Trotter

# LP-BASED COMBINATORIAL PROBLEM SOLVING

K. HOFFMAN

*Operations Research Division, National Bureau of Standards, Gaithersburg, Maryland 20899, USA*

and

M. PADBERG

*Graduate School of Business Administration, New York University, New York, New York, USA*

## Abstract

A tutorial outline of the polyhedral theory that underlies linear programming (LP)-based combinatorial problem solving is given. Design aspects of a combinatorial problem solver are discussed in general terms. Three computational studies in combinatorial problem solving using the polyhedral theory developed in the past fifteen years are surveyed: one addresses the symmetric traveling salesman problem, another the optimal triangulation of input/output matrices, and the third the optimization of large-scale zero-one linear programming problems.

## Keywords and phrases

Integer programming, polyhedral theory, facets, cutting planes, traveling salesman problems, triangulation of matrices, large-scale zero-one problems, software design, computational testing.

## 1. Introduction

Several recent computational studies in combinatorial optimization have focused on applying the polyhedral theory developed in the past fifteen years to actual problem solving. The resulting computational advances in the state-of-the-art have been considerable, especially as far as the *exact* optimization of large-scale combinatorial

The second author's work was supported in part by NSF Grant No. MCS 8304634.
An earlier version of this paper was published in NATO ASI Series VF15, *Computational Mathematical Programming* (© Springer-Verlag, Berlin – Heidelberg, 1985).

Disc Math 124 (132-153) 1994

optimization problems is concerned. Indeed, a common aspect shared by all of the computational studies surveyed in this paper is that all of them address problems belonging to the class of NP-hard combinatorial optimization problems for which to date no technically good (or polynomially bounded) algorithms are known. Of course, these empirial studies will not resolve the fundamental question of whether or not exponential worst-case behavior is the best that we can expect. However, it is entirely possible that – as has been observed in several cases – a gulf exists between the empirical record and the theoretical (worst-case) analysis of numerical problem solving. For instance, the fact that the simplex method exhibits exponential performance on some artificially constructed problems has in no way discouraged the many users who rely daily on the simplex method for the optimization of linear programming problems having thousands of variables.

Mounting empirical evidence indicates that both pure and mixed integer programming problems can be solved to optimality in economically feasible computation times by methods derived from our improved understanding of the underlying polyhedral theory. This theory leads to new cutting-planes, that in a well-defined sense are best-possible cutting-planes and that are thus different from the traditional cutting-plane described in the pre-1985 textbooks on integer programming. While the theory developed is in most cases (still) incomplete – meaning that ~~as~~ some point in the calculation recourse must be made to enumerative methods such as branch and bound (with embedded linear programs) – the cutting planes derived from the polyhedral theory differ from the traditional ones in an additional important aspect: The latter ones are known to have a high density of non-zero coefficients and thus lead to explosive storage requirements, whereas the polyhedral approach leads to inequalities that in most cases preserve sparsity and have moderate storage requirements.

In sect. 2 of this paper, we give a tutorial outline of the polyhedral theory that underlies the computational studies referred to above. In the third section, we discuss some design aspects of a combinatorial problem solver in general terms. Much of the implementational detail is left out. For more detailed descriptions, see Bernal et al. [4], Padberg and Grötschel [59], and Crowder et al. [11]. In sects. 4, 5 and 6, we survey three computational studies concerning the symmetric traveling salesman problem, the optimal triangulation of input/output matrices, and the optimization of large-scale sparse zero-one linear problems without special structure. Space and time limitations prohibit us from discussing the recent papers by Martin and Schrage [45] and Van Roy and Wolsey [65] that address the optimization of large-scale *mixed* zero-one linear programming problems and are based on results on the facial structure of such problems developed in a series of recent paper by Baranyi et al. [2,3], Padberg et al. [58], and Van Roy and Wolsey [63,64].

## 2. Combinatorial problem solving and linear programming

Combinatorial optimization problems can be stated in the following way: Given a finite set $E$, a family $\mathcal{F}$ of subsets of $E$ and weights $c_e \in \mathbb{R}$ for all elements $e \in E$, we are interested in finding a member $F^*$ of $\mathcal{F}$ such that

$$c(F^*) = \sum_{e \in F^*} c_e \tag{2.1}$$

is as small (or alternatively, as large) as possible. To 'model' combinatorial optimization problems, one associates with every element $e \in E$ a variable $x_e$, i.e. a component of a vector $x \in \mathbb{R}^E$ indexed by $e$. (Note that for notational convenience, we write $\mathbb{R}^E$ rather than $\mathbb{R}^{|E|}$.) With every subset $F \subseteq E$ one associates a vector $x^F \in \mathbb{R}^E$, called the *incidence* or *characteristic* vector of $F$, defined as follows:

$$x_e^F = \begin{cases} 1 & \text{for } e \in F \\ 0 & \text{for } e \notin F. \end{cases} \tag{2.2}$$

Thus, to every subset $F \subseteq E$ there corresponds a unique zero-one vector in $\mathbb{R}^E$, and vice versa. For any family $\mathcal{F}$, we let $P_{\mathcal{F}}$ be the convex hull of the incidence vectors of all members of $\mathcal{F}$, i.e.,

$$P_{\mathcal{F}} = \operatorname{conv}\{x^F \in \mathbb{R}^E \mid F \in \mathcal{F}\}. \tag{2.3}$$

$P_{\mathcal{F}}$ is a polytope in $\mathbb{R}^E$ and every extreme point of $P_{\mathcal{F}}$ uniquely corresponds to a member of $\mathcal{F}$, and vice versa.

We can solve the combinatorial optimization problem (2.1) by solving the problem

$$\min\{c^T x \mid x \in P_{\mathcal{F}}\} \tag{2.4}$$

since, by the linearity of the objective function of (2.4), every optimum solution to (2.1) corresponds to an optimum extreme point solution of (2.4), and vice versa.

By the theorem of Weyl [67], there exists a finite (integer) matrix $H$ and a vector $h$ (of integers) such that

$$P_{\mathcal{F}} = \{x \in \mathbb{R}^E \mid Hx \leqslant h\} \tag{2.5}$$

holds and, consequently, our combinatorial optimization problem (2.1) becomes the linear program

$$\min\left\{c^T x \mid Hx \leqslant h\right\}. \tag{2.6}$$

Evidently, if we know the description of $P_{\mathcal{F}}$ by way of linear inequalities (2.5), then we can use linear programming methods such as the simplex method to solve the combinatorial optimization problem (2.4).

Of course, the description (2.5) of the polytope $P_{\mathcal{F}}$ is not unique, as one can append any non-negative linear combination of the rows of $Hx \leqslant h$ to this system without changing the solution set. One is thus led to consider *minimal* systems of linear inequalities that describe $P_{\mathcal{F}}$ (i.e. linear inequalities that define *facets* of the polytope $P_{\mathcal{F}}$). We recall that the dimension of $P_{\mathcal{F}}$, denoted $\dim P_{\mathcal{F}}$, is the affine rank of $P_{\mathcal{F}}$ minus one, where the affine rank of any set $S \subseteq \mathbb{R}^E$ is the cardinality of the largest *affinely independent* subset of $S$. An inequality $f^T x \leqslant f_0$, where $f \in \mathbb{R}^E$ and $f_0 \in \mathbb{R}$ defines a facet of $P_{\mathcal{F}}$ if we have

(i) $P_{\mathcal{F}} \subseteq \{x \in \mathbb{R}^E \mid f^T x \leqslant f_0\}$ and

(ii) $\dim(P_{\mathcal{F}} \cap \{x \in \mathbb{R}^E \mid f^T x = f_0\}) = \dim P_{\mathcal{F}} - 1$.

Property (i) is referred to as the 'validity' property of the inequality $f^T x \leqslant f_0$ with respect to the polytope $P_{\mathcal{F}}$, whereas property (ii) expresses the condition that the hyperplane $f^T x = f_0$ is generated by the (extreme) points of a facet of $P_{\mathcal{F}}$. In the linear description (2.5) of $P_{\mathcal{F}}$, we can drop all the rows of $Hx \leqslant h$ that do not define facets of $P_{\mathcal{F}}$. Thus, a minimal system of linear inequalities defining $P_{\mathcal{F}}$ is obtained. If $P_{\mathcal{F}}$ is of full dimension (i.e. if $\dim P_{\mathcal{F}} = |E|$), then such a minimal system is unique modulo multiplications of its inequalities by positive scalars. In the general case, i.e. if $\dim P_{\mathcal{F}} \leqslant |E|$, then such a minimal system is unique modulo multiplication by positive scalars *and* linear combinations of the equations defining the smallest affine subspace of $\mathbb{R}^E$ that contains $P_{\mathcal{F}}$.

There are several ways in which the family $\mathcal{F}$ of subsets of $E$ arises in actual combinatorial problem solving. Typical for *integer programming on graphs* is the situation that one looks for a well-defined combinatorial configuration in a graph (e.g. a maximum-weight stable (or independent) node-set or edge-set, a minimum-weight Hamiltonian cycle, or a minimum-weight clique-covering of all nodes). It is clear that in these cases, the ground-set $E$ corresponds either to all the nodes or to all the edges (or in certain instances to both all nodes and edges) of the given graph. The combinatorial configurations that one looks for define the elements of the family $\mathcal{F}$.

In other situtations, one starts with a system of linear inequalities in $n$ zero-one variables that correspond to the 'formulation' of some underlying 'real-world' problem arising in economics, the management of public or private enterprises, engineering, etc.

In this case, one has $E = \{1, \ldots, n\}$, and the family $\mathfrak{F}$ of subsets $F$ of $E$ is given by the correspondence

$$F \in \mathfrak{F} \iff x^F \in \{x \in \mathbb{R}^n \mid Ax \leqslant b,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\} \quad (2.7)$$

where $A$ is some given $m \times n$ matrix of rationals and $b$ is a vector with $m$ rational components. Of course, in the case of an integer program on a graph, one can always write down a 'formulation' of the problem by way of linear inequalities in zero-one variables as well, thus obtaining a linear system $Ax \leqslant b$ whose zero-one solutions correspond uniquely to the combinatorial configurations that one looks for in the graph.

We will now assume that – no matter what combinatorial problem is considered – we have a 'formulation' of the optimization problem at hand and denote by

$$D = \{x \in \mathbb{R}^n \mid Ax \leqslant b,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\} \quad (2.8)$$

the set of feasible solutions. Indeed, *any* formulation will do and will be called the 'user-supplied' formulation. The combinatorial optimization problem can then be stated as

$$\min \{c^T x \mid x \in D\}. \quad (2.9)$$

Letting

$$\mathfrak{F} = \{F \subseteq \{1, \ldots, n\} \mid x^F \in D\},$$

where $x^F$ is the characteristic vector of $F$, we have that

$$P_{\mathfrak{F}} = \operatorname{conv}(D), \quad (2.10)$$

where conv $(D)$ means the convex hull of $D$ in $\mathbb{R}^n$, i.e. the convexification of the discrete set $D$ in $\mathbb{R}^n$.

The following concept of ***relaxation*** is typical for linear programming approaches to combinatorial problem solving. Denoting

$$P = \{x \in \mathbb{R}^n \mid Ax \leqslant b,\ 0 \leqslant x_j \leqslant 1 \text{ for } j = 1, \ldots, n\}, \quad (2.11)$$

we obtain a polytope that satisfies

$$P_{\mathcal{J}} \subseteq P \tag{2.12}$$

and thus we have the easy (but important) consequence

$$\min\{c^T x \mid x \in P_{\mathcal{J}}\} \geqslant \min\{c^T x \mid x \in P\}. \tag{2.13}$$

Furthermore, one can readily prove that every extreme point of $P_{\mathcal{J}}$ is an extreme point of $P$, but of course *not* conversely since $P$ will, in general, have many more extreme points than $P_{\mathcal{J}}$. However, if the second problem in (2.13) produces a zero-one solution vector, then we have solved the combinatorial optimization problem (2.9).

There are only few instances where the preceding (naive) relaxation idea has been proven to solve the combinatorial optimization problem for all possible choices of the objective function; for instance, this is the case when the matrix $A$ is totally unimodular or perfect (see Padberg [50] for a related survey). But even if the relaxation procedure does not work for all possible choices of the objective function, it *may* still work for the particular objective function of the user-supplied formulation of the optimization problem at hand. Taking this chance seems worthwhile since we know how to solve linear programming problems efficiently. Indeed – except for certain purely enumerative attempts at solving combinatorial optimization problems – all *proven* methods for solving general combinatorial optimization problems use this relaxation idea as their point of departure. For instance, the only integer programming method implemented up to 1982 in the form of *commercially* available software – branch and bound – uses precisely this naive relaxation of the combinatorial optimization problem. (For post-1982 developments, see IBM [28].) The trouble with such ordinary branch and bound (empirically observed in many instances of combinatorial optimization problems) is that the 'integrality gap' in relation (2.13) may be very large. As a case in point, Crowder et al. [11] report on a zero-one problem with 548 variables where the left-hand side of (2.13) produced a value of 8691.0, while the right-hand side of (2.13) produced a value of 315.3 for the user-supplied formulation. Predictably, commercially available, state-of-the-art branch and bound codes failed to close this enormous integrality gap even when the computer (an IBM 370/168) was permitted to run for several hours of CPU time. The way out of this situation is to use *improved* formulations of the combinatorial optimization problem employing the polyhedral approach to combinatorial optimization outlined above.

Returning to the 'ideal' situation where we know a complete linear description (2.5) of the underlying polytope, it is in general out of the question to generate and store all of the data of the linear program (2.6) in the computer. This is so because – while $H$ is guaranteed to be a *finite* matrix – it can still be expected to be extremely large. Indeed, in most cases of interest, the number of rows of $H$ is known to grow exponentially and worse with the parameter $n$, (i.e. with number of variables of the problem). An alternative to *enumerating* all known rows of $H$ is to solve periodically the following problem:

(2.14) *Facet-identification problem:* Given a point $\bar{x} \in \mathbb{R}^E$ and a polytope $P_{\mathcal{T}}$, find a facet-defining linear inequality $f^T x \leqslant f_0$ for $P_{\mathcal{T}}$ which is violated by $\bar{x}$ (i.e. such that $f^T \bar{x} > f_0$ holds), or prove that no such inequality exists (i.e. that $\bar{x} \in P_{\mathcal{T}}$ holds).

Using (2.14) as a *subroutine* in a linear programming code, we can generate violated facet-inequalities 'on the fly' as we need them to cut off fractional linear programming optima. More precisely, we think of the following iterative procedure for solving combinatorial problems (see e.g. Padberg and Grötschel [59]).

(2.15) *Relaxation method for combinatorial problems*

(2.15.1) (Initialization). Let $(\mathrm{LP}_0)$ be a valid linear programming relaxation of the combinatorial problem under consideration (e.g. the user-supplied formulation without the integer restrictions). Set $k = 0$ and go to (2.15.2).

(2.15.2) (LP-solver). Solve $(\mathrm{LP}_k)$, let $x^k$ be an optimal solution to $(\mathrm{LP}_k)$ and go to (2.15.3).

(2.15.3) (Facet-identification). Solve the facet-identification problem (2.14) for $x^k$ and the polytope $P_{\mathcal{T}}$.

(2.15.3.1) If one or more violated facet-inequalities for $P_{\mathcal{T}}$ are found, define $(\mathrm{LP}_{k+1})$ to be $(\mathrm{LP}_k)$ amended by the violated facet-inequalities, replace $k + 1$ by $k$, go to (2.15.2).

(2.15.3.2) If a violated facet-inequality does not exist, stop.

If a finitely convergent LP solver is used in step (2.15.2), then the relaxation method (2.15) is finitely convergent. Moreover, if the simplex method is used in step (2.15.2), then one readily shows that termination in (2.15.3.2) occurs at a zero-one feasible solution to the combinatorial optimization problem. On the theoretical side, it has been proven independently by several authors that the combinatorial optimization problem (2.4) is solvable in polynomial time if and only if the facet-identification problem (2.14) is solvable in polynomial time for rational $\bar{x} \in \mathbb{R}^E$. (See Grötschel et al. [20], Padberg and Rao [56], and Karp and Papadimitriou [31]; see also Padberg and Grötschel [59] for a related summary.)

