation is basically as follows. About fifteen vehicles are stationed at a single central depot and must serve about 500 geographically dispersed customers. Each vehicle has a given capacity. Each customer has a given demand and must be served within a specified time interval. The travel times between the locations of the depot and the customers are given. We have to find a collection of routes for the vehicles, each starting and finishing at the depot and collectively visiting all customers, while respecting the capacity constraints of the vehicles and the time constraints of the customers. We would like to minimize the total travel time.

Problems of this type and size must be solved daily. There are several reasons why, at the present time, it is not possible to completely automate their solution. On the one hand, the models that arise are *hard*, in a well defined sense. Better solutions are obtained if the computer system and its user cooperate and divide the tasks in accordance with their respective — and complementary — capabilities. On the other hand, the problem situations are *soft*.

Feasibility constraints can be stretched, and optimality on one criterion will be weighed against the values of secondary criteria. This is not the place to advocate the benefits of man-machine interaction in complex decision situations. Suffice it to refer to Anthonisse, Lenstra and Savelsbergh [1988] and to mention that CAR has been designed and built as an interactive system, which does not make decisions but only supports decision making by the people who are in charge. The system is being used by several firms in the Netherlands.

2.2. *Model formulation*

We present an integer programming formulation of the single-depot vehicle routing problem. For the time being, we ignore the time windows of the customers.

The data of the problem are as follows. There are m vehicles. The capacity of vehicle h is equal to Q_h ($h = 1, \ldots, m$). The depot is indexed by $i = 1$ and the customers by $i = 2, \ldots, n$; the demand of customer i is equal to q_i ($i = 2, \ldots, n$). Finally, there is a matrix $(c_{ij})_{i,j=1}^{n}$ of travel times. As to the decision variables, let

$$y_{hi} = \begin{cases} 1 & \text{if vehicle } h \text{ visits customer } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$x_{hij} = \begin{cases} 1 & \text{if vehicle } h \text{ visits customers } i \text{ and } j \text{ in sequence,} \\ 0 & \text{otherwise.} \end{cases}$$

The problem is now to minimize

$$\sum_{h=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{hij}$$

subject to

$$\sum_{h=1}^{m} y_{hi} = \begin{cases} m & \text{for } i = 1, \\ 1 & \text{for } i = 2, \ldots, n, \end{cases} \tag{2.1}$$

$$\sum_{i=2}^{n} q_i y_{hi} \leqslant Q_h \qquad \text{for } h = 1, \ldots, m, \tag{2.2}$$

$$y_{hi} \in \{0,1\} \qquad \text{for } h = 1, \ldots, m,\ i = 1, \ldots, n, \tag{2.3}$$

$$\sum_{j=1}^{n} x_{hij} = \sum_{j=1}^{n} x_{hji} = y_{hi} \qquad \text{for } h = 1, \ldots, m,\ i = 1, \ldots, n, \tag{2.4}$$

$$\sum_{i,j \in U} x_{hij} \leqslant |U| - 1 \qquad \text{for } h = 1, \ldots, m,\ U \subset \{2, \ldots, n\}, \tag{2.5}$$

$$x_{hij} \in \{0,1\} \qquad \text{for } h = 1, \ldots, m,\ i,j = 1, \ldots, n. \tag{2.6}$$

The conditions (2.1) ensure that each customer is allocated to one vehicle and that the depot is allocated to each vehicle. The conditions (2.2) are the vehicle capacity constraints. The conditions (2.4) ensure that a vehicle which arrives at a customer also leaves that customer. The conditions (2.5) are the subtour elimination constraints, in a form that differs from (1.2).

This formulation is due to Fisher and Jaikumar [1981]. They observed that it consists of a number of interlinked subproblems, namely, a generalized assignment problem and a collection of m traveling salesman problems. The *generalized assignment problem* is the problem of minimizing

$$\sum_{h=1}^{m} f_h(y_{h1}, \ldots, y_{hn})$$

subject to (2.1-3). Here, the y_{hi} are the decision variables and $f_h(y_{h1}, \ldots, y_{hn})$ is the minimum time duration of a tour through the depot and the cluster of customers defined by $\{i \mid y_{hi} = 1\}$. These are *traveling salesman problems*, i.e., $f_h(y_{h1}, \ldots, y_{hn})$ is equal to the minimum value of

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{hij}$$

subject to (2.4-6). Here, the x_{hij} are the decision variables and the y_{hi} prescribe the allocation of customers to vehicles.

The traveling salesman problem is *NP*-hard, and so is the generalized assignment problem, even if its criterion function is linear in the y_{hi}. However, these subproblems have been well studied.

2.3. *Solution approaches*

Fisher and Jaikumar originally proposed to solve the single-depot vehicle routing problem to optimality by an iterative process, which can be viewed as an application of Benders decomposition. Replacing each $f_h(y_{h1}, \ldots, y_{hn})$ by a lower linear support, they solve the generalized assignment problem, which provides a *lower bound* on the overall solution value and a tentative *clustering* of the customers into vehicles. They then solve the m resulting traveling salesman problems, which yields an *upper bound* on the overall solution value and a tentative *routing* for each vehicle. In the second iteration, the generalized assignment problem is solved again, with an improved lower linear support derived from the solution obtained in the first iteration, and the process continues. As soon as lower bound and upper bound are equal, an optimal solution has been obtained.

This approach is notable for its conceptual value, not for its computational efficiency. It motivated the development of an approximation algorithm. This is a *cluster first-route second* approach, which essentially consists of the first iteration of the optimization procedure. Much depends on the linearization of the functions f_h that is chosen. Fisher and Jaikumar [1981] propose to select a *seed point* $s(h)$ for each vehicle h; $s(h)$ is a customer who is centrally located in the area that is to be covered by vehicle h. They compute the 'extra mileage costs' $d_{hi} = c_{1i} + c_{is(h)} - c_{1s(h)}$, which should approximate the routing costs incurred if customer i is served by vehicle h. They then replace each $f_h(y_{h1}, \ldots, y_{hn})$ by $\sum_{i=1}^{n} d_{hi} y_{hi}$ and solve the linear generalized assignment problem. Finally, they solve a traveling salesman problem for each collection of customers allocated to the same vehicle.

Large generalized assignment problems can be solved close to optimality [Fisher, Jaikumar and Van Wassenhove, 1986]. Large traveling salesman problems are usually solved by edge exchange methods of the type discussed in Section 1.3.

We should draw the reader's attention to a small but crucial problem that we encountered during the development of CAR. For an unconstrained traveling salesman problem, it takes constant time to process a single edge exchange, as long as the number of edges involved is

bounded by a constant. In the presence of time windows, however, testing feasibility of the route that results from an edge exchange requires an amount of time that is linear in the number of cities. Savelsbergh [1990] developed techniques for implementing local search subject to time windows without an increase in overall time complexity. He extended these techniques to handle other side constraints such as multiple time windows per customer, mixed collections and deliveries, and precedence constraints.

2.4. *Related models and applications*

Vehicle routing problems can be modeled and solved in many different ways. Surveys are given by Bodin, Golden, Assad and Ball [1983] and Christofides (Chapter 12 in Lawler, Lenstra, Rinnooy Kan and Shmoys [1985]).

Desrochers, Lenstra, Savelsbergh and Soumis [1988] review the state of the art regarding routing with time window constraints. Next to standard vehicle routing problems with time windows, which are especially relevant in the context of school bus routing, they discuss pickup and delivery problems with time windows, which arise in dial-a-ride situations. The characteristic difference between the two problem types is that, in the latter case, pickup and delivery of the same commodity occurs in a single route. The models they present are based on integer programming, dynamic programming, and set partitioning.

After this discussion of node routing problems, the next section deals with *arc routing*. The salesman is replaced by the postman .

3. Multiple postmen

3.1. *Sprinkling highways*

In the winter, the highways in the Netherlands are sprinkled with salt to prevent them from becoming slippery. Some highways have built-in sensors, which indicate when the road temperature drops below a certain threshold; in other cases, the critical point in time is determined by visual inspection. Safety regulations require that, when the signal for preventive sprinkling is given, all highways in a region should be handled within a time period of 45 minutes. The salt sprinklers are stationed at various depots along the highways. They can carry an amount of salt that suffices for a period of 45 minutes. When sprinkling, they drive at a reduced speed. The highway system is such that a road segment may have to be traversed without being sprinkled. How many sprinklers are needed, and how should their routes be constructed?

This is a slight simplification of a problem that was handled by ORTEC Consultants in Rotterdam, in cooperation with the first author. The project resulted in a prototype program that, on a small problem instance, reduced the number of routes from seventeen to thirteen. Actual instances involve about 150 depots and 500 routes.

3.2. *Model formulation*

Highways are one-way streets. The highway system is therefore modeled as a directed graph. The road junctions and the salt depots are the nodes, and the road segments are the arcs. The graph is strongly connected. Each arc has two weights, indicating the time needed to traverse the arc while sprinkling and while driving without sprinkling, respectively. A feasible solution is a collection of directed walks, one for each vehicle, such that each walk starts at one of the

depot nodes, it is indicated for each occurrence of an arc in a walk whether it is sprinkled or not, each arc is sprinkled exactly once, and no walk exceeds a given upper bound in length. Note that a walk does not have to be closed; the time the vehicles need to return to their depots is irrelevant. A solution is optimal if the sum of the squared differences between upper bound and actual walk lengths is minimum. This criterion models the objective to make the walk lengths large on the one hand and more or less equal on the other.

