PART IV
CHAPTER 2

Approaches to Hard Combinatorial Optimization Problems

Martin GRÖTSCHEL *)

Institut für Ökonometrie und Operations Research
Universität Bonn
Nassestrasse 2
D-5300 Bonn 1
West Germany

Contents

*) Supported by Sonderforschungsbereich 21 (DFG), Institut für Ökonometrie und Operations Research, Universität Bonn, West-Germany.

MODERN APPLIED MATHEMATICS – Optimization and Operations Research
Edited by B. KORTE

This paper is an extensive survey of those problems considered in combinatorial optimization which are usually termed 'hard' and of the methods with which these problems are attacked. In Section 1 we give an introduction to the subject and define our notation. Section 2 surveys complexity theory. In particular, it describes precisely what the notions 'hard' and 'easy' mean. A large number of examples of combinatorial optimization problems and some of their applications are given in Section 3. The design of heuristic algorithms is treated in Section 4. We discuss worst-case bounds, approximation schemes, and probabilistic analysis. The principle methods available for the exact solution of a combinatorial optimization problem are introduced in Section 5. The basic ingredients of a branch and bound algorithm are described in Section 6, an example of such a method for the symmetric travelling salesman problem is also given. In Section 7 Lagrangean relaxation is discussed. Polyhedral combinatorics and cutting plane methods for combinatorial optimization problems are described in Section 8.

1. Introduction and notation

1.1. Combinatorial optimization problems

Combinatorial optimization can be considered as the branch of mathematics that studies the following kind of problems:

1.1. *Given a finite number of objects, say* $\mathcal{F}$, *and an 'objective' function* $f: \mathcal{F} \to S$ (*where* S *is an ordered set*) *which associates with every object a 'cost', 'value', 'weight', 'distance' or the like. Find an element of* $\mathcal{F}$ *whose cost is minimum or maximum with respect to some criterion.*

Since $\mathcal{F}$ is finite, an optimal object can of course be found by completely enumerating $\mathcal{F}$ (if the objects in $\mathcal{F}$ are computationally accessible in some way), thus the main task of combinatorial optimization is to design algorithms which do better than that or prove that no algorithm better than complete enumeration exists.

Most of the problems studied in the early days of combinatorial optimization came from operations research, industrial management, logistics, engineering, computer science and military applications. But problems of this kind arise almost everywhere, and therefore combinatorial optimization has found successful applications in fields like archeology, biology, chemistry, economics, geography, linguistics, physics, sociology and others. Due to its wide range of applicability and due to the fact that most of the real-world problems appear to be hard to solve, the literature on combinatorial optimization has grown explosively in the recent years, cf. Kastning (1976), Hausmann (1978).

It appears that many of the optimization problems occuring in practice can be put in the following form which is more special than 1.1.

1.2. *Let E be a finite set and $\mathcal{F}$ a set of subsets of E called the set of feasible solutions. Let $c: E \to \mathbb{R}$ be an objective function. Find a feasible set $I^* \in \mathcal{F}$ such that*

$$\sum_{e \in I^*} c(e) = \max \left\{ \sum_{e \in I} c(e) \mid I \in \mathcal{F} \right\}.$$

This kind of problem is called a *linear objective combinatorial optimization problem*. Since from now on we will consider only this class of problems, we skip the phrase 'linear objective' to avoid repetition. For notational convenience we often abbreviate $\sum_{e \in I} c(e)$ by $c(I)$ in the sequel. Therefore, a combinatorial optimization problem is given by a triple $(E, \mathcal{F}, c)$, where E is the ground set, $\mathcal{F} \subseteq \mathcal{P}(E)$ is the set of feasible solutions, c is a linear objective function, and a feasible solution I^* has to be found such that $c(I^*)$ is maximum (or minimum).

1.2. Basic concepts of graph theory

The most frequently studied problems in combinatorial optimization can be formulated in the language of graph theory whose basic concepts will now be introduced.

A *graph* G is a pair $[V, E]$ where V is a finite nonempty set of elements called *nodes* and E is a finite set of two-element subsets of V called *edges*. For convenience we shall denote an edge $\{v, w\}$ by vw. The number of nodes of G is called the *order* of G, the number of edges the *size* of G. The order will usually be denoted by n and the size by m. The graph of order n in which any two nodes are adjacent is called *complete* and is denoted by K_n.

If $e = vw$ is an edge then e is said to *join* v and w, we also say that v and w are the *endnodes* of e, that e is *incident* with v and w, and that v and w are *adjacent* or *neighbours*. Two edges with a common endnode are called *adjacent*. The set of neighbours of a node v is denoted by $\Gamma(v)$, and the set of edges incident with v by $\omega(v)$, $d(v) := |\Gamma(v)|$ is the *degree* of v. The set of edges having one endnode in $W \subseteq V$ and the other in $V \setminus W$ is called a *cut* and is denoted by $\omega(W)$.

A *stable set* in G is a set of nodes no two of which are adjacent, a *clique* is a set of nodes every two of which are adjacent.

If for every node $v \in V$ an integer $b_v \geqslant 0$ is given, then an edge set M such that $|M \cap \omega(v)| \leqslant b_v$ holds for all $v \in V$ is called a *b-matching*. Thus a *1-matching* (i.e. $b_v = 1 \ \forall\, v \in V$) is a set of edges no two of which are adjacent. A b-matching M is called *perfect*, if every node v has degree b_v in M.

A *walk* of *length* k is a sequence of k edges $e_1, e_2, \ldots, e_k$ where every edge is of the form $e_i = v_{i-1} v_i$, $i = 1, \ldots, k$. If, in addition, all edges e_i, $i = 1, \ldots, k$, are distinct, then a walk is called a *trail*. If all nodes v_i, $i = 0, \ldots, k$ are distinct a walk is called a *path*. A walk is denoted by $v_0, v_1, \ldots, v_k$. The set of edges we obtain from a path $v_1, v_2, \ldots, v_k$ of length $k - 1$ by adding the edge $v_1 v_k$ is called a *cycle of length* k and is denoted by $[v_1, \ldots, v_k]$.

If G is a graph of order n, then a cycle of length n (path of length $n - 1$) is called *hamiltonian*. A trail $v_0, v_1, \ldots, v_m$ which contains every edge of G exactly once and for which $v_0 = v_m$ holds is called an *eulerian tour*. A graph G which contains a hamiltonian cycle (eulerian tour) is called *hamiltonian* (*eulerian*).

A set of edges in a graph G of order n which does not contain a cycle is called a *forest*, a forest with $n - 1$ edges is called a *spanning tree* or just a *tree* of G.

Two nodes $u, v \in V$ are said to be connected if G contains a path from u to v. A graph is *connected* if every two nodes are connected.

A graph $G = [V, E]$ is called *bipartite* if its node set can be partitioned into two disjoint, nonempty stable node sets. One can show that a graph G is bipartite if and only if G contains no cycle of odd length.

A *directed graph* or *digraph* D is a pair (V, A) where V is a finite nonempty set of elements called *nodes*, and A is a finite set of ordered pairs of distinct nodes of V called *arcs*. Arcs are denoted by (v, w) for $v, w \in V$. The arc $a = (v, w)$ is said to be *incident from* v and *incident to* w, v is the *tail* of a and w the *head* of a, v and w are also called *endnodes* of a, v is the *predecessor* of w and w the *successor* of v. The set of predecessors of $v \in V$ is denoted by $\Gamma^-(v)$, the set of successors by $\Gamma^+(v)$. The set of arcs incident to v is denoted by $\omega^-(v)$, the set of arcs incident from v by $\omega^+(v)$. *Diwalks* (directed walks), *ditrails* (directed trails), *dipaths* (directed paths) and *dicycles* (directed cycles) are defined analoguously to graphs taking into account that consecutive arcs a_l, a_{l+1} have to be of the form $a_l = (v_{l-1}, v_l)$, $a_{l+1} = (v_l, v_{l+1})$. The digraph of order n in which every node is the predecessor and successor of every other is called *complete*.

Good books on graph theory which treat the above mentioned and further concepts in depth are Behzad, Chartrand and Lesniak-Foster (1979), Berge (1973), Bollobas (1978) and Bondy and Murty (1976).

Since the subject of combinatorial optimization is progressing so fast and the most interesting concepts and topics are changing so rapidly, it is more difficult to find good up-to-date books on combinatorial optimization. The most recent ones are of Lawler (1976) and Gondran and Minoux (1979) which treat the class of problems which we shall call 'easy' later on. Christofides (1975) mainly discusses so-called 'hard' problems. Older books are Hu (1969), Garfinkel and Nemhauser (1972) and Burkard (1972). There exist good surveys on various topics. A general survey is Klee (1980), some others will be mentioned below. Fridman (1978) gives an account of the recent developments of this subject in the Soviet Union.

2. Complexity theory

2.1. Problems and instances

In mathematics (and elsewhere) the word 'problem' is used in different meanings. For our purposes, a *problem* will be a general question to be answered which has several parameters (or variables) whose values are left open. A problem is defined by giving a description of all its parameters

and specifying what properties an (optimal) *solution* is required to satisfy. If all the parameters are set to certain values we speak of an *instance* of the problem.

For example, the *stable set problem* is the following: Given a graph $G = [V, E]$ and a function $c: V \to \mathbb{R}$, find a stable set in G such that the sum of its node weights is as large as possible. Thus, the open parameters of the stable set problem are the graph G and the objective function c. If a particular graph and a particular function are given, we have an instance of the stable set problem.

An algorithm is said to *solve* a problem P if for any instance I of P, the algorithm produces an (optimal) solution. The objective of algorithm design is of course to find the 'most efficient' algorithm for solving a problem.

2.2. *Efficiency measures*

The notion of efficiency needs some clarification. In the early days of mathematical programming efficiency was usually tested empirically by running the programmed algorithm on several data sets, measuring the time and storage space needed and fitting these measures to some curves. Although such data are often very helpful in practice, there is no guarantee that the same program would behave similarly on other data sets.

In order to capture certain aspects of 'efficiency' computer scientists have defined various *complexity measures*. We shall only elaborate on one, namely *time complexity*, since time requirements are usually the most important ones in judging the efficiency of an algorithm.

It is obvious that for almost any imaginable problem the running time of an algorithm to solve a problem instance depends on the 'size' of the instance, e.g. to calculate the maximum weight stable set of a graph with large order should take longer than for one with small order. It is customary to measure the size of a problem instance informally by some parameters. In the stable set case, the size could be specified by the number of nodes and edges of the graph and the numbers c_v, for all $v \in V$, of the objective function.

This kind of informal measurement can be rigorously formalized by considering an *encoding scheme* which maps problem instances into strings of symbols describing them. In real-world computers these strings usually consist of zeros and ones, since the binary encoding appears to be the

most convenient one for our present computer technology. The *size* (or *input length*) of the problem instance is then defined to be the length of this string of symbols. Such a string can be used to input the instance into a real (or hypothetical) machine. In order to prove interesting mathematical theorems involving the 'size' of problem instances one has to make sure that the results do not considerably change by taking (slightly) different encoding schemes and (slightly) different machines. The machines most frequently used in theoretical analysis are the Turing machine and the RAM machine which are in some sense (in particular, for our purposes) equivalent. For both machines different choices of (reasonable) encoding schemes have almost no effect on the results we are interested in. For the reader not familiar with these concepts, it is completely sufficient to consider a real-world computer instead of a Turing machine.

Given an encoding scheme and a machine model, the *time complexity function* f: $\mathbb{N} \to \mathbb{N}$ of an algorithm expresses its time requirements by giving, for each possible size or input length $n \in \mathbb{N}$, the largest amount of time needed by the algorithm to solve a problem of that size.

The time requirements are measured by counting the number of 'elementary steps' the algorithm performs to solve the problem instance. Informally, such operations as addition, multiplication, comparison are considered elementary steps, keeping in mind, however, that, if large numbers are involved, operations with these numbers require more than one elementary step.

When a proper measure of time complexity is found, 'efficiency' has to be defined. We say that a function $f(n)$ is $O(g(n))$, in words f is of order g, if there is a constant c such that $|f(n)| \leqslant c\,|g(n)|$ for all integers $n \geqslant 0$. A *polynomial time algorithm* is an algorithm whose time complexity function $f(n)$ is $O(p(n))$ for some polynomial p: $\mathbb{N} \to \mathbb{N}$. Any algorithm whose time complexity function cannot be bounded by a polynomial is said to be a *superpolynomial* or *exponential time algorithm*. It is well known that exponential functions grow much faster than polynomials, therefore polynomial time algorithms are much more desirable than exponential ones. Hence, we shall say that an algorithm is *efficient* or *good* if it runs in polynomial time, keeping, however, in mind, that a good algorithm is not necessarily efficient in practice, since an $O(n^{1000})$-algorithm is of almost no practical value. Such cases of useless good algorithms occur quite rarely. Many of the good algorithms for problems of practical interest are also efficient in real life.

2.3. *The classes* P *and* NP

We now want to distinguish between hard and easy problems. In order to be able to use a formally convenient apparatus we shall analyse *decision problems* only which are problems having only two possible solutions, either 'yes' or 'no'. A 'natural' problem of this kind is "Does a graph G contain a hamiltonian cycle?" Optimization problems can be transformed into decision problems as follows: Let $(E, \mathcal{F}, c)$ be a combinatorial maximization problem and B be a number (lower bound), the corresponding decision problem is "Is there a feasible solution whose value is at least B?", e.g. "Is there a stable set in the graph whose sum of node weights is at least B?".

The class of all those decision problems for which a polynomial time algorithm exists is called the *class P*. More correctly, given an encoding scheme E, say binary encoding, and a computer, say a Turing machine; let π be a decision problem and every instance of π be encoded in E. Then π belongs to class P if there is a polynomial $p_\pi(n)$ and a Turing machine program which for every instance of π halts with a correct answer ('yes' or 'no') after at most $O(p_\pi(n))$ elementary steps where n is the size of the instance encoded in E. As we shall see in Section 3 there are many decision problems which are not known to be in P, e.g., "Is a graph hamiltonian?".

Although there is no known polynomial algorithm for checking whether a graph has a hamiltonian cycle or not, it is simple to verify hamiltonicity. Suppose someone claims that a certain graph is hamiltonian; if he displays a hamiltonian cycle (by whatever means he has found it), then everybody can easily convince himself that this claim is correct or not. ('Easily' means that there is a polynomial time algorithm deciding whether a given set of edges is a hamiltonian cycle or not.) If he claims that the graph is non-hamiltonian, however, no simple proof of this fact is known.

This informal discussion leads us to the concept of nondeterministic algorithms. A *nondeterministic algorithm* is composed of two stages. Given an encoded problem instance I, the first stage guesses some object and encodes it. Then the encoded problem instance and the encoded object will be given as input to the checking stage in which the algorithm tries to find out whether the object yields a positive answer to the given instance of the decision problem.

E.g. in the case of the stable set problem "Is there a stable set in the graph $G = [V, E]$ whose sum of node weights is at least B?", the first stage

guesses and encodes a set S of $k \leqslant |V|$ nodes of G and the second stage checks whether S is a stable set of G with weight sum at least B.

We say that a nondeterministic algorithm solves a decision problem π in polynomial time if there exist two polynomials $p_\pi(n)$ and $q_\pi(n)$ such that the following two properties hold for all instances I of π:

(1) If the answer to I is 'yes' then there exists an object S (i.e. an object that can be guessed) which satisfies

(a) if n is the input length of I then S can be encoded in $O(p_\pi(n))$ time (nothing is said about how S is computed or guessed),

(b) if I and S are used as inputs for the checking stage and n' is the input length of S then the answer will be 'yes' in $O(q_\pi(n + n'))$ time.

(2) If the answer to I is 'no', then there exists no object S which will lead the checking stage to respond 'yes' when I and S are used as inputs.

Notice that this definition is nonsymmetric in 'yes' and 'no'. A polynomial nondeterministic algorithm will only run in polynomial time if the input gives rise to a 'yes' answer, if the answer is 'no' then nothing is said about the performance of the algorithm, it may, for example, run forever.

Informally, the *class* NP is defined to be the class of all decision problems that, under reasonable encoding schemes, can be solved by polynomial time nondeterministic algorithms. The class NP can be rigorously defined by introducing nondeterministic machines, e.g. nondeterministic Turing machines, and precisely specifying the more intuitive notions presented above.

Our discussion above shows that the hamiltonian cycle problem is in NP, since, if a hamiltonian cycle exists it can be 'guessed', then it can be encoded in polynomial time, and to check that this cycle is a hamiltonian cycle of the given graph can also be done in polynomial time.

It is obvious that $P \subseteq NP$ holds, and it appears to be as obvious that $P \neq NP$, since nondeterministic algorithms seem to be more powerful than deterministic ones. However, despite enormous research efforts the problem of whether or not P equals NP is still one of the major open problems in mathematics.

Cook (1971) introduced a class of decision problems which are in a well-defined sense the hardest problems in NP. Suppose we have two decision problems π and π' and a fixed encoding scheme. A *polynomial transformation* is an algorithm which, given an encoded instance of π, produces in polynomial time an encoded instance of π' such that the following holds: For every instance I of π, the answer to I is 'yes' if and only if

the answer to the transformation of I (as an instance of π') is 'yes'. Clearly, if there is a polynomial algorithm to solve π' then by polynomially transforming any instance of π to an instance of π' there is also a polynomial algorithm to solve π. We call a decision problem π NP-*complete* if $\pi \in$ NP and every other problem in NP can be polynomially transformed to π. Thus, every NP-complete problem π has the following property: if π can be solved in (deterministic) polynomial time then all NP-problems can be solved in (deterministic) polynomial time, i.e. if π is NP-complete and if $\pi \in P$ then $P =$ NP. This justifies saying that NP-complete problems are the hardest NP-problems. Both the stable set decision problem and the hamiltonian cycle problem discussed above are NP-complete problems.

2.4. *Hard and easy problems*

Unfortunately, optimization problems cannot be completely expressed as decision problems. There is, however, a satisfactory way to treat almost all combinatorial optimization problems. This goes as follows.

As shown above we can formulate a maximization problem as a decision problem by additionally introducing a bound B and asking 'Is there a feasible solution whose value is at least B'. Supposing there is a polynomial algorithm to solve the optimization problem, we first compute the optimal solution and its value, then we compare the optimal value with the bound B and hence are able to solve the decision problem in polynomial time. Informally we call an optimization problem Q NP-*hard* if there is an NP-complete decision problem which can be polynomially reduced to Q. For example, the decision problem 'Is there a stable set in G whose value is at least B?' can be shown to be NP-complete, thus the stable set optimization problem is NP-hard.

Very often one can use polynomial algorithms for decision problems to solve the corresponding optimization problem in polynomial time. For example, suppose we have a polynomial algorithm for the stable set decision problem. Assume for convenience that all node weights c_v, $v \in V$, are nonnegative integers, then the value of the maximum weight stable set cannot be larger than $n\alpha$, where $\alpha = \max\{c_v \mid v \in V\}$, and it is not smaller than zero. Now ask whether there is a stable set of value at least $n\alpha/2$, if yes ask whether there is a stable set of value at least $3n\alpha/4$, otherwise ask whether there is a stable set of value at least $n\alpha/4$, and continue by successively halving the remaining interval of uncertainty. Since the optimum

value is integer, this decision problem has to be solved at most $\lceil \log_2(n\alpha) \rceil + 1$ times. If the decision problem could be solved in polynomial time, then the optimum value could be calculated in polynomial time with this so-called *binary search method* given above, and similarly, the optimum solution could also be computed in polynomial time. Informally, let us call an optimization problem NP-*easy* if there exists a decision problem $\pi \in$ NP such that the optimization problem can be polynomially reduced (e.g. by a binary search method) to π. For example, finding a maximum weight stable set in a graph is NP-easy since it can be solved by calling the stable set decision problem at most a polynomial number of times.

An optimization problem which is both NP-hard and NP-easy is called NP-*equivalent*. By definition, if P $\neq$ NP then no NP-hard problem can be solved in polynomial time, and if P = NP then every NP-easy problem can be solved in polynomial time. Therefore, all NP-equivalent problems can be solved in polynomial time if and only if P = NP.

For our convenience we shall henceforth call all problems *hard* which are NP-hard or NP-complete, and all those problems which are solvable by a polynomial time (deterministic) algorithm are called *easy*.

Our introduction to some aspects of complexity theory is more or less informal; all these concepts and other related ones can be defined rigorously in order to make a clear distinction between 'hard' and 'easy' possible (although at present it is not known whether these measures of 'hard' and 'easy' are really different). For further reading about these topics we recommend Aho, Hopcroft and Ullman (1975), Bachem (1982), and Garey and Johnson (1979).

3. Examples and applications

In this section we give an overview on various combinatorial optimization problems and their applications to real-life situations. This list is certainly not complete but contains many of those problems which have a wide range of applications or which are of particular theoretical interest.

3.1. Independence systems. Let E be a finite set and $\mathcal{I}$ be a subset of the power set of E. If $\mathcal{I}$ satisfies the following two axioms:

$$\emptyset \in \mathcal{I}, \tag{I.1}$$

$$I \subseteq J \in \mathcal{I} \Rightarrow I \in \mathcal{I}, \tag{I.2}$$

then $(E, \mathcal{I})$ or just $\mathcal{I}$ is called an *independence system*. If $c: E \to \mathbb{R}$ is an objective function, then $\max\{c(I) \mid I \in \mathcal{I}\}$ is the associated optimization problem.

Most of the subsequent problems of this section can be formulated as independence systems either directly or by making a slight transformation as follows: if $\mathcal{I} \subseteq \mathcal{P}(E)$ then

$$\tilde{\mathcal{I}} := \{I \subseteq E \mid \exists\, J \in \mathcal{I} \text{ with } I \subseteq J\}$$

is called the *independence system associated with* $\mathcal{I}$ or the *monotonization* of $\mathcal{I}$. It is often simple to transform the objective function $c: E \to \mathbb{R}$ into an objective function $\tilde{c}: E \to \mathbb{R}$ such that the set of optimal solutions of $(E, \mathcal{I}, c)$ and $(E, \tilde{\mathcal{I}}, \tilde{c})$ coincide and that $\tilde{c}(I) = c(I) + k \;\forall\, I \in \mathcal{I}$ where k is a constant.