This relaxation method (even when the theoretically non-polynomial simplex method is used) has proven computationally efficient also in cases where a complete linear description (2.5) of the associated polytope is known. For instance, for the problem of finding a maximum-weight matching in a graph, Edmonds [14] has found a complete linear description of the polytope. An algorithm utilizing the facet-identification problem (2.14) as a subroutine was given by Padberg and Rao [57]. This algorithm was implemented by Grötschel and Holland [19a] and found to have an equal or better empirical performance on large-scale problems than the theoretically proven polynomial algorithm by Edmonds [14] as modified by Burkard and Derigs [6]. There are other classes of problems where the above algorithmic idea works.

In general, however, we know of many problems for which complete linear descriptions (2.5) of the associated polytope are unknown to date. Indeed, in view of the NP-hard characteristic of the general zero-one problem, such *complete* characterizations may prove to be elusive forever for many practically important combinatorial optimization problems. Yet, just as ordinary branch and bound used only the most naive form of relaxation of the combinatorial optimization problem (and proved to be successful in solving sufficiently many practical problems), we can utilize the above polyhedral description to improve upon the naive relaxation of a combinatorial optimization problem by adding a *partial* list of facet-inequalities in the relaxation method (2.15). If it so happens that we do not *find* a violated facet-inequality in this method, either because it does not exist or because we just do not yet know the full description, or because we do not yet know how to *algorithmically* generate a known facet-inequality, then we can at termination of (2.15.3.2) still resort to branch and bound and have an *improved* chance of proving optimality by the latter method due to a much 'tighter' relaxation.

The *challenge* in combinatorial optimization is thus to find a sufficiently large sub-system $H^*x \leqslant h^*$ of the complete system $Hx \leqslant h$ of the polytope $P_{\mathcal{J}}$ that permits one to reduce the integrality gap in relation (2.13). For we choose such a subsystem $H^*x \leqslant h^*$ carefully, then we obtain

$$\min\{c^Tx \mid x \in P_{\mathcal{J}}\} \geqslant \min\{c^Tx \mid H^*x \leqslant h^*\} \geqslant \min\{c^Tx \mid x \in P\}. \quad (2.16)$$

The smaller the integrality gap between the 'improved' linear formulation and the optimization problem in zero-one variables, the more likely the chances of optimizing the problem by branch and bound. As a case in point, in the example by Crowder et al. [11] referred to above, sufficiently many inequalities were generated automatically to yield an objective function value of 8643.5 for the 'improved' formulation. As it so happened, with the remaining integrality gap being so small, the entire optimization (including a branch and bound phase) of this previously unsolvable problem took 0.91 minutes of CPU time on an IBM 370/168 starting from scratch with the user-supplied formulation.

We conclude this tutorial outline by giving a general example that illustrates the major points of this section.

(2.17) *Example:* Consider the problem of finding a maximum-weight stable (or independent) node set in a finite undirected graph (without loops and multiple edges) $G = (V,E)$ with node set $V$ and edge set $E$. We recall that $S \subseteq V$ is a stable set if

$v, w \in S$ implies $(v, w) \notin E$, i.e. $v$ and $w$ are not connected by an edge of $G$. In terms of the general notation of this section, the ground set $E$ is the node set $V$ of $G$, while the family $\mathcal{F}$ is the set of all stable subsets of $V$. $P_{\mathcal{F}}$ is the *vertex-packing polytope*. Introducing zero-one variables $x_v$ for all $v \in V$, we can formulate the problem as follows:

$$\text{(P1)} \qquad \begin{array}{lll} \text{maximize} & \sum_{v \in V} c_v x_v & \\ \text{subject to} & x_v + x_w \leqslant 1 & \text{for all } e = (v, w) \in E \\ & x_v = 0 \text{ or } 1 & \text{for all } v \in V, \end{array}$$

where $c_v$ for $v \in V$ are the node weights for a given problem instance. The formulation (P1) correctly models the problem, but is rather 'weak' from a linear programming point of view. Let $C$ be the node set of any clique of $G$ (a node set of a maximal *complete* subgraph of $G$) and $\mathcal{G}$ be any family of cliques of $G$ that *cover all* edges of $G$. That is, $\mathcal{G}$ may (but need not) be the family of all cliques of $G$. Then we can formulate the same problem as follows:

$$\text{(P2)} \qquad \begin{array}{lll} \text{maximize} & \sum_{v \in V} c_v x_v & \\ \text{subject to} & \sum_{v \in C} x_v \leqslant 1 & \text{for all } C \in \mathcal{G} \\ & x_v = 0 \text{ or } 1 & \text{for all } v \in V. \end{array}$$

Denote $Z_G$ the optimal objective function value of (P1) and (P2), $Z_{\text{P1}}$ the optimal objective function value of the (naive) linear programming relaxation of (P1) and $Z_{\text{P2}}$ likewise for the formulation (P2). Noting that we are maximizing, we obtain, as in (2.16), the following inequalities:

$$Z_G \leqslant Z_{\text{P2}} \leqslant Z_{\text{P1}} . \tag{2.18}$$

Moreover, Padberg [49] has shown that the inequalities of (P2) define facets of the polytope $P_{\mathcal{F}}$, whereas the inequalities of (P1), in general, do not. Now consider the special case of finding a maximum-cardinality stable set in an odd anti-hole (the complement graph of an odd cycle without chords) having $n$ nodes with $n \geqslant 7$ (see Padberg [52]). One readily shows that $Z_{\text{P1}} = n/2$, since $x_v = 1/2$ for all $v \in V$ is a feasible solution of (P1) satisfying all inequalities (except non-negativity) as equations, and an odd anti-hole is a regular graph of degree $n - 3$ having $n(n - 3)/2$ edges. Taking $\mathcal{G}$ to be the family of all maximum-cardinality cliques of the odd anti-hole, one finds $Z_{P2} = 2 + 2/(n - 1)$, while $Z_G = 2$. The maximum-cardinality cliques are, of course, a proper subset of all cliques of an odd anti-hole and their totality does not define the associated polytope $P_{\mathcal{J}}$. (See Padberg [49,52,53], Trotter [62], and others.) However, this example demonstrates the basic idea: A proper subset $H^*x \leqslant h^*$ of the totality of all facet-defining inequalities $Hx \leqslant h$ of the polytope $P_{\mathcal{J}}$ can have a dramatic effect upon the goodness of the resulting (naive) linear programming relaxation of the combinatorial optimization problem because the second formulation permits one to obtain $Z_G$ by simply rounding down. From a user's point of view, of course, any formulation of the problem 'will do'. In terms of the relaxation method (2.15), formulation (P1) provides the first linear programming relaxation ($\mathrm{LP}_0$). Then a combinatorial problem solver based on (2.15) generates facet-defining inequalities to yield formulation (P2). Thus an improved formulation of the combinatorial problem is generated automatically. Branch and bound or any other enumeration method may still be needed since, at present, as for most integer programming problems of practical interest, the complete description of the vertex-packing polytope is unknown.

## 3. Design aspects of a combinatorial problem solver

As discussed in sect. 2, we will assume that the combinatorial optimization problem is of the form

$$\min\{c^T x \mid Ax \leqslant b,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\}, \tag{3.1}$$

where $c$ is a vector with $n$ components, $A$ is an $m \times n$ matrix and $b$ is a vector with $m$ components. An overall framework for a combinatorial problem solver is given in fig. 1. This flowchart is designed for a software package to optimally solve large-scale zero-one problems having thousands of zero-one variables and can, alternatively, be used as a heuristic procedure for obtaining 'reasonable' feasible solutions, and at the same time a true lower bound on the optimal objective function value of (3.1). The latter provides, of course, a yardstick to measure the 'reasonableness' of the solutions found.

In this design, we start by reading the data of the user-supplied formulation into the computer. In a 'preprocessing' phase, the formulation is inspected automatically and permanent problem reductions may be carried out. Such problem reductions concern the fixing of certain variables due to feasibility and/or optimality considerations to either zero or one, the clearing of rows of common divisors greater then one, the 'cleaning up' of the user-supplied formulation such as reducing 'Big M's', the 'classification' of constraints, and the detection of special structures. Some of the

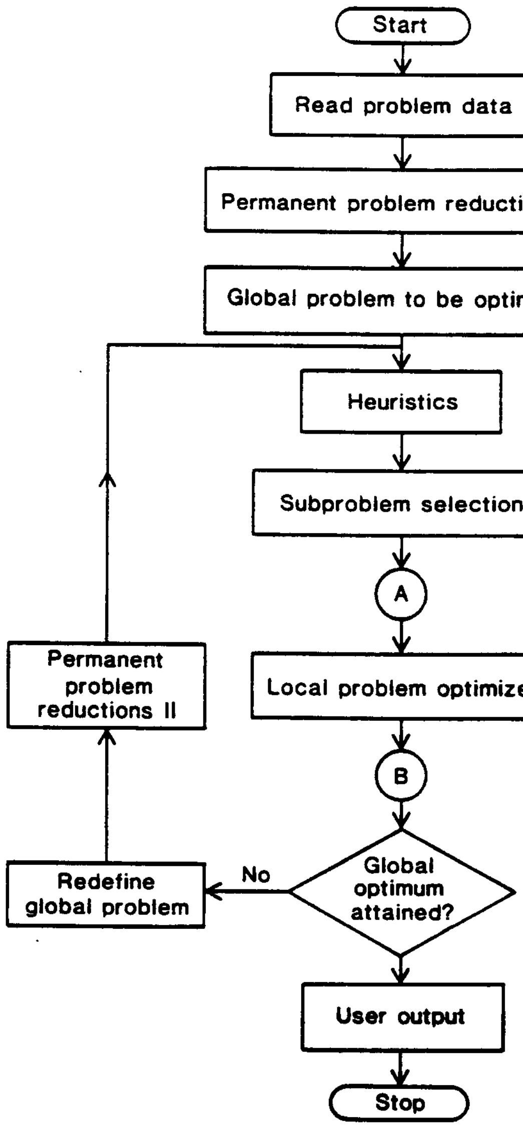

Fig. 1. The overall flowchart.

methods used in this preprocessing phase are discussed in the paper by Crowder et al. [11] and will be elaborated in a forthcoming paper by Bernal et al. [4]. After preprocessing, we thus obtain the definite version of the zero-one problem to be optimized, which we will call the 'global problem' to be optimized.

In the next phase, heuristics are used to find an upper bound on the objective function of (3.1). While satisfactory heuristics exist for many specially structured zero-one problems (e.g. the vertex packing problem, the set-covering problem, the traveling salesman problem), quick and efficient heuristics for general zero-one problems are scarce. One such heuristic is the pivot-and-complement heuristic due to Balas and Martin [1]. It is important to note that a good upper bound permits the 'fixing' to zero or one of certain variables permanently once a good *lower* bound is also known.

Since this framework is to be used to solve zero-one problems having possibly thousands of variables, one proceeds next with the selection of a subproblem for actual optimization. How to select a subproblem is clear in the context of integer programming on graphs: In the case of a traveling salesman problem with several hundred cities, for instance, one has to optimize over tens of thousands of variables (corresponding to the edges of the graph). Clearly, the bulk of these variables will be zero in an optimal solution; thus, it will be sufficient to consider the problem on a sparse subgraph of the entire problem induced by the 'short' edges of the graph. To prove the optimality of a solution of the sparse problem to the global problem, one needs to 'price out' the variables temporarily set equal to zero. If their reduced costs exceed the gap between the linear programming optimum and the integer optimum, then they were correctly set to zero. Otherwise, the subproblem is redefined. By a judicious choice of the subproblems selected (e.g. by selecting larger and larger subproblems), one can ensure that the process is convergent. In general, the subproblem selection can be done on the basis of a normalized cost criterion. i.e. we can order the variables by increasing ratios

$$\frac{c_j}{\left(1 + \sum_{i=1}^{m} a_{ij}^2\right)^{1/2}} \tag{3.2}$$

and use a cut-off value for determining the variables of the actual subproblem to be considered. At later stages of the calculation, such selection may be based on normalized reduced cost, rather than the original cost. Currently, except for integer programs on graphs, it is not clear how to select a suitable subproblem so as to ensure zero-one feasibility of the selected subproblems (provided one knows that problem (3.1) is feasible). Of course, if a (good) feasible zero-one solution has been found heuristically, or during the course of computation, one can always include the variables that are in a feasible solution in each of the subproblems, thus ensuring the feasibility of all selected subproblems. We note that in the case of a traveling salesman problem, criterion (3.2) is identical to the one based on the original cost $c_j$ and was successfully used by Crowder and Padberg [10] and Padberg and Hong [54] to optimize a 318-city problem involving 50 403 zero-one variables.

The most important phase of the combinatorial problem solver is the phase called the 'local problem optimizer'. Included in this sub-system is a problem set-up

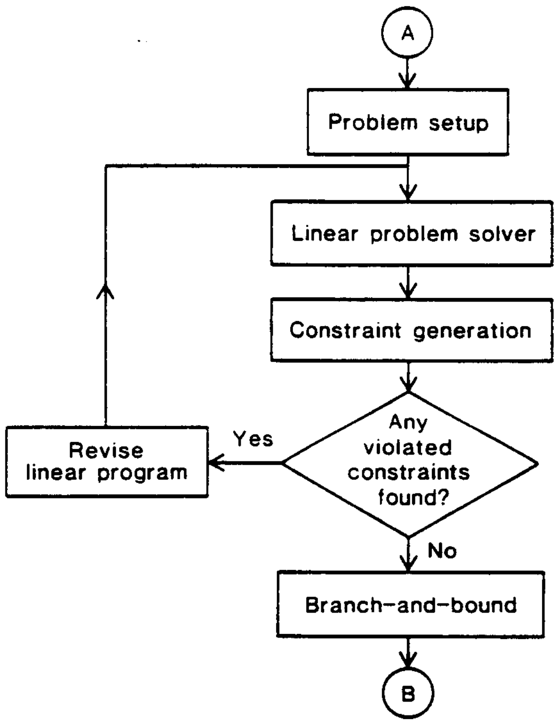

Fig. 2. The local problem optimizer.

routine, a linear program optimizer. a package of problem-dependent subroutines for the constraint generation, as well as a branch and bound routine. In other words, this is the relaxation method (2.15) of sect. 2 amended by a branch and bound procedure. Examples of how the constraint generation (2.14) of sect. 2 is carried out for different problem strutures are discussed in later sections of this paper. Figure 2 shows the flow-chart of the 'local problem optimizer'.

Having obtained an optimal solution for the subproblem considered in the local problem optimizer, one proceeds to extend ('lift') the newly generated constraints to include all zero-one variables of the global problem. Such an extension is theoretically and computationally possible using the various 'lifting' procedures described in the literature. (See Padberg [51] and Wolsey [68] for the case of general zero-one problems.) More specific results apply to specially structured zero-one problems (e.g. the traveling salesman problem, vertex packing, set covering). One then computes a lower bound on the optimal objective function value of (3.1) (e.g. by optimizing the linear programming relaxation of the global problem). Assuming that a feasible zero-one solution was found (either by a heuristic or by the local problem optimizer), we then have an upper bound and a lower bound on the optimal objective function value of (3.1). At this point, one may be able to fix variables based on the reduced costs of the linear program and the gap between the LP solution and the feasible zero-one solution found. This fixing of variables may enable further permanent problem reduction (Crowder et al. [11]). A new subproblem is then selected among those variables that were not fixed at either zero or one. The latter has to be done in such a manner as to guarantee convergence (the strategies for the selection will depend on the structure of the actual problem on hand). For *general* zero-one problems, at present we always use the global problem (i.e. we bypass the subproblem selection) because the sizes considered so far permitted us to do so.