The sprinkling problem belongs to the class of *arc routing* problems, which was already mentioned in Section 1.4. In designing a solution method for our problem, we will relate it to the standard arc routing problem, the *directed Chinese postman problem.* This problem is formulated as follows: given a strongly connected arc-weighted directed graph, find the shortest closed directed walk that traverses each arc at least once. It is solvable in polynomial time, as will be indicated in Section 3.3.

Our problem is essentially a *multi-postman problem* with some complexifying characteristics: there are several depots, to which the postmen do not have to return, and there is a quadratic cost function. Irrespective of the objective, the problem of deciding if two postmen stationed at one depot can do the job is already *NP*-complete.

3.3. *Solution approach*

We follow an approximative solution strategy. In contrast to the approach of Section 2.3, it is a *route first-cluster second* algorithm. That is, we first construct a single closed directed walk that contains all the arcs and has minimum length, and we then break it up into smaller directed walks, one for each vehicle. The routing problem is nothing but the directed Chinese postman problem; the clustering problem will be solved by dynamic programming.

We first discuss the routing phase. The closed walk that is to be determined may have to traverse some of the arcs more than once. Suppose that, if it traverses an arc k times, we add $k-1$ duplicate arcs to the graph. The closed walk is thereby transformed into an *Eulerian tour*, which traverses each arc exactly once. Recall that a directed graph is Eulerian (i.e., has an Eulerian tour) if and only if it is strongly connected and, for each node, the indegree is equal to the outdegree. The directed Chinese postman problem is therefore equivalent to finding a minimum-weight collection of duplicate arcs, the addition of which makes the indegree and outdegree of each node equal to each other.

Let V^+ be the set of nodes for which the indegree is larger than the outdegree; let d_i^+ denote the difference for each $i \in V^+$. Let V^- be the set of nodes for which the outdegree is larger than the indegree; let d_j^- denote the difference for each $j \in V^-$. Note that $\Sigma_{i \in V^+} d_i^+ = \Sigma_{j \in V^-} d_j^-$. Further, let c_{ij} be equal to the length of the shortest path from $i \in V^+$ to $j \in V^-$. The decision variables x_{ij} will indicate the number of duplicates of the shortest path from $i \in V^+$ to $j \in V^-$ that have to be added to the graph. The problem is to find x_{ij} for which

$$\Sigma_{i \in V^+} \Sigma_{j \in V^-} c_{ij} x_{ij}$$

is minimized subject to

$$\sum_{i \in V^-} x_{ij} = d_j^- \quad \text{for all } j \in V^-,$$
$$\sum_{j \in V^+} x_{ij} = d_i^+ \quad \text{for all } i \in V^+,$$
$$x_{ij} \in \mathbb{N} \cup \{0\} \quad \text{for all } i \in V^+, j \in V^-.$$

This is the *linear transportation problem*, which can be solved in polynomial time (see Chapter 2).

In the sprinkling problem, each arc has to be sprinkled only once, so that we have to compute the shortest path lengths c_{ij} using the arc weights that correspond to driving without sprinkling. We solve the linear transportation problem and add duplicate arcs to the graph in accordance with its optimal solution. The resulting graph usually contains many Eulerian tours. We select one using a heuristic rule, which incorporates various secondary criteria that are beyond the scope of this discussion.

We now turn to the clustering phase. We choose a starting point of the Eulerian tour and list the arcs in the order in which they occur, say, $a_1, a_2, \ldots, a_m$. Each arc a_h has a weight w_h. If there are several occurrences of the same arc, then the first one is assumed to be the original arc (and has the higher weight) and the others are the duplicates (with the lower weight). We will restrict our attention to walks that correspond to subsequences of $(a_1, a_2, \ldots, a_m)$.

Let W denote the given upper bound on walk length. For each arc a_h $(h = 1, \ldots, m)$, let v_h be the shortest distance from any depot to the tail of a_h, and let I_h be the set of indices i such that the subsequence $(a_h, \ldots, a_i)$ can be traversed by a single vehicle, i.e., $v_h + w_h + \cdots + w_i \leqslant W$ for all $i \in I_h$. The minimum cost z_h of an optimal partitioning of the subsequence $(a_h, \ldots, a_m)$ into feasible walks can now be computed by a simple recursion:

$$z_{m+1} = 0,$$
$$z_h = \min_{i \in I_h} \{(W - (v_h + w_h + \cdots + w_i))^2 + z_{i+1}\} \quad \text{for } h = m, m-1, \ldots, 1.$$

The optimum solution has value z_1. Since walks starting with duplicate arcs may be disregarded, we can restrict the computation to those indices h for which a_h is an original arc, and define $z_h = z_{h+1}$ if a_h is a duplicate arc.

Although the directed Chinese postman problem in the routing phase and the partitioning problem in the clustering phase are both solved to optimality, the solution obtained is only approximate. This is due to the decomposition of the solution process into two phases and to the heuristic choice of an Eulerian tour. The entire algorithm runs in polynomial time.

3.4. *Related models and applications*

The Chinese postman problem was originally formulated on *undirected* graphs, by Guan [1962]. It can be solved by shortest path and matching techniques; see, e.g., Lawler [1976]. While both the undirected and the directed case can be solved in polynomial time, the postman problem becomes *NP*-hard if it is *mixed*, *windy*, *rural*, *multiple*, or *capacitated*, and also if the postman is replaced by a *stacker crane*. Lenstra and Rinnooy Kan [1981] and Johnson and Papadimitriou (Chapter 5 in Lawler, Lenstra, Rinnooy Kan and Shmoys [1985]) review complexity results and approximation algorithms for these variants.

Shortest path algorithms are discussed by Lawler [1976]. They are used as subroutines in many other combinatorial algorithms, e.g., for the linear transportation problem and the

minimum cost flow problem (see Chapter 2).

Knuth and Plass [1981] describe an interesting application of shortest paths that arose during the development of the TEX text processing system. A paragraph of text is to be broken into lines. Nodes correspond to feasible breaking points, and arcs to feasible lines. With each arc, a weight is associated that measures the quality of the breaks at its endpoints and of the line in between. The determination of these weights is not easy, but it is a typographical rather than a mathematical question. As the resulting directed graph is acyclic, a shortest path can be found by a very simple algorithm.

4. Linear ordering

4.1. *Ranking priorities*

In 1970, a Dutch trade union was planning its policy for the future. Nine action items were listed:

(1) increasing the retirement payments as well as the pensions for widows and orphans;
(2) increasing the payment for loss of working hours due to frost;
(3) increasing the holiday allowance;
(4) increasing the pensions for widows and orphans;
(5) introduction of capital growth sharing;
(6) reduction of working hours;
(7) increasing the number of days off;
(8) education of young people at full pay;
(9) increasing the retirement payments.

Being a democratic organization, the trade union decided to involve its members. One thousand union representatives and 326 ordinary members were asked to rank the items in order of decreasing importance. But how does one aggregate 1,000 or 326 individual rankings into a single one? Anthonisse (private communication) proposed a simple and elegant model.

4.2. *Model formulation*

By ranking n items, an individual expresses $n(n-1)/2$ preferences, one for each pair of items. One way to evaluate an overall ranking is by counting the total number of individual preferences that are consistent with it.

Suppose c_{ij} is the number of people who prefer item i to j, for $i,j = 1, \ldots, n$. An ordering ρ of $\{1, \ldots, n\}$ defines a priority ranking in the sense that i is ranked higher than j if $\rho(i) < \rho(j)$. The total number of preferences that are consistent with a ranking ρ is given by

$$\sum_{i,j:\rho(i)<\rho(j)} c_{ij},$$

and the problem is to find a ranking ρ that maximizes this number. This is the *linear ordering problem*. It generalizes the *feedback arc set problem* (see Section 4.4) and is therefore *NP*-hard.

A formulation in terms of 0-1 variables is easily obtained. Let $x_{ij} = 1$ indicate that $\rho(i) < \rho(j)$. The problem is then to maximize

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

subject to

$$x_{ii} = 0 \qquad \text{for } i = 1, \ldots, n, \tag{4.1}$$

$$x_{ij} + x_{ji} = 1 \qquad \text{for } i,j = 1, \ldots, n,\ i < j, \tag{4.2}$$

$$x_{ij} + x_{jk} + x_{ki} \leqslant 2 \qquad \text{for } i,j,k = 1, \ldots, n,\ i < j < k \text{ or } i < k < j, \tag{4.3}$$

$$x_{ij} \in \{0,1\} \qquad \text{for } i,j = 1, \ldots, n. \tag{4.4}$$

The conditions (4.3) represent the transitivity of ρ: if we rank i above j and j above k ($x_{ij} = x_{jk} = 1$), then we rank i above k ($x_{ki} = 0$, so $x_{ik} = 1$).

