# A Cutting Plane Algorithm for the Linear Ordering Problem

MARTIN GRÖTSCHEL, MICHAEL JÜNGER AND GERHARD REINELT

*Universitat Augsburg, Augsburg, West Germany*

(Received August 1982; accepted October 1983)

The linear ordering problem is an NP-hard combinatorial optimization problem with a large number of applications (including triangulation of input-output matrices, archaeological seriation, minimizing total weighted completion time in one-machine scheduling, and aggregation of individual preferences). In a former paper, we have investigated the facet structure of the 0/1-polytope associated with the linear ordering problem. Here we report on a new algorithm that is based on these theoretical results. The main part of the algorithm is a cutting plane procedure using facet defining inequalities. This procedure is combined with various heuristics and branch and bound techniques. Our computational results compare favorably with the results of existing codes. In particular, we could triangulate all input-output matrices, of size up to 60 × 60, available to us within acceptable time bounds.

IN THIS PAPER we report on a new algorithm for the solution of linear ordering problems. The algorithm is based on our investigations of the polytope associated with this problem (Grötschel et al. [1982a]) and combines heuristics, cutting plane, and branch and bound techniques. The main core of the algorithm is a cutting plane method utilizing the linear programming code of IBM's MPSX-package. Our computational results compare favorably with previous codes for this problem. In particular, we could triangulate all input-output matrices available to us within reasonable time bounds (this application was the starting point of our study). Before summarizing several applications of the linear ordering problem, we state several definitions that will permit us to describe the problem.

A directed graph or *digraph* $D = (V, A)$ consists of a finite nonempty set $V$ of *nodes* and a set $A$ of *arcs* that are ordered pairs of different elements of $V$. (Loops and parallel arcs are of no interest for our purposes.) The number of nodes of $V$ is called the *order* of $D$. If $a = (u, v) \in A$ is an arc, then the nodes $u, v$ are the *endnodes* of $a$. For a set

*Subject classification:* 133 triangulation of input-output tables, 432 polyhedral combinatorics, 825 cutting plane methods.



**Operations Research**
Vol. 32, No. 6, November-December 1984

0030-364X/84/3206-1195 $01.25

$B$ of arcs, we let $V(B)$ denote the set of nodes appearing as endnodes of arcs in $B$.

A *graph* $G = [V, E]$ consists of a finite nonempty node set $V$ and a set $E$ of edges that are unordered pairs of different nodes called the endnodes of the edge. If $G = [V, E]$ is a graph, then a digraph $D$ is called an *orientation* of $G$ if, whenever $ij \in E$, $D$ contains an arc $(i, j)$ or $(j, i)$ but not both.

If $D = (V, A)$ is a digraph, then every digraph $D' = (V', A')$ with $V' \subseteq V$ and $A' \subseteq A$ is called a *subdigraph* of $D$. We also say that $D$ contains $D'$ and that $D'$ is contained in $D$.

A set of arcs $P = \{(v_1, v_2), (v_2, v_3), \cdots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k)\}$ in $D = (V, A)$ such that $v_i \neq v_j$ for $i \neq j$ is called a $(v_1, v_k)$-*dipath* of *length* $k - 1$. If $P$ is a $(v_1, v_k)$-dipath and $(v_k, v_1) \in A$, then $C = P \cup \{(v_k, v_1)\}$ is called a *dicycle of length* $k$ or $k$-*dicycle*. A digraph $D = (V, A)$ or just an arc set $A$ that contains no dicycle is called *acyclic*.

A digraph $A$ is called *complete* if for every pair $i, j$ of nodes, $D$ contains the arcs $(i, j)$ and $(j, i)$. Up to isomorphism, there is only one complete digraph of order $n$. We will denote this digraph by $D_n = (V, A_n)$. A graph $G$ is *complete* if every two nodes of $G$ are adjacent. The complete graph of order $n$ is denoted by $K_n$. A *tournament* is a digraph $D = (V, A)$ such that for every two nodes $u$ and $v$, $A$ contains exactly one arc with endnodes $u$ and $v$. A tournament on $n$ nodes is an orientation of the complete graph $K_n$.