In the next three sections, we describe facets of the polytope $P_{\mathcal{F}}$ and the status of the respective facet-identification problem (2.14) for three classes of combinatorial problems: the traveling salesman problem, the triangulation of large input/output matrices, and large sparse zero-one problems having no special structure. These three types of problems are chosen because computational studies have proven that cutting planes related to the facets of the underlying polytope are an indispensable tool for the exact solution to these problems. A summary of the computational results of these studies will also be provided in these sections.

## 4. The symmetric traveling salesman problem

In the symmetric traveling salesman problem (hereafter abbreviated to TSP), one is interested in finding a shortest round-trip or *tour* through $n$ cities such that every city is visited exactly once. The traveling salesman problem has its historical origins in graph theory. Euler [15] (translated in Biggs et al. [5]) studied related problems, as did Sir William Rowan Hamilton in 1859 (see Hankins [25]). The problem appears to have been formulated and coined the 'messenger boy's problem' by Menger [46]. In the seminal work by Dantzig et al. [12], a tour of forty-nine cities is optimized. This computational advancement is a milestone in the history of mathematical programming calculations. Since that time, the problem has been studied extensively in the applied mathematics and operations research literature. (See Lawler et al. [35] for an up-to-date survey of the traveling salesman problem.)

Consistent with graph-theoretic notation, let $G = (V, E)$ denote the complete, undirected graph having $n$ nodes (representing the cities), and let the vector $c$ with components $c_e$ for all edges $e \in E$ denote the symmetric distance table of the graph (i.e. if edge $e$ connects nodes $i$ and $j$ of the graph, then $c_e$ is the (symmetric) distance between cities $i$ and $j$). The assumption that $G$ is complete is for notational convenience only since, in a minimization context, missing edges of the original graph can be adjoined by assigning arbitrarily large distances to them.

A round-trip or tour through the $n$ cities corresponds to a Hamiltonian cycle in the graph $G$ and thus the family $\mathcal{F}$ of sect. 2 is the family of all subsets of edges of $G$ that form a Hamiltonian cycle. The ground set of sect. 2 is the set $E$ of all edges of $G$ and the zero-one vectors of interest to us are those that correspond to subsets of edges forming a Hamiltonian cycle in $G$. In an undirected complete graph with $n$ nodes there are exactly $\frac{1}{2}(n-1)!$ Hamiltonian cycles and thus in $\mathbb{R}^m$, where $m = \frac{1}{2}n(n-1)$, we have exactly that many zero-one points of interest to us. Taking the convex hull of these $\frac{1}{2}(n-1)!$ points, we obtain the *traveling salesman polytope* over which we wish to minimize the linear form given by the vector $c$.

To formulate the TSP as an integer program, denote by $A$ the $n$-by-$m$ node-edge incidence matrix of $G$ where, as before, $m = \frac{1}{2}n(n-1)$. Furthermore, for any vector $x \in \mathbb{R}^E$ with components $x_e$ for all $e \in E$, we denote for any subset $W \subseteq V$

$$x(W) = \sum_{e \in E(W)} x_e, \tag{4.1}$$

where $E(W)$ is the set of edges in $G$ with both ends in the node set $W$. The TSP is the following minimization problem:

$$\text{minimize} \quad cx \tag{4.2}$$

$$\text{subject to} \quad Ax = \underline{2} \tag{4.3a}$$

$$x(W) \leqslant |W| - 1 \quad \text{for all non-empty proper subsets } W \subseteq V \tag{4.3b}$$

$$\left.\begin{array}{l} 0 \leqslant x_e \leqslant 1 \quad (4.3\text{c}) \\ x_e \text{ integer} \quad (4.3\text{d}) \end{array}\right\} \text{ for all } e \in E,$$

where $\underline{2}$ is the vector with $n$ components equal to 2 and $|W|$ is the cardinality of $W$. The constraints (4.3a) express the condition that every node must be met be exactly two edges in every tour. Conditions (4.3b) express the conditions that cycles having less than $n$ edges are of no interest to us as they do not correspond to Hamiltonian cycles in the graph and are called ***subtour-elimination constraints*** (see Dantzig et al. [12]). (Note that the upper bound constraints are duplicated for notational convenience in (4.3c).) It is important to note that the requirement (4.3d) is essential, since fractional solutions will occur. This of course means that additional linear inequalities are needed to describe the traveling salesman polytope if the integrality requirements (4.3d) are dropped.

A first (naive) linear programming relaxation of the traveling salesman problem as used in the relaxation procedure (2.15) is given by (4.2), (4.3a), (4.3b) (4.3c). If the optimal solution of this linear program is integer, we have solved the traveling salesman problem. Otherwise we need more (facet-defining) inequalities to eliminate (chop off) the fractional linear programming optimum. If we run out of known facet-defining inequalities to do this job before obtaining a tour, we then resort to branch and bound. In the next section we discuss classes of inequalities that define facets for the traveling salesman polytope.

The astute reader will have noticed that this naive relaxation involves *exponentially many* constraints and thus – *prima facie* – it is not implementable. Indeed, historically, this has been the reason that the above formulation has been ignored in the literature. There are several formulations reported in the literature that replace the above inequality system by one that is polynomial in the number $n$ of nodes of the graph (typically, however, at the expense of introducing additional variables). Whatever the merits of these other formulations, we know that half of the constraints of (4.3b) define facets of the traveling salesman polytope. Thus, they are *essential* in defining the polytope. The way out of the dilemma is to include the constraints (4.3b) in those that are generated 'on the fly'. The real question, therefore, is not how many constraints we have, but whether or not we can find *violated* (facet-defining) inequalities efficiently. The answer to this question is positive for subtour elimination constraints as well as other inequalities.

## 4.1. FACETS OF THE SYMMETRIC TRAVELING SALESMAN POLYTOPE

The set of linear inequalities (4.3a), (4.3b) and (4.3c) defines a polytope that contains (properly) the traveling salesman polytope. We now wish to append to this set of constraints facet-inducing inequalities which will 'tighten' the above formulation when the requirement '$x$ integer' is ignored. There are many types of facet-inducing inequalities known for the problem (see Grötschel and Padberg [21]), of which we will discuss three types that can all be described by the inequality

$$\sum_{i=0}^{k} x(W_i) \leqslant |W_0| + \sum_{i=1}^{k} (|W_i| - 1) - \langle k/2 \rangle, \tag{4.4}$$

where $\langle\cdot\rangle$ denotes rounding up to the closest integer. The sets $W_i$ are proper subsets of $V$ satisfying the following conditions for $i = 0, 1, \ldots, k$:

$$|W_0 \cap W_i| \geqslant 1, \quad i = 1, \ldots, k, \tag{4.5}$$

$$|W_i \setminus W_0| \geqslant 1, \quad i = 1, \ldots, k, \tag{4.6}$$

$$W_i \cap W_j = \emptyset \quad 1 \leqslant i < j \leqslant k, \tag{4.7}$$

$$k \text{ odd}. \tag{4.8}$$

The edge set $C = \cup_{i=0}^{k} E(W_i)$ is called a *comb* in $G$ (see Grötschel and Padberg [21] and Padberg and Grötschel [59]). The inequalities (4.4) are called *comb constraints.* For $k = 1$ and $|W_0| = 1$, we retrieve the *subtour-elimination constraints* (4.3b) introduced above. A comb is a *2-matching constraint* (Edmonds [14]) if the inequalities in both (4.5) and (4.6) hold as equalities. A comb is a *Chvátal-comb* (Chvátal [9]) if the requirement (4.7) is dropped and the inequality (4.5) is required to hold as an equality. A thorough discussion of comb inequalities can be found in Grötschel and Padberg [21], where it is proven that comb inequalities belong to the class of linear inequalities that occur in any minimal linear constraint system for the TSP (modulo a multiple of the rows of $A$). The linear programming formulation of the symmetric traveling salesman problem provided by (4.2)–(4.4) is still only an approximation, however, since the associated linear constraint set is not complete (i.e. it admits basic solutions that are non-integer). Furthermore, as indicated above, one would not wish to list all

Table 1*

| No. of cities | No. of subtour elimination constraints | No. of comb constraints |
|---|---|---|
| 6 | 25 | 60 |
| 7 | 56 | 2 100 |
| 8 | 119 | 42 840 |
| 9 | 246 | 667 800 |
| 10 | 501 | 8 843 940 |
| 15 | 16 368 | 1 993 711 339 620 |
| 20 | $0.5 \cdot 10^{6}$ | $1.5 \cdot 10^{18}$ |
| 30 | $0.5 \cdot 10^{9}$ | $1.5 \cdot 10^{31}$ |
| 40 | $0.5 \cdot 10^{12}$ | $1.5 \cdot 10^{45}$ |
| 50 | $0.5 \cdot 10^{15}$ | $10^{60}$ |
| 120 | $0.6 \cdot 10^{36}$ | $2 \cdot 10^{179}$ |

*Extracted from Padberg and Grötschel [59].

known facets since even for very small TSPs, computers are not now, nor ever will be, capable of handling systems of linear equalities of the size needed (see table 1). But to stress this important point again, it is *not* the number of constraints needed to fully describe the polytope that matters in computation; rather it is the question of whether or not we are able to find *violated* constraints efficiently, i.e. in polynomial time. Using the classification of all comb constraints shown in table 1, we are faced with the following facet-identification problems (2.14):

(4.9) *Identification of subtour elimination constraints* (SEC):

Given a point $\bar{x} \in \mathbb{R}^E$ satisfying $0 \leqslant \bar{x}_e \leqslant 1$ for all $e \in E$, find a set $W \subseteq V$, $2 \leqslant |W| \leqslant n-1$, such that $\bar{x}(W) > |W| - 1$ holds or prove that no such $W \subseteq V$ exists.

(4.10) *Identification of 2-matching constraints* (2C):

Given a point $\bar{x} \in \mathbb{R}^E$ satisfying $0 \leqslant \bar{x}_e \leqslant 1$ for all $e \in E$, find a set $W \subseteq V$ and a set $T$ of edges $e_1, \ldots, e_{|T|}$ in $E(|T| \geqslant 3)$ with the properties

(a) $e_i$ has one endpoint in $W$ $\quad i = 1, \ldots, |T|$

(b) $e_i \cap e_j = \emptyset$ $\quad 1 \leqslant i < j \leqslant |T|$

(c) $|T|$ odd,

such that $\bar{x}(W) + \bar{x}(T) > |W| + \lfloor |T|/2 \rfloor$ holds or prove that no such $W \subseteq V$ and $T \subseteq E$ exist.

(4.11) *Identification of comb constraints* (Co):

Given a point $\bar{x} \in \mathbb{R}^E$ satisfying $0 \leqslant \bar{x}_e \leqslant 1$ for all $e \in E$, find sets $W$, $W_1, \ldots, W_k \subseteq V$ with the properties

(a) $|W \cap W_i| \geqslant 1$ $\quad i = 1, \ldots, k$

(b) $|W_i \setminus W| \geqslant 1$ $\quad i = 1, \ldots, k$

(c) $W_i \cap W_j = \emptyset$ $\quad 1 \leqslant i < j \leqslant k$

(d) $k$ odd,

such that

$$\bar{x}(W) + \sum_{i=1}^{k} \bar{x}(W_i) > |W| + \sum_{i=1}^{k} (|W_i| - 1) - \langle \tfrac{k}{2} \rangle$$

holds or prove that no such sets $W, W_1, \ldots, W_k \subseteq V$ exist.

In Padberg and Hong [54], a heuristic is provided for the identification of violated subtour elimination constraints. The algorithm requires a total computational effort which is polynomially bounded. For a complete answer to the question of whether or not there exists a subtour elimination constraint which chops off some feasible point $x$, the algorithm by Gomory and Hu [18] must be applied to the graph obtained by the refined shrinking procedure described by Padberg and Grötschel [59]. If all calculations are performed carefully in both algorithms, then the identification of subtour elimination constraints can be carried out by a polynomially bounded algorithm (i.e. problem (4.9) is solvable in polynomial time).

Padberg and Rao [57] have shown that a modification of the minimum-cut algorithm by Gomory and Hu [18] can be used to identify violated 2-matching constraints, and that this algorithm solves problem (4.10) in polynomial time. The Padberg and Rao algorithm, as well as a reduction heuristic for 2-matching constraints, are described in Padberg and Grötschel [59].

For the general comb inequalities, no polynomial exact algorithm is known to date. Padberg and Grötschel [59] give a heuristic procedure whose effectiveness remains to be tested in numerical problems.

A complete description of the methodology for the facet-identification problem (2.14) for symmetric TSPs (heuristics and exact algorithms to the extent known to date) can be found in Padberg and Grötschel [59]. The known results indicate that for a large class of facets of the traveling salesman polytope, the facet-identification problem (2.14) is solvable in polynomial time.

## 4.2. COMPUTATIONAL RESULTS FOR LARGE-SCALE SYMMETRIC TRAVELING SALESMAN PROBLEMS

In this section we give a brief summary of the papers by Crowder and Padberg [10], Grötschel [19], and Padberg and Hong [54]. There are, however, several other studies (unpublished) that utilized polyhedral information in the actual solution of symmetric traveling salesman problems such as the solution of a 125-city problem done by students of W.R. Pulleyblank at the University of Grenoble (France) and the solution of a 68-city problem done by students of L. Wolsey at the Catholic University of Louvain-la-Neuve (Belgium).

Related studies concerning the use of cutting planes in the solution of symmetric traveling salesman problems have been carried out by Land [34] and Miliotis [47,48], Fleischmann [16,17], and earlier by Hong [27]. While these studies reported good results for medium-sized traveling salesman problems, their methodology did not focus on testing the polyhedral theory described here, which is the intent of our brief survey.

Figure 3 depicts the overall flow of calculations to be performed in implementing the relaxation method (2.15) as applied to traveling salesman problems, when one ignores the aspect of subproblem selection. The very bottom loop in this flowchart is necessary because a zero-one solution found by branch and bound may correspond to a collection of subtours and thus needs to be excluded. The three studies that we will survey used flowcharts differing in minor details. While Grötschel [19] carried out the constraint identification visually, both Crowder and Padberg [10] and Padberg and Hong [54] used fully automated procedures (no manual intervention) in their respective software.

The first instance of a symmetric traveling salesman problem involving more than 100 cities that was solved using only subtour elimination, 2-matching and comb constraints is a 120-city problem given by the data in the *Deutscher General Atlas*

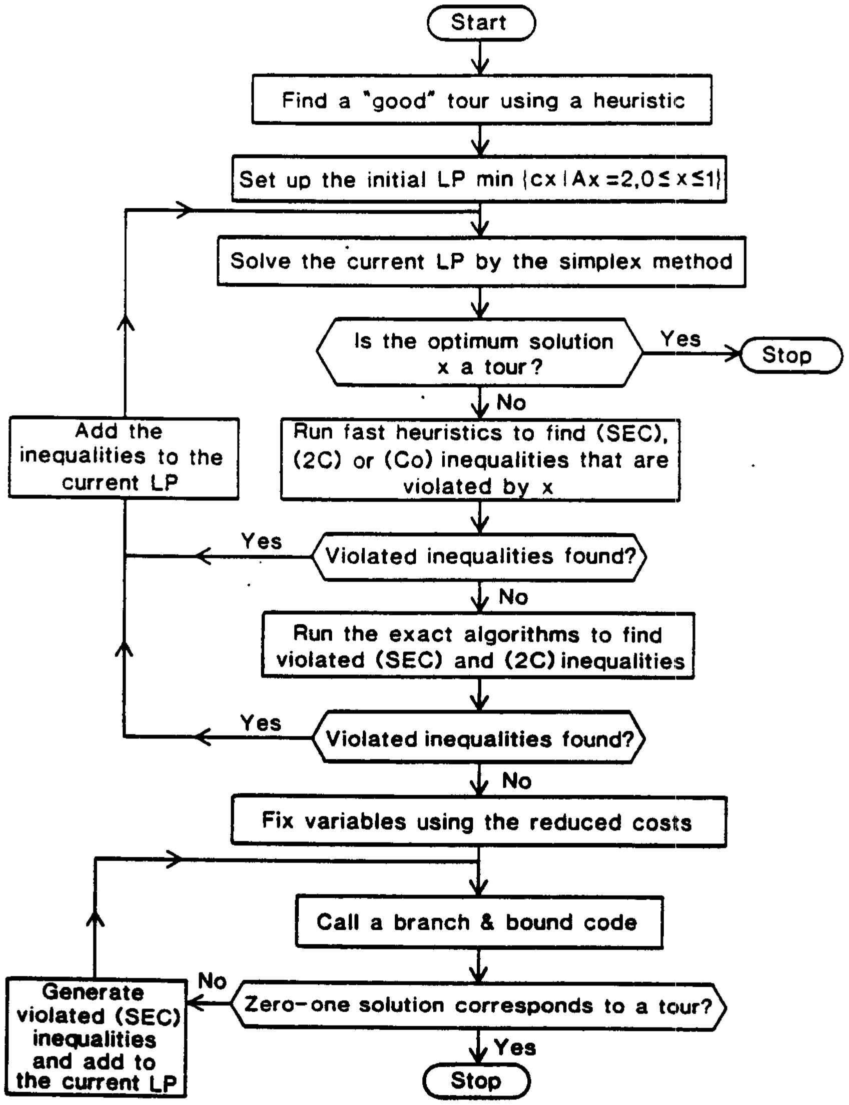
Start
Find a "good" tour using a heuristic
Set up the initial LP min {cx | Ax = 2, 0 ≤ x ≤ 1}
Solve the current LP by the simplex method
Is the optimum solution x a tour?
Yes
Stop
No
Run fast heuristics to find (SEC), (2C) or (Co) inequalities that are violated by x
Add the inequalities to the current LP
Violated inequalities found?
Yes
No
Run the exact algorithms to find violated (SEC) and (2C) inequalities
Violated inequalities found?
Yes
No
Fix variables using the reduced costs
Call a branch & bound code
Zero-one solution corresponds to a tour?
No
Generate violated (SEC) inequalities and add to the current LP
Yes
Stop

Fig. 3. Flowchart.