4.3. *Solution approaches*

For the trade union's problem, Anthonisse replaced (4.4) by $x_{ij} \in \{0,1\}$ by $x_{ij} \geqslant 0$, solved the resulting linear programming problem, and obtained an integral solution. It has been observed more often that, for this problem type, the linear programming relaxation gives an optimal solution to the integer program or at least an excellent upper bound.

The best available algorithm for the linear ordering problem uses polyhedral techniques, very much in the spirit of the polyhedral approach to the symmetric traveling salesman problem (see Section 1.3). The constraints (4.3) define facets of the linear ordering polytope, but, again, more classes of facets have been identified. Reinelt [1985] describes this approach in detail. He also reviews earlier optimization and approximation algorithms for the linear ordering problem.

4.4. *Related models and applications*

Another application of the linear ordering problem is the triangulation of input-output matrices. Here, there are n industry sectors and c_{ij} denotes the supply from sector i to sector j. The sectors have to be ordered 'from raw material to consumer'.

The linear ordering problem is equivalent to the *acyclic subgraph problem*: given a directed graph $G = (V,A)$ with weights associated with the arcs, find an acyclic directed graph $G' = (V,A')$ with $A' \subset A$ such that the sum of the weights of the arcs in A' is maximum. If all arc weights are equal, the problem reduces to the *feedback arc set problem*: given a directed graph $G = (V,A)$, find a minimum-cardinality set of arcs that intersects each directed circuit in G. We leave it to the reader to sort out the details and refer to Jünger [1985] and Reinelt [1985] for further information on models, algorithms and applications.

5. Clique partitioning

5.1. *Distinguishing types of professions*

In 1969, the Interfaculty of Actuarial Sciences and Econometrics of the University of Amsterdam wished to revise the curriculum in econometrics. 'Econometrics' is used here in a broad sense and includes mathematical economics, empirical econometrics, statistics, and operations research. A committee was installed to investigate what professional econometricians in practice would demand of a curriculum.

The committee interviewed 45 econometricians employed in government, industry and consultancy. Each of them was given a list of 24 problem situations and activities and a list of 24 methods and techniques, and was asked to indicate which problems he had been working on

during the last two years and which techniques he had applied. The committee then first analyzed the lists of problems to determine which types of professionals could be distinguished. Secondly, the lists of techniques were used to design a curriculum for each type. It should be noted that no one intended to directly implement the results. There are many reasons why an actual university curriculum could differ from what present practice views as desirable.

The first problem was modeled and solved by techniques from combinatorial optimization; details will be given in Sections 5.2 and 5.3. The result was a clear distinction between two groups: the 'measurers' (proper econometricians) and the 'regulators' (operations researchers). The second problem was a statistical exercise, which need not concern us here. A full account is given by Cramer, Kool, Lenstra and De Leve [1970].

5.2. *Model formulation*

Given are a set V of persons and a set P of problem situations. For each person $i \in V$, there is a set $P_i \subset P$ of problems on which i has worked. For each problem $p \in P$, there is a set $V_p \subset V$ of persons who worked on p, with $V_p = \{i : p \in P_i\}$. We will write $V = \{1, \ldots, n\}$, $\bar{P}_i = P \setminus P_i$ $(i \in V)$, and $\bar{V}_p = V \setminus V_p$ $(p \in P)$.

We wish to partition V into a number of mutually disjoint groups in such a way that two persons i and j are allocated to the same group if P_i and P_j are similar and to different groups if P_i and P_j are dissimilar. In other words, the partition X of V we look for should be a reasonable aggregation of the given partitions $(V_p, \bar{V}_p)$ $(p \in P)$. In the case that $|P| = 1$, we could take $X = (V_1, \bar{V}_1)$, and there would be complete agreement between input and output. In the general case, we choose to minimize the sum, over all problems $p \in P$, of the number of disagreements between X and $(V_p, \bar{V}_p)$.

The data can be represented by numbers y_{ijp} $(1 \leqslant i < j \leqslant n,\ p \in P)$ such that $y_{ijp} = 1$ if i and j are in agreement regarding problem p, i.e., $p \in (P_i \cap P_j) \cup (\bar{P}_i \cap \bar{P}_j)$, and $y_{ijp} = 0$ otherwise. Similarly, X can be described by decision variables x_{ij} $(1 \leqslant i < j \leqslant n)$ such that $x_{ij} = 1$ if i and j are allocated to the same group and $x_{ij} = 0$ otherwise. Since $z^2 = z$ for any $z \in \{0,1\}$, the total number of disagreements can now be written as

$$\begin{aligned}
&\textstyle\sum_{p \in P} \sum_{1 \leqslant i < j \leqslant n} (x_{ij} - y_{ijp})^2 \\
&= \textstyle\sum_p \sum_{i<j} (1 - 2y_{ijp}) x_{ij} + \sum_p \sum_{i<j} y_{ijp} \\
&= \textstyle\sum_{i<j} c_{ij} x_{ij} + C,
\end{aligned}$$

where

$$\begin{aligned}
c_{ij} &= \textstyle\sum_p (1 - 2y_{ijp}) = |P| - 2(|P_i \cap P_j| + |\bar{P}_i \cap \bar{P}_j|), \\
C &= \textstyle\sum_p \sum_{i<j} y_{ijp} = \sum_{i<j} (|P_i \cap P_j| + |\bar{P}_i \cap \bar{P}_j|).
\end{aligned}$$

The constant term C can be dropped. The problem is then to minimize

$$\textstyle\sum_{1 \leqslant i < j \leqslant n} c_{ij} x_{ij}$$

subject to the condition that the x_{ij} define a partition of V:

$$
\begin{aligned}
&x_{ij}+x_{jk}-x_{ik} \leqslant 1 \quad && \text{for } i,j,k = 1,\ldots,n,\ i<j<k,\\
&x_{ij}-x_{jk}+x_{ik} \leqslant 1 \quad && \text{for } i,j,k = 1,\ldots,n,\ i<j<k,\\
&-x_{ij}+x_{jk}+x_{ik} \leqslant 1 \quad && \text{for } i,j,k = 1,\ldots,n,\ i<j<k,\\
&x_{ij} \in \{0,1\} && \text{for } i,j = 1,\ldots,n,\ i<j.
\end{aligned}
$$

Note that the problem has a trivial solution if all c_{ij} have the same sign.

The problem can also be formulated in terms of graphs. Consider the complete graph $K_n = (V,E)$ on n nodes with a weight $c_e \in \mathbb{Z}$ for each edge $e \in E$. The problem is to partition V into nonoverlapping subsets so as to minimize the sum of the weights of the edges whose ends are in the same subset. This is called the *clique partitioning problem*, and it is known to be *NP*-hard.

The above discussion follows Grötschel and Wakabayashi [1989]. Note that the number of groups in the partition is not specified in advance but computed as part of the solution.

Let us briefly consider the case in which the number of groups is not free but fixed to, say, m. This *clique m-partitioning problem* can be stated as follows: given the complete graph $K_n = (V,E)$ on n nodes with a weight $c_e \in \mathbb{Z}$ for each edge $e \in E$, color each node with one of m colors so as to minimize the sum of the weights of the edges whose ends receive the same color. This formulation generalizes the *graph coloring problem*, where all c_e are equal to 0 or 1 and we are interested in the existence of an m-coloring with value 0. The graph coloring problem is solvable in polynomial time for $m = 2$ and *NP*-complete for any $m \geqslant 3$. The clique 2-partitioning problem is also known as the *max cut problem*, which is already *NP*-hard in itself.

Carlson and Nemhauser [1966] were the first to consider the clique m-partitioning problem. They gave a *quadratic programming* formulation. Let $x_{hi} = 1$ indicate that node i receives color h. The problem is then to minimize

$$\tfrac{1}{2} \sum_{h=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{hi} x_{hj}$$

subject to

$$
\begin{aligned}
&\sum_{h=1}^{m} x_{hi} = 1 \quad && \text{for } i = 1,\ldots,n,\\
&x_{hi} \in \{0,1\} && \text{for } h = 1,\ldots,m,\ i = 1,\ldots,n.
\end{aligned}
$$

It is not hard to see that, if $x_{hi} \in \{0,1\}$ is replaced by $x_{hi} \geqslant 0$, then there exists an integral optimal solution. It can also be proved that an integral feasible solution satisfies the Karush-Kuhn-Tucker conditions if and only if it cannot be improved by giving any single node another color.

5.3. *Solution approaches*

Grötschel and Wakabayashi [1989] developed a cutting plane algorithm for the clique partitioning problem, along the same lines as the polyhedral approaches mentioned in Sections 1.3 and 4.3. They were able to solve problem instances with up to 158 nodes, without ever having to resort to tree search.

In 1969, when the problem at the University of Amsterdam occurred, a more heuristic approach was taken. First, for various values of m, the number of groups was fixed at m and a

good partitioning into m groups was computed. Secondly, the results were analyzed and an appropriate value of m was determined. The weights were defined by

$$c_{ij} = \sum_{p \in P} \frac{|P| - |P_i \cap P_j| - .9|\bar{P}_i \cap \bar{P}_j|}{|P_i \cap P_j| + 1}.$$

Note that a positive agreement between i and j counts more heavily than a negative agreement. This choice was made after some experiments with small problem instances. The results obtained were, however, quite robust with respect to small perturbations of the weights.