A *linear ordering* (or permutation) of a finite set $V$ with $|V| = n$ is a bijective mapping $\sigma : [1, 2, \cdots, n] \to V$. For $u, v \in V$, we say that $u$ is "better than" or "before" $v$ if $\sigma^{-1}(u) < \sigma^{-1}(v)$. We sometimes write $\langle i_1, i_2, \cdots, i_n \rangle$ to denote the linear ordering $\sigma(j) = i_j$ for $j = 1, \cdots, n$. In many applications, a "value" or a "cost" can be associated with a linear ordering in the following way. For every two elements $u, v \in V$, a value $c_{uv}$ and a value $c_{vu}$ are given which can be interpreted as the profit we obtain from having $u$ "before" $v$ or $v$ "before" $u$ in a linear ordering. Then, the total value of a linear ordering is given by

$$\sum_{\sigma^{-1}(u)<\sigma^{-1}(v)} c_{uv}.$$

Given a linear ordering of the nodes $V$ of a digraph, the arc set $\{(u, v) \mid \sigma^{-1}(u) < \sigma^{-1}(v)\}$ forms an acyclic tournament on $V$, and similarly, if $(V, T)$ is an acyclic tournament, then this induces a linear ordering of $V$. Using this graph theoretical interpretation, we can state an instance of the *linear ordering problem* as follows.

Given a complete digraph $D_n = (V, A_n)$ with arc weights $c_{ij}$ for all $(i, j) \in A_n$, find an acyclic tournament $(V, T)$ in $D_n$ such that

$$c(T) := \sum_{(i,j)\in T} c_{ij}$$

is as large as possible.



For ease of notation, whenever we shall use the word tournament in the sequel we shall mean the arc set of a tournament.

The linear ordering problem has been known by several other names in the literature, e.g., Young [1978], calls it a permutation problem, and Korte and Oberhofer [1968, 1969] call it a triangulation problem. In most cases, these names reflect the application from which this optimization problem is derived.

A closely related problem is the *acyclic subgraph problem* where a digraph $D = (V, A)$ with arc weights $c_{ij}$ for all $(i, j) \in A$ is given and one must find an acyclic arc set $B \subseteq A$ that maximizes $c(B)$. The acyclic subgraph problem and the linear ordering problem can be transformed into each other by a simple construction. Therefore, a good algorithm for one problem yields a good one for the other. Another version of this problem is the *feedback arc set problem* where in a digraph $D = (V, A)$ with arc weights $c_{ij}$ for all $(i, j) \in A$ a set $F \subseteq A$ must be found that intersects every directed cycle and for which $c(F)$ is minimum. If $F \subseteq A$ is such a set then, clearly, $B := A \backslash F$ is a maximum acyclic arc set, and vice versa.

The linear ordering problem (or one of its equivalent versions) has many real-world applications. Lists of such applications can be found in Korte and Oberhofer [1968] or Lenstra [1973, 1977]. We shall mention only a few of these applications.

In input-output analysis, the economy of a region (usually a state) is divided into $n$ sectors, and an $(n, n)$-input-output table $X$ is constructed where the entry $x_{ij}$ denotes the amount of deliveries from sector $i$ to sector $j$ in a certain year. The problem of permuting the rows and columns of $X$ simultaneously such that the sum of the entries of the permuted matrix above the main diagonal is as large as possible is called *triangulation problem*. Considering an entry $x_{ij}$ of an $(n, n)$-input-output matrix $X$ as the weight of the arc $(i, j)$ of $D_n$, one obtains a linear ordering problem. Triangulated input-output matrices allow interesting interpretations of the structure of an economy and comparisons between different countries, cf. Wessels [1981].

If a group of persons wants to order $n$ alternatives (wines, investments) according to "desirability," each person could make up its own preference scheme (saying that he prefers some alternative $i$ to alternative $j$; even weights could be assigned to such preferences). By choosing $c_{ij}$ to be the number of times a person prefers $i$ to $j$ (or the sum of the weights assigned to "$i$ preferred to $j$") and solving the linear ordering problem, one obtains a linear ordering that is as "consistent" as possible with the individual preferences. This application often is referred to as *aggregation of individual preferences*. If every person of the group must choose between every pair of alternatives, the problem is sometimes referred to as *ranking by paired comparison*. Applications of this method can be found, for

instance, in marketing (product positioning and design, cf. Slater [1961] for the composition of dogfood) and psychology (hierarchy problems).