(Mairs Geographischer Verlag, Stuttgart, 1967/68). In Grötschel [19], no recourse to branch and bound or other enumerative methods was needed in order to obtain an optimal tour (i.e. only the relaxation method (2.15) for combinatorial optimization problems was used).

In order to find and to prove optimality of this problem, thirteen iterations of the relaxation method (2.15) were necessary. In this process, only 96 subtour elimination and comb constraints of the total universe of $10^{179}$ such constraints were generated to establish optimality of the tour. More precisely,

36 subtour elimination constraints,
25 2-matching constraints, and
35 (other) comb constraints

were needed to find and to prove optimality of the tour.

A comprehensive study using the facet-defining inequalities for the TSP was performed by Padberg and Hong [54]. They sampled 74 symmetric traveling salesman problems in order to broadly assess the computational value of facet-defining linear inequalities for the solution of traveling salesman problems. We note that out of the total 74 problems, only 54 were solved (to optimality), but in all cases, excellent lower bounds were obtained when compared to the solutions found by using the heuristic due to Lin and Kerningham [40].

In order to evaluate the value of facet-defining inequalities towards the goal of proving optimality, Padberg and Hong [54] used the following approach: Given a heuristically obtained solution, they solved the LP relaxation problem (4.2, 4.3a, 4.3c) (i.e. the 2-matching relaxation of the symmetric TSP) with the heuristically obtained tour as a starting point. Using the same heuristically obtained solution, they then ran the problem a second time, generating facet-defining inequalities. This run either terminated with an optimal tour or, in case of an inability to identify a suitable new constraint, defaulted with a lower bound for the problem obtained by solving the, by now, enlarged linear programming problem. This approach yielded two values: VALUE1 is the objective function value without cuts and VALUE2 is the objective function value with cuts. If TOUR denoted the minimum length tour of the problem, then the following ratio is a good proxy for measuring the added value of the additional work:

$$\text{RATIO} = (\text{VALUE2} - \text{VALUE1})/(\text{TOUR} - \text{VALUE1}).$$

Note that RATIO is zero if no improvement is obtained (e.g. if no constraint was generated which altered the LP values), while RATIO is one if the constraint-generation procedure terminates with the optimal tour. RATIO ranges from zero to one inclusively, and due to taking both differences and a ratio, the measure is invariant under scaling and translating the data. As the exact value of TOUR is known only *a posteriori*, it is worth pointing out that RATIO is a *conservative* valid measure of the improvement (if TOUR is a possibly *suboptimal* value obtained heuristically). It should further be noted that the same starting solution for the respective linear programs is used for the computation of both VALUE1 and VALUE2, which is known to greatly impact the performance of simplex methods.

Here we will discuss only two of several parts of the computational study by Padberg and Hong [54]. In one part, they ran fifty-five randomly generated Euclidean TSPs using the pseudo-random number generator by Lin and Kerningham [40], which generates coordinates $x_i$ and $y_i$ with values between 1 and 1000 for $i = 1, \ldots, n$. Because coordinates are generated as pairs, the same random seed for $n + m$ cities produces a graph that is properly contained in the graph on the first $n$ cities, thus permitting one to study how increasing $n$ affects the added value of the facet-inducing inequalities. Ten different problems were run for $n = 15, 30, 45, 60$ and $75$, respectively, using ten different seeds for the random number generator. Furthermore, five Euclidean problems with $n = 100$ due to Krolak et al. [33] were included in this statistical part of the study, although they are different from the others.

Table 2

Fifty-five Euclidean TPSs

| $n$ | | 15 | 30 | 45 | 60 | 75 | 100 |
|---|---|---|---|---|---|---|---|
| Ratio | $\mu$ | 1.0 | 0.99 | 0.93 | 0.92 | 0.88 | 0.92 |
| | $\sigma$ | 0.0 | 0.03 | 0.11 | 0.10 | 0.09 | 0.02 |
| TOUR | $\mu$ | 3 555 | 4 738 | 5 566 | 6 297 | 6 878 | 21 507 |
| | $\sigma$ | 383 | 314 | 224 | 181 | 224 | 525 |
| GAP1 | $\mu$ | 224 | 352 | 379 | 452 | 387 | 1507 |
| | $\sigma$ | 121 | 100 | 149 | 133 | 93 | 313 |
| GAP2 | $\mu$ | 0.0 | 5 | 24 | 38 | 50 | 120 |
| | $\sigma$ | 0.0 | 15 | 57 | 44 | 44 | 43 |
| PIVOT2 | $\mu$ | 11 | 34 | 47 | 76 | 87 | 167 |
| | $\sigma$ | 2 | 7 | 13 | 30 | 23 | 40 |
| ΔPIVOT | $\mu$ | 2 | 13 | 11 | 36 | 31 | 97 |
| | $\sigma$ | 2 | 5 | 12 | 29 | 22 | 38 |
| TIME2* | $\mu$ | 0.33 | 1.37 | 4.46 | 14.47 | 30.52 | 108.74 |
| | $\sigma$ | 0.03 | 0.26 | 1.27 | 6.82 | 10.81 | 39.97 |
| ΔTIME* | $\mu$ | 0.07 | 0.46 | 1.39 | 6.25 | 11.64 | 50.4 |
| | $\sigma$ | 0.03 | 0.23 | 1.23 | 6.63 | 10.63 | 31.7 |
| CUTS | $\mu$ | 3 | 12 | 15 | 26 | 28 | 72 |
| | $\sigma$ | 2 | 5 | 8 | 13 | 12 | 18 |
| OPTIM | | 10 | 9 | 5 | 4 | 3 | 0 |

*Seconds, IBM 370/168 MVS.

[illegible]
TOUR – [illegible]
[illegible] LP.
[illegible]
* [illegible] opt.

Table 2 contains all the relevant statistics for this part of the study. The entries in table 2 were obtained by averaging the respective individual figures, and their mean $\mu$ is given with $\sigma$ being the standard deviation. The top row of table 2 contains the value RATIO. As it is to be expected, RATIO declines with increasing $n$. TOUR is the tour length obtained by the heuristic.

The bottom line of table 2 (OPTIM) specifies the number of times the linear program terminated by proving the optimality of the heuristically obtained tour. GAP1 measures the average difference between TOUR and the objective function value of the (initial) linear program (4.2), (4.3a) and (4.3c) (i.e. the 2-matching relaxation of the symmetric TSP). GAP2 is the crucial measure in evaluating the constraint-generation procedure; it is the difference between TOUR and the objective function value of the amended linear program. PIVOT2 is the average of the total number of pivot operations carried out by the constraint-generating program. $\Delta$PIVOT is the average increment of the pivot count over what it takes to solve the initial linear program. Likewise, TIME2 specifies the total CPU time of the constraint-generating program, $\Delta$TIME the average increment over the respective times for the initial linear program. Note that to terminate with an average lower bound of 21 387 for $n$ = 100, it took on average 108.74 seconds of CPU time, while it took on average 50.4 seconds of CPU time less than 108.74 seconds to obtain an average lower bound of 20 000. (The numbers in the corresponding rows labelled $\sigma$ are the respective standard deviations from the sample problems.) Finally, CUTS is the average number of constraints that were generated and amended to the original linear program. Thus for the 100-city problems, the initial linear program has 100 rows and 4950 variables, while at termination of the constraint-generation procedure, the linear program increased on average to 172 rows and 5022 variables, a truly modest increase given the complexity of the problem and the goodness of the bound obtained. For the individual figures for all of the sample problems, the reader is referred to the original paper (Padberg and Hong [54]).

Another part of this study was done in order to permit a limited comparison of the performance of the constraint-generation procedure vis-a-vis other approaches. Therefore, a number of test problems that have been used by other researchers were solved. The results are summarized in table 3. The heading 'without cuts' refers to the solution of the (initial) linear program: TIME1 is the CPU time in seconds, PIVOT1 the pivot count, VALUE1 the objective function. TOUR refers to the minimum length tour or the value of the best tour found by the heuristic or during earlier runs with the TSP code, and is the initial solution for the TSP code in the run reported in table 3. The heading 'with cuts' refers to the constraint-generation procedure: VALUE2 is the objective function value of the linear program with cuts, the first column under PIVOT2 refers to the total number of pivots, the second column under PIVOT2 refers to the number of pivots carried out after the default in the constraint-generation procedure (i.e. the second column is already counted in the first). CUTS specifies the total number of cuts generated in the run, its second column the number of cuts that were dropped after defaulting. TIME2 is the total execution time to termination in CPU seconds. RATIO is the value discussed in the introduction to this section.

DAN42 is the 42-city version of the 49-city problem due to Dantzig et al. [12]. The solution was proven to be optimal in 3.10 seconds of CPU time after adding

Table 3
Eight problems from the literature

| | Without cuts | | | With cuts | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem | TIME1 | PIVOT1 | VALUE1 | TOUR | VALUE2 | PIVOT2 | | CUTS | | TIME2 | RATIO |
| DAN42 | 2.57 | 30 | 641 | 699 | 699 | 37 | 0 | 9 | 0 | 3.10 | 1.0 |
| GRO48 | 4.09 | 33 | 4 769 | 5 046 | 5 031$\frac{1}{16}$ | 83 | 9 | 32 | 8 | 9.16 | 0.95 |
| HEL48 | 3.69 | 33 | 11 197 | 11 461 | 11 461 | 38 | 0 | 10 | 0 | 4.30 | 1.0 |
| TOM57 | 7.79 | 44 | 12 633$\frac{1}{2}$ | 12 955 | 12 940 | 61 | 4 | 22 | 1 | 10.40 | 0.95 |
| KROL70 | 16.33 | 53 | 623$\frac{1}{2}$ | 675 | 673$\frac{1}{4}$ | 120 | 8 | 44 | 10 | 31.91 | 0.98 |
| GRO120 | 111.20 | 69 | 6 662$\frac{1}{2}$ | 6 951 | 6 928$\frac{1}{4}$ | 166 | 17 | 49 | 4 | 171.44 | 0.92 |
| KNU121 | 4.54 | 45 | 328 | 349 | 343$\frac{1}{2}$ | 74 | 13 | 10 | 1 | 7.25 | 0.76 |
| LIN318 | 670.8 | 251 | 38 765$\frac{1}{2}$ | 41 349 | 41 236$\frac{112}{240}$ | 578 | 70 | 171 | 64 | 1 751.46 | 0.96 |

nine constraints. (These constraints include one type not discussed here; see Padberg and Hong [54].) GRO48 is a 48-city problem due to Grötschel (48 cities with distances given in Shell's road atlas of Germany). The program terminated with a lower bound of 5032 for the optimum tour after 9.16 seconds of CPU time; the best tour found by the heuristic has a length of 5046. HEL48 is the 48-city problem due to Held and Karp [26]. The solution was proven to be optimal in 4.30 seconds of CPU time after adding ten constraints. TOM57 is the 57-city problem due to Thompson and Karp [61]. A lower bound of 12 940 for the optimum length tour of 12 955 was obtained after 10.40 seconds of CPU time. (Optimality was proven by Held and Karp [26].) KROL70 is a 70-city problem due to Krolak et al. [33]. After 31.91 seconds of CPU time, a lower bound of 674 on the heuristically obtained best tour of length 675 was obtained. GRO120 is the 120-city problem due to Grötschel described above. KNU 121 is a supersparse 121-city problem due to Knuth [32]. 7.25 seconds of CPU time were used to obtain a lower bound of 344 on the optimum tour length of 349, published in the New York Times.

LIN318 is a 318-city problem, the data of which are published by Lin and Kerningham [40]. The data come from an actual problem involving the routing of a numerically controlled drilling machine through three identical sets of 105 points each plus three outliers. As the drilling is done by a pulsed laser, drilling time is negligible and the problem becomes a standard traveling salesman problem. The only exception to the standard form is that a particular start and end point are to be used: the resulting Hamiltonian path problem can, however, easily be accommodated within the linear programming framework by assigning a negative distance to the particular arc. The distance table of the complete graph on 318 points (with the exception of one edge) was computed from the coordinates published by Lin and Kerningham [40].

The most surprising outcome of the computational study by Padberg and Hong [54] is that only very few additional facet-defining inequalities are needed in order to obtain an excellent lower bound on the minimum tour length and, in some cases, to also prove optimality. Consistently, the results obtained in the statistical part of the computational study are at most 0.5% off the optimal tour length: the standard deviations are consistently small as well. The results for the test problems from the literature including the – by 1980 standards – truly large-scale traveling salesman problems with 120 and 318 cities, respectively, generally out-performed the results that one might expect based on the statistical part of the study. In particular, the bound for the 120-city problem obtained this way indicates that the solution is within 0.04% of the optimum tour, and the bound for the 318-city problem indicates that the solution is within 0.26% of the minimum length Hamiltonian path through the 318 points. With the resulting (remaining) gap between the best tour found and the bound obtained by the use of facet-inducing inequalities being so relatively small, it was entirely realistic to expect that any good branch and bound procedure would enable one to solve large-scale traveling salesman problems to optimality, as was done by Crowder and Padberg [10].

Crowder and Padberg selected the same ten problems from the literature as those chosen in the previous study (as shown in table 4), with KRO124 through KRO128 being the five 100-city problems due to Krolak et al. [33]. Several changes in the code used by Padberg and Hong [54] were implemented to improve the lower bounds. These changes concerned the identification of additional subtour-elimination constraints and the determination of the 'hard core' of the respective TSPs that remained to be optimized. To this end, a subroutine was written to fix non-basic variables of the last LP optimum at either zero or one, using the available upper and lower bound on the optimal objective function value. The basic idea for this 'variable fixing' can be found in the paper by Dantzig et al. [12]. It is very effective and, on average, a reduction of the $n(n-1)/2$ variables of a TSP with $n$ cities to 5% of that number was obtained. The reduction in the case of the 318-city problem was even more impressive and was between 2% – 3% of the original variables.