None of the optimization algorithms for the clique m-partitioning problem that were available in 1969 could handle instances with $n = 45$. An iterative improvement procedure was developed, with k ($1 \leqslant k \leqslant n$) as an input parameter. At each step, the existing partitioning is considered and the k persons are determined whose individual transition to another group would result in the largest decrease of the criterion value. These k persons are then optimally reallocated by a branch and bound algorithm. If no further improvements are possible, a local optimum has been obtained, which could be called *k-opt*. As mentioned before, a solution is 1-opt if and only if it satisfies the Karush-Kuhn-Tucker conditions of the Carlson-Nemhauser formulation. For $m = 2$, this procedure always produced the same local optimum, for any starting solution and for any value of k between 1 and 20. For $m = 3$, several local optima were obtained, but their criterion values are close together. The same is true for $m = 4$.

The question is now how to determine the best value of the number m of groups. It is obvious that, if more groups are added, the optimal criterion value will decrease. Cramer, Kool, Lenstra and De Leve [1970] used a simulation experiment to estimate the decreases that can be solely ascribed to enlarging the number of groups. They generated a number of random surveys among 45 'colorless' people and computed locally optimal solutions for these into two, three and four groups. If one goes from one to two groups, then an average of 42 percent of the total weight remains, with a very small variance, as compared to only 30 percent in case of the actual survey data. If more groups are added, the decreases for the simulation and for the survey are about the same. It was concluded that there is a clear distinction between two, but no more than two, groups.

5.4. *Related models and applications*

The relation of the *clique m-partitioning problem* to the graph coloring problem and the max cut problem has already been pointed out.

A variant of the model occurs if upper bounds on the group sizes are specified. Arnold, Beckwith and Jones [1973] describe an application that arises when one is organizing a scientific meeting: nodes correspond to sessions, weights to conflicts between sessions, and colors to time slots. They used a combination of random sampling and local search.

Kernighan and Lin [1970] consider the problem of partitioning V into two equal-size subsets so as to minimize the total weight of the edges whose ends are in *different* subsets. They propose a variable-depth search method, which is a precursor of their algorithm for the traveling salesman problem (see Section 1.3).

The *clique partitioning problem* serves as a model for a variety of clustering problems. Grötschel and Wakabayashi [1989] give several examples, such as the classification of animals with

respect to morphological and behavioral characteristics, and the classification of member countries of the United Nations on the basis of voting behavior. All these problem situations can be captured under the general heading of the *aggregation of binary relations*, which is a central topic of interest in the area of qualitative data analysis.

6. Test cover

6.1. *Recognizing diseases*

In 1979, the CWI received a request from the Department of Mathematics at the Agricultural University in Wageningen: 'Enclosed you will find a 0-1 matrix B with 63 rows and 28 columns. We define a 0-1 matrix A with 63 rows and 378 columns. Each column of A corresponds to a pair of columns of B (note that $378 = 28 \cdot 27/2$) and is obtained by adding those columns modulo 2. We would be interested in a solution to the set covering problem on A^T.'

The *set covering problem* is here the problem of finding a minimum number of rows of A such that, in each column, at least one of these rows has a 1. The difficulty of solving large set covering problems as well as our professional curiosity motivated us to transform backwards. We identified the rows of B with tests, the columns with items, and each entry (i,j) with the result of test i applied to item j. The problem is then to find a minimum number of tests such that, for each pair of items, at least one of these tests distinguishes the two items. This is the *test cover problem.*

Before discussing these models and their relation in more detail, let us clarify the practical background. It turned out that tests and items were plant varieties and plant diseases, respectively. A minimum number of varieties that discriminates between all diseases should provide an efficient and economical setup for recognizing diseases. The problem arose as part of a project carried out at the Research Institute of Plant Protection in Wageningen in cooperation with the International Maize and Wheat Improvement Center (CIMMYT). We provided a program that incorporates the algorithm described in Section 6.3. It computed an optimal solution of seven tests for the above instance (the best known solution used eight) and has been used successfully on many other instances.

6.2. *Model formulation*

Given is a finite set N and a family $\mathcal{T}$ of subsets of N; N contains the items and $\mathcal{T}$ is the collection of tests. A *test cover* is a subfamily $\mathcal{T}' \subset \mathcal{T}$ such that, for each pair $\{j,k\} \subset N$, there is a test $T \in \mathcal{T}'$ such that T distinguishes between j and k, i.e., $|T \cap \{j,k\}| = 1$. The problem is to find a test cover of minimum cardinality.

A more familiar problem is the set covering problem: given a finite set M and a family $\mathcal{S}$ of subsets of M, determine a minimum-size subfamily $\mathcal{S}' \subset \mathcal{S}$ for which $\cup_{S \in \mathcal{S}'} S = M$.

As suggested above, we can formulate each instance of the test cover problem as a set covering problem. We define an element in M for each pair of items in N, and we create a subset $S \in \mathcal{S}$ for each test $T \in \mathcal{T}$; the elements in S are precisely the pairs of items in N that are distinguished by T. It is immediate that a subfamily $\mathcal{S}' \subset \mathcal{S}$ covers M if and only if the corresponding subfamily $\mathcal{T}' \subset \mathcal{T}$ is a test cover.

Through a relation with an optimization problem on graphs, which we will briefly discuss in

Section 6.4, it turns out that the test cover problem is already *NP*-hard if $|T| \leqslant 2$ for all $T \in \mathcal{T}$.

6.3. *Solution approaches*

Any algorithm for set covering can be used for the test cover problem. However, the quadratic blowup is not encouraging, and it even appears that the resulting set covering instances are particularly hard ones. Although all *NP*-hard combinatorial optimization problems are polynomially equivalent, it generally pays off to develop an algorithm that is specific to the problem at hand.

The problem instance described in the introduction was solved to optimality by a straightforward combination of approximation and branch and bound. In the first phase, a greedy algorithm constructs a reasonable solution and a local search algorithm tries to improve on it. The greedy algorithm selects, at each step, the test that distinguishes the greatest number of pairs that are not yet distinguished by the tests selected so far. Iterative improvement then produces a good solution, the value of which serves as an initial upper bound for the branch and bound process. In this second phase, the tests $T \in \mathcal{T}$ are put in nonincreasing order of 'distinguishing power' $\min\{|T|, |N-T|\}$ and the subfamilies of $\mathcal{T}$ are enumerated in lexicographic order. Subfamilies are eliminated by a lower bound which is based on the simple observation that, for distinguishing n items, at least $\lceil \log_2 n \rceil$ tests are needed.

One of the first papers on the test cover problem is by Moret and Shapiro [1985]. They investigate the worst-case behavior of approximation algorithms and the derivation of lower bounds. A negative result is that the greedy algorithm can perform as badly as for the general set covering problem and produce solutions that are off by a factor of $\Theta(\log|N|)$. The above project has inspired further work on approximation and optimization algorithms for the test cover problem.

6.4. *Related models and applications*

An interesting special case of the test cover problem occurs if all tests have cardinality 2. In terms of a graph G with node set N and edge set $\mathcal{T}$, a test cover is an edge subset $\mathcal{T}' \subset \mathcal{T}$ such that no two nodes in N have the same incidence relations with respect to $\mathcal{T}'$. This turns out to be equivalent to the requirement that the subgraph $G' = (N, \mathcal{T}')$ has no isolated edges and at most one isolated node. One can establish a strong relation between this problem and the problem of finding a maximum number of node-disjoint paths of length 2 in a graph. This implies *NP*-hardness of the restricted test cover problem and has interesting consequences for the worst-case behavior of approximation algorithms.

Moret and Shapiro [1985] mention applications of the test cover problem in fault testing and diagnosis, pattern recognition, and biological identification.

7. Bottleneck extrema

7.1. *Locating obnoxious facilities*

In a regional development plan, a number of sites has been identified at which residential quarters as well as industrial areas will be located. The industrial areas will accommodate obnoxious facilities. The problem is how to select sites for the industrial areas so as to minimize the

inconvenience caused to the residential quarters.

Our presentation follows Hsu and Nemhauser [1979]. We did not encounter the problem in practice, but we decided to include it here because it nicely illustrates the elegant theory of bottleneck extrema.

7.2. *Model formulation*

Let us assume that there are n sites, k of which will be industrial areas. We construct a graph $G = (V,E)$ with $V = \{1, \ldots, n\}$ and $\{i,j\} \in E$ if and only if sites i and j influence each other, i.e., if an industrial area at one of the sites would be a nuisance to a residential quarter at the other site. With each edge $e = \{i,j\} \in E$, a weight d_e is associated, representing the distance between sites i and j.

Recall that, for all $U \subset V$ with $U \neq \emptyset$ and $U \neq V$, $\delta(U)$ is the set of edges with exactly one end in U. The set $\delta(U)$ is called a *cut*; its removal from G disconnects the node sets U and $V \setminus U$. If $|U| = k$, then $\delta(U)$ will be called a *k-cut*.