The problem of minimizing total weighted (or average weighted) completion time in one-machine scheduling is another important equivalent version of the linear ordering problem, cf. Boenchendorf [1982].

Glover et al. [1974] describe an application of this problem in archaeology where a "most probable" chronological ordering of pottery types must be found. Here $c_{ij}$ denotes a weighted sum over all graves at which pottery type $i$ was found below type $j$.

The linear ordering problem can also be used to break ties in sports tournaments or to rank the players. Suppose in a fencing tournament (soccer tournament) each fencer (team) fights each other once. Usually, the participants are ordered lexicographically by the number of victories (or points) using the differences (quotients, etc.) of the scores to break ties. Letting $c_{ij}$ denote the number of times person (team) $i$ has landed a hit (scored a goal) on person (team) $j$, then an optimum solution of this linear ordering problem might better reflect the relative individual performance of the fencers (teams), than would the usual ways of ordering, since the linear order better accounts for the "quality" of the individual results (high difference of goals).

There is, however, one major obstacle in the application of the linear ordering problem. Namely, the problem is NP-hard, and it seems, therefore, unlikely that algorithms can be found that solve all such problems in polynomial time. On the other hand, this observation does not necessarily mean that it is impossible to devise algorithms that are reasonably fast for problems coming from certain classes of applications, and that can be used successfully in practice.

We shall describe such an algorithm here. Our method works very well for triangulating input-output matrices, but requires much more computational time if certain kinds of random objective functions are used.

## 1. THEORETICAL BACKGROUND OF THE ALGORITHM

Our algorithm is based on our results about the polyhedron associated with the linear ordering problem described in Grötschel et al. [1982a]. We shall briefly summarize those aspects of these results that are important for the description and "philosophy" of the method.

Let $D_n = (V, A_n)$ be the complete digraph of order $n$, and set

$$\mathcal{T}_n := \{T \subseteq A_n \mid (V, T) \text{ is an acyclic tournament}\}.$$

Given weights $c_{ij}$ for all $(i, j) \in A_n$, the linear ordering problem can be stated as $\max\{c(T) \mid T \in \mathcal{T}_n\}$. This problem can be formulated in polyhedral terms in the following way. Let $\mathbb{R}^m$, $m := |A_n| = n(n-1)$, denote the real vector space where every component of a vector $x \in \mathbb{R}^m$ is indexed by an arc $(i, j) \in A_n$. For convenience, we write $x_{ij}$ instead of $x_{(i,j)}$. For every arc set $A \subseteq A_n$, define the incidence vector $x^A \in \mathbb{R}^m$ of $A$ as follows: $x^A_{ij} = 1$ if $(i, j) \in A$, and $x^A_{ij} = 0$ if $(i, j) \notin A$. The *linear ordering polytope* $P^n_{LO}$ on $D_n$ is the convex hull of the incidence vectors of all acyclic tournaments in $D_n$, i.e.

$$P^n_{LO} = \text{conv}\{x^T \in \mathbb{R}^m \mid T \in \mathcal{T}_n\}. \tag{1}$$

By definition, every vertex of $P^n_{LO}$ corresponds to an acyclic tournament and vice versa. Thus, the linear ordering problem can be solved (in principle) via the linear program $\max c^T x$, $x \in P^n_{LO}$. The problem with this approach, however, is that $P^n_{LO}$ is given as the convex hull of $n!$ points in $\mathbb{R}^m$ while the application of linear programming techniques requires a representation of $P^n_{LO}$ by means of linear equations and inequalities. Since the linear ordering problem is NP-hard, it is very unlikely that a complete (and nonredundant) system of equations and inequalities describing $P^n_{LO}$ can ever be found. We were, however, able to find systems of equations and inequalities describing $P^n_{LO}$ partially. Moreover, we could prove that any complete and nonredundant system describing $P^n_{LO}$ must contain these equations and inequalities (or equivalent forms of these).