In table 4, $n$ is the number of cities and $m = n(n-1)/2$ is the number of variables (the number of edges of the graph) except in the case of KNU121, a super-sparse problem having 222 edges only and 121 nodes. $n_R$ is the number of rows after running the problem with the changed TSP code, using the best available tour as a starting solution for the linear program: $m_R$ is the number of variables that were either basic at the final linear programming optimum or that could not be fixed at their respective non-basic value. TIME is the total execution time to obtain LPVALUE at termination. RATIO is the ratio described in this section. Three runs were made for

Table 4

Ten large-scale TSPs

| Problem | $n$ | $m$ | $n_R$ | $m_R$ | TIME* | RATIO | LPVALUE |
|---|---|---|---|---|---|---|---|
| GRO48 | 48 | 1 128 | 86 | 104 | 9 | 0.95 | 5 031.06 |
| TOM57 | 57 | 1 596 | 89 | 91 | 7 | 0.98 | 12 948.5 |
| KRO124 | 100 | 4 950 | 187 | 248 | 66 | 0.97 | 21 225.31 |
| KRO125 | 100 | 4 950 | 170 | 446 | 122 | 0.91 | 21 978.00 |
| KRO126 | 100 | 4 950 | 183 | 185 | 52 | 0.98 | 20 730.08 |
| KRO127 | 100 | 4 950 | 192 | 220 | 114 | 0.97 | 21 257.48 |
| KRO128 | 100 | 4 950 | 203 | 334 | 213 | 0.93 | 21 970.83 |
| GRO120 | 120 | 7 140 | 199 | 239 | 149 | 0.97 | 6 934.89 |
| KNU121 | 121 | 222 | 139 | 182 | 9 | 0.90 | 346.5 |
| LIN318A | 318 | 50 403 | 496 | 1 372 | 2 231 | 0.97 | 41 269.83 |
| LIN318B | 318 | 50 403 | 495 | 1 144 | 2 283 | 0.97 | 41 269.00 |
| LIN318C | 318 | 50 403 | 495 | 1 208 | 2 295 | 0.97 | 412 820.0 |

*Seconds, IBM 370/168 MVS.

Table 5

Breakdown of constraints by type

| Problem | Subtour | 2-matching | Comb | Total |
|---|---|---|---|---|
| GRO48 | 21 | 16 | 1 | 38 |
| TOM57 | 16 | 16 | 0 | 32 |
| KRO124 | 54 | 32 | 1 | 87 |
| KRO125 | 40 | 28 | 2 | 70 |
| KRO126 | 43 | 37 | 3 | 83 |
| KRO127 | 55 | 36 | 1 | 92 |
| KRO128 | 45 | 57 | 1 | 103 |
| GRO120 | 51 | 28 | 0 | 79 |
| KNU121 | 18 | 0 | 0 | 18 |
| LIN ~~KNU~~ 318 | 157 | 20 | 0 | 177 |
| Total | 500 | 270 | 9 | 779 |
| Percent | 64.3 | 34.6 | 1.1 | 100 |

problem LIN318, labeled A, B and C. LIN318A is a run of LIN318 started with the suboptimal tour of length 41 349 found by Padberg and Hong [54] using the revised software system just described. The run LIN318B used the optimal tour length 41 345 found by Crowder and Padberg [10] as a starting point – a check to ensure the validity of the software output. The third run, LIN318C, used the same information as LIN318B, but with a factor of 10 in the data to increase the precision of the distance calculation.

Table 5 gives a breakdown of the facet-inducing linear inequalities that were generated in the respective runs with the (changed) TSP code to get the linear programs of table 4. As noted earlier, the constraint identification for (general) comb inequalities was rather ineffective.

Finally, table 6 presents the results of using the IBM package MIP/370 as the branch and bound package to prove optimality for each of these problems. The first two columns display the problem size: $n_R$ is the number of rows and $m_R$ is the number of columns of the (pure) zero-one problem to be optimized. The column labeled MIP gives the number of times the MIP/370 program was called, i.e. the number of passes through the bottom loop of the flowchart in fig. 3 in which branch and bound calculations were required to prove optimality of a tour. The first of the two columns labeled NODES gives the number of nodes of the branch and bound tree generated to find the optimum zero-one solution, and the second column gives the total number of nodes generated in MIP/370 to prove optimality. Likewise, the first of the two columns labeled TIME gives the time spent in the branch and bound procedure to find the optimum zero-one solution, whereas the second one gives the total time spent in the MIP/370 program to prove optimality. (Thus, in problem GRO48 it took 17 nodes and 2 seconds of CPU time to find the optimum and one additional node, and an additional second to prove optimality.) The respective times have been rounded to the nearest second: in some cases, the MIP times add up to the total execution times, indicating that MPSX/370 solved the linear program in less than one second. The total CPU time for the MPSX/370 computation of solving the linear program (plus constraint generation and re-optimization where applicable), as well as the time spent in the MIP/370 program are reported in the last but one column labeled TOTAL TIME.

A first glance at table 6 shows that the computation times as well as the total branch and bound effort spent on optimizing the zero-one linear programming problems are extremely low. To optimize the (previously unsolved) problem GRO48, a total of 18 nodes had to be generated and the entire execution time of the MPSX-MIP/370 routine took 5 seconds of CPU time. The 'gap' between the linear program value of the corresponding TSP output and the optimal tour is 14.94, i.e. roughly 0.3% of the optimal tour length. This fact, together with the high cutting power of the facet-defining subtour-elimination and comb constraints, is responsible for the low computation time. The same can be said about virtually every other problem.

Table 6

| Name | $n_R$ | $m_R$ | MIP | NODES | | TIME | | $\Delta n$ | NODES | | TIME | | $\Delta n$ | NODES | | TIME | | TOTAL TIME* | OPT TOUR LENGTH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GRO48 | 86 | 104 | 1 | 17 | 18 | 2 | 3 | | | | | | | | | | | 5 | 5 046 |
| TOM57 | 89 | 91 | 1 | 2 | 3 | 0 | 1 | | | | | | | | | | | 6 | 12 955 |
| KRO124 | 187 | 248 | 1 | 8 | 15 | 2 | 7 | | | | | | | | | | | 13 | 21 282 |
| KRO125A | 170 | 445 | 2 | 133 | 147 | 59 | 70 | 1 | 115 | 146 | 51 | 79 | | | | | | 159 | 22 141 |
| KRO125B | 170 | 445 | 2 | 187 | 201 | 98 | 110 | 1 | 160 | 191 | 80 | 108 | | | | | | 220 | 22 141 |
| KRO126 | 183 | 185 | 1 | 5 | 5 | 1 | 2 | | | | | | | | | | | 9 | 20 749 |
| KRO127 | 192 | 220 | 2 | 4 | 12 | 2 | 7 | 1 | 2 | 12 | 0 | 7 | | | | | | 21 | 21 294 |
| KRO128 | 203 | 334 | 1 | 6 | 22 | 5 | 11 | | | | | | | | | | | 24 | 22 068 |
| GRO120 | 199 | 239 | 2 | 17 | 19 | 5 | 8 | 1 | 18 | 20 | 5 | 8 | | | | | | 24 | 6 942 |
| KNU121 | 139 | 182 | 3 | 14 | 25 | 1 | 2 | 1 | 15 | 23 | 1 | 2 | 3 | 15 | 23 | 1 | 2 | 16 | 349 |
| LIN318A | 496 | 1 372 | 3 | 22 | 22 | 50 | 52 | 5 | 35 | 36 | 81 | 86 | 1 | 21 | 71 | 75 | 163 | 319 | 41 345 |
| LIN318B | 495 | 1 144 | 2 | 41 | 41 | 123 | 125 | 4 | 54 | 58 | 129 | 146 | | | | | | 321 | 41 345 |
| LIN318C | 495 | 1 208 | 2 | 188 | 188 | 349 | 359 | 4 | 150 | 186 | 368 | 445 | | | | | | 818 | 413 589 |
| LIN318D | 495 | 1 208 | 2 | 216 | 216 | 365 | 375 | 4 | 200 | 230 | 408 | 483 | | | | | | 862 | 413 589 |

*Seconds, IBM 370/168 MVS.

Computational and theoretical research on traveling salesman problems continues along the lines described here. We quote from Padberg and Grötschel [54]:

'318 cities should not be the end of the story. The codes we have reported on here have used only some of the heuristics known to date. Currently, new LP-based cutting plane procedures are being developed which contain exact subroutines for the identification of subtour elimination and 2-matching constraints, as well as several new heuristics and new implementation details to speed up various communication procedures between subroutines and to overcome certain problems with respect to space. These methods may lead to another jump in the range of solvable problem sizes.'

## 5. The triangulation of input/output matrices

The results described in this section are summarized from papers by Grötschel et al. [22–24]. The triangulation of input/output matrices is an interesting application of an NP-hard combinatorial optimization problem known as the *linear ordering problem*, described as follows: Following Leontief [36], the economy of a region or country is divided into $n$ sectors, and an input/output matrix $A$ with $n$ columns and $n$ rows is constructed with the entry $a_{ij}$ denoting the amount of flow of goods from sector $i$ to sector $j$ in a certain year. The 'triangulation' problem consists of permuting the rows and columns of $A$ simultaneously such that the sum of the entries above the main diagonal of the permuted matrix is as large as possible. As to the problem's economic interest and significance, Leontief writes in a 1963 article in *Scientific American* [38]:

'The triangulation of a real input/output table – that is, the discovery of its peculiar structural properties and its interindustry dependence – is a challenging task. It is complicated by the fact that one must take into account not only the distinction between zero and non-zero entries, but also the more important difference between their actual numerical magnitudes.'

(See also Leontief [37,39].)

Let $\alpha_0$ denote the *largest* sum of the entries above the main diagonal of the permuted matrix that can be achieved by simultaneous row and column permutations and let $\alpha$ be the sum of all off-diagonal elements of the input/output matrix $A$. Then the number

$$\lambda(A) = 100\ \frac{\alpha_0}{\alpha}$$

is called the *degree of linearity* of the economy represented by the input/output matrix $A$ and is used to measure the mutual dependence or circularity of the sectors of the economy under consideration. To know $\alpha_0$, it is necessary to solve the *triangulation* or *linear ordering problem.*

The linear ordering problem is described as follows: A linear ordering of a finite set $V$ with $|V| = n$ is a one-to-one mapping $\sigma$ from $\{1, \ldots, n\}$ to $V$. For $i, j \in V$ we say that $i$ *precedes* $j$ if $\sigma^{-1}(i) < \sigma^{-1}(j)$. In economic applications, a 'profit' (or a 'cost') can be associated with a linear ordering in the following way: For every pair of elements $i, j \in V$, the two given values $c_{ij}$ and $c_{ji}$ are interpreted as the profit that accrues if $i$ precedes $j$ or if $j$ precedes $i$, respectively, in a linear ordering $\sigma$. Assuming additivity of profits, the total profit of a linear ordering $\sigma$ is

$$\sum_{(i,j) \in T(\sigma)} c_{ij}, \tag{5.0}$$

where $T(\sigma) = \{(i, j) | \sigma^{-1}(i) < \sigma^{-1}(j)\}$, and $(i, j)$ denotes the *ordered* pair $i$ and $j$. In the linear ordering problem, one is interested in finding a linear ordering $\sigma$ that maximizes the objective function (5.0). Of course, in the triangulation problem the profits are given by the flow values $a_{ij}$ of the input/output matrix $A$ and an optimal linear ordering gives the recording of the rows and columns of this matrix so as to get as close as possible to a triangular form.

Graphically, a linear ordering $\sigma$ induces an *acyclic orientation* of the edges of the complete graph $K_n$ having $n$ nodes corresponding to the elements of $V$: an edge connecting nodes $i$ and $j$ is oriented (or directed) *from i to j* if $(i, j) \in T(\sigma)$. The induced orientation of the edges of $K_n$ is acyclic in the sense that the resulting directed graph does not contain any directed cycle, since in a linear ordering no element can precede itself. On the other hand, every acyclic orientation of the edge set of $K_n$ induces a linear ordering on the nodes of $K_n$. Consider the complete directed graph $D_n = (V, A_n)$ having $n$ nodes and arc set $A_n$. A linear ordering $\sigma$ corresponds uniquely to a *partial graph* $(V, T)$ of $D_n$ such that $(V, T)$ induces an acyclic orientation of the undirected graph $K_n$ and vice versa. In the terminology of graph theory, an acyclic orientation of $K_n$ is called an *acyclic tournament* in the directed graph $D_n$. Then the family $\mathcal{F}$ of sect. 2 is the following set:

$$\mathcal{F}_n = \{T \subseteq A_n \mid T \text{ is an acyclic tournament in } D_n\},$$

where for notational simplicity we identify 'tournament' with 'arc set of a tournament'. Since $|A_n| = n(n-1)$, the *linear ordering polytope* $P^n_{\mathrm{LO}}$, i.e. the convex hull of the incidence vectors of all acyclic tournaments in $D_n$, is the polytope

$$P^n_{\mathrm{LO}} = \mathrm{conv}\{x^T \in \mathbb{R}^{n(n-1)} \mid T \in \mathcal{F}_n\}\ ,$$

over which we wish to maximize the linear form given by the profits $c_{ij}$ for all $i \neq j$.

Having described the combinatorial objects we are looking for as certain subsets of the arc set $A_n$ of the complete directed graph $D_n$, we can now proceed to formulate the linear ordering problem as a linear program in zero-one variables. Let $x_{ij} = 1$ if $i$ precedes $j$, $x_{ij} = 0$ otherwise. The linear ordering problem is the following maximization problem:

$$\text{maximize} \quad \sum_{i,j} c_{ij} x_{ij} \tag{5.1}$$

$$\text{subject to} \quad x_{ij} + x_{ji} = 1 \qquad \text{for all } i \neq j \tag{5.1a}$$

$$\left.\begin{aligned} x_{ij} + x_{jk} + x_{ki} &\leqslant 2 \\ x_{ik} + x_{kj} + x_{ji} &\leqslant 2 \end{aligned}\right\} \quad \text{for all } 1 \leqslant i < j < k \leqslant n \tag{5.1b}$$

$$x_{ij} \geqslant 0 \tag{5.1c}$$

$$x_{ij} \ \text{integer}\ . \tag{5.1d}$$

Note that variables $x_{ii}$ are not needed, hence are not defined. The inequalities (5.1a) express the condition that either $i$ preceded $j$ or $j$ precedes $i$ for all $i \neq j$. Inequalities (5.1b) exclude directed cycles (dicycles) of length 3 and are all that are needed to exclude *all* directed cycles in conjunction with (5.1a). The condition (5.1d) is essential. This of course means that additional linear inequalities are needed to describe the linear ordering polytope $P^n_{\mathrm{LO}}$ when condition (5.1d) is dropped.

## 5.1. FACETS OF THE LINEAR ORDERING POLYTOPE

The following theorem from Grötschel et al. [23] summarizes the partial description of the linear ordering polytope $P^n_{\mathrm{LO}}$ known to date. Related interesting structural properties of this polytope can be found in the paper by Young [69], who studied this problem because of its interest to voting theory.

THEOREM

Let $D_n = (V, A_n)$ be the complete directed graph having $n \geqslant 6$ nodes.

(i) The dimension of $P^n_{\mathrm{LO}}$ is $n(n-1)/2$.

(ii) The inequalities $x_{ij} \geqslant 0$ for all $(i, j) \in A_n$ define (trivial) facets of $P^n_{\mathrm{LO}}$.

(iii) The 3-dicycle inequalities (5.1b) define facets of $P^n_{\mathrm{LO}}$.

(iv) Let $U = \{u_1, \ldots, u_k\}$, $W = \{w_1, \ldots, w_k\} \subseteq V$ be disjoint sets of nodes of $D_n$ of cardinality $3 \leqslant k \leqslant n/2$ and call the arc set

$$A = \{(u_i, w_i) \mid i = 1, \ldots, k\} \cup \{(w_i, u_j) \mid i \neq j \in \{1, \ldots, k\}\}$$

a $k$-fence. Then every $k$-fence inequality

$$x(A) \leqslant k^2 - k + 1 \tag{5.2}$$

defines a facet of $P^n_{\mathrm{LO}}$ that is distinct from the facets of part (ii) and (iii). (Note that $k$-fences are particular orientations of the complete bipartite graph $K_{k,k}$.)