One way of minimizing annoyance is to maximize the minimum distance between any residential quarter and its closest industrial area. The problem is then to find a k-cut whose minimum edge weight is maximum:

$$\max_{U:|U|=k} \min_{e \in \delta(U)} d_e. \tag{7.1}$$

We present a polynomial-time algorithm for this problem in the next section.

7.3. *Solution approach*

The solution of (7.1) is based on the theory of bottleneck extrema developed by Edmonds and Fulkerson [1970]. We outline some of their results below.

Let S be a finite set. A *clutter* $\mathcal{C}$ on S is a collection of subsets of S such that no set in $\mathcal{C}$ includes any other set in $\mathcal{C}$. The *blocker* $\mathcal{B}$ of $\mathcal{C}$ is the collection of subsets of S that intersect all sets in $\mathcal{C}$ and are minimal under inclusion. Note that $\mathcal{B}$ is again a clutter.

Edmonds and Fulkerson proved two duality results. First, $\mathcal{C}$ is the blocker of its blocker $\mathcal{B}$. Secondly, for any real-valued function f on S,

$$\max_{C \in \mathcal{C}} \min_{e \in C} f_e = \min_{B \in \mathcal{B}} \max_{e \in B} f_e. \tag{7.2}$$

They also presented a simple threshold algorithm for the max-min problem and an analogous dual threshold method for the min-max problem. The choice between these two algorithms depends on the relative efficiency of recognizing members of $\mathcal{C}$ and $\mathcal{B}$. We will describe the dual method.

Suppose that the elements of S are indexed so that $f_1 \leqslant f_2 \leqslant \cdots \leqslant f_{|S|}$. Suppose further that $e^* \in S$ is such that $\{1, \ldots, e^* - 1\}$ does not include a member of $\mathcal{B}$ but that $\{1, \ldots, e^*\} \supset B^*$ for some $B^* \in \mathcal{B}$. Then B^* solves the min-max problem. By bisection search over S, e^* can be found in $O(\log|S|)$ iterations, where each iteration tests for inclusion of a member of $\mathcal{B}$.

We now return to problem (7.1). It suffices to consider only minimal k-cuts: if $E' \subset E''$, then $\min_{e \in E'} d_e \geqslant \min_{e \in E''} d_e$. We let S correspond to the edge set E, f to the weight function d, and $\mathcal{C}$ to the collection of minimal k-cuts; note that $\mathcal{C}$ is a clutter. Problem (7.1) can now be stated as $\max_{C \in \mathcal{C}} \min_{e \in C} f_e$.

Rather than solving the recognition problem for members of $\mathcal{C}$, we will characterize its blocker $\mathfrak{B}$ and give an $O(n^2)$ membership test for $\mathfrak{B}$. We thus obtain an $O(n^2 \log n)$ algorithm for computing $\min_{B \in \mathfrak{B}} \max_{e \in B} f_e$ and, by (7.2), for solving (7.1).

The crucial observation to make is that a subset $B \subset E$ intersects all k-cuts if and only if its complement $E \setminus B$ is not a k-cut. It immediately follows that $\mathfrak{B}$ contains the minimal $B \subset E$ such that no collection of connected components of the graph $G_B = (V,B)$ contains exactly k nodes.

As to the membership test, consider some $E' \subset E$. Suppose $G_{E'} = (V,E')$ has m components and let component h have a_h nodes, for $h = 1, \ldots, m$. Then $E' \supset B$ for some $B \in \mathfrak{B}$ if and only if

$$\sum_{h=1}^{m} a_h x_h = k,$$
$$x_h \in \{0,1\} \text{ for } h = 1, \ldots, m$$

has no solution. This can be tested by dynamic programming in $O(km) = O(n^2)$ time.

7.4. *Related models*

Discrete location theory is one of the cornerstones of combinatorial operations research. Its basic problem types are the following. In the *uncapacitated plant location problem*, one has to find a set of locations such that the sum of the setup costs for facilities and the transportation costs between customers and facilities is minimized. In the *k-median problem*, one has to locate k facilities so as to minimize the sum of the distances between each customer and its closest facility, while in the *k-center problem*, the maximum of these distances is to be minimized.

Mirchandani and Francis [1990] collected a number of survey articles on discrete location theory.

8. Minimum cost flow

8.1. *Two-dimensional proportional representation*

Gooi en Vechtstreek is a region in the Netherlands, just east of Amsterdam. There is a regional council, the members of which are appointed by and from the participating local councils. The composition of the regional council should be a fair representation of the local interests as well as of the political views in the region.

Each local council has an odd number of members. It is prescribed that a quarter of them, rounded to the nearest integer, is appointed to the regional council. This should take care of a proportional representation of local interests. However, if the allocation of seats to political parties is left completely to the local councils, then there is an obvious danger that overall disproportionalities in the representation of political views will occur. Coordination is required.

It has been agreed that the chairman of the regional cooperation uses the outcome of the local elections to determine the number of seats to be allocated to each party from each local council. The result of his reflections has the status of an advice to the local councils. In current practice, he applies the method proposed by Anthonisse [1984] in arriving at his advice. This method is described below.

8.2. *Model formulation*

We first briefly consider the question of one-dimensional proportional representation. This problem has attracted a lot of interest, in politics as well as in mathematics. It can be formulated as follows.

Suppose there are P parties, V votes, and S seats. Party p has received v_p votes, with $\sum_{p=1}^{P} v_p = V$. Ideally, party p should receive $s_p = Sv_p/V$ seats. However, the s_p are generally non-integral and have to be rounded. Let a bivariate function f be given, where $f(s_p,x_p)$ measures the distance between the ideal number of seats s_p and the actual allotment x_p. The problem is then to find $x_1,\ldots,x_P$ such that

$$\sum_{p=1}^{P} f(s_p,x_p)$$

is minimized subject to

$$\begin{aligned} &\sum_{p=1}^{P} x_p = S, \\ &x_p \in \mathbb{N} \cup \{0\} \qquad \text{for } p = 1,\ldots,P. \end{aligned} \tag{8.1}$$

Various methods for solving this problem have been proposed. Hamilton's method of the greatest remainders corresponds to $f(s_p,x_p) = |x_p - s_p|$, Jefferson's method of the greatest divisors to $f(s_p,x_p) = (x_p - (s_p - \tfrac{1}{2}))^2/s_p$, and Webster's method to $f(s_p,x_p) = (x_p - s_p)^2/s_p$. For a historical and mathematical overview of one-dimensional proportional representation, we refer to Balinski and Young [1982]. They list a number of properties a perfect method of apportionment should satisfy and show that no such method exists. They argue convincingly that Webster's method comes closest towards 'meeting the ideal of one man, one vote'.

As to the problem of two-dimensional proportional representation, suppose there are P parties and M municipalities. In the local council of municipality m, party p has v_{mp} seats. The size of council m is $V_{m\cdot} = \sum_{p=1}^{P} v_{mp}$, the regional strength of party p is $V_{\cdot p} = \sum_{m=1}^{M} v_{mp}$, and $V = \sum_{m=1}^{M}\sum_{p=1}^{P} v_{mp}$. Let $w_m = \lfloor (V_{m\cdot}+1)/4 \rfloor$ denote the number of members of council m that are to be appointed in the regional council, and let $S = \sum_{m=1}^{M} w_m$ be the total number of seats in the regional council. Ideally, party p should receive $s_p = SV_{\cdot p}/V$ regional seats, $t_{mp} = Sv_{mp}/V$ of which should come from municipality m. The decision variables are x_p, the actual allotment to party p, and y_{mp}, the number of members of party p to be appointed from the council of municipality m. In the formulations below, f can be any convex bivariate distance function.

The problem is solved in two stages. First, we find the x_p such that

$$\sum_{p=1}^{P} f(s_p,x_p)$$

is minimized subject to

$$\sum_{p=1}^{P} y_{mp} = w_m \quad \text{for } m = 1, \ldots, M, \tag{8.2}$$

$$\sum_{m=1}^{M} y_{mp} = x_p \quad \text{for } p = 1, \ldots, P, \tag{8.3}$$

$$y_{mp} \leqslant v_{mp} \quad \text{for } m = 1, \ldots, M,\ p = 1, \ldots, P, \tag{8.4}$$

$$y_{mp} \in \mathbb{N} \cup \{0\} \quad \text{for } m = 1, \ldots, M,\ p = 1, \ldots, P, \tag{8.5}$$

$$x_p \in \mathbb{N} \cup \{0\} \quad \text{for } p = 1, \ldots, P.$$

Secondly, given the x_p, we find the y_{mp} such that

$$\sum_{m=1}^{M} \sum_{p=1}^{P} f(t_{mp}, y_{mp})$$

is minimized subject to (8.2-5).

The first problem obviously has a feasible solution: any sample of w_m from the V_m. members satisfies (8.2,4,5), and (8.3) then defines the x_p. Given these x_p, the second problem is also feasible, as the constraints remain the same. The reader may wonder why we have not simplified the first stage by relaxing (8.2-5) into (8.1). The reason is that this may yield an infeasible second stage.