Optimization problems over independence systems are in general hard. We shall see, however, that if additional requirements are introduced they may be solvable in polynomial time.

A special case of independence system are matroids (an excellent reference for matroid theory is Welsh (1976)).

3.2. The matroid problem. If $(E, \mathcal{I})$ is an independence system and $F \subseteq E$, then a subset $B \subseteq F$ with $B \in \mathcal{I}$ is called a *basis* of F if there is no $B' \in \mathcal{I}$ with $B' \subseteq F$ and B' strictly contains B.

An independence system $(E, \mathcal{I})$ is called a *matroid* if the following additional axiom is satisfied.

> For every subset $F \subseteq E$, every two bases of F have the same cardinality. (I.3)

If $c: E \to \mathbb{R}$ is an objective function and $(E, \mathcal{I})$ a matroid, then the matroid problem $(E, \mathcal{I}, c)$ is the problem of solving $\max\{c(I) \mid I \in \mathcal{I}\}$.

Matroid problems $(E, \mathcal{I}, c)$ can be solved with the following algorithm:

3.3. Greedy algorithm. 1. Order the positive elements of E such that $c(e_1) \geqslant c(e_2) \geqslant \cdots c(e_n) > 0$ holds. (This can be done in time $O(n \log n)$.)

2. Set $F = \emptyset$.

3. DO $i = 1$ TO n: IF $F \cup \{e_i\} \in \mathcal{I}$ THEN set $F := F \cup \{e_i\}$. END

It can be shown that the set F after termination of the algorithm is the optimum solution of the matroid problem and that F satisfies a certain

performance guarantee in case $(E, \mathcal{F})$ is a general independence system (cf. Edmonds (1971), Jenkyns (1976), Korte and Hausmann (1978), and Theorem 4.7).

If it is possible to check in polynomial time whether a set I belongs to $\mathcal{F}$, then the greedy algorithm obviously runs in polynomial time.

An example of a matroid is the following from linear algebra. Consider a finite set of vectors $x_1, x_2, \ldots, x_n$, and set $E = \{1, \ldots, n\}$. Let $\mathcal{F}$ be the set of all sets I of indices such that the vectors x_i, $i \in I$, are linearly independent (or affinely independent); then $(E, \mathcal{F})$ is a matroid. For our purposes the following example is more interesting.

3.4. The spanning tree problem. Let $G = [V, E]$ be a graph and $\mathcal{F}$ the set of all forests in G, then $(E, \mathcal{F})$ is a matroid. One can show that if G is connected then the bases of E are exactly the spanning trees of G. Thus, if we have an objective function $c: E \to \mathbb{R}_+$, then every optimal solution will be a spanning tree. Since the forests constitute a matroid, the maximum weighted spanning tree can be found by the greedy algorithm. However, there are algorithms which are more efficient than the greedy algorithm, cf. Prim (1957), Dijkstra (1959), Cheriton and Tarjan (1976).

If certain additional properties are required of the spanning tree then the new problem often becomes hard. For example, if we require that no node in the optimum spanning tree T has degree larger then an integer k, or that there are at least k nodes with degree 1, or that T contains no path of length larger than k, then the restricted spanning tree problem is hard. For further examples of this kind see Garey and Johnson (1979) p. 206ff.

There are certain applications of the spanning tree problem, e.g. the so called *connector problem* which is the following. Consider a set of towns (branches of a company) where a railway system (communication system) should be set up such that every town (branch) can be reached from every other. The objective is to design a network such that the total construction costs are minimal. Such applications are rather artificial and almost never occur in such a pure form. The importance of the spanning tree problem is that it is a close relative of other hard problems (we shall see later what that means) and that spanning tree calculations are often used as subroutines, cf. Algorithm 6.5.

3.5. 2-matroid intersection. Suppose there are two matroids $(E, \mathcal{F}_1)$ and $(E, \mathcal{F}_2)$ on the same ground set E and let $\mathcal{F} = \mathcal{F}_1 \cap \mathcal{F}_2$. If $c: E \to \mathbb{R}$ is

an objective function, then

$$\max\{c(I) \mid I \in \mathcal{I}\}$$

is called a 2-*matroid intersection problem.*

In general, the intersection of two matroids is not a matroid, but a surprising result is that 2-matroid intersection problems are easy. For polynomial algorithms to solve such problems see Lawler (1975), Edmonds (1979), Grötschel, Lovász and Schrijver (1981a) and Frank (1981).

One can show that every independence system is representable as the intersection of (possibly a very large number of) matroids. The above result, unfortunately, cannot be generalized to more than two matroids, since there are hard problems which are representable as the intersection of three matroids (e.g. the asymmetric travelling salesman problem).

A special case of the 2-matroid problem is:

3.6. The branching problem. Given a digraph $D = (V, A)$ and a function $c: A \to \mathbb{R}$. A *branching* B is a set of arcs such that B contains no cycle (in the undirected sense) and that every node of D is the head of at most one arc in B. The optimum branching problem is to find either a maximum weighted branching or a minimum weighted spanning branching.

It is easy to see that the independence system of branchings is the intersection of the forest-matroid on D and the matroid $\mathcal{I} := \{F \subseteq A \mid$ every node of D is the head of at most one arc of $F\}$. There exist very efficient methods to solve branching problems, cf. Edmonds (1967), Tarjan (1977).

As in the case of the spanning tree problem the main importance of the branching problem lies in its close relation to hard problems and therefore the possibility of using branching algorithms as subroutines of a branch and bound algorithm, cf. Section 7.

A further special case of the 2-matroid-intersection problem (and hence easy) is the following.

3.7. The bipartite matching or assignment problem. There are various ways to formulate this problem and thus different names are used in the literature. The *bipartite matching problem* is the following. Given a bipartite graph $G = [V, E]$ where the nodes are partitioned into two stable sets V_1, V_2 such that $V_1 \neq \emptyset \neq V_2$, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and given a function $c: E \to \mathbb{R}$. Find a maximum matching in G, i.e. a set of edges

such that no two edges have a common endnode and whose sum of edge weights is as large as possible.

The special case where $|V_1| = |V_2|$ is often called the *assignment problem* and has the following interpretation. Given n persons (or men) and n jobs (or women) let c_{ij}, $1 \leqslant i, j \leqslant n$, be a qualification measure for person i to successfully do job j (a measure of empathy between man i and woman j). Find an assignment of persons to jobs (assignment (marriage) of men to women) such that the total qualification measure (sum of empathy coefficients) is as high as possible.

The assignment problem can also be formulated on a digraph. Here one usually assumes that a complete diagraph $D = (V, A)$ with weights $c_{ij} \in \mathbb{R}$ $\forall$ $(i, j) \in A$ is given. One wants to find a set of arcs B such that every node of D is the head and the tail of exactly one arc in B and that the sum of the weights of the arcs of B is as small as possible. (Instead of maximizing empathy one can also minimize antipathy!) It is easy to see that every solution B of the assignment problem is the union of dicycles such that every node is contained in exactly one dicycle.

The so-called 'Hungarian Method' to solve the bipartite matching problem was one of the first good algorithms in combinatorial optimization and had a strong influence on further algorithm design, (cf. Kuhn (1955)). A survey of existing assignment and matching algorithms can be found in Lawler (1976), and in Burkard and Derigs (1980). Applications of the assignment problem are obvious. It is also of particular interest as a relaxation of the asymmetric travelling salesman problem, cf. 3.12 and Section 6.

3.8. The b-matching problem. If the bipartite graph in the previous problem (3.7) is replaced by an arbitrary graph we obtain the 1-matching problem. A further generalization goes as follows. Given a graph $G = [V, E]$, nonnegative integers b_v for all $v \in V$ and costs $c_e \in \mathbb{R}$ for all $e \in E$. There are three closely related problems:

(a) Find a b-matching in G of maximum total weight. This problem is called the *binary b-matching problem.*

(b) Find nonnegative integers x_e for all $e \in E$ such that $\sum_{v \in e} x_e \leqslant b_v$ for all $v \in V$ and $\sum_{e \in E} c_e x_e$ is as large as possible. We call this problem the *uncapacitated b-matching problem.*

(c) Given in addition nonnegative integers a_e, $e \in E$, find integers x_e for all $e \in E$ such that the following holds:

$$0 \leqslant x_e \leqslant a_e \quad \text{for all } e \in E,$$

$$\sum_{v \in e} x_e \leqslant b_v \quad \text{for all } v \in V,$$

$$\sum_{e \in E} c_e x_e \text{ is maximum.}$$

This problem is called the (*edge-*) *capacitated b-matching problem.*

Although none of these problems is a matroid or 2-matroid intersection problem, each is easy, i.e. can be solved in polynomial time (cf. Edmonds (1965), Edmonds and Johnson (1980), Pulleyblank (1973), Cunningham and Marsh (1978), Burkard and Derigs (1970)). Matching problems have many applications, e.g. in oil drilling, the design of telephone networks and physics (the planar spin glass problem within a magnetic field), cf. Barahona (1980), Devine (1973).

The *b*-matching problem seems to be close to the border between 'hard' and 'easy' since many slight variations of this problem turn out to be hard. For example, consider the uncapacitated 2-matching problem. Cornuejols and Pulleyblank (1980) have developed a polynomial algorithm which constructs an optimum *uncapacitated 2-matching without triangles* (cycles of length three), and Papadimitriou has shown that the problem of finding an optimum *uncapacitated 2-matching without a k-cycle* ($k \geqslant 5$ and odd) is hard. A further hard problem in matching theory is the problem of finding a *minimum maximum matching*, i.e. a matching which is not contained in any larger matching of a graph and which has as few edges as possible, (cf. Yannakakis and Gavril (1980)).

3.9. Shortest paths. Given a digraph $D = (V, A)$ and a 'distance' function $c: A \to \mathbb{R}$ which associates a length c_{ij} with every arc (i, j). If u and v are two nodes of D, then the problem of finding a directed path P from u to v such that $\sum_{(i,j) \in P} c_{ij}$ is as small as possible is called the *shortest path problem.*

This problem as well as several variations of it (shortest dipaths from one node to all others, shortest dipath between every pair of nodes, the corresponding problems for undirected graphs, etc.) are solvable in polynomial time. A good survey of the best known methods to date can be found in Lawler (1976). On the other hand, some variations of the shortest path problem which are hard are listed in Garey and Johnson (1979).

3.10. The network flow problem. One of the most important combinatorial optimization problems is the following. Given a digraph $D = (V, A)$ and a nonnegative 'capacity' c_{ij} for every arc $(i, j) \in A$. Let $s, t \in V$ be any nodes, call s the *source* and t the *sink* of D. A vector x having a component x_{ij} for every arc $(i, j) \in A$ is called a feasible s–t-flow in D if the following holds:

$$0 \leqslant x_{ij} \leqslant c_{ij} \quad \text{for all } (i, j) \in A, \tag{1}$$

$$\sum_{i \in \Gamma^-(j)} x_{ij} - \sum_{i \in \Gamma^+(j)} x_{ji} = \begin{cases} -v & \text{if } j = s, \\ 0 & \text{if } j \in V \setminus \{s, t\}, \\ v & \text{if } j = t. \end{cases} \tag{2}$$

(Condition (1) says that a feasible s–t-flow has to satisfy the capacity conditions, while condition (2) states that except for the source and the sink the flow going into node j also leaves j (conservation of flows)).

The number v associated with every flow vector x is called the *value of* x, and the *network flow problem* is to find a feasible s–t-flow whose value is as large as possible.

If, in addition, a cost is associated with every flow through an arc we can ask for a feasible s–t-flow with a certain (e.g. maximum) value whose cost is as small as possible. This and other variations of the network flow problem are solvable in polynomial time. The network flow problem has numerous real-world applications, (cf. Glover and Klingman (1977)), and some of the famous theorems of network flow theory like the max-flow min-cut theorem have far-reaching consequences in the theory of combinatorial optimization. The classical monograph on flow theory is Ford and Fulkerson (1962), more recent surveys on implementations of network flow algorithms can be found in Hu (1969), Lawler (1976), Ali, Helgason, Kennington and Lall (1978), Cheung (1980), Glover, Klingman, Mote and Whitman (1979), Kennington and Helgason (1980).

If the flows have to satisfy additional requirements then these modified network flow problems usually turn out to be hard. For a list of such variations of the network flow problem see Garey and Johnson (1979) p. 214ff.

3.11. The Chinese postman problem. A postman has to deliver mail in certain streets of a city the lengths of which are known. He looks for a minimum length walk starting at the post office and ending there. In terms

of graphs we can state this problem as follows: Given a graph $G = [V, E]$ and distances c_e for all $e \in E$. Find a shortest walk through the graph that passes through every edge at least once.

As shown by Edmonds and Johnson (1973) this problem can be solved in polynomial time using the shortest path and the 1-matching algorithms.

Clearly, the Chinese postman problem can also be formulated in a directed version for digraphs. This problem is also easy. But if we consider the Chinese postman problem on a mixed graph, i.e. a graph with directed and undirected edges, then it becomes hard, cf. Papadimitriou (1976).

The preceding list of easy problems (3.2, 3.4–3.11) is a collection of some of the most interesting easy problems, interesting from both the theoretical and practical point of view. As mentioned above, slight variations of these problems often turn out to be hard. This indicates that most of the real-world problems are hard, since these almost never occur in the pure form of the models described above. The above problems are also important because they are often used as relaxations of hard problems, i.e. good algorithms for the easy problems are used as subroutines for branch and bound procedures for hard problems, (cf. Section 6).

One of the most prominent and most intensively studied hard problems is the following.

3.12. The asymmetric travelling salesman problem (ATSP). A salesman lives in a city, say 1, wants to travel through cities $2, 3, \ldots, n$ and then return to his hometown. The distances between the cities are known and he wants to find a tour which is as short as possible. The graph theoretic formulation is as follows: Given a complete digraph $D = (V, A)$ of order n (i.e. between every pair of nodes i, j there is an arc from i to j and from j to i), and a 'distance' function $c: A \to \mathbb{R}$, find a hamiltonian dicycle (or tour) of minimum total distance.

3.13. The symmetric travelling salesman problem (STSP). In case the distance from city i to city j is the same as the one from j to i, we call the travelling salesman problem symmetric. Of course, the symmetric problem is a special case of the asymmetric one but practical experience has shown that algorithms for the asymmetric problem perform in general badly on symmetric ones, so the latter need special treatment. The usual

graph theoretical representation of the symmetric TSP is: Given a complete (undirected) graph $K_n = [V, E]$, find a shortest hamiltonian cycle (tour) through K_n.

The symmetric and the asymmetric travelling salesman problem have numerous real-world applications. There is, of course, the obvious one of finding a shortest roundtrip through several cities, but such optimization problems also occur in routing numerically controlled machines, e.g. backboard wiring, automatic soldering, automatic drilling (e.g. of computer chips) and the like. Similarly, all problems where regular maintenance or control of certain facilities is required (e.g. telephone booths, measuring stations for meteorological or pollution data) can be formulated as TSPs. Other applications include the sequencing of fluids in a multi-product pipeline, certain vehicle routing problems, the clustering of data arrays, jobshop scheduling with no wait in process, the sequencing of colors in varnishing machines, or the sequencing of different profiles in a rolling mill. Some of these examples and their formulation as TSP's can be found in Lenstra and Rinnooy Kan (1975), Lin and Kernighan (1973), and Steckhan and Thome (1972), and a book edited by Lawler, Lenstra, and Rinnooy Kan which will appear in 1982.

In many practical applications the pure TSP as described above cannot be used since additional constraints exist which require a similar but different model formulation. Some general types of combinatorial optimization problems which include the TSP are the following:

3.14. General routing problems. Consider a mixed graph G with node set V, (undirected) edges E, (directed) arcs A, and a cost function $c: E \cup A \to \mathbb{R}$. Furthermore, subsets $V' \subseteq V$, $E' \subseteq E$ and $A' \subseteq A$ are given. We seek a walk T in G of minimum total cost which contains the nodes V', the edges E' and the arcs A' such that all arcs of A' have the same orientation in T. This problem is called the *single vehicle routing problem.*

Obviously, if $E = E_n$, $A = A' = E' = \emptyset$ and $V' = V$, we obtain the symmetric TSP, and if $E' = E$, $A' = A$, and $V' = \emptyset$ we have the mixed Chinese postman problem. The problem where E is arbitrary, $A = A' = V' = \emptyset$ and E' is some subset of E is called the *rural postman problem.* If E and A are arbitrary $V' = E' = \emptyset$ and $A' = A$ we have the so-called *stacker crane problem.*

A natural extension is the *m-vehicle routing problem* where the purpose is to find m walks, each containing a common distinguished node (the depot) and collectively containing the sets V', E' and A', such that the sum of the total costs of the walks is minimized.

These general routing problems are, of course, hard and only a very few special cases like the directed and undirected (but not the mixed) Chinese postman problem can be solved in polynomial time.

Surveys on various aspects of vehicle routing can be found in the special issue of the journal *Networks* (Vol. 11, No. 2, 1981), e.g. the complexity of vehicle routing problems is treated in Lenstra and Rinnooy Kan (1981). Several methods for these problems are also described in Christofides, Mingozzi and Toth (1979). A further example of a practical application which is a variant of the TSP is given in Gensch (1978).

3.15. Scheduling problems. A general class of machine scheduling problems can be formulated as follows. A job J_i $(i = 1, \ldots, n)$ consists of a sequence of operations each of which corresponds to the uninterrupted processing of J_i on some machine M_j $(j = 1, \ldots, m)$ during a given period of time. Each machine can handle at most one job at a time. What is according to some overall optimality criterion the optimal processing order on each machine?

There is a large variety of different optimality criteria (e.g. minimizing total tardiness, minimizing maximum lateness, minimizing total cost) and further assumptions on the operations, jobs and machines (e.g. precedence constraints, group of parallel nonidentical machines), that have been discussed in the literature. For extensive surveys of the above stated scheduling problems, as well as others, we refer to Conway et al. (1967), Coffman (1976), Rinnooy Kan (1976) and the special issue of the journal *Operations Research* on Scheduling (Vol. 1, No. 1, 1978). There are many easy scheduling problems, but most of the practically relevant problems are hard. A classification of the complexity of scheduling problems can be found in Rinnooy Kan (1976), Graham et al. (1979).

Location problems come up in great variety. Some typical problems of this kind are the following:

3.16. Location problems. Given a graph (or digraph) where nodes represent communities, areas or the like and edges represent a road system. Where

should a hospital, police station, fire station or the like be 'optimally' located. 'Optimally' could mean here that the sum of the distances from the emergency station to the communities is as small as possible.

Another location problem can be stated as follows. Let $M = \{1, \ldots, m\}$ be a set of customer locations (markets) and $N = \{1, \ldots, n\}$ be a set of possible plant (warehouse, factory, facility service station) locations, say cities. Let $c_{ij} \geqslant 0$ be the value of serving customer $i \in M$ from city $j \in N$. Where should k plants be located to maximize total value? More precisely, suppose each customer should be served from the highest value available plant and that every plant can serve every customer. The value of placing a plant at every location $S \subseteq N$ is given by

$$c(S) = \sum_{i=1}^{m} \max_{j \in S} c_{ij}.$$

Therefore, this so-called *uncapacitated* (or *simple*) *location problem* can be stated as

$$\max\{c(S) \mid S \subseteq N,\ |S| = k\}.$$

All the above mentioned (and most other) location problems are hard. A nice analysis and applications of the simple plant location problem to the location of bank accounts are given in Cornuejols, Fisher and Nemhauser (1977), for a survey of the literature on this problem cf. Krarup and Pruzan (1979). Further location problems are treated in Christofides (1975).

3.17. Set covering, set partitioning, set packing. Given a finite set E and subsets $S_1, S_2, \ldots, S_n$ of E. Associate with each set S_j a cost c_j. Find a set $J \subseteq \{1, \ldots, n\}$ such that

(a) $\sum_{j \in J} c_j$ is minimum, and for every $i \in E$ there is *at least* one $j \in J$ with $i \in S_j$,

(b) $\sum_{j \in J} c_j$ is minimum (or maximum), and for every $i \in E$ there is *exactly* one $j \in J$ with $i \in S_j$,

(c) $\sum_{j \in J} c_j$ is maximum, and for every $i \in E$ there is *at most* one $j \in J$ with $i \in S_j$.

Problem (a) is called *set covering problem*, problem (b) *set partitioning problem*, problem (c) *set packing problem*.

It is very convenient to formulate these problems in terms of 0/1-matrices. Define a matrix A with $m = |E|$ rows and n columns where

$$a_{ij} := \begin{cases} 1 & \text{if } i \in E \text{ is contained in set } S_j, \\ 0 & \text{otherwise.} \end{cases}$$

Then the set covering (partitioning, packing) problem is given by

$$\begin{array}{ll} \min & \geqslant \\ \min\{c^T x \mid Ax = \mathbb{1}_m,\ x_i \in \{0,1\},\ i = 1, \ldots, n\} & \\ \max & \leqslant \end{array}$$

where $\mathbb{1}_m = (1, \ldots, 1)^T$ is an m-vector.

The literature on applications of these problems is rather extensive. Some practical problems which can be modeled as set covering, partitioning or packing problems are: information retrieval, truck deliveries, political districting, railroad and airline crew scheduling, tanker routing, switching circuit design, stock cutting, assembly line balancing, and some facility location problems. For a bibliography of such applications we refer to Balas and Padberg (1975).

3.18. Node covering, node partitioning, node packing. Special (but particularly interesting) cases of the preceding problems (3.17) are the node covering, node partitioning and node packing problems. Here a graph $G = [V, E]$ is given and our ground set is the edge set E. The subsets S_j of E are the sets $S_j := \omega(j)$, $j = 1, \ldots, n$, i.e. S_j is the set of edges incident with node j. The cost function associates with every S_j (or equivalently with every node j) a cost c_j. We therefore seek an optimal set J of nodes such that every edge is incident with at least one node of J (covering), incident with exactly one node of J (partitioning), incident with at most one node of J (packing).

The matrix A corresponding to these problems is therefore the edge–node incidence matrix of G. Note that the node packing problem is the same as the stable set problem introduced in Section 2. Both names are common in the literature.