(v) Let $M \subseteq A_n$ be an arc set (called Möbius-ladder) which is the union of $k \geqslant 3$, $k$ odd, dicycles $C_1, \ldots, C_k$ satisfying the following properties:

(1) The length of $C_i$ is three or four, $i = 1, \ldots, k$.

(2) The cycles $C_i$, $C_{i+1}$, $i = 1, \ldots, k-1$, resp. $C_1$, $C_k$ have exactly one arc, say $e_i$ resp. $e_k$, in common. No other pair of cycles has a common arc.

(3) Each node $u$ in the vertex set of $M$ is contained in at least three arcs of $M$.

(4) If two dicycles $C_i$, $C_j$, $2 \leqslant i + 1 < j \leqslant k$, have a node, say $v$, in common, then $v$ belongs to all dicycles $C_i, C_{i+1}, \ldots, C_j$ or to all dicycles $C_j, \ldots, C_k, C_1, \ldots, C_i$.

(5) Given any dicycle $C_j$, $j \in \{1, \ldots, k\}$, the set $M \setminus \{e_i \mid i \in J\}$ contains exactly one dicycle, namely $C_j$, where

$$J = \{1, \ldots, k\} \cap (\{j-2, j-4, \ldots\} \cup \{j+1, j+3, \ldots\}),$$

Then the Möbius-ladder inequality

$$x(M) \leqslant |M| - \frac{k+1}{2} \tag{5.3}$$

defines a facet of $P^n_{\mathrm{LO}}$. The Möbius-ladder inequality for $M = C_1 \cup C_2 \cup C_3$, where the $C_i$ are dicycles of length four, is a 3-fence inequality. In all other cases, no inequality (5.3) is equivalent to any of the inequalities of part (ii), (iii), and (iv).

To use this theorem algorithmically, one first notes that because of the simplicity of the constraints (5.1a), one can eliminate half the number of the variables of the problem. One then rewrites the inequalities of the theorem accordingly. To begin the calculation, one considers the problem

$$\text{maximize} \quad \sum_{i<j} \bar{c}_{ij} x_{ij}$$

$$\text{subject to} \quad 0 \leqslant x_{ij} \leqslant 1 \quad \text{for all} \quad 1 \leqslant i < j \leqslant n\,,$$

where $\bar{c}_{ij} = c_{ij} - c_{ji}$. This problem can be solved trivially.

One then adds the (transformed) 3-dicycle inequality constraints (5.1b). Constraints of this type, while polynomial in $n$, are too many to list; consequently, they are generated as cutting planes 'on the fly'. That is, all 3-dicycle inequalities are enumerated, but only those that are violated by the solution to the current linear programming relaxation are added.

On the other hand, there is no known-to-date polynomial method to solve the facet-identification problem (2.14) for the $k$-fence inequalities described in (5.2). Heuristics for finding violated constraints can be found in the paper by Grötschel et al. [23].

The Möbius ladders (5.3) are even more difficult to handle, and a general approach for obtaining such constraints is unknown at present. Heuristics have been devised for three special Möbius-ladder inequalities (again, see Grötschel et al. [23]).

## 5.2. COMPUTATIONAL RESULTS FOR THE TRIANGULATION OF INPUT/OUTPUT MATRICES

Given the current state of information regarding the polyhedral structure of the linear ordering problem, we now summarize the computational results Grötschel et al. [22,23] obtained with an algorithm using the facet-cuts described above. All experiments were run on the IBM 370/168 of the Rechenzentrum der Universität Bonn. These authors state that the range of matrices considered in their study is representative of almost all input/output matrices that have been compiled in Europe so far. The study comprises three sets of input/output matrices: one set is provided by the European Community, the second one by the West German Institut für Wirtschaftsforschung, and the third one by the West German Statistisches Bundesamt.

The European Community compiles (44 X 44) input/output matrices for all members of the EC. This input/output program is of considerable importance since here, for all countries, all sectors are defined in a (more or less) identical way, so that structural comparisons between these countries can be made. These I/O tables can be obtained from the Office Statistique des Communautés Européennes (EUROSTAT, B.P. No. 1907, Kirchberg, Luxembourg) on tape, free of charge. In this study, the authors chose the (most important) matrices which are the (44 X 44) matrices of intermediate consumption at current prices for the years 1959, 1965, 1970, 1975. EUROSTAT has thirty of these matrices. A list of the matrices together with their degree of linearity (the computation of which requires the optimization of the linear ordering problem for the given input/output matrix), and the respective running times can be found in table 7. The running times in table 7 include the total time used including input and output of data, paging, etc. In every cutting plane generation step, all cutting planes that were found were added to the LP. In twenty-eight of the thirty matrices, an integral solution was obtained after the initial phase, and only for the matrices BELGIUM 59 and GERMANY 70 was it necessary to enter the branch and bound stage. Both problems were solved with only one branching operation. Thus, the branch and bound tree consisted of three nodes (including the root).

The second class of input/output matrices have the size 56 X 56 and are compiled by the Deutsche Institut für Wirtschaftsforschung (DIW), Berlin, for the Federal Republic of Germany during the years 1954–1972. The degrees of linearity and the running times are listed in table 8. The version of the code used was the same as for the EUROSTAT matrices. All nine triangulation problems could be optimized without entering the branch and bound stage.

Thirdly, the authors ran the same version of the code on three (60 X 60) input/output matrices compiled for the years 1970, 1974 and 1975 by the Statistisches Bundesamt (Postfach 5528, D-6200 Wiesbaden, West Germany) for the Federal Republic of Germany. The three tables 'Input/Output Tabelle zu Ab-Werk-Preisen-Inländische Produktion 70, 74, 75' are available from this government agency. The results can be found in table 9. The 1974 and 1975 problems could be solved using nothing but 3-dicycle inequalities. For the 1970 matrix, only one branching operation was necessary to obtain the optimum solution. Up to this point, the authors used a version of the code where only 3-dicycle inequalities are generated. In a second version of the code, other heuristics were used only if no violated 3-dicycle inequalities were found. The only problems where the new code produces results different from the one described above are those input/output tables for which the branch and bound phase has to be called, i.e. for which 3-dicycle inequalities did not suffice to solve the problem.

For each of these problems, one violated Möbius-ladder constraint together with many 3-dicycle cuts was sufficient to get the optimal solution. Perhaps luckily, in every real-world application of this study the authors were able to solve the linear ordering problem purely by LP cutting plane techniques without resorting to branch and bound.

Thus, the linear ordering problem appears to be another class of NP-hard problems which can be optimized in computationally reasonable times using facet-defining linear inequalities in a linear-programming-based approach to combinatorial problem solving.

Table 7

| Input/Output table | | Degree of linearity | CPU time in (min : sec) (IBM 370/168) |
|---|---|---|---|
| T59I11XX.B | (Italy 59) | 88.443 | 1 : 02.10 |
| T59D11XX.B | (Germany 59) | 88.148 | 0 : 45.84 |
| T59F11XX.B | (France 59) | 85.516 | 0 : 44.02 |
| T59B11XX.B | (Belgium 59) | 83.818 | 1 : 12.84 |
| T59N11XX.B | (Netherlands 59) | 82.991 | 0 : 51.83 |
| T65L11XX.B | (Luxemburg 65) | 90.805 | 0 : 28.95 |
| T65I11XX.B | (Italy 65) | 85.812 | 1 : 34.07 |
| T65B11XX.B | (Belgium 65) | 85.030 | 1 : 15.41 |
| T65F11XX.B | (France 65) | 84.395 | 1 : 10.87 |
| T65D11XX.B | (Germany 65) | 83.007 | 1 : 29.05 |
| T65W11XX.B | (Eur-6 65) | 82.921 | 0 : 56.78 |
| T65N11XX.B | (Netherlands 65) | 82.585 | 1 : 15.86 |
| T70L11XX.B | (Luxemburg 70) | 89.808 | 0 : 58.87 |
| T69R11XX.B | (Ireland 69) | 87.862 | 0 : 56.85 |
| T70K11XX.B | (Denmark 70) | 85.893 | 0 : 54.56 |
| T70I11XX.B | (Italy 70) | 85.613 | 1 : 13.68 |
| T70F11XX.B | (France 70) | 85.115 | 1 : 16.51 |
| T70B11XX.B | (Belgium 70) | 84.132 | 0 : 54.96 |
| T70W11XX.B | (Eur-6 70) | 83.401 | 0 : 54.72 |
| T70D11XX.Ba | (Germany 70a) | 83.181 | 1 : 12.06 |
| T70N11XX.B | (Netherlands 70) | 82.810 | 1 : 08.82 |
| T70X11XX.B | (Eur-9 70) | 82.668 | 0 : 53.76 |
| T70D11XX.Bb | (Germany 70b) | 82.199 | 1 : 36.69 |
| T70U11XX.B | (United Kingdom 70) | 80.636 | 0 : 36.86 |
| T75E11XX.B | (Spain 75) | 87.715 | 1 : 04.01 |
| T75K11XX.B | (Denmark 75) | 86.054 | 1 : 04.47 |
| T75I11XX.B | (Italy 75) | 85.136 | 1 : 10.56 |
| T75N11XX.B | (Netherlands 75) | 82.943 | 0 : 52.04 |
| T75D11XX.B | (Germany 75) | 82.773 | 1 : 13.87 |
| T75U11XX.B | (United Kingdom 75) | 82.678 | 1 : 12.05 |

minimum CPU time: 0 : 28.95,
maximum CPU time: 1 : 36.69,
average CPU time: 1 : 04.09.

Table 8

| Input/Output table | Degree of linearity | CPU time in (min : sec) (IBM 370/168) |
|---|---|---|
| DIW56N54 | 81.299 | 4 : 56.71 |
| DIW56N58 | 81.605 | 4 : 24.14 |
| DIW56N62 | 81.435 | 4 : 56.12 |
| DIW56N66 | 81. 791 | 4 : 47.15 |
| DIW56N67 | 81.063 | 5 : 01.19 |
| DIW56N72 | 79.812 | 5 : 10.75 |
| DIW56R54 | 80.785 | 4 : 47.61 |
| DIW56R58 | 81.061 | 4 : 50.81 |
| DIW56R66 | 81.941 | 4 : 33.77 |
| DIW56R67 | 81.801 | 5 : 08.58 |
| DIW56R72 | 79.966 | 4 : 57.03 |

minimum CPU time: 4 : 24.14,
maximum CPU time: 5 : 10.75,
average CPU time: 4 : 52.16.

Table 9

| Input/Output table | Degree of linearity | CPU time in (min : sec) (IBM 370/168) |
|---|---|---|
| Input/Output-Tabelle 1970 zu Ab-Werk-Preisen-inländische Produktion | 83.186 | 13 : 25.01 |
| Input/Output-Tabelle 1974 zu Ab-Werk-Preisen-inländische Produktion | 83.965 | 9 : 32.95 |
| Input/Output-Tabelle 1975 zu Ab-Werk-Preisen-inländische Produktion | 83.799 | 9 : 57.76 |

## 6. Large-scale zero-one linear programming problems

In this section, we describe two computational studies (Crowder et al. [11] and Bernal et al. [4]) which apply the polyhedral theory to the solution of *general large-scale* zero-one programming problems. The zero-one programming problems that are considered here have the following form:

$$\text{(ZOP)} \qquad \min \{cx \mid Ax \leqslant b,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\},$$

where $A$ is an $m$-by-$n$ matrix with arbitrary rational entries, and $b$ and $c$ are vectors of length $m$ and $n$, respectively, with rational entries. Historically, the interest in pure zero-one problems appears to originate with mathematical programming models in finance (see, for example, the paper by Lorie and Savage [41] and the work by Weingartner [66]). It would, however, be wrong to conclude that these zero-one problems are limited to finance models; rather, they enjoy broad applicability.

In all test problems, the costs $c_j$ are non-negative numbers. The problems have a *sparse* constraint matrix $A$ (i.e. the total number of non-zero elements $a_{ij}$ of $A$ divided by the product of $m$ and $n$ is typically less than 0.05). Furthermore, while $A$ is permitted to have rows with entries equal to $+1$, $-1$ and $0$ (e.g. special ordered set constraints), the problems considered here primarily have a substantial number of rows in which the non-zero elements of $A$ have a significant variance within each row. Indeed, if $A$ has only $+1$, $-1$ and $0$ entries, or if all but a few rows of $A$ have this property, then neither problem preprocessing nor constraint generation as done in these studies can be expected to be very effective. This, indeed, was the case for several of the test problems having this property.

The approach taken to solve these problems consists of a combination of problem preprocessing, cutting planes and clever branch and bound strategies. The preprocessing and overall flow of the procedure was discussed in sect. 3. We now provide the details of constraint generation for this class of problem. For the various improvements of the standard branch and bound procedure implemented by Crowder et al. [11], we refer to reader to the original paper. Computational results will then be presented.

### 6.1. A POLYHEDRAL APPROXIMATION FOR SPARSE ZERO-ONE LINEAR PROGRAMS

The zero-one problem (ZOP) with a single linear constraint (i.e. the case where $m = 1$) is called the *knapsack problem*. The facial structure of the associated polytope has been studied thoroughly, even though a complete list of the linear inequalities that define the knapsack polytope remains unknown. Let $(a_i, b_i)$ denote the $i$th row of $(A, b)$ and assume for simplicity that all constraints of (ZOP) are general constraints having arbitrary rational entries. Let

$$P_I^i = \operatorname{conv}\{x \in \mathbb{R}^n \mid a^i x \leqslant b_i,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\} \tag{6.1}$$

denote the convex hull of the zero-one solutions to the single inequality $a^i x \leqslant b_i$, where $i \in \{1, \ldots, m\}$. $P_I^i$ is the *knapsack polytope* associated with constraint $i$ of problem (ZOP). Likewise, we let

$$P_I = \operatorname{conv}\{x \in \mathbb{R}^n \mid Ax \leqslant b,\ x_j = 0 \text{ or } 1 \text{ for } j = 1, \ldots, n\} \tag{6.2}$$

denote the convex hull of zero-one solutions to the entire constraint set of problem (ZOP). $P_I$ is the zero-one polytope associated with problem (ZOP) and clearly we have

$$P_I \subseteq \bigcap_{i=1}^{m} P_I^i , \tag{6.3}$$

i.e. $P_I$ is contained in the intersection of all the knapsack polytopes $P_I^i$, $i = 1, \ldots, m$. In general, equality (6.3) does not hold, but does if, for example, problem (ZOP) decomposes totally into $m$ knapsack problems. This is the case if every variable $x_j$ appears in exactly one of the $m$ constraints $a^i x \leqslant b_i$ for $i = 1, \ldots, m$. If we have a large-scale zero-one programming problem with a *sparse* matrix $A$ and with no apparent special structure, it is reasonable to expect that the intersection of the $m$ knapsack polytopes $P_I^i$ provides a fairly good approximation to the zero-one polytope $P_I$ over which we wish to minimize a linear objective function. This is a working hypothesis and the computational results confirm that it is a reasonable assumption. Practically speaking, this assumption permits one to concentrate on the individual rows of the constraint set of problem (ZOP) when one derives valid inequalities for the polytope $P_I$. While the preceding approach is clearly valid for any zero-one problem, it can not be expected to lead to substantial computational gains if, for instance, the matrix $A$ is rather dense or if almost all entries in $A$ equal $+1$, $-1$ and $0$. In these two cases, the constraint generation must be done differently (see, for example, the survey by Padberg [53]).