Both problems can be modeled in terms of minimum cost flows (see Chapter 2) and as such be solved in polynomial time. At the first stage, we define a network with a source α, a node m for each municipality, a node p for each party, and a sink ω. There are arcs (α,m) with given flow values w_m, arcs (m,p) with capacities v_{mp}, and unconstrained arcs (p, ω); $f(s_p,x_p)$ denotes the cost of sending x_p units of flow through the arc (p, ω). A feasible solution $((x_p),(y_{mp}))$ now corresponds to a flow of value S from α to ω, with flow values y_{mp} through the arcs (m,p) and x_p through (p, ω). The conditions (8.2) and (8.3) are flow conservation constraints at the nodes m and p; the conditions (8.4) represent the capacity constraints of the arcs (m,p). We have to find a feasible flow that minimizes the total costs of the flows x_p through the arcs (p, ω). At the second stage, we use the same network, except that the arcs (m,p) have costs $f(t_{mp},y_{mp})$ and the arcs (p, ω) have given flow values x_p. We now have to find feasible flows y_{mp} through the arcs (m,p) of minimum total costs.

8.3. *Solution approaches*

Minimum cost flow problems are treated in depth in Chapter 2. Minoux (1986) gives a polynomial-time algorithm for finding minimum cost integral flows with separable convex cost functions.

8.4. *Related models*

Following Anthonisse's work, Balinski and Demange [1989] pursued an axiomatic approach to two problems, one of which generalizes the problem considered above. Given are a nonnegative matrix $V = (v_{mp})$ and a positive integer S; one may give v_{mp} and S the same interpretation as before. In addition, nonnegative integers w_m^-, w_m^+, x_p^-, x_p^+ are given. An *allocation* is defined as a matrix $Y = (y_{mp})$ of the same dimensions as V with $w_m^- \leqslant \Sigma_p y_{mp} \leqslant w_m^+$, $x_p^- \leqslant \Sigma_m y_{mp} \leqslant x_p^+$, and $\Sigma_m \Sigma_p y_{mp} = S$. What should it mean to say that an allocation Y is proportional to V? And what should this mean for an *apportionment*, i.e., an *integer* allocation?

9. Interval scheduling

9.1. *Dealing with nasty clients*

A Dutch firm, primarily engaged in the retail trade, had decided to diversify and had acquired a large number of summer cottages. A client can make a reservation at any one of the firm's branches and is immediately told whether a cottage is still available for the period (s)he is applying for. Only at a later stage is it determined in which cottage each accepted client will spend the holidays. This procedure gave rise to a number of questions.

Does there exist a simple rule that indicates whether a client can be accepted? Yes, there does, as we will clarify below: cottages can be assigned to clients in their desired periods if and only if, at any time, the number of clients is no greater than the number of cottages. How about a method that assigns the accepted clients to a minimum number of cottages? This exists as well: assign the clients to cottages in order of their starting times, giving priority to cottages used before.

Both questions could be answered during the first contact with the firm's employee who sought the advice of the CWI. All seemed well until, while leaving, a trivial complication crossed his mind: a client can reserve a specific cottage by paying Hfl. 25 upon application and is then preassigned. This appears to have a dramatic effect on the problem's computational complexity. The above necessary and sufficient condition for acceptance remains valid only under the assumption that the clients would be willing to move into another cottage now and then. But under the more realistic assumption that these people do not want to move when they are on holiday, the problem turns out to be *NP*-complete.

We never heard from our client again. The complications caused by the nasty clients are probably trivial indeed and do not prohibit the application of the existing methods.

The above account follows Anthonisse and Lenstra [1984]. We will deal with the technical details below.

9.2. *Model formulation*

We rephrase the problem in scheduling terminology. Cottages will be represented by machines and clients by jobs. There are m identical parallel machines, each of which can handle at most one job at a time. There are n independent jobs j, which need processing on one of the machines during the time interval (s_j,t_j) $(j = 1,\ldots,n)$. First of all, we are interested in the minimum number of machines needed to process all jobs. The solution of this problem goes back to Dantzig and Fulkerson [1954].

Let us define a *partial order* $\rightarrow$ on the set of jobs. We say that $j \rightarrow k$ whenever $t_j \leqslant s_k$, i.e., when job j is completed before job k starts. A *chain* in the job set is a subset $\{j_1,j_2,\ldots,j_k\}$ with $j_1 \rightarrow j_2 \rightarrow \cdots \rightarrow j_k$. The jobs in a chain can be consecutively scheduled on one machine, and, conversely, any schedule on one machine corresponds to a chain in the job set. The minimum number of machines needed to process all jobs is therefore equal to the minimum number of chains into which the job set can be partitioned.

Jobs j and k are *unrelated* if neither $j \rightarrow k$ nor $k \rightarrow j$. An *antichain* is a set of pairwise unrelated jobs. Any two jobs in an antichain overlap in time; by a property of intervals, this is equivalent to saying that all jobs in an antichain overlap at a certain time.

We now invoke Dilworth's chain decomposition theorem: for every partially ordered set, the

minimum number of chains needed to cover all elements is equal to the maximum number of elements in an antichain. Or: the minimum number of machines needed to process all jobs is equal to the maximum number of jobs that require simultaneous processing. For fast algorithms that actually assign the jobs to a minimum number of machines, we refer to Section 9.3.

Another way of modeling the interval scheduling problem is in terms of *interval graphs*. Associated with the job set, we define the interval graph $G = (\{1, \ldots, n\}, E)$ where $\{j,k\} \in E$ if and only if jobs j and k overlap in time. We recall that a *clique* in a graph is a subset of pairwise adjacent nodes, a *stable set* is a subset of pairwise nonadjacent nodes, and the *chromatic number* is the smallest k for which the node set can be partitioned into k stable sets. Clearly, a clique of G corresponds to an antichain in the partially ordered set, an stable set of G corresponds to a chain, and the chromatic number of G is equal to the minimum number of machines that can accommodate all jobs.

For interval graphs, it is true in general that the chromatic number is equal to the maximum clique size. This result parallels Dilworth's decomposition theorem for partially ordered sets. A minimum coloring and a maximum clique in an interval graph can be found in polynomial time. We refer to Chapter 4 for details.

We now turn to the situation in which some clients have been preassigned to specific cottages during certain periods. Machines will now correspond to maximal idle periods of the cottages and jobs to unassigned clients. Machine i is available during the interval (a_i, b_i) $(i = 1, \ldots, m)$; job j requires processing during the interval (s_j, t_j) $(j = 1, \ldots, n)$. Note that, in contrast to the previous problem, the machines are not identical anymore. The question whether a client can be accepted boils down to the following problem: is it possible to pack the intervals (s_j, t_j) into the intervals (a_i, b_i)?

We may try to generalize the partial order model to this situation. We introduce dummy jobs $n+i$ requiring processing in $(-\infty, a_i)$ and $n+m+i$ requiring processing in (b_i, ∞), for $i = 1, \ldots, m$. Again, we write $j \rightarrow k$ if and only if job j is completed before job k starts. A feasible schedule corresponds to a decomposition of the job set into m chains, where the ith chain starts with job $n+i$ and ends with job $n+m+i$ $(i = 1, \ldots, m)$. Conversely, because $\{n+1, \ldots, n+m\}$ and $\{n+m+1, \ldots, n+2m\}$ are antichains, any chain in a decomposition of the job set into m chains must start at some $n+i$ $(1 \leqslant i \leqslant m)$ and end at some $n+m+i'$ $(1 \leqslant i' \leqslant m)$. Unfortunately, there is nothing to guarantee that $i = i'$, and therefore a chain does not necessarily correspond to a schedule on one machine. It is not hard to see, however, that from such a chain decomposition a *preemptive* schedule can be constructed, in which the processing of a job may be interrupted on one machine and continued on another machine.

The nonpreemptive problem appears to be much harder. Kolen, Lenstra and Papadimitriou [1991] give a polynomial-time algorithm for the case of fixed m. They also prove that the general case is *NP*-complete, by relating the problem to a generalization of interval graph coloring.

9.3. *Solution approaches*

We restrict ourselves here to algorithms for the interval scheduling problem on identical machines.

Ford and Fulkerson [1962] give a simple $O(n^2)$ algorithm for decomposing a partially ordered set into chains. This so-called *staircase rule* finds a minimum number of chains in the

case that the elements can be numbered so that $j \leqslant k$ implies that all predecessors of j are included in those of k. This condition holds for the partial order defined on the job set. The rule works as follows. Find the smallest element, in terms of the numbering. Repeatedly, find the smallest successor of the last element found, until no successor exists. Delete the chain that has been found, and repeat the process.

Gupta, Lee and Leung [1979] give an $O(n\log n)$ algorithm, which builds the chains in parallel rather than in series. This rule, which was informally stated in Section 9.1, is as follows. Put all of the machines on a stack S of idle machines. Order the s_j and t_j ($j = 1, \ldots, n$) in nondecreasing order, where a t_j precedes an s_k in case of a tie; this yields a nondecreasing sequence $u_1, u_2, \ldots, u_{2n}$. Then, for $k = 1, \ldots, 2n$, do the following: if u_k corresponds to s_j, then assign job j to the machine on top of S, and remove this machine from S; if u_k corresponds to t_j, then put the machine to which job j was assigned on top of S.