In general, the node covering (partitioning, packing) problems are hard and thus the set covering (partitioning, packing) problems are as well. Due to its interesting properties the stable set problem has been the subject of intensive theoretical studies. As a consequence of these studies it has been

shown that the stable set problem can be solved in polynomial time for some classes of graphs, e.g. line graphs, bipartite graphs, $K_{1,3}$-free graphs (Minty (1980)) and perfect graphs (Grötschel, Lovász and Schrijver (1981 b)).

One of the outstanding problems in mathematics was the so-called 4-color conjecture which states that a geographical map can be colored with four colors such that no two neigbouring countries have the same color. This problem was solved by Appel and Haken (1976). Formulated in terms of graph theory the 4-color problem says that the nodes of a planar graph can be colored with four colors such that no two nodes with the same color are adjacent.

3.19. Coloring problems. More generally we can ask, given a graph G, what is the minimum number of colors needed to color all nodes such that no two adjacent nodes have the same color?

Note that in every node coloring of a graph the nodes with the same color form a stable set. Suppose that for every node $v \in V$ a positive integer c_v is given. Then the *weighted node coloring problem* can be stated as follows: Find stable sets $S_1, S_2, \ldots, S_t$ in G and positive integers $\lambda_1, \lambda_2, \ldots, \lambda_t$ such that

$$\sum_{v \in S_l} \lambda_l \geqslant c_v \quad \text{for every node } v \in V$$

and such that

$$\sum_{l=1}^{t} \lambda_l$$

is as small as possible.

The general (weighted and nonweighted) node coloring problem is hard, but for some classes of graphs the coloring problem can be solved in polynomial time, e.g. perfect graphs (cf. Grötschel, Lovász and Schrijver (1981 b)). Applications of the coloring problem include loading problems, the scheduling of examination timetables and some resource allocation problems, cf. Christofides (1975), Chapter 5.

3.20. The acyclic subdigraph problem. The *acyclic subdigraph problem* can be stated as follows: Given a digraph $D = (V, A)$ with a nonnegative weight function $c: A \to \mathbb{R}_+$, find an acyclic subdigraph (i.e. a set of arcs which contains no directed cycle) of maximum total weight. Expressed

equivalently, we want to find a linear order of the nodes of D such that the sum of the arc weights which are consistent with this order is as large as possible.

The acyclic subdigraph problem is also known by several other names, e.g. the *triangulation problem* or the *feedback arc set problem*. The triangulation problem occurs in input–output analysis and is usually stated in the following form: Given an (n, n)-matrix A, find a simultaneous permutation of the rows and colums of A such that the sum of the elements above the main diagonal of the permuted matrix is as large as possible. This problem is clearly an acyclic subdigraph problem for a complete digraph where the arcs (i, j) carry the weights a_{ij}. There exists a large body of literature on the acyclic subdigraph problem in various nonmathematical fields. Some applications are ranking by paired comparison, Slater (1961), aggregation of individual preferences, determination of ancestry relationship, Glover et al. (1974), and triangulation of input–output matrices, Korte and Oberhofer (1968). For surveys of various aspects of this problem we refer to Lenstra Jr. (1973) and Kaas (1980).

3.21. Knapsack problems. Suppose n different items (types of scientific equipment) are considered for filling a knapsack (for inclusion on a space vehicle). For every item $j = 1, \ldots, n$, a value $c_j > 0$ and a weight $a_j > 0$ per unit are given. The knapsack (space vehicle) has a total weight limit b. The problem of maximizing the total value of the equipment taken is called the (unbounded or integer) *knapsack problem*. More formally the knapsack problem is

$$\max \sum_{j=1}^{n} c_j x_j, \qquad \sum_{j=1}^{n} a_j x_j \leqslant b,$$

$$x_j \geqslant 0 \text{ and integer } j = 1, 2, \ldots, n,$$

where x_j represents the number of times item j is included.

If we additionally require $x_j \in \{0, 1\}$, i.e. that at most one piece of item j is taken, the problem is called the 0/1 *or binary knapsack problem*.

The special case of the binary knapsack problem in which $c_j = a_j$ for $j = 1, \ldots, n$ holds is called the *subset sum problem*.

If $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and A is an (m, n)-matrix such that all entries of c, b and A are nonnegative, then the problem

$$\max c^{\mathrm{T}} x, \qquad Ax \leqslant b, \quad x \geqslant 0, \; x \in \mathbb{Z}^n$$

is often referred to as the *multi-dimensional knapsack problem*, since the objective function plus each one of the constraints $A_i x \leqslant b_i$, $x \geqslant 0$, $x \in \mathbf{Z}^n$ define an (one-dimensional) knapsack problem.

Although these problems look rather simple (in particular the subset sum problem), they are hard and as difficult to solve optimally as general routing problems or the like.

3.22. Planar embedding of a graph. Any graph can clearly be drawn in the plane by representing every node as a point and every edge by a line connecting the two points which represent its endnodes. A graph which can be drawn in the plane in such a way that no two edges (i.e. the corresponding lines) cross is called *planar*. A problem often occuring in designing an integrated circuit or a printed-circuit board is the following: Is a given graph planar, or, what is the minimum number of edges whose removal results in a planar graph?

Hopcroft and Tarjan (1973) developed an algorithm with which planarity of a graph $G = [V, E]$ can be checked in $O(|V|)$ steps. However, Yannakakis (1979) showed that the problem of finding the minimum number of edges F such that $G \setminus F$ is planar is hard.

The problem 3.22 described above gives rise to more general questions about node and edge removal to find subgraphs with certain properties. Examples of such problems are the following:

3.23. Node and edge deletion problems. Suppose π is a property of a graph (or of a digraph), (e.g. π could be one of the following properties: planar, bipartite, acyclic, hamiltonian etc.). Given a graph G, find the minimum number of nodes (or edges) whose deletion results in a subgraph having property π.

Lewis and Yannakakis (1980) have shown that the node deletion problem is hard for the class of properties π which are hereditary and nontrivial. Here a property π is called *nontrivial*, if π is true for infinitely many graphs and false for infinitely many graphs, and π is called *hereditary* if in any graph having property π all node induced subgraphs also have property π. It follows from this result that finding the minimum number of nodes to be deleted such that the resulting graph is e.g. planar, outerplanar, bipartite, acyclic, degree-constrained, chordal, complete, or without edges is hard.

With respect to edge deletion, Yannakakis (1979) proved that for a graph G finding the minimum number of edges F such that $G \setminus F$ has property π is hard for the following properties π: without cycles of specified length l, without any cycle of length $\leqslant l$, connected and degree-constrained, planar, outerplanar, bipartite, transitive (and some others).

These results of course imply that the weighted versions of the problems listed are hard. These include e.g., the so-called *max cut problem*, where a graph $G = [V, E]$ and an objective function $c: E \to \mathbb{R}_+$ are given, and we seek a cut $\omega(W)$ such that $\sum_{e \in \omega(W)} c_e$ is as large as possible. This follows from the above results since finding a maximum weighted cut is obviously equivalent to finding a set of edges whose deletion results in a bipartite subgraph such that the sum of the weights of these edges is as small as possible. (Note that finding a minimal weighted (nonempty) cut is easy, since it can be solved by polynomial network flow algorithms).

The list of hard problems (3.1, 3.12–3.23) and the other hard ones mentioned is just a small collection of such problems. We have introduced some of the most attractive problems but there are hundreds of other combinatorial optimization problems which are also hard and for which we shall likely never find efficient algorithms. An extensive list of hard problems is compiled in Garey and Johnson (1979), but after publication of this book many more problems have been shown to be hard.

4. Heuristic algorithms

Due to the results of complexity theory it seems very unlikely that exact algorithms can be designed which are able to solve all large-scale hard problems with moderate computational effort. Unfortunalely, most combinatorial optimization problems arising from real-world applications are both large and hard, therefore methods have to be found that quickly produce feasible solutions which are (in a sense to be made precise) reasonable.

Heuristic methods are widely used by practitioners with (more or less) satisfactory success. For many years the judgement of the quality and effectiveness of heuristic methods was largely based on empirical computational experience. That is, some test runs on 'representative' real world and some 'representative' randomly generated problem instances were performed and the method which yielded the 'best results on the average' was chosen to be used.

In recent years researchers have started to investigate heuristic methods more seriously. Increased research efforts have led to a more general treatment of the design of heuristics, to better algorithms for various hard problems, and to more sophisticated methods to judge the performance or the quality of heuristic algorithms. Two new tools for evaluating heuristics—'worst case analysis' and 'probabilistic analysis'—are now available and will be discussed in the sequel. Similarly, it has turned out that there are only a few principles which are the basic essentials of most heuristic algorithms. The best known of these are the greedy and interchange techniques which will be studied in the next section.

4.1. Some principles of design for heuristics

Without doubt the most frequently used principles in designing heuristics are the greedy method (cf. 3.3) and its variants. We have mentioned that all matroid problems (3.2) can be solved with the greedy algorithm, but of course, the greedy algorithm can be applied to any maximization problem over an independence system (3.1). For general independence systems, however, there is no guarantee that the greedy algorithm produces an optimal solution. Consider the following example:

4.1. *The greedy algorithm for the stable set problem.* Suppose a graph $G = [V, E]$ of order n and a weight function $c: V \to \mathbb{R}$ are given, and we want to find a stable node set such that the sum of the node weights is as large as possible, (cf. 3.18). The greedy algorithm for this problem works as follows:

1. Order the positive node weights such that
$$c(v_1) \geqslant c(v_2) \geqslant \cdots \geqslant c(v_k) \geqslant 0.$$
2. Set $W = \emptyset$.
3. DO $i = 1$ TO k;
IF $W \cup \{v_i\}$ is a stable node set, THEN add v_i to W. END

In this case the greedy algorithm may behave very poorly, as the following graph shows. Let $G = [V, E]$ be a star, i.e. a graph such that node 1 is linked to all other nodes by an edge and such that there are no other edges. Clearly, the nodes 2, 3, ..., n form a stable set. If the node weights are all equal to one, then the greedy algorithm might have chosen node 1 first, not being able, then, to add any further node. Thus the greedy algo-

rithm produces a solution of total weight 1, while the optimum solution has weight $n - 1$.

Clearly, the greedy algorithm can also be formulated for minimization problems. For the travelling salesman problem this can be done as follows.

4.2. *The greedy algorithm for the symmetric travelling salesman problem.* Let K_n be the complete graph of order n and denote by c_{ij} the distance between city i and city j.

1. Order the intercity distances in a list which monotonically increases.
2. Let $[V, T]$ be the graph with n nodes and no edge, i.e. $T = \emptyset$.
3. Pick the smallest current distance, say c_{ij}. If the graph $[V, T \cup \{ij\}]$ has no node of degree three and contains no cycle of length less than n, then add ij to T.
4. Remove distance c_{ij} from the list. If the list is empty, STOP, otherwise GOTO 3.

The above given two examples are straightforward applications of the greedy technique. There are various other modifications of this method. We shall outline some of these variations with respect to the travelling salesman problem. In principle all these algorithms work as follows:

1. Start with a basic structure (this is often not a feasible solution and could be a node, the empty set, a short cycle or the like).
2. Enlarge the present structure using a 'myopic' optimization rule, i.e. a rule which takes a locally best solution (but which does not guarantee global optimality) and which is easy to implement.

For the symmetric (and similarly for the asymmetric) TSP the following well-known heuristic procedures can be considered as applications of this method.

4.3. *More TSP-heuristics.*

(a) *Nearest neighbour.*

1. Start with any node.
2. Choose a node not yet chosen but closest to the last chosen node, link it to the last chosen node by an edge and continue.

This method clearly produces a hamiltonian path which we can transform into a tour by joining the nodes chosen first and last.

(b) *Nearest insertion.*

1. Start with a cycle $[i_1, i_2, i_3]$ of lenght three.

2. Find a node k not in the present cycle which is closest to any node in the cycle.

3. Find an edge, say $i_s i_{s+1}$, of the cycle such that

$$c_{i_s k} + c_{k i_{s+1}} - c_{i_s i_{s+1}}$$

is minimal.

4. Insert node k between i_s and i_{s+1}. If the present cycle is hamiltonian, STOP, otherwise GOTO 2.

(c) *Farthest insertion.*

The same as nearest insertion, except for Step 2 where 'closest to' is replaced by 'farthest from'.

(d) *Cheapest insertion.*

The same as nearest insertion, except that Step 2 and Step 3 are replaced by

2'. Find a node k not in the present cycle and an edge $i_s i_{s+1}$ of the present cycle such that

$$c_{i_s k} + c_{k i_{s+1}} - c_{i_s i_{s+1}}$$

is minimal.

The greedy technique as described above usually halts when a feasible solution or feasible solution that cannot be enlarged is found (e.g. a tour in the case of the TSP, a basis of the ground set in an independence system). Therefore, such methods are also often called feasible solution construction (tour construction) methods. A straightforward idea is to continue this process by manipulating the present solution in order to find a better one. The most commonly used improvement technique is the so called *interchange method* which in principle works as follows:

Given a feasible solution, say T.

1. Remove some elements from the present feasible solution T to obtain a (not necessarily feasible) solution T'.

2. Construct all feasible solutions containing T'.

3. Choose the best one of these and go to 1.

An application of this technique for the travelling salesman problem is the following method, cf. Lin and Kernighan (1973).

4.4. *k-interchange heuristic for the STSP.* Given a symmetric travelling salesman problem.

1. Construct a tour, say T, e.g. by one of the methods of 4.3.
2. Let L be the list of all k-element subsets of the edge set of T.
3. If L is empty, STOP.
4. Choose a k-element subset from L, say $K = \{e_1, \ldots, e_k\}$. Remove all edges e_i, $i = 1, \ldots, k$, from T to obtain an edge set T'. Construct all possible tours that contain T'. Let S be the shortest tour obtained this way.
5. If T is not longer than S, then remove K from L and GOTO 3.
6. If S is shorter than T, set $T := S$ and GOTO 2.

It should be noted that the amount of computation needed for 4.4 for large k is very great. It is not known whether the worst-case running time of 4.4, even for $k = 2$, is polynomial in n. Even checking that a given tour cannot be improved by a 3-interchange (such a tour is called 3-optimal) is computationally quite expensive, namely $O(n^3)$. Nevertheless, practical experience has shown that, among the computational feasible heuristics, combinations of the 2- and 3-interchange method are the best available heuristics for the STSP to date.

An interchange heuristic for the uncapacitated location problem. cf. 3.16, is the following:

4.5. *Interchange heuristic for the uncapacitated location problem.* Given an uncapacitated location problem, cf. 3.16.

1. Given a feasible solution, say $S \subseteq N$, of the uncapacitated location problem consisting of k locations (e.g. obtained by the greedy method). Let all locations of S be unlabeled.
2. If all locations of S are labeled, STOP.
3. Pick any unlabeled location $j \in S$ and iteratively replace it by each of the unused locations $N \setminus S$.
4. If none of these new solutions is better than S, label the present location j and GOTO 2.
5. Otherwise take the best new solution (or the first one better than S), say S', set $S := S'$, remove all labels and GOTO 3.

Most of the known heuristics for combinatorial optimization problems have one of the two (or both) principles described above as basic ingredients. Usually a greedy technique is used to obtain a starting solution for an

interchange heuristic which then often improves the greedy solution considerably.

It is hard to judge which of the variants of the greedy or interchange heuristics is the best in practice. Our own computational experience for the TSP, for instance, shows that among the four TSP-heuristics described in 4.3 the farthest intertion method 4.3 (c) is by far the best (see also Golden et al. (1980)). All these, however, are outperformed by the 2- or 3-interchange heuristic defined in 4.4. Although the second statement might have been expected, it is hard to find a convincing reason for the (empirical) fact that the farthest insertion method is better than, say, the cheapest insertion algorithm.

4.2. Worst-case analysis

It is somewhat unsatisfactory to have to rely completely on data of computational experiments to judge the quality of a heuristic. To measure the performance of a heuristic algorithm several indicators have been introduced. One such index is the so-called worst-case bound. The worst-case bound (if it exists) gives the maximum relative error that a heuristic algorithm can make given any instance of a particular class of problems. More precisely, we define an ε-approximate algorithm as follows:

4.6. Definition. Let $\mathscr{P}$ be a combinatorial optimization problem, let $\varepsilon > 0$ be a fixed constant, and let H be an algorithm that generates a feasible solution for every instance of $\mathscr{P}$. For every instance $(E, \mathscr{I}, c)$ of $\mathscr{P}$ we denote by $c(I_{\text{opt}})$ the value of the optimal solution and by $c(I_H)$ the value of the solution I_H generated by H. We furthermore assume that $c(I_{\text{opt}}) > 0$. We say that H is an *ε-approximate algorithm* if

$$\frac{|c(I_H) - c(I_{\text{opt}})|}{c(I_{\text{opt}})} \leqslant \varepsilon.$$

To make the definition meaningful we require for maximization problems that ε be less than one. If H is an ε-approximate algorithm for a maximization problem we have

$$\frac{c(I_H)}{c(I_{\text{opt}})} \geqslant 1 - \varepsilon,$$

and if H is an ε-approximate algorithm for a minimization problem, Definition 4.6 implies

$$\frac{c(I_H)}{c(I_{\text{opt}})} \leqslant 1 + \varepsilon.$$

Usually the numbers $1 - \varepsilon$ (resp. $1 + \varepsilon$) are called the *worst-case bounds* of an algorithm for a maximization (resp. minimization) problem.

Thus if $\mathscr{P}$ is a maximization problem and H a 1/5-approximate algorithm we know that every solution generated by H is at least 4/5 as good as the optimal solution, i.e. in the worst case the solution I_H of algorithm H is 20% off optimality, or expressed more positively and less quantitatively, we have a guarantee that the solution given by H is not too bad.

In the recent years intensive studies have been carried out to show that certain well-known heuristics have a certain worst-case bound, to design algorithms with increasingly better bounds, or to show that for a certain problem there is no polynomial algorithm at all with any worst-case bound, unless P = NP.

One of the nicest results in this area is the worst-case bound of the greedy algorithm for general independence systems, (cf. Jenkyns (1976), Korte and Hausmann (1978)).

4.7. Theorem. *Let $(E, \mathscr{I})$ be any independence system and $c: E \to \mathbb{R}_+$ be an objective function. For any $F \subseteq E$ denote by $r_*(F)$ the minimum cardinality of a basis of F and by $r^*(F)$ the maximum cardinality of a basis of F. Let I_G be a greedy solution and I_{opt} an optimal solution of the problem* $\max\{c(I) \mid I \in \mathscr{I}\}$, *then the following holds*

$$q_{\mathscr{I}} := \min_{F \subseteq E} \frac{r_*(F)}{r^*(F)} \leqslant \frac{c(I_G)}{c(I_{\text{opt}})} \leqslant 1.$$

In other words, the greedy algorithm is a $(1 - q_{\mathscr{I}})$-approximate algorithm for any maximization problem over an independence system $(E, \mathscr{I})$. Since by axiom (1.3) of 3.2 for matroids $r_*(F) = r^*(F)$ holds for all $F \subseteq E$, Theorem 4.7 also proves the optimality of the greedy algorithm for matroids.

For particular combinatorial optimization problems it is easy to obtain bounds for this so-called *rank-quotient* $q_{\mathscr{I}}$. For example, for the stable set problem on a graph G with n nodes we have $q_{\mathscr{I}} \geqslant 1/(n-1)$. So the star mentioned after 4.1 gives a worst-case example for the stable set greedy

algorithm, that is, any other possible greedy solution for any other graph of order n is at least as good as $1/(n-1)$ times the optimal value. For further examples, cf. Korte and Hausmann (1978).

Cornuejols, Fisher and Nemhauser (1977) have shown that the greedy algorithm for the uncapacitated location problem 3.16 is an 1/e-approximate algorithm, i.e. the error of the greedy solution is not more than 37%. They have also analysed the interchange heuristic 4.5 which turns out to be a $(k-1)/(2k-1)$-approximate algorithm. These results were generalized to the problem of maximizing a submodular set function in Nemhauser, Wolsey and Fisher (1978) and Fisher, Nemhauser and Wolsey (1978). Similar worst-case bounds for the greedy and interchange heuristic were obtained.

The present situation of the triangulation problem (acyclic subdigraph problem) (3.20) is quite strange. Korte and Hausmann (1978) have shown that the greedy algorithm can be arbitrarily bad, i.e. for the greedy algorithm there is no constant ε (independent of n) which applies to all triangulation problems. This means that for problem instances of increasing size the possible relative error becomes larger and larger. There is, however, an absolutely trivial heuristic which guarantees at least one half of the optimum. Pick any feasible solution, i.e. linear order of the nodes of the digraph, and consider the converse linear order. Since the sum of the values of these two solutions is the sum of all arc weights and since this is an upper bound for the optimum, the better of the two solutions is at least half as good as the optimal one. It is interesting to note that no polynomial heuristic algorithm is known to date which has a better performance guarantee than this.

The situation of worst-case analysis of minimization problems is somewhat different from that of maximization problems. For a given maximization problem and a given heuristic one can usually obtain a worst-case bound independent of the objective function but often dependent upon the size of the instance (e.g. the order of a graph, the number of rows of a matrix, or cf. Theorem 4.7). For minimization problems such objective function independent bounds frequently do not exist at all. A typical negative result of this kind is the following:

4.8. Theorem. *For any $\varepsilon > 0$, there exists a polynomial ε-approximate algorithm for the symmetric travelling salesman problem if and only if* P = NP.

Theorem 4.8 states that finding a polynomial ε-approximate algorithm (even for $\varepsilon = 10^{100}$ or larger) is as hard as finding a polynomial exact optimization method for the STSP. This result implies that none of the algorithms for the STSP described in 4.2, 4.3 and 4.4 has a performance guarantee.