## 6.2. FACETS OF THE KNAPSACK POLYTOPE

Now consider a single inequality $a^i x \leqslant b_i$ of the constraint system of problem (ZOP). Using the variable substitution $x_j = 1 - x_j$ where necessary, one brings the inequality into a form where all non-zero coefficients are positive. Dropping the index $i$ for notational convenience, one therefore considers (without loss of generality) a linear inequality

$$\sum_{j \in K} a_j x_j \leqslant a_0 , \tag{6.4}$$

where the coefficients $a_j$ are positive rational numbers and the variables $x_j$ assume the values zero or one. Let $S \subseteq K$ be such that

$$\sum_{j \in S} a_j > a_0 \quad \text{and} \quad \sum_{j \in S} a_j - a_k \leqslant a_0, \quad \text{for all } k \in S \tag{6.5}$$

hold. Then $S$ is called a *minimal cover* with respect to (6.4) and obviously every zero-one solution to (6.4) satisfies the inequality

$$\sum_{j \in S} x_j \leqslant |S| - 1, \tag{6.6}$$

where $|S|$ denotes the cardinality of the set $S$ (see, for example, Dantzig [13] p. 520). Suppose next that $S^* \subseteq K$ and $t \in (K \setminus S^*)$ satisfy

$$\sum_{j \in S^*} a_j \leqslant a_0 \quad \text{and}$$

$$Q \cup \{t\} \text{ is a minimal cover for every } Q \subseteq S^* \text{ with } |Q| = k, \tag{6.7}$$

where $k$ is an integer number satisfying $2 \leqslant k \leqslant |S^*|$. Due to the one-element role of the index $t$ and because $k$ is some integer number, the set $S^* \cup \{t\}$ is called a $(1, k)$-configuration and the following inequalities are valid for the 0-1 solutions to (6.4):

$$(r - k + 1)x_t + \sum_{j \in T(r)} x_j \leqslant r, \tag{6.8}$$

where $T(r) \subseteq S^*$ varies over all subsets of cardinality $r$ of $S^*$, and $r$ varies over all integers from $k$ to $|S^*|$, inclusively. If $k = |S^*|$ holds in (6.7), then a $(1, k)$-configuration is a minimal cover. Thus, in general, the class of inequalities associated with $(1, k)$-configurations properly contains the class of inequalities associated with minimal covers. Both inequalities (6.6) and (6.8) are best possible ones if $K = S$ holds in the first case, or if $K = S^* \cup \{t\}$ holds in the second case, i.e. in these cases the respective inequalities define facets of the associated knapsack polytope (see Padberg [53,55]). In general, however, these inequalities must be 'lifted' to obtain facets of the knapsack polytope associated with (6.4), i.e. they must be extended appropriately to the variables $x_j$ with index $j$ in $K \setminus S$, or with index $j$ in $K \setminus S^* \setminus \{t\}$, respectively.

The extension is done over the respective variables by the following recursive lifting procedure: Initially, one sets $f_j = 1$ for all $j \in S$, $f_0 = |S| - 1$, or $f_j = 1$ for all

$j \in S^*$, $f_t = (r - k + 1)$, $f_0 = r$ and $S = S^* \cup \{t\}$, respectively. For the iterative step, let $k \in K \setminus S$ and determine

$$z_k = \max\left\{ \sum_{j \in S} f_j x_j \mid \sum_{j \in S} a_j x_j \leqslant a_0 - a_k,\ x_j = 0 \text{ or } 1 \text{ for all } j \in S \right\}. \quad (6.9)$$

Define $f_k = f_0 - z_k$, redefine $S$ to be $S \cup \{k\}$, and repeat until $K \setminus S$ is empty. The resulting inequality $f_x \leqslant f_0$ defines a facet for the knapsack polytope associated with (6.4). Obviously, the lifting procedure requires the solution of several zero-one problems. But by relaxing (6.9) to a linear program, one can approximate the lifting procedure and thus produce 'almost' facet-defining inequalities for the knapsack polytope efficiently.

We note that the support of an inequality obtained by lifting (6.6) or (6.8) is contained in the support of the inequality (6.4), i.e. $a_j = 0$ implies that $f_j = 0$ holds. Therefore, the inequalities that are generated preserve the sparsity of the constraint matrix. This is an additional difference between these methods and the traditional cutting planes described in the textbooks on integer programming. The traditional cutting planes are typically rather dense and thereby lead to explosive storage requirements.

Since the number of possible minimal cover and $(1, k)$-configurations for (6.4) is exponential in the number of variables of the constraint (6.4), one can not list *a priori all* possible minimal covers and $(1, k)$ configurations for each row of the problem. One must therefore generate such constraints 'on the fly', i.e. generate them in the course of computation as they are needed. To this end, one starts by solving the linear program

$$\min\{cx \mid Ax \leqslant b,\ 0 \leqslant x_j \leqslant 1 \text{ for } j = 1, \ldots, n\} \quad (6.10)$$

and obtains an optimal solution $\bar{x}$. If $\bar{x}$ is a zero-one solution, one can stop: $\bar{x}$ solves the problem (ZOP). Otherwise, taking any knapsack row of $A$ (i.e. a non-SOS row), one solves the following problem in approximation to the facet-identification problem (2.14):

***Constraint identification problem:*** Given $\bar{x}$, find a minimal cover inequality (6.6) or a $(1, k)$-configuration inequality (6.8) that chops off $\bar{x}$, if such an inequality exists.

The constraint-identification problem is solved, in turn, for each row of the original constraint matrix $A$ that qualifies; the resulting inequalities that are identified are 'lifted' and appended to the linear programming problem; the augmented problem is reoptimized; and the procedure is repeated until a zero-one solution is found, until no more constraints are found, or until the gain in the objective function value after a number of such iterations becomes too small (e.g. is less than one unit in terms of the units of the objective function). When one of the latter two possibilities occurs, one resorts to branch and bound.

The question is then to devise a procedure that solves the constraint-identification problem. Consider the zero-one knapsack problem

$$\min\left\{ \sum_{j\in K} (1-\bar{x}_j)s_j \mid \sum_{j\in K} a_j s_j > a_0, s_j = 0 \text{ or } 1 \text{ for all } j \in K \right\}, \qquad (6.11)$$

where the inequality is strict in the knapsack constraints. It follows that there is a minimal cover inequality (6.6) that chops off $\bar{x}$ if and only if the optimal objective function value of (6.11) is less than one. As can be easily verified, problem (6.11) is constructed in such a manner that its solution finds a most violated minimum cover inequality.

The formulation of the constraint-identification problem for minimal cover inequalities involves the solution of a zero-one knapsack problem for which, to date, no technically good algorithm is known, but which can be approximated efficiently by its linear relaxation. For (1, $k$)-configurations, a formulation of the constraint-identification problem in a tractable form such as (6.11) is not known at present. Rather, both computational studies reported here use *ad hoc* procedures for finding (1, $k$)-configurations. For details about these procedures, see Crowder et al. [11] and Bernal et al. [4].

We note that facets for the knapsack problem with special ordered sets were described by Johnson and Padberg [29,30]. The study by Crowder et al. [11] does not use these results, but rather a surrogate constraint is formed from the original knapsack constraint and a non-overlapping subset of SOS constraints. Bernal et al. [4] compare results of using the surrogate strategy to that of using the theoretical results directly.

Both studies resort to branch and bound procedures when it is no longer possible to generate constraints which cut off the current LP solution. Crowder et al. [11] use IBM's MIP370 package with an *ad hoc* upper bound procedure. Bernal et al. [4] use Marsten's IP83 code, an extension of XMP to integer programming (see Marsten [42,43]). This code is an all-FORTRAN portable code which attempts to incorporate both breadth-first and depth-first strategies (via parametric analysis). For a complete description of the approach, see Marsten and Morin [44] and the dissertation by Singhal [60]. Included in IP83 is the heuristic by Balas and Martin [1] for determining a feasible integer solution to the problem.

## 6.3. COMPUTATIONAL RESULTS FOR SPARSE ZERO-ONE PROGRAMS

Crowder et al. [11] used a set of real-world large-scale zero-one linear programming problems for their tests. Table 10 summarizes the main characteristics of the

Table 10

Test problem summary

| Name | VARS | ROWS | NON-SOS | VARIANCE* | DENS | $z_{lp}$ | $z_{ip}$ |
|---|---|---|---|---|---|---|---|
| P0033 | 33 | 16 | 11 | 49.6 (33.9) | 17.4 | 2 520.6 | 3 089.0 |
| P0040 | 40 | 24 | 13 | 39.6 (92.8) | 11.3 | 61 796.5 | 62 027.0 |
| P0201 | 201 | 134 | 107 | 0.7 (1.0) | 5.0 | 6 875.0 | 7 615.0 |
| P0282 | 282 | 242 | 44 | 34.3 (16.2) | 2.0 | 176 867.5 | 258 411.0 |
| P0291 | 291 | 253 | 14 | 3.5 (2.3) | 1.9 | 1 705.1 | 5 223.75 |
| P0548 | 548 | 177 | 94 | 123.9 (88.3) | 1.8 | 315.3 | 8 691.0 |
| P1550 | 1550 | 94 | 2 | 8.4 (8.4) | 7.4 | 1 706.5 | 1 708.0 |
| P1939 | 1939 | 109 | 2 | 3.5 (3.5) | 4.8 | 2 051.1 | 2 066.0 |
| P2655 | 2655 | 147 | 2 | 2.3 (2.3) | 3.4 | 6 532.1 | 6 548.0 |
| P2756 | 2756 | 756 | 386 | 149.2 (89.4) | 0.4 | 2 688.7 | 3 124.0 |

*After problem preprocessing.

test problem set. The columns headed VARS, ROWS and NONSOS contain the number of variables, the total number of constraints, and the number of those constraints that are not of the special ordered set type. The column headed VARIANCE gives the mean of the intra-row standard deviations of the absolute values of the non-zero elements of the non-SOS rows of the constraint matrix after preprocessing; the number in parentheses is the standard deviation of this aggregate statistic. As an example, P2756 has 756 rows, of which 386 are non-SOS rows. Computing for each non-SOS row of P2756 the standard deviation of the absolute values of its non-zero elements and averaging over all 386 rows yields an average intra-row variance of 149.2, a substantial variation of the non-zero coefficients; the number 89.4 indicates that the variation is substantial among the non-SOS rows of the constraint matrix, but is somewhat less, on average, than the variation within each non-SOS row. In problems P1550, P1939 and P2655, the figures indicate that only one of the two non-SOS rows contributes to the standard deviation, the other row having identical non-zero elements. Indeed, these problems are set partitioning problems with two additional constraints: one requires that the sum of all variables be a certain number; the other is a knapsack constraint with positive coefficients which, however, have little variation. The column headed DENS specifies the density of the original constraint matrix plus the cost row (i.e. the total number of non-zero elements of $A$ and $c$ divided by the product of $m$ and $n$ and multiplied by 100). Finally, $Z_{lp}$ denotes the optimal objective function value of the linear programming relaxation of problem (ZOP) in user-supplied form, while $Z_{ip}$ denotes the optimal zero-one objective function value.

It can be inferred from the problem characteristics that one can not expect the constraint generation as discussed in subsects. 6.1 and 6.2 to be equally effective on all of the ten test problems.

Crowder et al. [11] performed all computational experiments on the IBM 370/168 computer, running the MVS operating system, at the Thomas J. Watson Research Center in Yorktown Heights, New York. FORTRAN programs were compiled using the FORTRAN H extended compiler. PL1 programs were compiled using the PLI optimizing compiler.

The zero-one preprocessor is the initial computational phase. As indicated, the effect of the preprocessor is to tighten the user-supplied linear programming formulation of the zero-one problem using the standard 'tricks' of integer programming to eliminate inactive constraints, and to fix variables at either zero or one. Table 11 summarizes the results of the initial preprocessor step on the test problem set. The

Table 11

Problem preprocessor summary

| Name | Original problem | | Eliminated | | TIME* |
|---|---|---|---|---|---|
| | ROWS | VARS | ROWS | VARS | |
| P0033 | 16 | 33 | 0 | 0 | 0.01 |
| P0040 | 24 | 40 | 0 | 0 | 0.02 |
| P0201 | 134 | 201 | 0 | 6 | 0.01 |
| P0282 | 242 | 282 | 20 | 0 | 0.02 |
| P0291 | 253 | 291 | 47 | 1 | 0.02 |
| P0548 | 177 | 548 | 20 | 21 | 0.03 |
| P1550 | 94 | 1550 | 0 | 0 | 0.19 |
| P1939 | 109 | 1939 | 0 | 0 | 0.18 |
| P2655 | 147 | 2655 | 0 | 0 | 0.30 |
| P2756 | 756 | 2756 | 17 | 22 | 0.50 |

*CPU minutes. IBM 370/168 MVS.

column headed Original problem specifies the number of rows and variables in the problem before preprocessing. The column headed Eliminated gives the number of rows and variables that the preprocessor successfully eliminated from the problem. The time for the initial preprocessor phase, in CPU minutes, is given under the TIME column.

Table 12

Constraint generation summary

| Name | Preprocessed problem | | | | Augmented problem | | | | TIME* |
|---|---|---|---|---|---|---|---|---|---|
| | VARS | ROWS | DENS | $Z^*_{lp}$ | PASS | ROWS | DENS | $Z^{**}_{lp}$ | |
| P0033 | 33 | 16 | 17.4 | 2 819.4 | 6 | 36 | 12.1 | 3 065.3 | 0.12 |
| P0040 | 40 | 24 | 11.3 | 61 829.1 | 4 | 29 | 13.8 | 61 862.8 | 0.12 |
| P0201 | 195 | 134 | 5.0 | 7 125.0 | 1 | 139 | 5.0 | 7 125.0 | 0.16 |
| P0282 | 282 | 222 | 1.5 | 176 867.5 | 14 | 462 | 1.3 | 255 033.1 | 1.00 |
| P0291 | 290 | 206 | 1.3 | 1 749.9 | 6 | 278 | 2.5 | 5 022.7 | 0.33 |
| P0548 | 527 | 157 | 1.9 | 3 125.9 | 9 | 296 | 1.3 | 8 643.5 | 0.57 |
| P1550 | 1550 | 94 | 7.4 | 1 706.5 | 1 | 94 | 7.4 | 1 706.5 | 0.81 |
| P1939 | 1939 | 109 | 4.8 | 2 051.1 | 1 | 110 | 4.8 | 2 051.1 | 0.74 |
| P2655 | 2655 | 147 | 3.4 | 6 532.1 | 2 | 149 | 3.4 | 6 535.0 | 2.04 |
| P2756 | 2734 | 739 | 0.4 | 2 701.1 | 8 | 1065 | 0.4 | 3 115.3 | 3.06 |

*CPU minutes, IBM 370/168 MVS.

The constraint-generation procedure is the second computational phase: It operates on the preprocessed problem to produce and solve a linear programming problem with a better (greater in the minimization case) optimal continuous objective function value. Table 12 summarizes the results of the first constraint-generation step on the test problem set. The columns headed Preprocessed problem specify the number of constraints, the number of variables, the density of the problem, and the optimal continuous objective function value $Z^*_{lp}$ of the problem produced by the preprocessor, before constraint generation. The Augmented problem section of the table indicates the number of constraint-generation 'passes' (i.e. the number of intermediate linear programming problems required), the number of constraints in the final linear programming problem produced, the density of its constraint matrix including the cost row, and its optimal continuous objective function value $Z^{**}_{lp}$. The time for the initial constraint-generation phase, in CPU minutes, is given under the heading TIME. This time includes all linear programming calculations in the constraint-generation phase, including the solution of the original linear program.