9.4. *Related models*

Several generalizations of the interval scheduling problem on identical parallel machines have been investigated.

Arkin and Silverberg [1987] analyze the case in which there is a weight associated with each job and a maximum-weight subset of jobs that can be scheduled on m machines is to be found. They develop an $O(n^2\log n)$ algorithm.

Fischetti, Martello and Toth [1987, 1989] consider two problem types. In the first one, each machine is available for a period of length b, which starts at the starting time of the first job assigned to it. In the second problem type, each machine can perform no more than b time units of processing. In both cases, the number of machines is to be minimized. They show that both problems are *NP*-hard and developed branch and bound algorithms for their solution.

Another extension involves hierarchies of machines and jobs. There are m classes of machines and m classes of jobs. All machines are available during the same time interval. A job in class i requires processing during a given interval and can only be assigned to machines in classes $1, \ldots, i$. Does there exist a feasible schedule? Kolen, Lenstra and Papadimitriou [1991] give a polynomial-time algorithm for the case that $m = 2$, using network flow techniques, and show that the case $m = 3$ is *NP*-complete. Subsequent work concerns optimization versions of this problem, where costs are associated with the machines.

10. Job shop scheduling

10.1. *Production planning*

Combinatorial optimization problems that arise in production planning tend to be both difficult to formulate and difficult to solve. That is, the problem is often characterized by constraints that are very specific to the situation at hand, and it is usually an easy matter to find many independent reasons for its *NP*-hardness. These observations may explain why the development and application of general software in the area of production planning is not nearly at the stage at which it is in vehicle routing.

In order to avoid complicating details, we have chosen to consider a standard problem type, the job shop scheduling problem, which is at the core of many practical production planning situations. It is described as follows. Given are a set of jobs and a set of machines. Each

machine can handle at most one job at a time. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. The purpose is to find a schedule, i.e., an allocation of the operations to time intervals on the machines, that has minimum length.

This problem allows a number of relatively straightforward mathematical formulations. In addition, it is extremely difficult to solve to optimality. This is witnessed by the fact that a problem instance with only ten jobs, ten machines and one hundred operations, published in 1963, remained unresolved until 1986.

10.2. *Model formulation*

Given are a set $\mathcal{J}$ of jobs, a set $\mathcal{M}$ of machines, and a set $\mathcal{O}$ of operations. For each operation $i \in \mathcal{O}$, there is a job $J_i \in \mathcal{J}$ to which it belongs, a machine $M_i \in \mathcal{M}$ on which it requires processing, and a processing time $p_i \in \mathbb{N}$. There is a binary relation $\rightarrow$ on $\mathcal{O}$ that decomposes $\mathcal{O}$ into chains corresponding to the jobs; more specifically, if $i \rightarrow j$, then $J_i = J_j$ and there is no $k \notin \{i,j\}$ with $i \rightarrow k$ or $k \rightarrow j$. The problem is to find a starting time S_i for each operation $i \in \mathcal{O}$ such that

$$\max_{i \in \mathcal{O}} S_i + p_i \tag{10.1}$$

is minimized subject to

$$S_i \geqslant 0 \qquad \text{for } i \in \mathcal{O}, \tag{10.2}$$

$$S_j - S_i \geqslant p_i \qquad \text{whenever } i \rightarrow j,\ i,j \in \mathcal{O}, \tag{10.3}$$

$$S_j - S_i \geqslant p_i \vee S_i - S_j \geqslant p_j \qquad \text{whenever } M_i = M_j,\ i,j \in \mathcal{O}. \tag{10.4}$$

The objective function (10.1) represents the schedule length, in view of (10.2). The conditions (10.3) are the job precedence constraints. The conditions (10.4) represent the machine capacity constraints, which make the problem *NP*-hard.

To obtain an *integer programming* formulation, we choose an upper bound T on the optimum and introduce a 0-1 variable y_{ij} for each ordered pair (i,j) with $M_i = M_j$, where $y_{ij} = 0$ ($y_{ij} = 1$) corresponds to $S_j - S_i \geqslant p_i$ ($S_i - S_j \geqslant p_j$). We now replace (10.4) by

$$\left.\begin{aligned} & y_{ij} \in \{0,1\} \\ & y_{ij} + y_{ji} = 1 \\ & S_i + p_i - S_j - T y_{ij} \leqslant 0 \end{aligned}\right\} \qquad \text{whenever } M_i = M_j,\ i,j \in \mathcal{O}. \tag{10.4'}$$

This formulation is closely related to the *disjunctive graph*. The disjunctive graph $G = (\mathcal{O}, A, E)$ has a node set $\mathcal{O}$, an arc set $A = \{(i,j) \mid i \rightarrow j\}$, and an edge set $E = \{\{i,j\} \mid M_i = M_j\}$; note that the arcs are directed and the edges are undirected. A weight p_i is associated with each node i. There is an obvious one-to-one correspondence between feasible values of the y_{ij} in (10.1, 10.2, 10.3, 10.4') and orientations of the edges in E for which the resulting digraph is acyclic. Given any such orientation, we can determine feasible starting times by setting each S_i equal to the weight of a maximum-weight path in the digraph finishing at i minus p_i; the objective value is equal to the maximum path weight in the digraph. The problem is now to find an orientation of the edges in E that minimizes the maximum path weight.

10.3. *Solution approaches*

Optimization algorithms for job shop scheduling proceed by branch and bound. A node in the search tree is usually characterized by an orientation of each edge in a certain subset $E' \subset E$. The question is then how to compute a lower bound on the value of all completions of this partial solution.

A trivial lower bound is obtained by simply disregarding $E \setminus E'$ and computing the maximum path weight in the digraph $(\mathcal{O}, A \cup E')$. A more sophisticated bound is based on the relaxation of the capacity constraints of all machines except one: a machine $M' \in \mathfrak{M}$ is selected, and the job shop problem is solved on the disjunctive graph $(\mathcal{O}, A \cup E', \{\{i,j\} \mid M_i = M_j = M'\})$. This reduces to a *single-machine* problem, where the arcs in $A \cup E'$ define release and delivery times for the operations that are to be scheduled on M'. Although it is an *NP*-hard problem, there exist fairly efficient algorithms for its solution. The single-machine bound generalizes all previously proposed bounds [Lageweg, Lenstra and Rinnooy Kan, 1977]. More recent (and more complicated) bounds use *surrogate duality relaxation* and *polyhedral techniques*.

A variety of branching schemes to generate the search tree and elimination rules to truncate it is available. For this and for more information on the lower bounds, we refer to Lawler, Lenstra, Rinnooy Kan and Shmoys [1991].

Most *approximation algorithms* for job shop scheduling use a dispatch rule, which schedules the operations according to some priority function. Adams, Balas and Zawack [1988] developed a *sliding bottleneck heuristic*, which employs an ingenious combination of schedule construction and iterative improvement, guided by solutions to single-machine problems of the type described above. They also embedded this method in a second heuristic that proceeds by partial enumeration of the solution space.

Van Laarhoven, Aarts and Lenstra [1991] applied the principle of *simulated annealing* to the job shop scheduling problem. This is a randomized variant of iterative improvement. It is based on local search, but accepts deteriorations with a small and decreasing probability in the hope of avoiding bad local optima and getting settled in a global optimum. In the present case, the neighborhood of a schedule contains all schedules that can be obtained by interchanging two operations i and j for which $M_i = M_j$ and the arc (i,j) is on a longest path.

The *computational merits* of all these algorithms are accurately reflected by their performance on the notorious 10-job 10-machine problem instance dating back to 1963.

The single-machine bound, maximized over all machines, has a value of 808. In 1975, McMahon and Florian used the single-machine bound and a branching scheme that constructs all left-justified schedules to arrive at a schedule of length 972, without proving optimality. In 1983, Fisher, Lageweg, Lenstra and Rinnooy Kan applied surrogate duality relaxation to find a lower bound of 813; the time requirements involved did not encourage them to carry on the search beyond the root of the tree. In 1984, Lageweg developed an improved implementation of the McMahon-Florian algorithm, with an adaptive search strategy, and found a schedule of length 930; he also computed a number of multi-machine bounds, ranging from a three-machine bound of 874 to a six-machine bound of 907. Two years later, Carlier and Pinson [1989] proved optimality of the value 930; they used a relaxation of the single-machine bound, a drastically different branching scheme, and many elimination rules. The main drawback of

all these enumerative methods, aside from the limited problem sizes that can be handled, is their sensitivity to particular problem instances and even to the initial value of the upper bound.

The computational experience with polyhedral techniques that has been reported in recent years is slightly disappointing in view of what has been achieved for the traveling salesman problem and the linear ordering problem. However, the investigations in this direction are still at an initial stage.

Dispatch rules show an erratic behavior. The rule proposed by Lageweg, Lenstra and Rinnooy Kan [1977] constructs a schedule of length 1082, and most other priority functions do worse. Adams, Balas and Zawack [1988] report that their sliding bottleneck heuristic obtains a schedule of length 1015 in ten CPU seconds, solving 249 single-machine problems on the way. Their partial enumeration procedure succeeds in finding the optimum, after 851 seconds and 270 runs of the first heuristic.