Such negative results raise the question whether there are special cases of the problem considered for which a worst-case bound can be derived. This is indeed sometimes the case. For example, for the STSP a special case (which is the usual one occurring in practice) is the so-called *euclidean symmetric travelling salesman problem.* Here we require that the *triangle inequality* holds for all distances, i.e., for all three different cities $i, j, k \in \{1, 2, \ldots, n\}$ the inequality

$$c_{ik} \leqslant c_{ij} + c_{jk}$$

has to be satisfied. This assumption simply says that the direct trip from city i to city k is not longer than the trip from i to j and then from j to k. For the heuristic algorithms defined in 4.3 Rosenkrantz, Stearns and Lewis (1977) have found worst-case bounds; the performance guarantee of the greedy algorithm (4.2) is due to Frieze (1979).

4.9. Theorem. *Applied to euclidean symmetric travelling salesman problems the algorithms described in* 4.2 *and* 4.3 *achieve the following worst-case bounds:*

(a) $$\frac{\text{length of greedy tour}}{\text{length of optimal tour}} \leqslant \frac{137}{60} + \left\lceil \frac{\log_2((n-8)/2)}{5\log_2(5/4)} \right\rceil,$$

(b) $$\frac{\text{length of nearest neighbour tour}}{\text{length of optimal tour}} \leqslant \tfrac{1}{2}[\log_2(n)] + \tfrac{1}{2},$$

(c) $$\frac{\text{length of nearest insertion tour}}{\text{length of optimal tour}} \leqslant 2,$$

(d) $$\frac{\text{length of farthest insertion tour}}{\text{length of optimal tour}} \leqslant 2\log_2(n) + 0.16,$$

(e) $$\frac{\text{length of cheapest insertion tour}}{\text{length of optimal tour}} \leqslant 2.$$

Judged from the worst-case bound the farthest insertion method is the worst of the five methods, although in practical applications this method

turns out to be the best of these five. For the k-interchange methods no worst-case bounds are known. However, Rosenkrantz, Stearns and Lewis (1977) have shown that there are instances of n-city euclidean STSP's for which

$$\frac{\text{length of } k\text{-interchange tour}}{\text{length of optimal tour}} = 2\left(1 - \frac{1}{n}\right).$$

For surveys of further results, complexity calculations of heuristic algorithms for the STSP, and the judgement of their practical performance we refer to Golden et al. (1980) and Lenstra and Rinnooy Kan (1979).

The worst-case bounds in Theorem 4.9 immediately lead one to ask whether the number two is the best possible bound, or can algorithms be invented with a better performance guarantee. This bound can in fact be improved with a nice algorithm designed by Christofides (1976).

4.10. *Christofides heuristic for the STSP.* Given an n-city euclidean STSP with distances c_{ij}, $1 \leqslant i < j \leqslant n$.

Step 1. Calculate a shortest spanning tree S of the complete graph K_n.

Step 2. Construct the complete subgraph G of K_n induced by the nodes having an odd degree with respect to S. (Note that G has an even number of nodes.)

Step 3. Calculate a minimum perfect 1-matching M in G.

Step 4. Construct the graph G' on n nodes which consists of all edges of S and all edges of M. (If an edge is in S and M, it appears twice in G'. By construction, H is connected and every node in G' has even degree, thus by Euler's theorem, G' is eulerian.)

Step 5. Construct an eulerian tour T in G'.

Step 6. Suppose $T = v_1, v_2, v_3, \ldots, v_k, v_1$ is the eulerian tour in G'. We construct a hamiltonian tour H from T as follows: Start in v_1, go to v_2, to v_3 etc. If a node, say v_i, is encountered which has already been visited, then eliminate v_i from T and continue until the last node v_1 is reached.

Since a hamiltonian path is a spanning tree of K_n, the shortest spanning tree is not longer than the minimum length tour. Furthermore, given the optimal tour in K_n, we convert it into a tour of G by skipping over all nodes not in G. Because of the triangle inequality this tour in G is not longer than the one in K_n. Now the tour in G provides us with two perfect 1-matchings in G by taking every other edge. The smaller of these two 1-matchings

is not longer than half of the length of the optimal tour. Thus, the same holds for the minimal perfect 1-matching in G. This implies that the eulerian tour is at most 50% longer than the optimal hamiltonian tour. Since the hamiltonian tour of Christofides' algorithm is constructed by taking shortcuts (these *are* really shortcuts since the triangle inequality holds) we obtain:

4.11. Theorem. *For every euclidean symmetric travelling salesman problem the following holds:*

$$\frac{\text{length of Christofides tour}}{\text{length of optimal tour}} < 1.5.$$

To date no polynomial time heuristic for the euclidean STSP with better worst-case bound is known.

Similar studies have been made with respect to many other hard combinatorial optimization problems, general surveys of some of the results obtained so far are given e.g. in Kannan and Korte (1978), Korte (1979), and in Fisher (1980). The case of scheduling problems is treated in Graham, Lawler, Lenstra and Rinnooy Kan (1979).

4.3. Approximation schemes

Search for algorithms with increasingly better worst-case bounds immediately leads to the problem of whether it is possible to design polynomial ε-approximate algorithms, (cf. Definition 4.6), for every $\varepsilon > 0$. Therefore the following concepts have been introduced.

4.12. Definition. Given a combinatorial optimization problem $\mathcal{P}$. An *approximation scheme* (AS) for $\mathcal{P}$ is an algorithm which, given any instance $(E, \mathcal{F}, c)$ of $\mathcal{P}$ and a desired degree of accuracy $\varepsilon > 0$, computes a solution with value $c(I_{AS})$ such that, if $c(I_{opt}) > 0$ is the value of the optimal solution of $(E, \mathcal{F}, c)$,

$$\frac{|c(I_{AS}) - c(I_{opt})|}{c(I_{opt})} < \varepsilon.$$

Observe the difference between an ε-approximate algorithm and an approximation scheme. An ε-approximate algorithm works for only one ε

(and of course all numbers larger than ε), while an approximation scheme is required to work for all values $\varepsilon > 0$.

4.13. Definition. An approximation scheme is a *polynomial time approximation scheme* (*PAS*), if for every fixed $\varepsilon > 0$ its running time is bounded by a polynomial in the length of the input (the input does not include ε).

4.14. Definition. A *fully polynomial approximation scheme* (*FPAS*) is a PAS whose time complexity function is bounded by a polynomial function of both the length of the input and $1/\varepsilon$.

An ε-approximate algorithm H whose complexity is $O(n^{1/\varepsilon})$, where n is the length of the input, is a PAS but not a FPAS, however, if its time complexity is $O(n^k \varepsilon^{-l})$, where k and l are fixed constants, then it is an FPAS. If the degree of accuracy ε is a rational number p/q then the encoding length of ε is $\lceil \log(p) \rceil + \lceil \log(q) \rceil$. So we might define an even better approximation scheme by requiring that its running time is polynomial in n and $\lceil \log(p) \rceil + \lceil \log(q) \rceil$. But it is easy to show that the existence of such an approximation scheme implies the existence of a polynomial time exact algorithm. Thus, an FPAS is in some sense the best approximate method we can expect for a hard problem.

Unfortunately, there are not too many hard problems for which fully polynomial approximation schemes exist. Most of them are knapsack or knapsack-related problems and some are special scheduling problems. To show the general principles of an FPAS we now describe the algorithm of Ibarra and Kim (1975) for the 0/1-knapsack problem

4.15. Algorithm. *FPAS of Ibarra and Kim.*

Given: positive integers b, a_j and c_j, $j = 1, \ldots, n$, and $\varepsilon > 0$ (we assume $a_j \leqslant b$, $\sum_{j=1}^{n} a_j > b$, $c_j/a_j \geqslant c_{j+1}/a_{j+1}$ w.l.o.g.).

Output: a feasible solution $\bar{x}$ of the problem

$$\max \sum_{j=1}^{n} c_j x_j, \qquad \sum_{j=1}^{n} a_j x_j \leqslant b, \quad x_j \in \{0, 1\},\ j = 1, \ldots, n, \quad \text{(KP)}$$

whose value is at most $(1 - \varepsilon)$ off optimality.

1. Let k be the largest index such that $\sum_{j=1}^{k} a_j \leqslant b$, then set $c_{\text{est}} := \sum_{j=1}^{k+1} c_j$. Note, if c_{opt} is the optimum value of (KP) then

$$c_{\text{opt}} \leqslant c_{\text{est}} \leqslant 2\, c_{\text{opt}}.$$

2. Set $s := (\varepsilon/3)^2 c_{\text{est}}$ and $t := \varepsilon/3\, c_{\text{est}}$ and partition the index set $N := \{1, \ldots, n\}$ into 'small' and 'large' indices $S := \{j \in N \mid c_j < t\}$, $L := \{j \in N \mid c_j \geqslant t\}$.

3. Solve the special type equality 0/1-knapsack problems

$$\min \sum_{j \in L} a_j x_j, \qquad \sum_{j \in L} \lceil c_j/s \rceil x_j = d, \quad x_j \in \{0, 1\},\ j \in L \qquad (\text{EKP}_d)$$

for all $d = 0, 1, \ldots, \lceil c_{\text{est}}/s \rceil$ optimally (using e.g. dynamic programming).

4. For $d = 0, 1, \ldots, \lceil c_{\text{est}}/s \rceil$ do: If x_j^d, $j \in L$, is the optimal solution of (EKP_d) and $\sum_{j \in L} a_j x_j^d \leqslant b$, then apply the greedy algorithm to

$$\max \sum_{j \in S} c_j x_j, \qquad \sum_{j \in S} a_j x_j \leqslant b - \sum_{j \in L} a_j x_j^d, \quad x_j \in \{0, 1\},\ j \in S$$

where the greedy rule is: choose an index j such that the so-called weight density c_j/a_j is as large as possible. Let x_j^d, $j \in S$, be this greedy solution.

5. Among all feasible solutions x_d, $d = 0, \ldots, \lceil c_{\text{est}}/s \rceil$, of (KP) generated in 3. and 4. choose the one with the best value.

This algorithm is a prototype method for all FPAS's. First the variables are partitioned into important and unimportant ones (L and S). Then the knapsack problem is considered for the important variables only. For these variables new knapsack problems are created by a scaling method which has the special feature that the best solution of the scaled problems does not differ too much from the optimal one and that this solution can be computed in time polynomial in n and $1/\varepsilon$. Finally the unimportant variables are filled in by some simple greedy technique.

The $\lceil c_{\text{est}}/s \rceil$ problems (EKP_d) can be easily solved by recursion (dynamic programming) setting $f(\nu, 0) = 0$, $\nu = 1, \ldots, |L|$; $f(1, d) = 0$ if $d = 0$; $f(1, d) = a_1$ if $d = \lceil c_1/s \rceil$, and $f(1, d) = \infty$ otherwise; and by $f(\nu, d) = \min\{f(\nu - 1, d),\ f(\nu - 1, d - \lceil c_\nu/s \rceil) + a_\nu\}$ for $2 \leqslant \nu \leqslant |L|$, $1 \leqslant d \leqslant \lceil c_{\text{est}}/s \rceil$, where $f(\nu - 1, d - \lceil c_\nu/s \rceil) = \infty$ if $d - \lceil c_\nu/s \rceil < 0$. The optimum values of (FKP_d) are given by the numbers $f(|L|, d)$. The running time of this recursive procedure is $O(n\varepsilon^{-2})$ which is also the overall running time of algorithm 4.15.

Korte and Schrader (1980) have shown that a combination of scaling and dynamic programming (called ε-dominance test) like the one presented in 4.15 is the essential ingredient of any FPAS.

However, there are many problems for which FPAS's cannot exist (if P $\neq$ NP). Namely, suppose there were a FPAS for the STSP. Then we

formulate the hamiltonian graph problem (which is NP-complete) as a STSP with $\{1, 2\}$-coefficients by setting $c_{ij} = 1$ if $ij \in E(G)$ and $c_{ij} = 2$ if $ij \notin E(G)$. The graph G is hamiltonian if and only if the minimum of this STSP is equal to n. If G is not hamiltonian, then the minimum is at least $n + 1$.

Now, if we run our assumed FPAS with accuracy $\varepsilon = 1/(n + 1)$ we can decide in time polynomial in n and $1/\varepsilon$ (i.e. in time polynomial in n which is not larger than the input length of this STSP) whether the optimum is less than $n + 1$, i.e. whether G is hamiltonian. Thus, the existence of an FPAS for the STSP implies P = NP. The definition of the c_{ij} in particular implies that the STSP is hard even if we restrict the coefficients of the objective function to be one or two.

This analysis clearly applies to all hard problems. More exactly, let $\mathcal{P}$ be a hard combinatorial optimization problem and $\mathcal{P}_k$ be the subset of problem instances $(E, \mathcal{I}, c)$ of $\mathcal{P}$ such that the numbers $|c_e| \in \mathbb{Z}_+$ are bounded by a fixed positive constant k. If $\mathcal{P}_k$ is also hard, then $\mathcal{P}$ cannot have an FPAS, unless P = NP. This observation was made by Garey and Johnson (1978). This result rules out the existence of an FPAS for the travelling salesman problem, the stable set problem, the acyclic subgraph prolem and the like. On the other hand FPAS's are known e.g. for the subset sum problem (Ibarra and Kim (1975)), the 0/1-knapsack problem (Lawler (1979)), the multiple choice knapsack problem (Chandra, Hirschberg and Wong (1976)), job sequencing with deadlines (Sahni (1976)) and minimizing finish time on two parallel nonidentical machines (Sahni (1976)).

The situation for polynomial approximation schemes is not very different. It can be shown that only for a very limited class of hard problems PAS's are likely to exist, (cf. Garey and Johnson (1979) and Korte and Schrader (1980)). Examples of such hard problems are the multidimensional knapsack problem (Chandra, Hirschberg and Wong (1976)), machine scheduling with certain precedence constraints (Ibarra and Kim (1978)). For the multidimensional knapsack problem one can also show that no FPAS can exist, so the result of Chandra, Hirschberg and Wong is the best possible with respect to our approximation classes.

4.4. Probabilistic analysis of heuristic algorithms

Although the worst-case analysis was (and still is) very successful in many cases by providing approximate algorithms with very good perform-

ance guarantee, it does not seem to be the proper model for evaluating the performance of all algorithms. In particular for problems like the TSP where no polynomial time approximation algorithms with finite worst-case bound exist one has to find other means to judge the quality of a heuristic algorithm.

A promising idea is to consider a probabilistic approach and design algorithms which guarantee optimal or near-optimal solutions on almost all problem instances. One allows the algorithm to fail to produce good solutions in some cases, but such events should be very rare. To formulate mathematically what 'almost all' means one has to introduce a probability distribution over the set of problem instances of each size and then find out how the algorithm performs on the average when problem instances are drawn from this distribution.

An assumption like this immediately leads to the question as to which probability distribution is realistic or whether a chosen distribution actually occurs in practical examples. Nevertheless, this approach leads to interesting insights into the behaviour of algorithms which cannot be obtained by worst-case analysis.

The general approach to probabilistic analysis is the following. Given a problem; e.g. the symmetric travelling salesman problem. Then with each instance I of the problem a size $|I|$ is associated; e.g. the number of cities. Then for each size n one assumes that a certain probability distribution S_n over the problem instances of size n is given; e.g. one considers travelling salesman problems where cities are points in a given square of the plane and where an instance of an n-city problem is chosen by drawing n points independently from a uniform distribution over the square (the 'distance' here is the euclidean distance in the plane). Let $X(I)$ be some predicate which is true or false for each problem instance I; e.g. $X(I)$ is true if a certain algorithm A when applied to the TSP-instance I produces a solution whose value $c_A(I)$ satisfies $c_A(I) \leqslant (1 + \varepsilon)\, c_{\text{opt}}(I)$ for some $\varepsilon > 0$, where $c_{\text{opt}}(I)$ is the optimum value of instance I. Let q_n be the probability that $X(I)$ does not hold when I is drawn from S_n; e.g. that the value $c_A(I)$ is not ε-close to $c_{\text{opt}}(I)$. Then one says that $X(I)$ *holds almost everywhere* if $\sum_{n=1}^{\infty} q_n < \infty$. This condition implies that if an infinite sequence of problems, one of each size, is chosen then with probability one, the predicate $X(\,)$ would be observed to fail only finitely often; e.g. that in the case of the TSP with probability one algorithm A would produce a bad solution only

finitely often. Note that this condition is an asymptotic one, so it does not say anything about a finite sample of 100-city problems.

Although some very interesting results about the probabilistic behaviour of certain algorithms have been obtained, this subject is still not deeply explored. Most of the algorithms analyzed so far are extremely simple and the probability distributions that have been studied may not satisfy practitioners. The mathematical analysis of these models is, however, quite difficult and even proofs of seemingly trivial cases are rather involved.

Probabilistic analysis of heuristic algorithms for the symmetric and asymmetric TSP, the stable set problem, the set covering problem, the coloring problem and some other hard problems has been carried out. A nice survey of such results can be found in Karp (1976). Two particularly interesting applications for the STSP and the ATSP are in Karp (1977) and Karp (1979), where for instance Karp shows that for the STSP defined on the unit square by chosing n points independently from a uniform distribution a very simple partitioning algorithm almost always produces a near-optimum solution.

A further approach is to study the probabilistic behaviour of the optimum solution if the cost function is chosen from some distribution and then design algorithms which make use of this behaviour and almost surely produce a near-optimum solution. Investigations of this kind can be found in Lueker (1978) and Weide (1980).

5. Exact optimization procedures

There is a trivial way to solve a combinatorial optimization problem exactly, namely by enumerating all feasible solutions computing their values and then choosing the best one. This method may be feasible for small instances but it can obviously not be carried out for most of the relevant practical problems. A striking consequence of complexity theory is that, unless P = NP, there will be no algorithm for the exact solution of a hard problem which has a significantly better running time for all problem instances. This negative result, however, should not discourage algorithm designers because it does not exclude the possibility of finding algorithms which perform well on the average or which can solve certain problems of practical interest (even very large ones) with a moderate computational effort.

In principle there are two methods available for solving hard optimization problems. One is the so-called branch and bound technique and the other the cutting plane method. Both approaches can be combined of course and can be enriched by various heuristic principles (so much so, indeed, that sometimes it is not apparent what the main ingredient is).

The guiding idea of the branch and bound technique is to enumerate all possible solutions in an 'intelligent' way. The enumeration is organized in such a way that at every step the universe of feasible solutions is partitioned into disjoint subsets and that at every step lower and/or upper bounds for the best solution within these subsets are computed. If (in case of maximization) this upper bound of a certain subset is smaller than the present best known feasible solution or the best known lower bound we can omit all solutions of this subset from further considerations. Otherwise we split this subset into smaller pieces obtaining a finer partition and continue. This technique may save a tremendous number of computations, but it is not guaranteed to terminate without having looked at all possible solutions.

It is apparant that the relative success of a branch and bound method heavily depends on the partitioning strategy and the quality of the bounds that are computed. The determination of bounds is the most important feature of such algorithms and a large part of the research effort in past years has gone into finding methods for computing good bounds with reasonable computational effort.

The principle behind 'cutting plane methods' is to use linear programming to solve combinatorial optimization problems. This is done by associating a linear programming problem with the combinatorial optimization problem under consideration which has the property that the integral feasible solutions of the linear program are exactly the feasible solutions of the combinatorial problem. In other words, we want to find an integer programming formulation for a combinatorial optimization problem. Given such a linear program then the LP is solved, e.g. by the simplex method (if this is possible). If the LP-solution is integral the optimum solution of the combinatorial problem is found. If the LP-solution is not integral one has to find an inequality (cutting plane) which is violated by the current LP-solution but satisfied by all integral solutions. This inequality is added to the current system of equations and inequalities and the procedure is continued.

Clearly, the problem here is to find 'suitable' integer programming formulations for the hard combinatorial optimization problems and to find efficient ways to generate 'good' cutting planes. The search for good cutting planes has created a very lively area of research going under the name polyhedral combinatorics, and which will be introduced in Section 8.2.

6. Branch and bound methods

Methods which skillfully enumerate the feasible solutions of a combinatorial optimization problem have been invented and reinvented many times. It is therefore not surprising that various names are used for these techniques and that no common terminology exists. Some of the names used are implicit or partial enumeration, divide and conquer methods, backtracking techniques, partitioning strategies and so on. By introducing sophisticated distinguishing features one may build up a hierarchy of these methods, but we shall abstain from such subtleties in terminology and call all these techniques branch and bounds methods.

6.1. The general principle

We shall now give a formal definition of this method with respect to a combinatorial optimization problem $\mathcal{P}$. Each instance of $\mathcal{P}$ is given by the triple $(E, \mathcal{F}, c)$, where E is a finite set, $\mathcal{F} \subseteq \mathcal{P}(E)$ is the set of feasible solutions, $c_e \in \mathbb{R}$ for all $e \in E$ and $c(I) := \sum_{e \in I} c_e$ is the value of each set $I \subseteq E$.

We assume that $\mathcal{P}$ is a maximization problem, so we seek

$$\max\{c(I) \mid I \in \mathcal{F}\}.$$

6.1. Relaxation assumption. *We assume that there is a relaxation $R(\mathcal{P})$ of $\mathcal{P}$ which can be solved efficiently. More precisely, we assume that for every instance $(E, \mathcal{F}, c)$ there exists an instance $R(E, \mathcal{F}, c)$ such that every element I of $\mathcal{F}$ corresponds to a feasible solution, say I', of $R(E, \mathcal{F}, c)$ and such that the values of I and I' coincide. Moreover, there should be an algorithm which solves every instance of $R(\mathcal{P})$ efficiently (relative to solving instances of $\mathcal{P}$).*

Since by definition

$$\max\{c(I) \mid I \in \mathcal{I}\} \leqslant \max\{c(x) \mid x \text{ feasible for } R(E, \mathcal{I}, c)\},$$

the optimum value of the relaxed problem gives an upper bound for the original problem.

For every combinatorial optimization problem $\mathcal{P}$ it is easy to find some relaxation $R(\mathcal{P})$. The problem is to find a good one, namely one such that for every instance $(E, \mathcal{I}, c)$ the optimum values of the maximization problem over $(E, \mathcal{I}, c)$ and $R(E, \mathcal{I}, c)$ do not differ 'too much' and that the computational effort for finding the optimum solution of $R(E, \mathcal{I}, c)$ is not 'too large'.