The real measure of the effectiveness of the constraint-generation procedure is the extent to which it closes the 'gap' between the optimal value of the objective function of the linear programming relaxation and the optimal zero-one objective function value. Table 13 summarizes these data for the first application of the constraint-generation procedure. The column headed $\Delta$ROWS specifies the number of

Table 13

Effect of constraint generation

| Name | $\Delta$ROWS | GAP | RATIO |
|---|---|---|---|
| P0033 | 20 | 269.6 | 0.92 |
| P0040 | 5 | 197.9 | 0.17 |
| P0201 | 5 | 490.0 | 0.00 |
| P0282 | 240 | 81 543.5 | 0.96 |
| P0291 | 72 | 3 473.8 | 0.94 |
| P0548 | 139 | 5 565.1 | 0.99 |
| P1550 | 0 | 1.5 | 0.00 |
| P1939 | 1 | 14.9 | 0.00 |
| P2655 | 2 | 15.9 | 0.19 |
| P2756 | 326 | 422.9 | 0.98 |

constraints generated by the procedure. The GAP is the difference between the optimal zero-one objective function value and the optimal continuous objective function value before constraint generation, but after problem preprocessing (i.e. the GAP is $Z_{ip} - Z_{1p}^{*}$). The figure RATIO is computed as RATIO $= (Z_{1p}^{**} - Z_{1p}^{*})/(Z_{ip} - Z_{1p}^{*})$. Table 13 clearly supports the statements made earlier as to when one should expect the constraint generation, as discussed in subsects. 6.1 and 6.2, to be effective: The problems where this part of the overall procedure shows a significant effect are those having a low density (with the exception of P0033), and a substantial number of knapsack constraints having a high variance of non-zero elements in each row. In the remaining test problems, few additional constraints were generated, and while the authors do not include comparative bench marks, the ones that were generated probably had a positive effect upon the overall computation times. Table 14 summarizes the time, in CPU minutes, for the entire system execution on the test problem set. The system remains in the branch and bound procedure until an integer solution is generated. If the integer solution is non-optimal and variables can be fixed based on reduced costs, then control returns to the preprocessing phase (see sect. 3). PIPX passes is the number of times the branch and bound procedure is called. The columns headed TIME specify the aggregate times spent in each of the three main computational phases, and the total time required to solve the test problem. In particular, the heading LP/constraint generation refers to the total time spent on solving the linear programs and generating all of the constraints.

Table 14

PIPX execution summary

| Name | PIPX passes | TIME* | | | |
|---|---|---|---|---|---|
| | | Preprocessor | LP constraint generation | Branch and Bound | Total |
| P0033 | 1 | 0.01 | 0.12 | 0.56 | 0.69 |
| P0040 | 2 | 0.03 | 0.13 | 0.02 | 0.18 |
| P0201 | 1 | 0.01 | 0.16 | 9.90 | 10.07 |
| P0282 | 1 | 0.02 | 1.00 | 11.70 | 12.72 |
| P0291 | 2 | 0.04 | 0.51 | 0.30 | 0.85 |
| P0548 | 2 | 0.12 | 0.70 | 0.09 | 0.91 |
| P1550 | 3 | 0.50 | 0.90 | 0.10 | 1.50 |
| P1939 | 3 | 0.49 | 0.86 | 13.64 | 14.99 |
| P2655 | 3 | 0.82 | 3.23 | 2.70 | 6.75 |
| P2756 | 2 | 1.07 | 4.03 | 49.32 | 54.42 |

*CPU minutes, IBM 370/168 MVS

All test problems considered during the course of this study were solved to optimality. The methodology described above for the solution of large-scale zero-one linear programming problems produced impressive – by 1983 standards – computational results, particularly on sparse problems having no apparent special structure. The authors obtained the test problems from various sources within and outside the IBM Corporation. The authors report no comparative data on previous solution attempts, but claim that most problems in this set were originally considered not amenable to exact solution in economically feasible computation times. The computational results presented in Crowder et al. [11] contradict this sentiment and strongly confirm the hypothesis that a combination of problem preprocessing, facet-defining cutting planes, and judicious use of branch and bound techniques permits, in reasonable computation times, the optimization of sparse large-scale zero-one linear programming problems, even when no apparent special structure is present.

Computational results from the Bernal et al. [4] study will be available the end of 1985. The authors report preliminary results that good preprocessing and the heuristic by Balas and Martin [1] further improve the results obtained by Crowder et al. [11]. The Bernal et al. [4] study is significant in that it is the only pre-1985 study known to these authors that attempts to use constraint generation within an all-FORTRAN machine-portable non-proprietary code. The test effort took place on a UNIVAC 1100 series machine which allows only 262 K words of storage. Within this significant machine memory restriction, the authors were still successful in solving the first six problems reported by Crowder et al., as well as a variety of other difficult zero-one problems not included in the Crowder et al. study.

## 7. Conclusions

The computational results reported in the last three sections of this report point to the suitability of using facet-defining cutting planes for the purpose of proving optimality in difficult, large combinatorial optimization problems. These constraints – generated only when needed – provide the means for reducing substantially the gap between the linear-programming relaxation and the integer program, thereby making large-scale combinatorial optimization problems tractable.

Codes which are LP-based and which use cutting plane methods are not restricted to these three classes of problems. Facets for many other specially structured problems are known, for example, plant location problems (see Cho et al. [7,8]); set covering, packing and partitioning problems (see Padberg [3]); and mixed 0-1 linear programming (see Martin and Schrage [45] and the papers of the C.O.R.E. group mentioned in the introduction). Code development for such problems will require procedures that generate the facet-defining inequalities via polynomially-bounded algorithms or at least generate violated inequalities which approximate the facet by efficient procedures. The codes are likely to consist of a preprocessor, a variety of heuristics for generating good feasible solutions, algorithms for generating facet-inducing cutting planes, and a branch and bound algorithm, following the general design outlines in sect. 3 of this paper. These codes will certainly be complex, but are likely to lead to methods for solving *to optimality* – with reasonable computational effort – many of the classes of difficult combinatorial problems for which only heuristic 'guesses' at the optimal solution are currently possible. The excellent computational results of the most recent study by Van Roy and Wolsey [65] show that the validity of this statement is not restricted to the (pure) zero-one combinatorial optimization problems considered in this survey, but extends as well to the case of mixed zero-one linear programming problems.

## References

[1] E. Balas and R. Martin, Pivot and complement – a heuristic for 0-1 programming, Management Science 26(1980)86.

[2] I. Baranyi, T. Van Roy and L. Wolsey, Strong formulations for multi-item capacitated lot sizing, CORE Report No. 8313, Université Catholique, Louvain-la-Neuve, Belgium (1983a).

[3] I. Baranyi, T. Van Roy and L. Wolsey, Uncapacitated, lot sizing: The convex hull of solutions, CORE Report No. 8314, Université Catholique, Louvain-la-Neuve, Belgium (1983b).

[4] J. Bernal, K.L. Hoffman and M. Padberg, Pure zero-one linear programming problems: A computational study, Tech. Rep., National Bureau of Standards, Gaithersburg, MD (1985).

[5] N. Biggs, E. Lloyd and R. Wilson, *Graph Theory 1736–1936* (Clarendon Press, Oxford, 1976).

[6] R. Burkard and U. Derigs, *Assignment and Matching Problems: Solution Methods with FORTRAN-Programs*, Springer Lecture Notes in Economics and Mathematical Systems, No. 184 (Springer-Verlag, Berlin, 1980).

[7] D. Chinhyung Cho, E.L. Johnson, M. Padberg and M.R. Rao, On the uncapacitated plant location problem I: Valid inequalities and facets, Mathematics of Operations Research 8 (1983)579.

[8] D. Chinhyung Cho, M. Padberg and M.R. Rao, On the uncapacitated plant location problem II: Facets and lifting theorems, Mathematics of Operations Research 8(1983)590.

[9] V. Chvátal, Edmonds polytopes and weakly Hamiltonian graphs, Math. Progr. 5(1973)29.

[10] H. Crowder and M. Padberg, Solving large-scale symmetric traveling salesman problems to optimality, Management Science 26(1980)495.

[11] H. Crowder, E. Johnson and M. Padberg, Solving large-scale linear zero-one programming problems, Oper. Res. 31(1983)803.

[12] G. Dantzig, D.R. Fulkerson and S. Johnson, Solution of a large-scale traveling salesman problem, Oper. Res. 2(1954)293.

[13] G. Dantzig, *Linear Programming and Extensions* (Princeton University Press, New Jersey, 1963).

[14] J. Edmonds, Maximum matching and a polyhedron with 0, 1 vertices, Journal of Research, National Bureau of Standards 69B(1965)125.

[15] L. Euler, *Commentationes Arithmeticae Collectae* (St. Petersburg, 1736).

[16] B. Fleischmann, The traveling salesman problem on a road network, Working Paper, Fachbereich Wirtschaftswissenschaften, Universität Hamburg (1981), revised July 1982.

[17] B. Fleischmann, Linear programming approaches to traveling salesman and vehicle scheduling problems, paper presented at the XI Int. Symposium on Mathematical Programming, Bonn, FRG (1982).

[18] R. Gomory and T.C. Hu, Multi-terminal network flows, J. SIAM 9(1961)551.

[19] M. Grötschel, On the symmetric TSP: Solution of a 120-city problem, Math. Progr. Studies 12(1980)61.

[19a] M. Grötschel and O. Holland, Solving matching problems with linear programming, Preprint No. 37, Mathematisches Institut, Universität Augsburg, FRG (1984).

[20] M. Grötschel, L. Lovasz and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, Combinatorics 1(1981)169.

[21] M. Grötschel and M. Padberg, Polyhedral aspects of the traveling salesman problem I: Theory, in: *The Travelling Salesman Problem*, ed. E. Lawler et al. (Wiley, Chichester, 1985).

[22] M. Grötschel, M. Jünger and G. Reinelt, On the acyclic subgraph polytope, WP 82215-OR, Institut für Operations Research, Universität Bonn, FRG (1982a).

[23] M. Grötschel, M. Jünger and G. Reinelt, Facets of the linear ordering polytope, WP 82217-OR, Institut für Operations Research, Universität Bonn, FRG (1982b).

[24] M. Grötschel, M. Jünger and G. Reinelt, Optimal triangulation of large real-world input/output matrices, Preprint No. 9, Mathematisches Institut, Universität Augsburg, FRG (1983).

[25] T. Hankins, *Sir William Rowan Hamilton* (John Hopkins University Press, Baltimore, 1980).

[26] Held and Karp, The traveling salesman and minimum spanning trees, Part I: Oper. Res. 18 (1970)1138; Part II: Math. Progr. 1(1971)6.

[27] S. Hong. A linear programming approach for the traveling salesman problem. Ph.D. Thesis, John Hopkins University, Baltimore (1972).

[28] IBM. Pure Integer Programming executor program description and operations manual. Program No. 5785-GBX. IBM SB11-5712-0. First edition, Aug. 1982.

[29] E. Johnson and M. Padberg, A note on the knapsack problem with special ordered sets, Oper. Res. Lett. 1(1981)38.

[30] E. Johnson and M. Padberg, Degree-two inequalities, clique facets and biperfect graphs, Ann. of Disc. Math. 16(1982)169.

[31] Karp and Papadimitriou, On linear characterizations of combinatorial optimization problems, 21st Annual Symposium on the Foundation of Computer Science, 1980, p. 1.

[32] D. Knuth, The traveling salesman problem, illustrative example in: *Frontiers of Science, from Microcosm to Macrososm,* by H. Sullivan, New York Times (February 24, 1976) p. 18.

[33] P. Krolak, W. Felts and G. Marble, A man-machine approach toward solving the traveling salesman problem, CACM 14(1971)327.

[34] A. Land, The solution of 100-city symmetric traveling salesman problems, Research Report, London School of Economics, London (1979).

[35] E. Lawler, *The Traveling Salesman Problem,* ed. E. Lawler, J.K. Lenstra, A. Rinooy Kan and D. Shmoys (Wiley, Chichester, 1985).

[36] W. Leontief, Quantitative input and output relations in the economic system of the United States, Review of Economic Systems 18(1936)105.

[37] W. Leontief, *The Structure of the American Economy 1919-1939* (Oxford University Press, New York, 1951).

[38] W. Leontief, The structure of development, Scientific American (1963).

[39] W. Leontief, *Input-Output Economics* (Oxford University Press, New York, 1966).

[40] S. Lin and B. Kerningham, An effective heuristic algorithm for the traveling salesman, Oper. Res. 21(1973)498.

[41] J. Lorie and L.J. Savage, Three problems in capital rationing, Journal of Business 28(1955) 229.

[42] R.E. Marsten, XMP: A structured library of subroutines for experimental mathematical programming, ACM Trans. on Mathematical Software 7(1981)481.

[43] R.E. Marsten, User's guide to IP83, Tech. Report, Department of Management Information Systems, University of Arizona (1983).

[44] R.E. Marsten and T.L. Morin, A hybrid approach to discrete mathematical programming, Math. Progr. 14(1978)21.

[45] R.K. Martin and L. Schrage, Subset coefficient reduction cuts for 0-1 mixed integer programming, Tech. Report, Graduate School of Business, University of Chocago, Illinois (1983).

[46] K. Menger, Botenproblem, in: *Ergebnisse eines Mathematischen Kolloquiums Wien, 1930,* Heft 2, ed. K. Menger (Leipzig, 1932) p. 11.

[47] T. Miliotis, Integer programming approaches to the traveling salesman problem, Math. Progr. 10(1976)367.

[48] T. Miliotis, Using cutting planes to solve the symmetric traveling salesman problem, Math. Progr. 15(1979)177.

[49] M. Padberg, On the facial structure of set packing polyhedra, Math. Progr. 5(1973)199.

[50] M. Padberg, Characterizations of totally unimodular, balanced and perfect matrices, in: *Combinatorial Programming: Methods and Applications,* ed. B. Roy (Reidel, Dordrecht, 1975) p. 275.

[51] M. Padberg, A note on zero-one programming, Oper. Res. 23(1975)833.

[52] M. Padberg, On the complexity of set packing polyhedra, Ann. Discr. Math. 1(1977)421.

[53] M. Padberg, Covering, packing and knapsack problems, Ann. Discr. Math. 4(1979)265.
[54] M. Padberg and S. Hong, On the symmetric traveling salesman problem: A computational study, Math. Progr. Studies 12(1980)78.
[55] M. Padberg, $(1, k)$-configurations and facets for packing problems, Math. Progr. 18(1980) 94.
[56] M. Padberg and M.R. Rao, The Russian Method for linear inequalities III: Bounded integer programming, INRIA, Rapport de recherche, Rocquencourt, revised May 1981, to appear in Math. Progr. Studies.
[57] M. Padberg and M.R. Rao, Odd minimum cut-sets and $b$-matchings, Math. Oper. Res. 7 (1982)67.
[58] M. Padberg, T. Van Roy and L. Wolsey, Valid linear inequalities for fixed charge problems, CORE Report No. 8232, Université Catholique, Louvain-la-Neuve, Belgium (1982), to appear in Oper. Res. (1985).
[59] M. Padberg and M. Grötschel, Polyhedral aspects of the traveling salesman problem II: Computation, in: *The Traveling Salesman Problem*, ed. E. Lawler et al. Wiley, Chichester, 1985).
[60] J. Singhal, Fixed order branch and bound methods for mixed integer programming, Dissertation, Department of Management Information Systems, University of Arizona (1982).
[61] G.L. Thompson and R.L. Karp, A heuristic approach to solving traveling salesman problems, Management Science 10(1964)225.
[62] L. Trotter, A class of facets for vertex-packing polyhedra, Discr. Math. 12(1975)373.
[63] T. Van Roy and L. Wolsey, Valid inequalities for mixed 0-1 programs, CORE Report No. 8316, Université Catholique, Louvain-la-Neuve, Belgium (1983).
[64] T. Van Roy and L. Wolsey, Valid inequalities and separation for uncapacitated fixed charge networks, CORE Report No. 8410, Université Catholique, Louvain-la-Neuve, Belgium (1984a).
[65] T. Van Roy and L. Wolsey, Solving mixed integer programs by automatic reformulation, CORE Report No. 8432, Université Catholique, Louvain-la-Neuve, Belgium (1984b).
[66] H.M. Weingartner, *Mathematical Programming and the Analysis of Capital Budgeting* (Prentice-Hall, Englewood Cliffs, NJ, 1963).
[67] H. Weyl, Elementare Theorie der konvexen Polyheder, Comm. Math. Helv. 7(1935)290, translated in: *Contributions to the Theory of Games*, Vol. 1 (Annals of Mathematics Studies, No. 24, Princeton, 1950) p. 3.
[68] L. Wolsey, Faces for a linear inequality in 0-1 variables, Math. Progr. 8(1975)165.
[69] H.P. Young, On permutations and permutation polytopes, Math. Progr. Studies 8(1978)128.