Five runs of the simulated annealing algorithm with a standard setting of the cooling parameters take 6000 seconds on average and produce an average schedule length of 942.4, with a minimum of 937. If 6000 seconds are spent on deterministic neighborhood search, which accepts only true improvements, then more than 9000 local optima are found, the best one of which has a value of 1006. Five runs with a much slower cooling schedule take about 16 hours each and produce solution values of 930 (twice), 934, 935 and 938. In comparison to other approaches, simulated annealing requires unusual computation times, but it yields consistently good solutions with a modest amount of human implementation effort and relatively little insight into the combinatorial structure of the problem type under consideration.

10.4. *Related models*

The theory of scheduling is concerned with the optimal allocation of scarce resources to activities over time. It has been the subject of extensive research over the past decades. The emphasis has been on the investigation of *deterministic machine scheduling* problems, in which each activity requires at most one resource at a time, each resource can perform at most one activity at a time, and all problem data are known in advance. This problem class is surveyed extensively by Lawler, Lenstra, Rinnooy Kan and Shmoys [1991]. The results for these problems have reached the level of detail that a computer program is being used to maintain a record of the complexity status of thousands of problem types.

We mention two natural extensions of this class that are of obvious practical importance. In *resource-constrained project scheduling*, an activity may require several resources to be performed and a resource may be able to handle several activities simultaneously. In *stochastic scheduling*, some problem parameters are random variables. Either class has generated an impressive literature of its own; see Lawler, Lenstra, Rinnooy Kan and Shmoys [1991] for references.

Acknowledgements

We gratefully acknowledge the valuable comments and suggestions of David Shmoys and the editors. The research of the second author was partially supported by NSF Grant CCR-89-96272 with matching funds from UPS and Sun, and by the Cornell Computational Optimization Project.

REFERENCES

J. ADAMS, E. BALAS, D. ZAWACK (1988). The shifting bottleneck procedure for job shop scheduling. *Management Sci. 34*, 391-401.

J.M. ANTHONISSE (1984). Proportional representation in a regional council. *CWI Newsletter 5*, 22-29.

J.M. ANTHONISSE, J.K. LENSTRA (1984). Operational operations research at the Mathematical Centre. *European J. Oper. Res. 15*, 293-296.

J.M. ANTHONISSE, J.K. LENSTRA, M.W.P. SAVELSBERGH (1988). Behind the screen: DSS from an OR point of view. *Decision Support Systems 4*, 413-419.

E.M. ARKIN, E.B. SILVERBERG (1987). Scheduling jobs with fixed start and end times. *Discrete Appl. Math. 18*, 1-8.

L.R. ARNOLD, R.E. BECKWITH, C.M. JONES (1973). Scheduling the 41st-ORSA-Meeting sessions: the visiting-fireman problem, II. *Oper. Res. 21*, 1095-1103.

M.L. BALINSKI, G. DEMANGE (1989). An axiomatic approach to proportionality between matrices. *Math. Oper. Res. 14*, 700-719.

M.L. BALINSKI, H.P. YOUNG (1982). *Fair Representation: Meeting the Ideal of One Man, One Vote*, Yale University Press, New Haven, CT.

R.G. BLAND, D.F. SHALLCROSS (1989). Large traveling salesman problems arising from experiments in X-ray crystallography: a preliminary report on computation. *Oper. Res. Lett. 8*, 125-128.

L. BODIN, B. GOLDEN, A. ASSAD, M. BALL (1983). Routing and scheduling of vehicles and crews: the state of the art. *Comput. & Oper. Res. 10*, 63-211.

J. CARLIER, E. PINSON (1989). An algorithm for solving the job-shop problem. *Management Sci. 35*, 164-176.

R.C. CARLSON, G.L. NEMHAUSER (1966). Scheduling to minimize interaction cost. *Oper. Res. 14*, 52-58.

J.S. CRAMER, A.I.M. KOOL, J.K. LENSTRA, G. DE LEVE (1970). *Beroep en Opleiding — een Enquête Onder Econometristen* [*Profession and Education: a Survey Among Econometricians*; in Dutch], Institute for Actuarial Sciences and Econometrics, University of Amsterdam.

G.B. DANTZIG, D.R. FULKERSON (1954). Minimizing the number of tankers to meet a fixed schedule. *Naval Res. Logist. Quart. 1*, 217-222.

M. DESROCHERS, J.K. LENSTRA, M.W.P. SAVELSBERGH, F. SOUMIS (1988). Vehicle routing with time windows: optimization and approximation. B.L. GOLDEN, A.A. ASSAD (eds.) (1988). *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, 65-84.

J. EDMONDS, D.R. FULKERSON (1970). Bottleneck extrema. *J. Combin. Theory 8*, 299-306.

M. FISCHETTI, S. MARTELLO, P. TOTH (1987). The fixed job schedule problem with spread-time constraints. *Oper. Res. 35*, 849-858.

M. FISCHETTI, S. MARTELLO, P. TOTH (1989). The fixed job schedule problem with working-time constraints. *Oper. Res. 37*, 395-403.

M.L. FISHER, R. JAIKUMAR (1981). A generalized assignment heuristic for vehicle routing. *Networks 11*, 109-124.

M.L. FISHER, R. JAIKUMAR, L. VAN WASSENHOVE (1986). A multiplier adjustment method for the generalized assignment problem. *Management Sci. 32*, 1095-1103.

L.R. FORD, JR., D.R. FULKERSON (1962). *Flows in Networks*, Princeton University Press,

Princeton, NJ.

M. GRÖTSCHEL, O. HOLLAND (1990). Solution of large-scale symmetric travelling salesman problems. *Math. Programming*, to appear.

M. GRÖTSCHEL, Y. WAKABAYASHI (1989). A cutting plane algorithm for a clustering problem. *Math. Programming 45*, 59-96.

MEIGU GUAN [KWAN MEI-KO] (1962). Graphic programming using odd or even points. *Chinese Math. 1*, 273-277.

U.I. GUPTA, D.T. LEE, J.Y.-T. LEUNG (1979). An optimal solution for the channel-assignment problem. *IEEE Trans. Comput. C-28*, 807-810.

M. HELD, R.M. KARP (1970). The traveling-salesman problem and minimum spanning trees. *Oper. 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.

W.-L. HSU, G.L. NEMHAUSER (1979). Easy and hard bottleneck location problems. *Discrete Appl. Math. 1*, 209-215.

M. JÜNGER (1985). *Polyhedral Combinatorics and the Acyclic Subdigraph Problem*, Heldermann, Berlin.

B.W. KERNIGHAN, S. LIN (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J. 49*, 291-307.

D.E. KNUTH, M.F. PLASS (1981). Breaking paragraphs into lines. *Software — Practice and Experience 11*, 1119-1184.

A.W.J. KOLEN, J.K. LENSTRA, C.H. PAPADIMITRIOU (1991). *Interval Scheduling Problems*, in preparation.

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1977). Job-shop scheduling by implicit enumeration. *Management Sci. 24*, 441-450.

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

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, D.B. SHMOYS (eds.) (1985). *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, D.B. SHMOYS (1991). Sequencing and scheduling: algorithms and complexity. S.C. GRAVES, A.H.G. RINNOOY KAN, P. ZIPKIN (eds.) (1991). *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, North-Holland, Amsterdam.

J.K. LENSTRA (1974). Clustering a data array and the traveling-salesman problem. *Oper. Res. 22*, 413-414.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1975). Some simple applications of the travelling salesman problem. *Oper. Res. Quart. 26*, 717-733.

J.K. LENSTRA, A.H.G. RINNOOY KAN (1981). Complexity of vehicle routing and scheduling problems. *Networks 11*, 221-227.

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. *Oper. Res. 21*, 498-516.

W.T. MCCORMICK, JR., P.J. SCHWEITZER, T.W. WHITE (1972). Problem decomposition and

data reorganization by a clustering technique. *Oper. Res. 20*, 993-1009.

M. MINOUX (1986). Solving integer minimum cost flows with separable convex cost objective polynomially. *Math. Programming Stud. 26*, 237-239.

P.B. MIRCHANDANI, R.L. FRANCIS (eds.) (1990). *Discrete Location Theory*, Wiley, New York.

B.M.E. MORET, H.D. SHAPIRO (1985). On minimizing a set of tests. *SIAM J. Sci. Statist. Comput. 6*, 983-1003.

M.W. PADBERG, G. RINALDI (1987). Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Lett. 6*, 1-7.

H.D. RATLIFF, A.S. ROSENTHAL (1983). Order picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Oper. Res. 31*, 507-521.

G. REINELT (1985). *The Linear Ordering Problem: Algorithms and Applications*, Heldermann, Berlin.

M.W.P. SAVELSBERGH (1990). *Computer Aided Routing*, CWI Tract, Centre for Mathematics and Computer Science, Amsterdam.

P.J.M. VAN LAARHOVEN, E.H.L. AARTS, J.K. LENSTRA (1991). Job shop scheduling by simulated annealing. *Oper. Res.*, to appear.