To give an example of possible relaxations consider the asymmetric travelling salesman problem (3.12). A complete digraph $D = (V, A)$ of order n is given with a distance function c_{ij} between every two nodes $i, j \in \{1, \ldots, n\}$, $i \neq j$, and we want to find the shortest hamiltonian dicycle (tour) in D. In the assignment problem on D, cf. 3.7, we have to find a set of arcs such that every node is the head and the tail of exactly one arc. This condition is met by every tour, i.e. every tour is a solution of the assignment problem. This implies that the minimum solution of the assignment problem with distances c_{ij} provides a lower bound for the ATSP. Since assignment problems can be solved efficiently, the assignment problem is a good (and often used) candidate for a relaxation of the ATSP.

A general description of a branch and bound procedure can be formulated as follows.

6.2. *Branch and bound method* (for a combinatorial maximization problem $\mathcal{P}$). Given an instance $(E, \mathcal{I}, c)$ of $\mathcal{P}$, we want to find $I^* \in \mathcal{I}$ such that $c(I^*) = \max\{c(I) \mid I \in \mathcal{I}\}$. We assume that a relaxation $R(\mathcal{P})$ of $\mathcal{I}$ is chosen.

1. Let L be a lower bound for our problem instance. (This may be obtained by a heuristic procedure or just by setting $L = -\infty$.)

2. Set $K = \{\mathcal{I}\}$. (K is the set of candidate sets.)

3. If $K = \emptyset$, STOP. The present best solution is the optimal one (all candidate sets have been explored).

4. *Branching Step.* If $K \neq \emptyset$, choose a 'suitable' element of K, say $S \in K$.

5. *Bounding Step.* (a) Relax the problem instance (S) $\max\{c(I) \mid I \in S\}$ of $\mathcal{P}$ to a 'suitable' problem instance (RS) of $R(\mathcal{P})$.

(b) Find the optimum solution, say x^*, of (RS). Let $c(x^*)$ be the value of x^*.

6. If $c(x^*) \leqslant L$ then discard S from K and GOTO 3. (The best solution in S has a value which is not larger than $c(x^*)$. Since a solution of $(E, \mathscr{F}, c)$ with value L is known, no solution in S can be better than the present best. Therefore we can eliminate all these solutions.)

7. If $c(x^*) > L$ and x^* corresponds to an element, say I^*, of S then set $L := c(x^*)$, replace the present best solution by I^* and discard S from K. GOTO 3. (The optimum solution of (RS) is a feasible solution of $(E, \mathscr{F}, c)$, so the optimum solution of (S) is found, and we do not have to consider the elements of S any more. Here in addition we can remove all elements from the candidate list for which an upper bound is known which is not larger than L.)

8. *Separation.* (We have $c(x^*) > L$ and x^* does not correspond to an element of S. In this case we have to split S into smaller pieces.) Partition S into nonempty subsets $S_1, \ldots, S_k$, discard S from K, add the sets $S_1, \ldots, S_k$ to K, and GOTO 3. (The partitioning should be done in such a way that book-keeping is not too difficult.)

The implementation of such a procedure is clearly nontrivial since —apart from designing an efficient code for the relaxed problem—it requires a lot of complicated book-keeping, e.g. the implicit representation of the current candidate sets, and the testing of many heuristic rules to find workably efficient strategies for selecting 'promising' candidate sets and suitable partitionings of candidate sets. Although the principle of a branch and bound procedure is a triviality the actual design and encoding of a sufficiently successful method is quite difficult.

6.2. A branch and bound algorithm for the STSP

We shall now give an example of a branch and bound method with respect to the symmetric travelling salesman problem. The most important step in the design of such a procedure is the choice of the relaxation. For our example we shall try the so-called 1-tree relaxation.

6.3. Definition. Given the complete graph $K_n = [V, E]$ with node set $\{1, \ldots, n\}$, and let K_{n-1} be the complete graph with node set $\{2, \ldots, n\}$.

Then a *1-tree* in K_n is an edge set which consists of a spanning tree of the graph K_{n-1} together with two distinct edges incident with node 1.

By definition, a spanning tree T' of K_{n-1} consists of $n-2$ edges, contains no cycle, and between every two nodes $u, v \in \{2, \ldots, n\}$ there is a unique path joining u and v. Thus, if we add two edges to T' which are incident with node 1 we obtain a set of n edges which contains exactly one cycle. A tour has also n edges and is a cycle, i.e., every tour is a 1-tree. The problem of finding a minimum length 1-tree is therefore a relaxation of the STSP.

Moreover, it is easy to find a minimum 1-tree. Namely, we first find a minimum spanning tree of K_{n-1} and then add the two shortest edges incident with node 1. There are various fast methods known with which minimum spanning trees can be obtained, e.g. the greedy algorithm or the Prim–Dijkstra method, cf. Kruskal (1956), Dijkstra (1959), Prim (1957). Thus the relaxed problem instances can be solved easily.

We now have to devise a separation and branching scheme and to define our candidate sets. Every candidate set S_i is described by a set R_i of edges which are contained in all tours of S_i and by a set F_i of edges such that no tour in S_i contains an edge of F_i. If S_i is split into several pieces S_j we have to make sure that the new edge sets R_j and F_j are defined in such a way that we obtain a partition of S_i. Initially, S_0 is the set of all tours in K_n and $R_0 = F_0 = \emptyset$.

Suppose in the branching step 4 of 6.2 we have chosen a candidate set S described by the edge sets R and F. Then we solve the following restricted 1-tree problem

$$\min\{c(T) \mid T \text{ is 1-tree in } K_n \text{ and } R \subseteq T,\ T \cap F = \emptyset\}.$$

If the optimum solution is a tour or the minimum is larger than the shortest known tour then S is discarded forever. If, however, the minimum is a nontour T^* whose value is smaller than the current upper bound, then we have to split S into several pieces. For this procedure we use the following rule, (cf. Volgenant and Jonker (1980)). Choose a node, say v, whose degree in T^* is larger than two and which is contained in the unique cycle in T^*. Clearly, such a node always exists. Note also that v is incident with at least two edges, say e_1 and e_2, which are not contained in R. Namely, if v were incident with two required edges then clearly all other edges incident with v were forbidden; and so v could not have a degree larger than two.

We define a partition of S by setting

$$\begin{aligned} R_1 &:= R \cup \{e_1, e_2\}, & F_1 &:= F, \\ R_2 &:= R \cup \{e_1\}, & F_2 &:= F \cup \{e_2\}, \\ R_3 &:= R, & F_3 &:= F \cup \{e_1\}, \end{aligned} \tag{6.4}$$

which clearly results in candidate sets S_1, S_2, S_3 which are disjoint and whose union is S. S is discarded from the candidate list, but before S_1, S_2, S_3 are added we extend the sets of required and forbidden edges. Note that in the case where v was on a required edge of R, R_1 then states that v must be contained in three edges which is impossible. So in this case S_1 is the empty set and is not added. Moreover, if in the sets of required edges R_1 or R_2 one node is now incident with two required edges then all other edges incident with this node can be forbidden. If the edge sets F_2 or F_3 contain $n - 2$ edges incident with one node then the two remaining edges incident with this node are required. So in our separation procedure we discard S from the candidate list and add two (S_2, S_3) or three (S_1, S_2, S_3) new candidate sets to our list. We still have to state which node v we use to perform our separation and how we branch. This will be done in the following complete description of the algorithm.

6.5. Algorithm. *Branch and bound algorithm for the STSP with* 1*-tree relaxation.* Suppose we have a complete graph K_n with distances $c_{ij} \geqslant 0$, $1 \leqslant i < j \leqslant n$. We describe a candidate set S by $S(R, F, L)$ where R are the required, and F the forbidden edges, and L is a known lower bound for the best tour in S.

1. Run the farthest insertion and the Christofides heuristic described in 4.3 (c) (resp. 4.10) to obtain tours T_1 and T_2. Then apply the 2-interchange, and then the 3-interchange method described in 4.4 to T_1 and T_2. Let T be the best tour obtained in this way and let U be its value. (This choice of heuristics is based on our computational experience with many travelling salesman problems.)

2. Let the list of candidate sets consist of the set $S(\emptyset, \emptyset, 0)$ consisting of all tours. (The shortest tour clearly has length at least zero.)

3. If the list of candidate sets is empty, STOP. (The present best tour T is the optimal one.)

4. *Branching step.* We suggest two versions (a) and (a'), a depth first search and a branching on the smallest lower bound.

(a) *Depth-first-search:* Choose the candidate set $S(R, F, L)$ which among the ones on the list is the last one that has been added to the list. (This method has the advantage that we quickly get tours and that our candidate list does not grow too fast (hopefully).)

(a') Among the candidate sets which have the smallest present lower bound choose the set $S(R, F, L)$ which was added as the last one to the list. (Here one hopes that the optimum tour is found quickly, so that the gap between the current upper and lower bounds is not too large.)

(b) Remove $S(R, F, L)$ from the candidate list.

5. *Bounding step.* Find the minimum 1-tree T^* of the problem

$$\min\{c(T_1) \mid T_1 \text{ is a 1-tree in } K_n \text{ with } R \subseteq T_1,\ T_1 \cap F = \emptyset\}.$$

(This can be easily solved with the greedy algorithm or the Prim–Dijkstra method.)

6. If $c(T^*) \geqslant U$, GOTO 3. ($S(R, F, L)$ does not contain a better tour than the present one.)

7. If $c(T^*) < U$ and T^* is a tour, then set $T := T^*$ and $U := c(T^*)$. Remove all sets $S'(R', F', L')$ from the candidate list with $L' \geqslant U$. GOTO 3. ($S(R, F, L)$ has been completely explored and a new current best tour was found.)

8. *Separation.* (In this case T^* is not a tour and $c(T^*) < U$.) (a) Find the unique cycle in T^*, say C, and let W be the set of nodes on C with degree larger than two.

(b) Among the nodes of W choose one which is contained in a required edge of R. If there is none, choose any node of W. (The reason for the rule in (b)) is to split S into two rather than three subsets. This rule can of course be defined in a more sophisticated way.) Call the chosen node v.

(c) Choose two edges of T^* incident with v and not in R. If there is a choice, take edge e_1 to be an edge not in C and e_2 to be an edge in C.

(d) Define the partions $S_1(R_1, F_1, L)$, $S_2(R_2, F_2, L)$, $S_3(R_3, F_3, L)$ respectively $S_2(R_2, F_2, L)$, $S_3(R_3, F_3, L)$ as described in (6.4) where $L := c(T^*)$ and add these to the candidate list. (Note that no tour in the sets S_i can be shorter than L.)

(e) GOTO 3.

A major problem is to find good data structures to handle the candidate list. This is usually done by keeping so-called *enumeration* or *branch and bound trees*. These are trees in which every node corresponds to a candidate

set and which are organized in such a way that all informations about a candidate set can be retrieved easily.

Consider e.g. our algorithm 6.5, where each candidate set S is characterized by two sets R, F and a number L. Suppose we solve a travelling salesman problem, run the heuristic algorithms in step 1, obtain an upper bound $U = 100$ and start the branch and bound procedure with $S(\emptyset, \emptyset, 0)$. We get a 1-tree with length 80 which is not a tour. We choose the edges $e_1 = 23$ and $e_2 = 28$ in step 5 and split $S(\emptyset, \emptyset, 0)$ into three pieces $S_1(\{23, 28\}, \emptyset, 80)$, $S_2(\{23\}, \{28\}, 80)$ and $S_3(\emptyset, \{23\}, 80)$. Now we branch to $S_2(\{23\}, \{28\}, 80)$ and obtain a 1-tree of length 85. We choose the edges $e_1 = 37$, $e_2 = 79$ and split S_2 into

$$S_4(\{23, 37, 79\}, \{28\} \cup (\omega(3) \setminus \{23, 37\}) \cup (\omega(7) \setminus \{37, 79\}), 85),$$

$$S_5(\{23, 37\}, \{28\} \cup (\omega(3) \setminus \{23, 37\}), 85),$$

$$S_6(\{23\}, \{28, 37\}, 85)$$

and continue. We keep track of this process by storing at each node of the enumeration tree the edges that are added to R (resp. to F) and the lower bound calculated at the node. This is represented in Fig. 1.

This tree has a root, namely S_0. Every node corresponds to a candidate set. If a candidate set is chosen in step 4 of 6.5 (say this is S_2 in the above

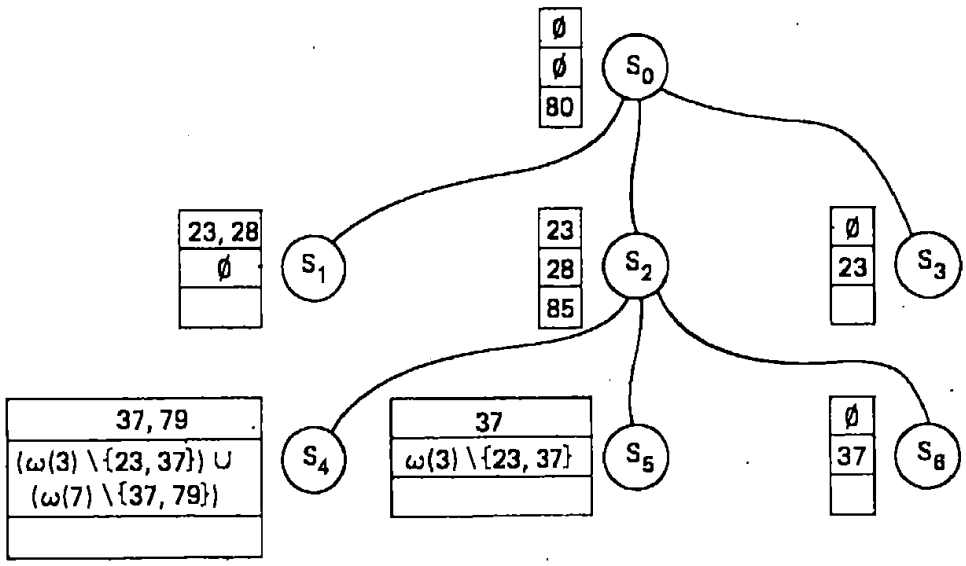

Fig. 1.

case) then the lower bound is calculated in step 5 of 6.5 and the result is recorded at the node (in our example: S_2). This indicates that the set (namely S_2) is now eliminated from our list. If we get to step 8 of 6.5 we split the current set into parts. This is done by creating sons of the present node (in our case the sons are S_4, S_5, S_6) and by also recording at every new node those edges which are now fixed. If we want to continue (with, say, S_5) then we have to go to the father (for S_5, this is S_2), from this to its father and so on until we reach the root S_0 to find out which edges are required and forbidden at the present node. The current lower bound is recorded at the father. If we do not get to step 8 of 6.5, then the tree will not be enlarged any further starting from the present node, and we can remove this node from the tree.

All branch and bound procedures have to keep track of the candidate sets, e.g., in the way described above. Of course, there are many variations, but it should be clear that for each candidate set some information has to be stored. So if the number of candidate sets grows we may run into storage problems. For real-world problems this is quite often the case.

Having coded our algorithm 6.5 we shall find out that all our steps (in particular the important step 5) can be executed very quickly but the branch and bound tree becomes tremendously large. This is due to the fact that the 1-tree relaxation does not give a very good bound for the travelling salesman problem. So we either need to consider a different relaxation or we have to invent a method which improves our 1-tree bound. Such a general method will be described in the next section.

7. Lagrangean relaxation

The method of increasing lower bounds (in the minimization case) or decreasing upper bounds (in the maximization case) we shall now discuss is applicable to any relaxation of a given hard problem. Whether or not it is computationally successful heavily depends on the algorithms available to solve the relaxed problem and its Lagrangean variant. There are various formulations of the Lagrangean approach. We shall describe it in such a way that it is directly applicable to the 1-tree relaxation algorithm for the STSP described in 6.5.

Given a combinatorial optimization problem $\mathcal{P}$, assume that for every instance $(E, \mathcal{F}, c)$ of $\mathcal{P}$ we have an integer programming formulation

which has the following form

$$\begin{aligned} z_{IP} := \min\ & c^T x, \\ & Ax = b, \qquad Dx \leqslant e, \\ & x \geqslant 0 \text{ and } x \text{ integral}, \end{aligned} \tag{IP}$$

where A is an (m, n)-matrix, D is an (k, n)-matrix and the vectors c, b, e are compatible. This means that every element of $\mathcal{S}$ corresponds to an integral solution of (IP) and vice versa.

Usually there are many ways to formulate an integer program corresponding to a combinatorial optimization problem. But it is not often the case that the obvious IP-formulations are the most suitable ones for the approach we are going to describe. To apply the method successfully skillful formulations have to be chosen.

We assume (and this is the most important assumption) that the constraints of (IP) have been partitioned in such a way into $Ax = b$, $Dx \leqslant e$ such that the following so-called *Lagrangean problem* can be solved easily (easily of course means easy relative to IP and $(E, \mathcal{S}, c)$):

$$\begin{aligned} f(u) := \min\ & c^T x + u^T(Ax - b), \\ & Dx \leqslant e, \\ & x \geqslant 0 \text{ and } x \text{ integral}. \end{aligned} \tag{LR_u}$$

The vector $u = (u_1, \ldots, u_m)^T$ is called the vector of *Lagrangean multipliers*. Thus, for every $u \in \mathbb{R}^m$ we have a Lagrangean problem (LR_u). The structure of the Lagrangean problem is such that we optimize over a subset of the constraints of (IP), i.e. enlarge the solution set, and absorb the constraints $Ax = b$ of (IP) which are not considered in (LR_u) into the objective function of (LR_u). The vector u serves to penalize feasible solutions x of (LR_u) which do not satisfy $Ax = b$ with equality. By choosing appropriate Lagrangean multipliers u one hopes to get an optimal solution of (LR_u) which is either a feasible solution of (IP) and therefore an optimal solution for (IP), or which gives a tight lower bound for the optimum value of (IP).

The problem of Lagrangean relaxation is to quickly find a 'good' u, i.e. a vector u such that $z_{IP} - f(u)$ is as small as possible. A best choice is of course a vector u^* such that

$$z_{LR} := f(u)^* = \max_{u \in \mathbb{R}^m} f(u) \tag{7.1}$$

which minimizes the gap $z_{IP} - f(u)$. Such a vector u^* provides the best lower bound obtainable with the chosen Lagrangean relaxation (LR_u).

To determine the best multipliers, u^*, several approaches are possible. We shall describe some of these. In general it is not easy to find a best u^*. However, usually near optimal Lagrangean multipliers $\bar{u}$ can be obtained quickly. So in actual applications of Lagrangean relaxation one tries to get u^* but usually stops (using some reasonable termination criteria) with a 'sufficiently good' approximation for a u^*.

7.1. Solving the Lagrangean relaxation by linear programming

We shall now show that the maximization problem (7.1) can be formulated as a linear program with many variables or constraints.

Note first that for every program (LR_u), $u \in \mathbb{R}^m$, the set of feasible solutions $R = \{x \in \mathbb{R}^n \mid Dx \leqslant e,\ x \geqslant 0,\ x \text{ integral}\}$ is the same. Moreover, since we consider combinatorial optimization problems, we can safely assume that R is finite. So we can represent R in the following form

$$R = \{x^t \in \mathbb{Z}^n \mid t = 1, \ldots, T\}$$

where T usually (e.g. in the case of combinatorial optimization problems) is a very large number.

Therefore, (7.1) can be written in the form

$$z_{LR} = \max_{u \in \mathbb{R}^m} \min\{c^T x^t + u^T(Ax^t - b) \mid t = 1, \ldots, T\}$$

or equivalently

$$z_{LR} = \max\{w \mid w \leqslant c^T x^t + u^T(Ax^t - b),\ t = 1, \ldots, T\}. \tag{7.2}$$

That is, z_{LR} can be determined by a linear program in the variables u_i, $i = 1, \ldots, n$, and w which has many constraints. The LP-dual of (7.2) is one with many variables (namely one for each x^t).

$$\begin{aligned} z_{LR} = \min & \sum_{t=1}^{T} \lambda_t c^T x^t, \\ & \sum_{t=1}^{T} \lambda_t A x^t = b, \\ & \sum_{t=1}^{T} \lambda_t = 1, \quad \lambda_t \geqslant 0,\ t = 1, \ldots, T. \end{aligned} \tag{7.3}$$

Although the linear programs (7.2) (resp. (7.3)) have many constraints (resp. variables), the simplex method can sometimes be used efficiently to solve these problems. For example one can solve (7.3) by generating at every pivot step the variable entering the basis by solving (LR_u) where u is the current value of the simplex multipliers. This method is known as the column generation technique and has been used in various areas of mathematical programming. Several special implementations of the simplex method (primal, dual, or primal–dual versions) have been considered for the special type program (7.3) in the literature, (cf. Held and Karp (1970), Fisher (1973), and Fisher, Northup and Shapiro (1975)), but in general this LP-approach seems to converge rather slowly to the desired optimum z_{LR} and is usually outperformed by the subgradient method described in the next section.

7.2. The subgradient method

Recall that program (7.1) is an unconstrained maximization problem where the function $f: \mathbb{R}^m \to \mathbb{R}$ is defined as the minimum of the integer program (LR_u). Since the set R of feasible solutions of (LR_u) can be written in the form

$$R = \{x^t \in \mathbb{Z}^n \mid t = 1, \ldots, T\},$$

the function f can be defined equivalently as

$$f(u) = \min\{c^T x^t + u^T(Ax^t - b) \mid t = 1, \ldots, T\}. \tag{7.4}$$

This implies immediately that f is a concave, piecewise linear function. So f is not differentiable everywhere and we cannot use the usual unconstrained optimization techniques of nonlinear programming.

However, for solving (7.1) a 'gradient-like' method can be used if we consider a slight generalization of the usual concept of a gradient.

7.5. Definition. Let $f: \mathbb{R}^m \to \mathbb{R}$ be a concave function and $u \in \mathbb{R}^m$. Then a vector $\pi \in \mathbb{R}^m$ is called a *subgradient* of f at u if the following subgradient inequality holds

$$f(v) - f(u) \leqslant \pi^T(v - u) \quad \forall\, v \in \mathbb{R}^m. \tag{7.6}$$

The set of all subgradients of f at u is called the *subdifferential* of f at u and is denoted by $\partial f(u)$.