A minimal representation of the affine hull of $P^n_{LO}$, i.e.

$$\text{aff}(P^n_{LO}) = \{x \in \mathbb{R}^m \mid x = \textstyle\sum_{i=1}^s \lambda_i x_i \quad \text{for some} \quad x_i \in P^n_{LO},\ \lambda_i \in \mathbb{R} \quad \text{with} \quad \lambda_1 + \cdots + \lambda_s = 1\},$$

is easy to obtain:

THEOREM 1. *Let $n \geq 2$, then the solution set of the system of equations*

$$x_{ij} + x_{ji} = 1 \quad \textit{for all} \quad i, j \in V,\ i \neq j \tag{2}$$

*is equal to $\text{aff}(P^n_{LO})$. Moreover, the matrix of the equation system has full rank $\binom{n}{2}$, which implies that the dimension of $P^n_{LO}$ equals $m - \binom{n}{2} = \binom{n}{2}$ and any linear description of $P^n_{LO}$ requires all these equations (and none less).*

An inequality $a^T x \leq b$ is called *valid* with respect to $P^n_{LO}$ if $P^n_{LO} \subseteq \{x \in \mathbb{R}^m \mid a^T x \leq b\}$, and a valid inequality defines a *facet* of $P^n_{LO}$ if $\dim(P^n_{LO}) = \dim(P^n_{LO} \cap \{x \mid a^T x = b\}) + 1$. Two valid inequalities $a^T x \leq b$, $c^T x \leq d$ are called *equivalent* if $P^n_{LO} \cap \{x \mid a^T x = b\} = P^n_{LO} \cap \{x \mid c^T x = d\}$. Any complete description of $P^n_{LO}$ requires at least one inequality to define every facet of $P^n_{LO}$. Such a system is *nonredundant* if for every facet of $P^n_{LO}$ there is exactly one inequality defining it, i.e., the system contains no equivalent inequalities. In Grötschel et al. [1982a], we have described several classes of valid inequalities for $P^n_{LO}$, and we proved that certain subclasses of these inequalities define facets of $P^n_{LO}$.

We shall describe here only those inequalities that are of interest later in our development.

THEOREM 2. *Let $D_n = (V, A_n)$ be the complete digraph of order $n \geq 6$.*

(a) *The trivial inequalities $x_{ij} \geq 0$ and $x_{ij} \leq 1$ define facets of $P_{LO}^n$ for all $(i, j) \in A_n$. Each inequality $x_{ij} \geqq 0$ is equivalent to the inequality $x_{ji} \leq 1$ (and vice versa) and to no other trivial inequality. Thus, among these inequalities it suffices to consider the system of inequalities*

$$x_{ij} \geq 0 \quad \text{for all} \quad (i, j) \in A_n. \tag{3}$$

(b) *The 3-dicycle inequalities*

$$x(C) \leq 2 \text{ for all 3-dicycles } C \subseteq A_n \tag{4}$$

*define facets of $P_{LO}^n$. None of these inequalities is equivalent to any of the inequalities (3). (Note that for every $k$-dicycle $C$ for $k \geq 4$, the inequality $x(C) \leq k - 1$ is valid but redundant with respect to $P_{LO}^n$.)*

(c) *Let $U = \{u_1, \cdots, u_k\}$, $W = \{w_1, \cdots, w_k\} \subseteq V$ be disjoint sets of nodes of cardinality $3 \leq k \leq n/2$. Let $A$ (called a $k$-fence) be the union of the arc sets $\{(u_i, w_i) \mid i = 1, \cdots, k\}$ (these arcs are called pales) and $\{(w_i, u_j) \mid i, j \in \{1, \cdots, k\}, i \neq j\}$ (called pickets), then the $k$-fence inequality*

$$x(A) \leq k^2 - k + 1 \tag{5}$$

*defines a facet of $P_{LO}^n$. No $k$-fence inequality is equivalent to any of the inequalities (3) and (4). (Note that $k$-fences are particular orientations of the complete bipartite graph $K_{k,k}$.)*

(d) *Let $M \subseteq A_n$ be an arc set (called a Möbius ladder) that is the union of dicycles $C_1, \cdots, C_k$ satisfying the following properties:*

(i) *$3 \leq k$ and $k$ odd.*

(ii) *The length of $C_i$ for $i = 1, \cdots, k$ is three or four.*

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

(iv) *Each node $u \in V(M)$ is contained in at least three arcs of $M$.*

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

(vi) *Given any dicycle $C_j$, $j \in \{1, \cdots, k\}$, set $J = \{1, \cdots, k\} \cap (\{j-2, j-4, \cdots\} \cup \{j+1, j+3, \cdots\})$, then the set $M \setminus \{e_i \mid i \in J\}$ contains exactly one dicycle, namely $C_j$.*