Note that if f is differentiable, then $\partial f(u) = \{\nabla f(u)\}$, so subgradients are a proper generalization of gradients. If f is a piecewise linear, concave function of the form (7.4), then the subgradients can be characterized as follows:

7.7. Remark. For any $\bar{u} \in \mathbb{R}^m$ set

$$\operatorname{eq}(\bar{u}) = \{t \in \{1, \ldots, T\} \mid f(\bar{u}) = c^T x^t + \bar{u}^T(Ax^t - b)\}.$$

Then

$$\partial f(\bar{u}) = \{\pi \in \mathbb{R}^m \mid \pi = \sum_{t \in \operatorname{eq}(\bar{u})} \mu_t (Ax^t - b), \sum_{t \in \operatorname{eq}(\bar{u})} \mu_t = 1, \mu_t \geqslant 0\}.$$

In other words, for functions f of the form (7.4), $\partial f(\bar{u})$ is a polytope since it is the convex hull of finitely many vectors. In particular, it follows that for every $t \in \operatorname{eq}(\bar{u})$ the vector $Ax^t - b$ is a subgradient of f at $\bar{u}$.

The well-known optimality criterion for differentiable concave functions also carries over to this more general case.

7.8. Theorem. *Let $f: \mathbb{R}^m \to \mathbb{R}$ be a concave function. Then a vector u^* solves the problem* $\max\{f(u) \mid u \in \mathbb{R}^m\}$ *if and only if* $0 \in \partial f(u^*)$.

This characterization of the optimum solution suggests the following iterative scheme.

7.9. Algorithm. *Subgradient method.* Given a concave function $f: R^m \to \mathbb{R}$ and a subroutine which at every point $u \in \mathbb{R}^m$ determines a subgradient (resp. checks whether $0 \in \partial f(u)$).

1. Choose an arbitrary vector $u^0 \in \mathbb{R}^m$, e.g. $u^0 = 0$, and set $i := 0$.
2. If $0 \in \partial f(u^i)$, then u^i is an optimal solution. STOP.
3. Determine a subgradient $\pi^i \in \partial f(u^i)$.
4. Choose a step length t^i and set $u^{i+1} := u^i + t^i \pi^i$.
5. Set $i := i + 1$ and GOTO 2.

It is presently unknown how to choose an optimum step length t^i, but Polyak (1967) showed that for a large variety of step lengths the subgradient method (7.9) converges.

7.10. Theorem. *If a concave function* $f: \mathbb{R}^m \to \mathbb{R}$ *attains its maximum and if the sequence* $(t^i)_{i\in\mathbb{N}}$ *of the step lengths satisfies the following conditions:*

(1) $t_i > 0$ *for all* $i \in \mathbb{N}$,

(2) $\lim_{i\to\infty} t_i = 0$.

(3) $\sum_{i\in\mathbb{N}} t_i = \infty$,

then

$$\lim_{i\to\infty} f(u^i) = \max_{u\in\mathbb{R}^m} f(u)$$

Loosely speaking, conditions (2) and (3) above state that the step length should go to zero, but not too fast, in order to converge to the maximum of f. There are various possibilities to accelerate convergence theoretically by choosing particular step lengths. However, in practical computations the following facts have been observed empirically by many researchers. The subgradient method produces substantial increases in the function value in the first steps but after a certain number of additional iterations the function value begins to oscillate without showing a tendency to converge.

Since in the case of combinatorial optimization problems subgradient methods are embedded in a branch and bound procedure this practical lack of convergence does not matter too much. One is usually satisfied with obtaining a good bound quickly and is not willing to pay the high computational cost for the calculation of a slightly better bound. For this reason, one does not often use the Polyak conditions of Theorem 7.10, but rather tries to find a reasonable step length formula which shows satisfactory convergence empirically. Such step length formulas are often highly dependent on the characteristics of the problem at hand.

Historically, nondifferential optimization has, to a large extent, been developed in the Soviet Union. Polyak (1978) surveys these developments and the most interesting results obtained so far. The subgradient method (7.9) is apparently due to Shor (1962). Good sources on nondifferential optimization are Rockafellar (1979) and Volume 3 of the Mathematical Programming Studies. The most influential papers (in the western literature) on the application of nondifferential optimization to combinatorial programming are probably those of Held and Karp (1970, 1971) (our present terminology was not used at that time). In a later paper, Held, Wolfe and Crowder (1974) pointed out the generality of this approach and showed

links to nondifferential optimization and to the large body of literature on this subject existing in the Soviet Union.

The literature on subgradient methods and their applications to combinatorial optimization has grown extensively in recent years. Surveys of applications of Lagrangean relaxation to integer programming can be found in Burkard (1980), Fisher (1981), Gavish (1978), Geoffrion (1974).

Fisher (1981) gives a long list of known applications of this approach to combinatorial optimization. Problems successfully treated include the STSP, ATSP, vehicle routing, knapsack problems, general integer programming, some generalized assignment problems, various scheduling problems, the multicommodity flow problem, several location problems, the set covering and partitioning problem, cluster analysis and several others. Summing up computational experiences with Lagrangean relaxation using subgradient techniques embedded in branch and bound schemes, one can say that this approach has certainly lead to substantial improvements of former algorithms.

7.3. A branch and bound algorithm for the STSP using 1-trees and Lagrangean relaxation

The algorithm for the STSP we are going to present is basically the method of Held and Karp, (cf. Held and Karp (1970, 1971)), including some later improvements by Helbig, Hansen and Krarup (1974), Smith and Thompson (1977) and Volgenant and Jonker (1980).

We have mentioned that our algorithm 6.5 using 1-tree relaxation is computationally not very successful due to the weak bounds obtained from 1-trees. We shall now take the same algorithm 6.5 and replace the bounding step 5 by a bounding step using a Lagrangean relaxation yielding a 1-tree problem.

The usual way to write the STSP as an integer program is the following:

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

$$\text{(1)} \qquad \sum_{i<j} x_{ij} + \sum_{j<i} x_{ij} = 2, \quad i = 1, \ldots, n,$$

$$\text{(2)} \qquad \sum_{ij\in E(W)} x_{ij} \leqslant |W| - 1 \quad \text{for all } W \subseteq \{2, 3, \ldots, n\}, \tag{7.11}$$

$$\left.\begin{array}{ll} \text{(3)} & x_{ij} \leqslant 1, \\ \text{(4)} & x_{ij} \geqslant 0, \\ \text{(5)} & x_{ij} \text{ integer} \end{array}\right\} \; 1 \leqslant i < j \leqslant n.$$

To get the desired 1-tree relaxation we have to reformulate (7.11) slightly but equivalently as follows:

$$\begin{aligned} &\min \sum_{1\leqslant i<j\leqslant n} c_{ij}x_{ij}, \\ (1') \qquad &\sum_{i<j} x_{ij} + \sum_{j<i} x = 2, \quad i = 2, \ldots, n, \\ (1'') \qquad &\sum_{j=2}^{n} x_{1j} = 2, \\ (1''') \qquad &\sum_{1\leqslant i<j\leqslant n} x_{ij} = n, \end{aligned} \tag{7.12}$$

and x satisfies (2), (3), (4), (5).

The two equations (1'') and (1''') can be written as four inequalities. So (7.12) is of the form $\min c^{\mathrm{T}}x$, $Ax = b$, $Dx \leqslant e$, $x \geqslant 0$, x integer; where $Ax = b$ represents the equations (1'), A is the node-edge incidence matrix of the nodes $2, \ldots, n$ and where $Dx \leqslant e$ represents the four inequalities arising from (1'') and (1''') and the inequalities (2), (3). With this definition, the program $\min c^{\mathrm{T}}x$, $Dx \leqslant e$, $x \geqslant 0$, x integer is nothing but the problem of finding a minimum 1-tree in K_n. Thus our Lagrangean problem derived from (7.12) reads

$$\begin{aligned} &f(u) := \min c^{\mathrm{T}}x + u^{\mathrm{T}}(Ax - b), \\ &Dx \leqslant e, \\ &x \geqslant 0 \text{ and integral}. \end{aligned} \tag{7.13}$$

Suppose for a given $u \in \mathbb{R}^{n-1}$ problem (7.13) is solved, then the optimum solution, say x^*, represents a 1-tree, say T^*. Since A is a node-edge incidence matrix, the vector $d \in \mathbb{R}^{n-1}$ with $d := Ax^*$ represents the degrees d_k of all nodes $k = 2, \ldots, n$ in T^*. Hence Remark 7.7 implies that the vector $\pi = (\pi_2, \ldots, \pi_n)^{\mathrm{T}} \in \mathbb{R}^{n-1}$ with $\pi_k := d_k - 2$, $k = 2, \ldots, n$, is a subgradient of our function f at u. This shows that for every u the optimum 1-tree calculation automatically provides us with a subgradient of f at u. Thus we can apply the subgradient algorithm.

We still have to choose the step lengths. We do not try to find the optimum solution, rather we want a method which rapidly improves the value of our 1-trees. Smith and Thompson (1977) empirically found the following step length formula which in our own computational experience has also exhibited good practical performance.

Let U be the current smallest upper bound for the optimal tour and L be the current largest lower bound for the optimal tour (both numbers are available in algorithm 6.5), let λ be a number satisfying $0 < \lambda \leqslant 2$ and let d_k be the degrees of the nodes $k = 2, \ldots, n$ of the current optimal 1-tree, then set

$$t := \frac{\lambda(U - L)}{\sum_{k=2}^{n} (d_k - 2)^2} \tag{7.14}$$

By adding a suitable termination criterion we now can formulate the new bounding step for algorithm 6.5:

7.15. *New bounding step 5' for algorithm* 6.5.

(a) Set $u_k^0 := 0$, $k = 2, \ldots, n$, $\lambda := 2$ and $i := 1$.

(b) Calculate the minimal 1-tree T_1^i of the Lagrangean problem

$$\min\{c(T_1) + \sum_{k=2}^{n} u_k^i (d_k(T_1) - 2) \mid T_1 \text{ 1-tree in } K_n \text{ with } R \subseteq T_1, \; T_1 \cap F = \emptyset\}$$

where $d_k(T_1)$ is the degree of node k in 1-tree T_1. Note that this problem can also be stated as $\min\{\bar{c}(T_1) \mid T_1 \text{ 1-tree} \ldots\}$ where $\bar{c}_{st} = c_{st} + u_s^i + u_t^i$, $1 \leq s < t \leq n$ (setting $u_1 = 0$), since by definition $\sum_{k=2}^{n} u_k^i = 0$.

(c) If T_1^i is a tour, set $T^* := T_1^i$ and GOTO 6.

(d) If the value of the current best 1-tree has not increased by at least 0.1 during the last $\lfloor n/8 \rfloor + 6$ iterations, GOTO (j).

(e) Define the new subgradient vector $\pi^i = (\pi_2^i, \ldots, \pi_n^i) \in \mathbb{R}^{n-1}$ by setting

$$\pi_k^i := d_k(T_1^i) - 2, \quad k = 2, \ldots, n.$$

(f) If $i \equiv 0 \pmod{\lfloor n/10 \rfloor + 4}$ set $\lambda := \lambda/2$ (the stepsize parameter λ is halved after every $\lfloor n/8 \rfloor + 4$ steps).

(g) Set

$$t^i := \frac{\lambda(U - L)}{\sum_{k=2}^{n} (\pi_k^i)^2}$$

If $t^i < 0.01$, GOTO (j) (the stepsize is too small for practical computations).

(h) Define the new Lagrangean multipliers

$$u^{i+1} := u^i + t^i \pi^i, \quad i = 2, \ldots, n.$$

(Do a step in the direction of the subgradient.)

(i) Set $i := i + 1$ and GOTO (b).
(j) Let T^* be the current best 1-tree of value L and GOTO 6.

It was empirically observed by Helbig Hansen and Krarup (1974) that in step 5' (b) of 7.15 different spanning tree algorithms should be used depending on the current status of the branch and bound scheme. Altogether, algorithm 6.5, where step 5 is replaced by step 5' described above (plus some additional tricks reported e.g. in Held and Karp (1970, 1971), Helbig Hansen and Krarup (1974), Smith and Thompson (1977) and Volgenant and Jonker (1980)) constitutes quite an efficient method for solving STSP's with up to about 100 cities.

8. Cutting plane methods

As mentioned in Section 5, the basic idea of cutting plane methods is to convert a combinatorial optimization problem into a linear program and to solve this linear program. Since the associated program typically has many constraints one starts with a subset of the complete equation and inequality system, solves the current LP, and if the optimum solution is not feasible for the combinatorial optimization problem, adds inequalities to cut off the optimum solution and continues. In the first part of this section we show how such linear programs can be obtained, in the second part, we discuss the computational problems of this approach, and in the third we outline an algorithm for the STSP based on this cutting plane approach. The standard cutting plane techniques based on Gomory's algorithm which are not discussed here can be found in every book on integer programming, e.g. Burkard (1972), Garfinkel and Nemhauser (1972). A good survey of existing cutting plane methods from a more general viewpoint is given by Wolsey (1979).

8.1. Polyhedral combinatorics

The area in which combinatorial optimization problems are studied by investigating certain polyhedra associated with these problems is frequently called *polyhedral combinatorics*. There are various ways to associate a polyhedron with a combinatorial optimization problem but we shall only outline the most common approach.

Given an instance $(E, \mathcal{F}, c)$ of a combinatorial optimization problem $\mathcal{P}$, then we associate with E the vector space $\mathbb{R}^E$ and with each element $e \in E$ we associate a variable x_e (resp. a component of a vector $x \in \mathbb{R}^E$ indexed by e). For every subset $F \subseteq E$ we define an *incidence vector* $x^F \in \mathbb{R}^E$ by setting

$$x_e^F := \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{if } e \notin F. \end{cases}$$

With the set of feasible solutions $\mathcal{F}$ we associate the following polytope (bounded polyhedron):

$$P_{\mathcal{F}} := \operatorname{conv}\{x^I \in \mathbb{R}^E \mid I \in \mathcal{F}\}. \tag{8.1}$$

That is, $P_{\mathcal{F}}$ is the convex hull of the incidence vectors of the feasible solutions $\mathcal{F}$. This definition implies that $P_{\mathcal{F}}$ is a subset of the unit hypercube in $\mathbb{R}^E$ and that every vertex of $P_{\mathcal{F}}$ corresponds to a feasible solution of $\mathcal{F}$ and vice versa. Thus, every instance $(E, \mathcal{F}, c)$ of $\mathcal{P}$ (w.l.o.g. we assume $\mathcal{P}$ is a maximization problem) can be solved by solving the maximization problem.

$$\max c^T x, \quad x \in P_{\mathcal{F}}. \tag{8.2}$$

Problem (8.2) is a linear programming problem since $P_{\mathcal{F}}$ is a polyhedron. The LP (8.2) is however not given in the usual way by means of an inequality system which is the one needed e.g. to apply the simplex algorithm or the ellipsoid method. The representation of a polytope as the convex hull of (many) points is not suitable for LP-algorithms. Fortunately, there exists a theorem of Weyl which states that for every polyhedron $P_{\mathcal{F}}$ of the form (8.1) there exists an $(m, |E|)$-matrix A and a vector $b \in \mathbb{R}^m$ such that

$$P_{\mathcal{F}} = \{x \in \mathbb{R}_E \mid Ax \leqslant b\}. \tag{8.3}$$

The number m of rows of A is usually very large, even for combinatorial optimization problems which are solvable in polynomial time, cf. (8.4)–(8.10). This number m must even be exponential in $|E|$ if $\mathcal{P}$ is a hard problem (if P $\neq$ NP). For otherwise we could construct from every instance $(E, \mathcal{F}, c)$ of $\mathcal{P}$ (in time polynomial in $|E|$ and the encoding of c) a linear program and then solve this LP in polynomial time with the ellipsoid method, (cf. Khachian (1979), Gács and Lovász (1981), Schrader (1982)).

On the other hand it was shown in Grötschel, Lovász and Schrijver (1981a) that in order to solve an LP of the form (8.2) the number of ine-

qualities to characterize $P_{\mathcal{F}}$ is not so important. What matters is the structure of the inequality system (8.3) (more detailed explanations of this will be given in Section 8.2). In order to obtain a polynomial algorithm for problem (8.2) it is therefore necessary to characterize the inequality system (8.3) for $P_{\mathcal{F}}$ exactly. Although there are constructive proofs of the theorem of Weyl, these general proof techniques usually do not suffice to obtain 'handy' descriptions of the polytopes $P_{\mathcal{F}}$. This makes it necessary to study every particular problem by means of special investigations.

The determination of complete and nonredundant systems of equations and inequalities defining the polyhedra associated with combinatorial optimization problems has been (and still is) one of the main topics of combinatorial combinatorics. This approach was very successful for the class of polynomially solvable problems. To date for most of these easy problems complete descriptions of the associated polyhedra are known explicitly. In the following we list some examples of this type. For convenience we shall abbreviate the sum $\sum_{e\in F} x_e$ by $x(F)$ in the sequel.

8.4. *The matroid polytope* (Edmonds (1971)). Let $(E, \mathcal{F})$ be a matroid and $P_{\mathcal{F}}$ be the convex hull of the incidence vectors of the elements of $\mathcal{F}$, then

$$P_{\mathcal{F}} = \{x \in \mathbb{R}^E \mid x_e \geqslant 0 \ \forall\, e \in E,\ x(F) \leqslant r(F) \ \forall\, F \subseteq E\}$$

where for every $F \subseteq E$, $r(F)$ (called the *rank* of F) denotes the cardinality of a basis of F, cf. 3.2.

Some of the inequalities above are superfluous. In fact, one can show that only those subsets $F \subseteq E$ are needed which are closed and inseparable. But this still leaves exponentially many inequalities in many cases. For example, consider the spanning tree problem, (cf. 3.4) which is a special case of the matroid problem.

8.5. *The forest polytope.* Let $K_n = [V, E]$ be the complete graph of order $n \geqslant 3$ and $\mathcal{F}$ the set of forests in K_n. Then

$$P_{\mathcal{F}} = \{x \in \mathbb{R}^E \mid x_e \geqslant 0 \ \forall\, e \in E,$$
$$x(E(W)) \leqslant |W| - 1 \ \forall\, W \subseteq V,\ |W| \geqslant 2\}.$$

This characterization of the so-called forest polytope is complete and nonredundant.

The following is a deep result of Edmonds on the intersection of two matroids.

8.6. *The 2-matroid intersection polytope* (Edmonds (1970)). Let $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$ be two matroids on the same ground set E. Let r_1 and r_2 be the rank functions with respect to $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$. Let $\mathcal{I} := \mathcal{I}_1 \cap \mathcal{I}_2$, then

$$P_{\mathcal{I}} = P_{\mathcal{I}_1} \cap P_{\mathcal{I}_2} = \{x \in \mathbb{R}^E \mid x_e \geqslant 0 \ \forall\ e \in E,$$
$$x(F) \leqslant \min\{r_1(F), r_2(F)\} \ \forall\ F \subseteq E\}.$$

The corresponding statement for three or more matroids is in general not true. A special case of 8.6 is the branching problem, cf. 3.6.

8.7. *The branching polytope* (Edmonds (1967)). Let $D = (V, A)$ be a digraph and $\mathcal{I}$ the set of all branchings in D. Let V^- be the set of nodes $v \in V$ such that either $\omega^-(v) = \emptyset$ or $|\Gamma^-(v)| = 1$, say $\Gamma^-(v) = \{w\}$, and $(v, w) \in A$. Then

$$P_{\mathcal{I}} = \{x \in \mathbb{R}^A \mid x_e \geqslant 0 \ \forall\ e \in A,$$
$$x(\omega^-(v)) \leqslant 1 \ \forall\ v \in V \setminus V^-,$$
$$x(A(W)) \leqslant |W| - 1 \text{ for all } W \subseteq V$$
$$\text{such that the induced subdigraph}$$
$$(W, A(W)) \text{ is strongly and 2-connected}\}. \quad (8.8)$$

This characterization of the branching polytope is complete and non-redundant, cf. Giles (1978), Grötschel (1979).

8.9. *The 1-matching polytope* (Edmonds (1965)). Given a graph $G = [V, E]$ and let $\mathcal{I}$ be the collection of 1-matchings in G. Then

$$P_{\mathcal{I}} = \{x \in \mathbb{R}^E \mid x_e \geqslant 0 \ \forall\ e \in E,\ x(\omega(v)) \leqslant 1 \ \forall\ v \in V,$$
$$x(E(W)) \leqslant (|W| - 1)/2 \ \forall\ W \subseteq V,$$
$$|W| \geqslant 3 \text{ and } |W| \text{ odd}\}.$$

8.10. *The perfect 2-matching polytope* (Edmonds (1965)). Given a graph $G = [V, E]$. Recall that a perfect (binary) 2-matching is a set M of edges such that every node is contained in exactly two edges of M. Let $\mathcal{I}$ be the

set of perfect 2-matchings in G. Then

$$P_{\mathcal{F}} = \{x \in \mathbb{R}^E \mid 0 \leqslant x_e \leqslant 1 \ \forall e \in E,\ x(\omega(v)) = 2 \ \forall v \in V,$$
$$\textstyle\sum_{i=1}^{k} x(E(W_i)) \leqslant |W_0| + \frac{1}{2}(k-1)$$
$$\text{where } W_0, \ldots, W_k \subseteq V \text{ such that}$$
$$|W_0 \cap W_i| = 1,\ |W_i \setminus W_0| = 1,\ i = 1, \ldots, k,$$
$$W_i \cap W_j = \emptyset,$$
$$1 \leqslant i < j \leqslant k,\ k \geqslant 1 \text{ and } k \text{ odd}\}.$$

In case G is complete, it is known which of these inequalities define facets of $P_{\mathcal{F}}$ (cf. Grötschel (1977b)).

There are many more such results about complete and nonredundant descriptions of polyhedra associated with polynomially solvable combinatorial optimization problems, (cf. Barahona (1980), Boulala and Uhry (1979), Chvátal (1975), Edmonds and Johnson (1970), Giles (1975), Grötschel (1977) and Pulleyblank (1973)).

No complete descriptions of polytopes $P_{\mathcal{F}}$ defined in (8.1) associated with hard problems have been given explicitly so far. There are various theoretical results which imply that it is very unlikely that such descriptions can be obtained at all (cf. Karp and Papdimitriou (1980), Grötschel, Lovász and Schrijver (1981)). Nevertheless, it is of theoretical and practical value to know at least partial descriptions of polyhedra associated with hard problems. Investigations of the structure of such polytopes have been made with respect to almost all hard problems and the number of papers on this subject is ever increasing. A list of references can be found in Kastning (1976) and Hausmann (1978). We only want to discuss the stable set and symmetric travelling salesman problem in more detail.

Suppose a graph $G = [V, E]$ is given, then the polytope

$$P(G) := \operatorname{conv}\{x^W \in \mathbb{R}^V \mid W \text{ a stable set in } G\} \tag{8.11}$$

is called the *stable set polytope* of G. By definition, no two adjacent nodes can be in a stable set. So, for every edge $vw \in E$, every incidence vector of a stable set satisfies the inequality $x_v + x_w \leqslant 1$. This implies that the

stable set polytope $P(G)$ is contained in the polytope

$$P'(G) := \{x \in \mathbb{R}^V \mid x_v \geqslant 0 \ \forall v \in V, \ x_v + x_w \leqslant 1 \ \forall vw \in E\}.$$

Moreover, it is easy to see that $P(G) = \text{conv}\{x \in P'(G) \mid x \text{ integral}\}$, i.e. every integral solution of

$$\max c^T x, \quad x \in P'(G) \tag{8.12}$$

is a solution of the stable set problem. Nemhauser and Trotter (1974) and (1975) have shown some interesting results about the relations between optimum solutions with respect to $P'(G)$ and $P(G)$. First of all they show the LP (8.12) is solvable with a fast polynomial time method. Secondly, if x' is an optimal solution to (8.12) and W is the set of indices $w \in V$ such that x'_w is zero or one, then there exists an optimal solution $\bar{x}$ of the stable set problem such that $\bar{x}_w = x'_w$ for all $w \in W$. This implies that having solved (8.12) one can fix the integral coefficients and need only consider the remaining fractional ones in the sequel.

There are, however, some disadvantages with the LP (8.12). The optimum value of (8.12) usually gives a very poor upper bound for the stable set problem. Moreover, Pulleyblank (1979) has shown that, under suitable assumptions, (8.12) will almost never have an optimum solution with integral coefficients. So the nice result of Nemhauser and Trotter above cannot be applied very often.

Because of these drawbacks concerning $P'(G)$ it is necessary to find tighter relaxations of the stable set problem. An obvious next step is to replace the edge inequalities in the definition of $P'(G)$ by clique inequalities. Namely, if W is a stable set in G and C is a clique, then $|W \cap C|$ is clearly at most one. This implies that every incidence vector x^W of a stable set $W \subseteq V(G)$ satisfies

$$x(C) \leqslant 1 \quad \text{for all cliques } C \subseteq V(G).$$

These inequalities are called *clique inequalities*. The polytope

$$P^*(G) := \{x \in \mathbb{R}^n \mid x_v \geqslant 0 \ \forall v \in V,$$

$$x(C) \leqslant 1 \text{ for all cliques } C \subseteq V\}$$

obviously satisfies $P(G) \subseteq P^*(G) \subseteq P'(G)$. It was shown by Fulkerson (see also Chvátal (1975)) that $P(G) = P^*(G)$ if and only if G is a perfect graph. This fact (among other things) was used in Grötschel, Lovász and Schrijver

(1981 b) to derive a polynomial time algorithm for the stable set problem in perfect graphs. Thus, at least for perfect graphs, the LP

$$\max c^{\mathrm{T}}x, \quad x \in P^*(G) \tag{8.13}$$

can be solved in polynomial time. However, it was also shown by Grötschel, Lovász and Schrijver (1981 b) that the problem (8.13) is in general as hard as the stable set problem itself. Furthermore, there are classes of graphs where (8.13) is hard but the stable set problem easy, and vice versa. Although the bound from (8.13) is much tighter than that of (8.12) it may still be arbitrarily bad (cf. Grötschel, Lovász and Schrijver (1981 b)).

The polytope $P^*(G)$ also has some nice relations to $P(G)$. The facets of $P^*(G)$ are exactly the trivial inequalities $x_v \geqslant 0$ for all $v \in V$ and the clique inequalities $x(C) \leqslant 1$ for all maximal cliques $C \subseteq V(G)$ (i.e. maximal with respect to set inclusion). It is easy to see that all facets of $P^*(G)$ are also facets of $P(G)$, so $P^*(G)$ is a polytope which is derived from $P(G)$ by removing some facet inequalities of $P(G)$.

Besides the maximal clique inequalities there are many more classes of facets of $P(G)$ known, cf. Chvátal (1975), Fulkerson (1971), Nemhauser and Trotter (1974), Padberg (1973), Trotter (1974), Balas and Zemel (1977). Some of these classes of inequalities were also exploited algorithmically, cf. Balas and Padberg (1975) for a survey of such results.

With respect to the STSP the following (symmetric) *travelling salesman polytope*

$$Q_T^n := \operatorname{conv}\{x^{\mathrm{T}} \in \mathbb{R}^m \mid T \text{ is a tour in } K_n\}, \tag{8.14}$$

where $m = n(n-1)/2$, is of primary interest. The integer programming formulation of the (STSP) presented in (7.11) is due to Dantzig, Fulkerson and Johnson (1954). It was shown in Grötschel and Padberg (1979 a, 1979 b) that the system of n equations given in (7.11)

$$x\big(\omega(v)\big) = 2 \quad \text{for all } v \in V \tag{8.15}$$

is one with maximum rank containing Q_T^n, and therefore that

$$\dim(Q_T^n) = n(n-3)/2 = m - n \tag{8.16}$$

is the dimension of Q_T^n. It also turns out that the *subtour elimination constraints* of (7.11),

$$x\big(E(W)\big) \leqslant |W| - 1, \tag{8.17}$$

and the trivial inequalities $0 \leqslant x_e \leqslant 1$ are facets of Q_T^n (cf. Grötschel and Padberg (1979b)). The travelling salesman polytope is not only a subset of the 1-tree polytope (cf. (7.12)), but also of the 2-matching polytope, cf. (8.10). So it is obvious to ask which of the facets of this polytope are also facets of Q_T^n. This problem is settled in Grötschel (1977a). The 2-matching inequalities of (8.10) are generalized by Chvátal (1973) and further by Grötschel and Padberg (1979a) to a class of inequalities called *comb inequalities* which are defined as follows:

Let $K_n = [V, E_n]$ be the complete graph on $n \geqslant 6$ nodes. Let $W_0, \ldots, W_k \subseteq V$ satisfy the following conditions:

(i) $|W_0 \cap W_i| \geqslant 1$ for $i = 1, \ldots, k$,

(ii) $|W_i \setminus W_0| \geqslant 1$ for $i = 1, \ldots, k$,

(iii) $|W_i \cap W_j| = 0$ for $1 \leqslant i < j \leqslant k$,

(iv) $k \geqslant 3$ and odd.

Then set

$$\sum_{i=0}^{k} x(E(W_i)) \leqslant |W_0| + \sum_{i=1}^{k} (|W_i| - 1) - \frac{k+1}{2}. \tag{8.18}$$

These comb inequalities (8.13) are also facets of Q_T^n. Note that in case $|W_0 \cap W_i| \geqslant 2$ for some i, the inequality (8.18) has coefficients which are not only equal to 0, 1 but also equal to 2.

The results on the facial structure of Q_T^n presented in Grötschel and Padberg (1979b) are summarized in the following

8.19. Theorem. *Let Q_T^n be the travelling salesman polytope for $n \geqslant 6$. Then the following holds:*

(a) $\dim Q_T^n = n(n-3)/2$.

(b) *For every $e \in E$, the trivial inequalities $x_e \geqslant 0$ and $x_e \leqslant 1$ define facets of Q_T^n.*

(c) *For every $W \subseteq V$ with $3 \leqslant |W| \leqslant n-3$, the corresponding subtour elimination constraint* (8.17) *defines a facet of Q_T^n.*

(d) *All comb inequalities* (8.18) *define facets of Q_T^n.*

Let us call two different inequalities equivalent if they define the same facet of Q_T^n. Then one can also show that the trivial inequalities, the sub-

tour elimination constraints and the comb inequalitites are mutually non-equivalent, and that for every subtour elimination constraint (comb inequality) there exists exactly one other subtour elimination constraint (comb inequality) which is equivalent to it. In particular, if we require additionally in Theorem 8.19 (c) that $1 \in W$, and in (d) that $1 \in W_0$, then the system of inequalities in (b), (c) and (d) of (8.19) is a nonredundant partial characterization of Q_T^n. This system is not at all complete. For example, other facets are known which are related to complicated graphs like hypohamiltonian, hypotraceable, or non-hamiltonian and hypomatchable graphs, (cf. Cornuejols and Pulleyblank (1980b), Grötschel (1977), Grötschel (1980b), Grötschel and Padberg (1979a, 1979b), Maurras (1975, 1976)). Recently Theorem 8.19 was considerably generalized by Grötschel and Pulleyblank (1981).

8.2. Cutting plane recognition

If G is a graph of order n then each of the polytopes (associated with easy problems) described in (8.4)–(8.10) has an exponential number of facets, and the partial descriptions of the stable set polytope $P(G)$ and the symmetric travelling salesman polytope Q_T^n presented above also contain an exponential number of facet inequalities. Thus it is impossible to represent these systems of linear inequalities in a computer and solve the corresponding LP's by the simplex method. For example, for the STSP there are about 10^{179} facets of Q_T^{120} known (cf. Grötschel (1980a)). Any sensible approach to solving such large linear programs must therefore try to avoid the use of such large numbers of inequalities and try to succeed in proving optimality with a few 'suitably chosen' inequalities.

The ellipsoid method shows that such large linear programs can indeed be solved in polynomial time provided cutting planes can be recognized in polynomial time. Let us make this more precise.

Suppose $P_{\mathcal{F}} \subseteq \mathbb{R}^n$ is a polytope of the form (8.1). For some technical reasons which do not restrict generality we assume that $P_{\mathcal{F}}$ is fully-dimensional.

8.20. *The optimization problem for* $P_{\mathcal{F}}$. Suppose a vector $c \in \mathbb{R}^n$ is given. Then the optimization problem for $P_{\mathcal{F}}$ is to find a vector x^* such that

$$c^T x^* = \max\{c^T x \mid x \in P_{\mathcal{F}}\}.$$

8.21. *The separation problem for* $P_{\mathcal{I}}$. Suppose a vector $y \in \mathbb{R}^n$ is given, then the separation problem for $P_{\mathcal{I}}$ is to prove that $y \in P_{\mathcal{I}}$ or to find a vector $d \in \mathbb{R}^n$ such that $d^T x < d^T y \ \forall\, x \in P_{\mathcal{I}}$.

The ellipsoid method (cf. Khachian (1979), Gács and Lovász (1981), Grötschel, Lovász and Schrijver (1981a)) can be stated in such a way that in every iteration a separation problem has to be solved. If this separation problem can be solved in polynomial time, then the ellipsoid method runs in polynomial time. The converse also holds, as has been shown by Grötschel, Lovász and Schrijver (1981a).

8.22. Theorem. *Let* $P_{\mathcal{I}}$ *be a full-dimensional polytope. Then the optimization problem for* $P_{\mathcal{I}}$ *is solvable in polynomial time if and only if the separation problem for* $P_{\mathcal{I}}$ *is solvable in polynomial time.*

Moreover, given an easy problem (this is, by definition, a problem for which the optimization problem for the associated polytope $P_{\mathcal{I}}$ is solvable in polynomial time), the proof of Theorem 8.22 yields a constructive polynomial time method to solve the separation problem for $P_{\mathcal{I}}$. This result in particular yields polynomial time methods which decide whether a given vector y is contained in any of the polytopes (8.4)–(8.10) and if not which produce facets of these polytopes separating vector y from the polytopes. For example, suppose a vector y is not contained in the 2-matching polytope for the complete graph K_n (8.10), then although this polytope has an exponential number of facets it takes only a polynomial number of steps to find a facet which is violated by y. The general algorithm of Grötschel, Lovász and Schrijver (1981a) can be applied to all easy problems. So, for every easy problem there exists a polynomial time cutting plane algorithm.

Clearly, if for a hard problem there were a polynomial separation algorithm, then Theorem 8.22 would imply P = NP, a very unlikely result. So we cannot expect such an algorithm for hard problems. It is even very unlikely that for a poytope $P_{\mathcal{I}}$ associated with a hard problem we can prove validity of a given inequality in polynomial time (cf. Karp and Papadimitriou (1980)). Here validity of an inequality $d^T x \leqslant d_0$ means that $P_{\mathcal{I}} \subseteq \{x \mid d^T x \leqslant d_0\}$.

Although these results indicate that we shall (probably) never be able to obtain theoretically good algorithms via cutting plane methods, it is still possible to design practically useful methods. The negative results state

that we will not be able to recognize all cutting planes, but they do not exclude the existence of large classes of facets for which the separation problem can indeed be solved. The search for classes of facets of polytopes associated with hard problems therefore should have the further objective of finding computationally efficient facet recognition algorithms. In cutting plane LP-approaches to hard problems we can then restrict attention to these tractable classes of facets.

Theorem 8.22 implies a nice result about relaxations of a hard problem. Namely, suppose $P_{\mathcal{S}}$ is a polytope associated with a hard problem and P_K is a polytope associated with an easy problem such that $P_{\mathcal{S}} \subseteq P_K$. Then we can recognize at least all those facet inequalities of $P_{\mathcal{S}}$ which are also facets of P_K in polynomial time. Therefore, if we can find several different easy relaxations $P_{\mathcal{S}_1}, \ldots, P_{\mathcal{S}_k}$ of $P_{\mathcal{S}}$ we can optimize over $\bigcap_{i=1,\ldots,k} P_{\mathcal{S}_i}$ in polynomial time using the ellipsoid method.

This general approach via Theorem 8.22 and the ellipsoid method has the theoretical advantage of being polynomial. It is, however, not very efficient in practice, since both the general optimization routine over $\bigcap_{i=1,\ldots,k} P_{\mathcal{S}_i}$ and the separation routine for every $P_{\mathcal{S}_i}$ use the ellipsoid method. Therefore, in practical computation instead of the ellipsoid method the simplex method is used. One starts with a small subset of inequalities, solves the corresponding LP with the simplex method, and checks whether the optimal solution $\bar{x}$ is feasible for the underlying combinatorial optimization problem. If the answer is yes, one stops, if the answer is no one generates one (or more) separating hyperplanes which cut off $\bar{x}$ and continues. Since one also does not want to use the ellipsoid method for this separation routine, other means have to be found to recognize cutting planes. In some cases good heuristics can be used or exact polynomial time methods can be constructed which check a whole class of facets to see whether these are satisfied or not. To date these problem-specific cutting plane recognition algorithms are not well developed in general and much is left to do. One of the exceptions is the symmetric travelling salesman problem for which we shall discuss cutting plane recognition algorithms in the next section.

8.3. A cutting plane algorithm for the STSP

In Theorem 8.19 we have mentioned that the following systems of equations and inequalities is a partial description of the travelling salesman polytope. It consists of n linearly independent equations and mutually non-

equivalent facet-defining inequalities:

$$\left.\begin{aligned} &x(\omega(v)) = 2 \quad \forall\, v \in V, \\ &x_e \leqslant 1 \quad \forall\, e \in E, \\ &x_e \geqslant 0 \quad \forall\, e \in E, \end{aligned}\right\} \tag{8.23}$$

$$x(E(W)) \leqslant |W| - 1 \quad \forall\, W \subseteq V \text{ with } 1 \in W,\ 3 \leqslant |W| \leqslant |V| - 3, \tag{SEC}$$

$$\sum_{i=0}^{k} x(E(W_i)) \leqslant |W_0| + \sum_{i=0}^{k} (|W_i| - 1) - (k+1)/2$$

for all $W_0, \ldots, W_k \subseteq V$ satisfying the conditions given in (8.18). (C)

Denote by $\overline{\overline{Q}}^n_{2M}$ the polytope defined by the equations and inequalities (8.23). The LP

$$\min c^T x, \quad x \in \overline{\overline{Q}}^n_{2M} \tag{8.24}$$

is small enough to be solvable by good simplex based LP-codes even for large n. (We have made experiments up to $n = 1000$ which were solved quite easily.) So $\overline{\overline{Q}}^n_{2M}$ is a suitable initial LP for a cutting plane approach to the STSP.

If we have an optimal solution to (8.24), say x, which does not correspond to a tour, then we have to check whether x satisfies the subtour elimination constraints (SEC). This can be done by the following heuristic. Represent x by the graph $G_x = [V, E_x]$ where $ij \in E_x$ if and only if $x_{ij} > 0$. Now pick an edge $ij \in E_x$ such that the corresponding x_{ij} is one. Then find the longest path in G_x containing ij and those edges pq such that $x_{pq} = 1$. If the endpoints of this path are joined by an edge of G_x, then x clearly violates the subtour elimination constraint generated by the set of nodes of this path. This heuristic is very fast, but does not always find a (SEC) if it exists. Crowder and Padberg (1980) note that this separation problem for subtour elimination constraints can be solved exactly by the Gomory–Hu procedure (cf. Gomory and Hu (1961)). The (SEC) inequalities can be equivalently written as cut constraints of the form $x(\omega(W)) \geqslant 2$, so if the Gomory–Hu algorithm finds a cut in G_x whose value is less than 2 then one of the cut constraints (resp. subtour elimination constraints) is violated. The Gomory–Hu procedure, however, has a worst-case running

time of $O(n^4)$, so it should not be used too often. If we denote by Q_S^n the polytope defined by (8.23) and (SEC), then we can optimize over Q_S^n efficiently (in practice) with the simplex method.

For the comb inequalities (C) no polynomial exact separation algorithm is known to date. One can design various heuristics but they do not guarantee that all comb inequalities are satisfied. Recently, Padberg and Rao (1980) devised a polynomial time method (which is also based on the Gomory–Hu procedure) to check whether all 2-matching inequalities are satisfied. The 2-matching inequalities constitute the subset of (C) where $|W_i| = 2$ for all $i = 1, \ldots, k$. So, it is possible to optimize efficiently over $Q_S^n \cap Q_{2M}^n$ where Q_{2M}^n denotes the 2-matching polytope for K_n (cf. (8.10)).

The flow chart shown in Fig. 2 gives an overview of the cutting plane algorithm described above.

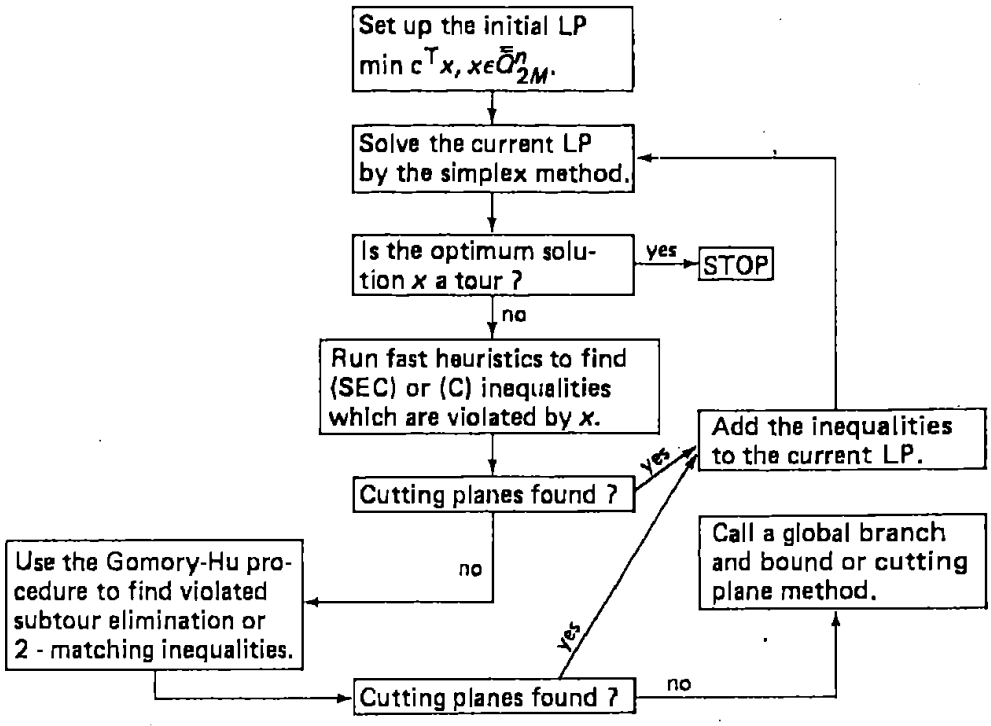

Fig. 2.

Of course, this cutting plane procedure (without calling a globally convergent method) does not always find a minimum tour, but the bounds for the minimum tour usually are so good that few iterations of a final branch and bound procedure suffice to obtain the optimum solution. Here one could also use general Gomory cutting planes instead. Variations of

this method have been implemented (cf. Grötschel (1980a), Miliotis (1978), Padberg and Hong (1980), Crowder and Padberg (1980)) and have shown very good computational results. The largest STSP solved to date is a 318-city problem, see Crowder and Padberg (1980). This is about three times the size that can successfully be attacked by the Lagrangean relaxation method described in 6.5 and 7.15. An interesting combination of cutting plane and Lagrangean techniques for the ATSP can be found in Balas and Christofides (1981) and for the set covering problem in Balas and Ho (1980).

The cutting plane approach described for the STSP here is certainly not restricted to the STSP. The same principles can be applied to any other hard problem provided sufficiently many facets of the corresponding polytope and reasonable cutting plane generation procedures are available.

The list of techniques describet in this paper for obtaining bounds for the optimum solution of hard problems (LP-relaxation, Langrangean relaxation, etc.) is not at all exhaustive. Many other methods have been developed which use mathematical tools which are quite different from the ones discussed before. For instance, procedures have been discussed in the literature which are based on nontrivial results from algebraic geometry or topology. A nice discussion of such methods with respect to the stable set problem can be found in Lovász (1981).