*Then, the Möbius ladder inequality*

$$x(M) \leq |M| - (k+1)/2 \tag{6}$$



*defines a facet of $P_{LO}^n$. 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 (6) is equivalent to any of the inequalities (3), (4), (5).*

Let us define

$$P_1^n := \{x \in \mathbb{R}^m \mid x \text{ satisfies (2) and (3)}\},$$

$$P_2^n := \{x \in \mathbb{R}^m \mid x \text{ satisfies (2), (3), and (4)}\},$$

$$P_3^n := \{x \in \mathbb{R}^m \mid x \text{ satisfies (2), (3), (4), and (5)}\},$$

$$P_4^n := \{x \in \mathbb{R}^m \mid x \text{ satisfies (2), (3), (4), (5), and (6)}\}.$$

Then, clearly,

$$P_{LO}^n \subseteq P_4^n \subseteq P_3^n \subseteq P_2^n \subseteq P_1^n,$$

and in fact for $n \geq 6$ the subset relation is strict for the three last relations. We do not know whether $P_{LO}^6 \neq P_4^6$, but we can prove that $P_{LO}^n \neq P_4^n$ for $n \geq 7$.

Thus, for $i = 1, \cdots, 4$, the four linear programs

$$\max\{c^T x \mid x \in P_i^n\}$$

are successively stronger linear relaxations of the linear ordering problem which (if solvable in reasonable time) can be used to obtain good upper bounds for the optimum value of the linear ordering problem.

Any linear program over $P_1^n$ is solvable by inspection (set $x_{ij} = 1$ if $c_{ij} \geq c_{ji}$ and set $x_{ij} = 0$ otherwise for $1 \leq i < j \leq n$).

There are $\binom{n}{2}$ equations (2) and $2\binom{n}{3}$ inequalities (3). Thus, the size of a linear program over $P_2^n$ is polynomial in the input data of the linear ordering problem which implies that $\max\{c^T x \mid x \in P_2^n\}$ can be solved in polynomial time with the ellipsoid method. (A more practical though theoretically nonpolynomial way to solve the problem will be described in the next section.)

The number of inequalities in both (5) and (6) is exponential to $n$, so there is no way to write them down explicitly and solve the corresponding linear program. In fact, we do not even know how to describe all Möbius ladders in $D_n$ in a "handy way." The large number of inequalities is, however, no obstacle in principle. If we could solve the separation problem for these inequalities in polynomial time, we would be able to solve linear programs over $P_3^n$ and $P_4^n$ in polynomial time; see Grötschel et al. [1981] for the theoretical background of this problem. The separation problem in our case could be formulated as follows:

*Given a vector $y \in \mathbb{Q}^m$ (we may assume that $y$ satisfies (2), (3) and (4)), decide whether $y$ satisfies the system of inequalities (5) or (6), and, if not, find a $k$-fence or Möbius ladder inequality violated by $y$.*

It is an interesting open problem whether the separation problem for (5) or (6) can be solved in polynomial time. We have only some simple heuristics for some subclasses of (5) and (6). But even simple heuristics may be helpful in finding cutting planes and obtaining good linear programming bounds.

## 2. DESCRIPTION OF THE ALGORITHM

The main part of our algorithm is a cutting plane procedure for the solution of the linear programming relaxation of the linear ordering problem described in Theorem 2. The algorithm is guaranteed to find a solution contained in $P_2^n$; if possible, we add further $k$-fence or Möbius ladder inequalities, and if the solution found this way is not integral, we continue with a branch and bound method that fixes variables in the linear program derived so far and may add further cutting planes.

Since, even for moderately sized linear ordering problems, the number of variables and inequalities is rather large, we have to be as economical as possible in order to keep the linear programs "reasonably sized." We shall later describe how we keep the number of inequalities "small" (in a heuristic manner). The number of variables can be reduced from $n(n-1)$ to $\binom{n}{2}$ by using the (simple-structured) equation system (2). Namely, we make the substitution

$$x_{ji} = 1 - x_{ij} \quad \text{for all} \quad 1 \le i < j \le n \tag{7}$$

and replace $x_{ji}$ for $j > i$ in all inequalities that occur by $1 - x_{ij}$. For every polyhedron $P \subseteq \mathbb{R}^m$, we let $\bar{P}$ denote the "projected" polyhedron in $\mathbb{R}^{\bar{m}}$, $\bar{n} = \binom{n}{2}$, which is obtained via this substitution.