The future code development for hard problems will probably consist of the design of hybrid algorithms consisting of various heuristics, cutting plane and branch and bound techniques combined with Lagrangean relaxation and other bounding methods. These algorithms will certainly be rather complex but there is strong hope that intensive research will lead to methods which can solve many of the hard problems arising from practical applications with reasonable computational effort.

References

A. V. Aho, J. E. Hopcroft and J. D. Ulman, *The Design and Analysis of Computer Algorithms* (Addison Wesley, Reading, MA, 1974).

A. I. Ali, R. V. Helgason, J. L. Kennington and H. S. Lall, Primal simplex network codes: state-of-the-art implementation technology, *Networks* 8 (1978) 315–339.

K. I. Appel and W. Haken, Every planar map is four colorable, *Bull. Amer. Math. Soc.* 82 (1976) 711–712.

A. Bachem, Concepts of algorithmic computation, *This book, Part I, Chapter 1.*

E. Balas and N. Christofides, A restricted Lagrangean approach to the travelling salesman problem, *Math. Programming* 21 (1981) 19–46.

E. Balas and A. Ho, Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study, *Math. Progr. Studies* Vol. 12 (1980) 37–60.

E. Balas and M. W. Padberg, Set partitioning, in: B. Roy, ed., *Combinatorial Programming: Methods and Applications* (Reidel, Dordrecht, 1975) 205–258.

E. Balas and E. Zemel, Critical cutsets of graphs and canonical facets of set-packing polytopes, *Mathematics of Operations Research* 2 (1977) 15–19.

F. Barahona, On the complexity of max cut, Raport de Recherche No. 186, IMAG, Université de Grenoble (1980).

M. Behzad, G. Chartrand and L. Lesniak-Foster, *Graphs and Digraphs*, (Prindle, Weber & Schmidt, Boston, 1979).

C. Berge, *Graphs and Hypergraphs* (North-Holland, Amsterdam, 1973).

B. Bollobás, *Extremal Graph Theory* (Academic Press, London, 1978).

J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications* (Macmillan, London, 1976).

M. Boulala and J. P. Uhry, Polytope des independants d'un graphe serieparallele, *Discrete Mathematics* 27 (1979) 225–243.

R. E. Burkard, *Methoden der ganzzahligen Optimierung* (Springer, Wien, 1972).

R. E. Burkard, Subgradient methods in combinatorial optimization, in: U. Pape, ed., *Graphs, Data Structures, Algorithms* (Hauser, München, 1980) 141–151.

R. E. Burkard and U. Derigs, *Assignment and Matching Problems: Solution Methods with* FORTRAN *Programs*, Lecture Notes in Economics and Mathematical Systems Vol. 184 (Springer, Berlin-New York, 1980).

A. K. Chandra, D. S. Hirschberg and C. K. Wong, Approximate algorithms for some generalized knapsack problems, *Theoretical Computer Science* 3 (1976) 293–304.

D. Cheriton and R. E. Tarjan, Finding minimum spanning trees, *SIAM J. Comput.* 5 (1976) 724–742.

T.-Y. Cheung, Computational comparison of 8 methods for the maximum network flow problem, *ACM Transactions on Mathematical Software* 6 (1980) 1–16.

N. Christofides, *Graph Theory, an Algorithmic Approach* (Academic Press, New York, 1975).

N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Working Paper, Carnegie-Mellon University, Pittsburgh (February 1976).

N. Christofides, A. Mingozzi and P. Toth, The vehicle routing problem, in: N. Christofides et al., eds., *Combinatorial Optimization* (Wiley, New York, 1979) 315–338.

V. Chvátal, Edmonds polytopes and weakly hamiltonian graphs, *Mathematical Programming* 5 (1973) 29–40.

V. Chvátal, On certain polyopes associated with graphs, *J. Comb. Theory B* 18 (1975) 138–154.

E. G. Coffman jr., editor, *Computer and Job-shop Scheduling Theory* (Wiley, New York, 1976).

R. W. Conway, W. L. Maxwell and L. W. Miller, *Theory of Scheduling* (Addison Wesley, Reading, MA, 1967).

S. A. Cook, The complexity of theorem proving procedures, *Proc. Third Annual ACM Symposium on Theory of Computing* (1971) 151–158.

G. Cornuejols, M. L. Fisher and G. L. Nemhauser, Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms, *Management Science* 23 (1977) 789–810.

G. Cornuejols and W. R. Pulleyblank, A matching problem with side conditions, *Discrete Mathematics* 29 (1980a) 135–159.

G. Cornuejols and W. R. Pulleyblank, The travelling salesman problem and {0, 2}-matchings, Report 80172-OR, Institut für Operations Research, Universität Bonn (1980b) to appear in *Annals of Discrete Mathematics* Vol. 16 (1982).

H. P. Crowder and M. W. Padberg, Solving large-scale symmetric travelling salesman problems to optimality, *Management Science* 26 (1980) 495–509.

W. H. Cunningham and A. B. Marsh III, A primal algorithm for optimum matching, *Math. Programming Studies* Vol. 8 (1978) 50–72.

G. B. Dantzig, D. R. Fulkerson and S. M. Johnson, Solution of a large-scale travelling salesman problem, *Operations Research* 2 (1954) 393–410.

M. D. Devine, A model for minimizing the cost of drilling dual completion oil wells, *Management Science* 20 (1973) 532–535.

E. W. Dijkstra, A note on two problems in connection with graphs, *Numer. Mathematik* 1 (1959) 269–271.

J. Edmonds, Maximum matching and a polyhedron with (0, 1) vertices, *J. Res. Nat. Bur. Stand.* 69 B (1965) 125–130.

J. Edmonds, Optimum branchings, *Journ. of Research of the National Bureau of Standards* 71 B (1967) 1396–1400.

J. Edmonds, Submodular functions, matroids and certain polyhedra, in: R. Guy et al., eds., *Combinatorial Structures and their Applications* (Gordon and Breach, New York, 1970) 69–87.

J. Edmonds, Matroids and the greedy algorithm, *Mathematical Programming* 1 (1971) 127–136.

J. Edmonds, Matroid intersection, *Annals of Discrete Mathematics* 4 (1979) 39–49.

J. Edmonds and E. L. Johnson, Matching: a well-solved class of integer linear programs, in: R. K. Guy et al., eds., *Combinatorial Structures and Their Applications* (Gordon and Breach, New York, 1970) 89–92.

M. L. Fisher, Optimal solution of scheduling problems using Lagrangean multipliers: Part I, *Operations Research* 21 (1973) 1114–1127.

M. L. Fisher, Lagrangean relaxation methods for combinatorial optimization, *Management Science* 27 (1981) 1–18.

M. L. Fisher, Worst-case analysis of heuristic algorithm, *Management Science* 26 (1980) 1–17.

M. L. Fisher, G. L. Nemhauser and L. A. Wolsey, An analysis of approximations for maximizing submodular set functions II, *Math. Programming Study* 8 (1978) 73–87.

M. L. Fisher, W. D. Northup and J. F. Shapiro, Using duality to solve discrete optimization problems, *Mathematical Programming Studies* Vol. 3 (1975) 56–94.

L. R. Ford and D. R. Fulkerson, *Flows in Networks* (Princeton University Press, Princeton, NJ, 1962).

A. Frank, A weighted matroid intersection algorithm, *J. of Algorithms* (1981) to appear.

A. A. Fridman, Modern trends in discrete optimization, *Matekon* XV (1978) 30–57.

A. Frieze, Worst-case analysis of algorithms for the travelling salesman problem, *Operations Research Verfahren* 32 (1979) 93–112.

D. R. Fulkerson, Blocking and anti-blocking pairs of polyhedra, *Mathematical Programming* 1 (1971) 168–194.

P. Gács and L. Lovász, Khachian's algorithm for linear programming, *Mathematical Programming Studies* 14 (1981) 61–68.

M. R. Garey and D. S. Johnson, Strong NP-completeness results: motivation, examples, and implications, *J. Assoc. Comput. Mach.* 25 (1978) 499–508.

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Farncisco, 1979).

R. S. Garfinkel and G. L. Nemhauser, *Integer Programming* (Wiley, London, 1972).

B. Gavish, On obtaining the 'best' multipliers for a Lagrangean relaxation for integer programming, *Comput. & Ops. Res.* 5 (1978) 55–71.

D. H. Gensch, An industrial application of the travelling salesman's subtour problem, *AIIE Transactions* 10 (1978) 362–370.

A. M. Geoffrion, Lagrangean relaxation for integer programming, *Math. Programming Studies* Vol. 2 (1974) 82–114.

R. Giles, Submodular functions, graphs and integer polyhedra, Doctoral Thesis, University of Waterloo, Canada (1975).

R. Giles, Facets and other faces of branching polyhedra, *Proceedings of the Fifth Hungarian Combinatorial Colloquium*, Bolyai Janos Mathematical Society (1978) 401–418.

B. Glover, T. Klastorin and D. Klingman, Optimal weighted ancestry relationships, *Management Science* 20 (1974) 1190–1193.

F. Glover and D. Klingman, Network applications in industry and government, *AIIE Transactions* 9 (1977) 363–376.

F. Glover, D. Klingman, J. Mote and D. Whitman, Comprehensive computer evaluation and enhancement of maximum flow algorithms, Research Report 356, Center of Cybernetic Studies, The University of Texas, Austin (October 1979).

B. Golden, L. Bodin, T. Doyle and W. Stewart jr., Approximate travelling salesman algorithms, *Operations Research* 28 (1980) 694–711.

R. Gomory and T. C. Hu, Multi-terminal networks flow, *J. SIAM* 9 (1961) 551–570.

M. Gondran and M. Minoux, *Graphes et Algorithmes* (Editions Eyrolles, Paris, 1979).

R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics* 5 (1979) 287–326.

M. Grötschel, *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme* (Hain, Meisenheim, 1977a).

M. Grötschel, The monotone 2-matching polytope on a complete graph, *Operations Research Verfahren* 26 (1977b) 72–84.

M. Grötschel, Strong blocks and the optimum branching problem, in: L. Collatz and W. Wetterling, eds., *Numerische Methoden bei graphentheoretischen und kombinatorischen Problemen*, Band 2, ISNM Vol. 46 (Birkhäuser, Basel, 1979) 112–121.

M. Grötschel, On the symmetric travelling salesman problem: solution of a 120 city problem, *Mathematical Programming Studies* Vol. 12 (1980a) 61–77.

M. Grötschel, On the monotone symmetric travelling salesman problem: hypohamiltonian/

hypotraceable graphs and facets, *Mathematics of Operations Research* 5 (1980b) 285–292.

M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* 1 (1981a) 169–197.

M. Grötschel, L. Lovász and A. Schrijver, Polynomial algorithms for perfect graphs, Report No. 81176, Institut für Operations Research, Universität Bonn (1981b) to appear in *Annals of Discrete Mathematics*.

M. Grötschel and M. W. Padberg, On the symmetric travelling salesman problem I: inequalities, *Mathematical Programming* 16 (1979a) 265–280.

M. Grötschel and M. W. Padberg, On the symmetric travelling salesman problem II: lifting theorems and facets, *Mathematical Programming* 16 (1979b) 281–302.

M. Grötschel and W. R. Pulleyblank, Clique free inequalities and the symmetric travelling salesman problem, Report No. 81196, Institut für Operations Research, Universität Bonn (1981).

G. H. Handler and P. B. Mirchandani, *Location on Networks: Theory and Algorithms* (MIT Press, Cambridge, MA, 1979).

D. Hausmann, editor, *Integer Programming and Related Areas: A Classified Bibliography* 1976–1978, Lecture Notes in Economics and Mathematical Systems Vol. 160 (Springer, Berlin, 1978).

K. Helbig Hansen and J. Krarup, Improvements of the Held–Karp algorithm for the symmetric travelling salesman problem, *Mathematical Programming* 7 (1974) 87–96.

M. Held and R. M. Karp, The traveling-salesman problem and minimum spanning trees, *Operations Research* 18 (1970) 1138–1182.

M. Held and R. M. Karp, The traveling-salesman problem and minimum spanning trees: Part II, *Mathematical Programming* 1 (1971) 6–26.

M. Held, P. Wolfe and H. P. Crowder, Validation of subgradient optimization, *Math. Programming* 6 (1974) 62–88.

J. E. Hopcroft and R. E. Tarjan, Efficient planarity testing, *SIAM J. Comput.* 2 (1973) 225–231.

T. C. Hu, *Integer Programming and Network Flow* (Addison-Wesley, Reading, MA, 1969).

O. H. Ibarra and C. E. Kim, Fast approximation algorithms for the knapsack and subset sum problems, *J. Assoc. Comput. Mach.* 22 (1975) 463–468.

O. H. Ibarra and C. E. Kim, Approximation algorithms for certain scheduling problems, *Math. of Operations Research* 3 (1978) 197–204.

T. A. Jenkyns, The efficacy of the 'greedy' algorithm, *Proc. 7th S–E Conf. Combinatorics, Graph Theory and Computing*, Utilitas Math., Winnipeg (1976) 341–350.

R. Kaas, A branch and bound algorithm for the acyclic subgraph problem, Report AE 20/80, University of Amsterdam (1980).

R. Kannan and B. Korte, Approximative combinatorial algorithms, Report No. 78107-OR, Institut für Operations Research, Universität Bonn (1978).

R. M. Karp, The probabilistic analysis of some combinatorial search algorithms, Memorandum No. ERL-M 581, University of California, Berkeley (April 1976).

R. M. Karp, Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane, *Mathematics of Operations Research* 2 (1977) 209–224.

R. M. Karp, A patching algorithm for the nonsymmetric traveling-salesman problem, *SIAM J. Comput.* 8 (1979) 561–573.

R. M. Karp and C. Papadimitriou, On linear characterizations of combinatorial optimization problems, Working Paper, University of California, Berkeley (1980) to appear in *SIAM J. Comput.*

C. Kastning, editor, *Integer Programming and Related Areas: A Classified Bibliography*, Lecture Notes in Economics and Mathematical Systems Vol. 128 (Springer, Berlin, 1976).

J. L. Kennington and R. V. Helgason, *Algorithms for Network Programming* (Wiley, New York, 1980).

L. G. Khachian, A polynomial algorithm in linear programming, *Doklady Academii Nauk SSSR* 244 (1979) 1093–1096 (English translation: *Soviet Math. Dokl.* 20 (1979) 191–194).

V. Klee, Combinatorial optimization: what is the state of the art?, *Mathematics of Operations Research* 5 (1980) 1–26.

B. Korte, Approximation algorithms for discrete optimization problems, *Annals of Discrete Mathematics* 4 (1979) 85–120.

B. Korte and D. Hausmann, An analysis of the greedy heuristic for independence systems, *Annals of Discrete Mathematics* 2 (1978) 65–74.

B. Korte and W. Oberhofer, Zwei Algorithmen zur Lösung eines komplexen Reihenfolgeproblems, *Unternehmensforschung* 12 (1968) 217–231.

B. Korte and R. Schrader, On the existence of fast approximation schemes, in: O. Mangasarian, R. R. Meyer, S. M. Robinson, eds., *Nonlinear Programming* Vol. 4 (Academic Press, New York, 1981) 415–437.

J. Krarup and P. Pruzan, Selected families of location problems, *Annals of Discrete Mathematics* 5 (1979) 327–388.

J. B. Kruskal, On the shortest spanning subtree of a graph and the travelling salesman problem, *Proc. of the American Math. Society* 7 (1956) 48–50.

H. W. Kuhn, The hungarian method for the assignment problem, *Naval Res. Logist. Quart.* 3 (1955) 253–258.

E. L. Lawler, *Combinatorial Optimization: Networks and Matroids* (Holt, Rinehart & Winston, 1976).

E. L. Lawler, Matroid intersection algorithms, *Math. Programming* 9 (1975) 31–56.

E. L. Lawler, Fast approximation algorithms for knapsack problems, *Math. of Oper. Res.* 4 (1979) 339–356.

H. W. Lenstra, jr., The acyclic subgraph problem, BW 26/73, Mathematisch Centrum, Amsterdam (July, 1973).

J. K. Lenstra, Sequencing by enumerative methods, Mathematisch Centrum, Amsterdam (1976).

J. K. Lenstra and A. H. G. Rinnooy Kan, Some simple applications of the travelling salesman problem, *Operational Research Quaterly* 26 (1975) 717–733.

J. K. Lenstra and A. H. G. Rinnooy Kan, Complexity of vehicle routing and scheduling problems, *Networks* 11 (1981) 221–227.

J. M. Lewis and M. Yannakakis, The node-deletion problem for hereditary properties is NP-complete, *Journal of Computer and System Sciences* 20 (1980) 219–230.

S. Lin and B. W. Kernigham, An effective algorithm for the traveling-salesman problem, *Operations Research* 21 (1973) 498–516.

L. Lovász, Bounding the independence number of a graph, Report No. 81175, Institut für Operations Research, Universität Bonn (1981) to appear in *Annals of Discrete Mathematics* Vol. 16 (1982).

G. S. Lueker, Maximization problems on graphs with edge weights chosen from a normal distribution, *Proc. Xth Annual ACM Symp. on Theory of Computing* (1978).

J. F. Maurras, Some results on the convex hull of the hamiltonian cycles of symmetric complete graphs, in: B. Roy, ed., *Combinatorial Programming, Methods and Applications* (Reidel, Dordrecht, 1975).

J. F. Maurras, Polytopes a sommets dans $\{0, 1\}^n$, Thèse Université Paris VII (1976).

P. Miliotis, Using cutting planes to solve the symmetric travelling salesman problem, *Mathematical Programming* 15 (1978) 177–188.

G. J. Minty, On maximal independent sets of vertices in claw-free graphs, *Journal of Combinatorial Theory B* 28 (1980) 284–304.

G. L. Nemhauser and L. E. Trotter jr., Properties of vertex packing and independence system polyhedra, *Mathematical Programming* 6 (1974) 48–61.

G. L. Nemhauser and L. E. Trotter jr., Vertex packings: structural properties and algorithms, *Mathematical Programming* 8 (1975) 232–248.

G. L. Nemhauser, L. A. Wolsey and M. L. Fisher, An analysis of algorithms for maximizing submodular set functions I, *Math. Programming* 14 (1978) 265–294.

M. W. Padberg, On the facial structure of set packing polyhedra, *Math. Programming* 5 (1977) 199–215.

M. W. Padberg and S. Hong, On the symmetric travelling salesman problem: a computational study, *Mathematical Programming Studies* Vol. 12 (1980) 78–107.

M. W. Padberg and M. R. Rao, Odd minimum cut-sets and b-matchings, Working Paper, Graduate School of Business Administration, New York University (revised version) (August 1980) to appear in *Math. Oper. Res.*

B. T. Polyak, A general method for solving extremum problems, *Soviet Math. Doklady* 8 (1967) 593–597.

B. T. Polyak, Subgradient methods: a survey of Sovjet research, in: C. Lemarechal and R. Mifflin, eds., *Nonsmooth Optimization* (Pergamon, Oxford, 1978) 5–30.

C. Papadimitriou, On the complexity of edge traversing, *J. Assoc. Comput. Mach.* 23 (1976) 544–554.

R. C. Prim, Shortest connection networks and some generalizations, *Bell System Techn. J.* 36 (1957) 1389–1401.

W. R. Pulleyblank, Minimum node covers and 2-bicritical graphs, *Math. Programming* 17 (1979) 91–103.

W. R. Pulleyblank, Faces of matching polyhedra, Doctoral Thesis, University of Waterloo, Canada (1973).

A. H. G. Rinnooy Kan, *Machine Scheduling Problems: Classification, Complexity and Computations* (Nijhoff, The Hague, 1976).

R. T. Rockafellar, *La Théorie des Sous-gradients et ses Applications à l'Optimisation* (Les Presses de l'Université Montréal, 1979); English translation: R & E 1, (Heldermann, Berlin, 1981).

D. J. Rosenkrantz, R. E. Stearns and P. M. Lewis, An analysis of several heuristics for the traveling salesman problem, *SIAM J. Comput.* 6 (1977) 563–581.

S. Sahni, Algorithms for scheduling independent tasks, *J. Assoc. Comput. Mach.* 1 (1976) 116–127.

R. Schrader, Ellipsoidal algorithms, *This book, Part II, Chapter 4.*

N. Z. Shor, Application of the gradient method for the solution of network transportation problems, *Notes Scientific Seminar on Theory and Application of Cybernetics and Operations Research*, Academy of Sciences, Kiev (1962) (in Russian).

P. Slater, Inconsistencies in a schedule of paired comparisons, *Biometrika* 48 (1961) 303–312.

T. H. C. Smith and G. L. Thompson, A Lifo implicit enumeration search algorithm for the symmetric traveling salesman problem using Held and Karp's 1-tree relaxation, *Annals of Discrete Math.* 1 (1977) 479–493.

H. Steckhan and R. Thome, Vereinfachungen der Eastmanschen Branch-and-Bound-Lösung für symmetrische Traveling Salesman Probleme, *Operations Research Verfahren* 14 (1972) 360–389.

R. E. Tarjan, Finding optimum branchings, *Networks* 7 (1977) 25–35.

L. E. Trotter jr., A class of facet producing graphs for vertex packing polyhedra, Technical Report No. 78, Dep. of Adm. Sciences, Yale University (Feb. 1974).

A. Volgenant and R. Jonker, A branch & bound algorithm for the symmetric traveling salesman problem based on 1-tree relaxation, Report AE 5/80, University of Amsterdam (1980).

B. W. Weide, Random graphs and graph optimization problems, *SIAM J. Comput.* 9 (1980) 552–557.

D. J. A. Welsh, *Matroid Theory* (Academic Press, New York, 1976).

L. A. Wolsey, Cutting plane methods, in: A. G. Holzman, ed., *Operations research Methodology* (Dekker, New York, 1979) 441–466.

M. Yannakakis, Edge deletion problems, Technical Report 249, Princeton University, Department of EECS (1979).

M. Yannakakis and F. Gavril, Edge dominating sets in graphs, *SIAM J. Appl. Math.* 38 (1980) 364–372.