If we have a linear program $\max\{c^T x \mid x \in P \subseteq \text{aff}\,(P_{LO}^n)\}$, we set

$$\bar{c}_{ij} := c_{ij} - c_{ji} \qquad 1 \le i < j \le n \tag{8}$$

and obtain a linear program

$$\max\{\bar{c}^T x \mid x \in \bar{P}\} \tag{9}$$

with a (trivial) bijection between the points in $P$ and $\bar{P}$ and such that the value of (9) differs from the value of $\{\max c^T x \mid x \in P\}$ only by the constant $C = \sum_{i<j} c_{ji}$.

Clearly, the polyhedron $\bar{P}_1^n$ is the unit hypercube in $\mathbb{R}^{\bar{m}}$. Thus, any linear program over $\bar{P}_1^n$ can be solved by inspection. The polyhedron $\bar{P}_2^n$ (the one which is most important for our purposes) is given by the following inequalities

$$0 \le x_{ij} \le 1, \qquad 1 \le i < j \le n, \tag{10}$$

$$x_{ij} + x_{jk} - x_{ik} \le 1, \qquad 1 \le i < j < k \le n, \tag{11}$$

$$-x_{ij} - x_{jk} + x_{ik} \le 0, \qquad 1 \le i < j < k \le n. \tag{12}$$



where the inequalities (11) and (12) are the projected versions of the 3-dicycle inequalities (4). For $1 \le i < j < k \le n$, inequality (11) corresponds to the dicycle $(i, j, k)$ and inequality (12) to the dicycle $(i, k, j)$. For the description of $\bar{P}_2^n$, we therefore need $\bar{m} = \binom{n}{2}$ variables and $s = 2\binom{n}{2} + 2\binom{n}{3}$ inequalities. It is important to note that

$$\bar{P}_{LO}^n = \text{conv}\{x \in \bar{P}_2^n \mid x \text{ integral}\}.$$

Thus, whenever a solution of a linear program $\max\{\bar{c}^T x \mid x \in \bar{P}_2^n\}$ turns out to be integral, we have found an optimum solution of $\max\{\bar{c}^T x \mid x \in \bar{P}_{LO}^n\}$, and hence an optimum solution of the linear ordering problem.

Linear programs over $\bar{P}_2^n$ could be solved in polynomial time with the ellipsoid method. The simplex method is, however, more efficient in practice. There are two possible strategies for the solution of such problems. Either we use all inequalities explicitly, or we generate them as cutting planes in an iterative manner. We have chosen the second approach (using of course additional cutting planes) since it seems to be more flexible and to result in better running times. Although, from a complexity theory point of view, the number of constraints for $\bar{P}_2^n$ is "small," there are, e.g., for $n = 60$, still 68,440 3-dicycle inequalities, 3,540 upper and lower bounds, and 1,770 variables to be handled. We do not know any commercial code that could solve such linear programs explicitly. For instance, MPSX allows at most 16,000 constraints. One could, of course, pass over to the dual of the linear program, but similar limits are also reached quite quickly. As we shall see later, "careful ways" to add cutting planes can permit us to maintain linear programs to be solved in manageable sizes.

Figure 1 contains a flowchart of the algorithm we have implemented. We shall now describe the boxes and arrows of the flowchart in more detail. In particular, we shall indicate where we used our empirical observations with various versions of our code to improve the running time in practice.

We assume that a linear ordering problem with objective function coefficients $c_{ij}$ for $1 \le i, j \le n$ is given. We set $\bar{c}_{ij} = c_{ij} - c_{ji}$ for $1 \le i < j \le n$.

*Box 1.* For various features of the branch and bound part of the algorithm (temporary and permanent fixing of variables, stopping criteria, improving performance, and lower storage requirements), we need a lower bound for the value of an optimum linear ordering. This bound is obtained via a heuristic that first calculates a greedy solution (node $i_1$ for which $\sum_{j=1, j\neq i_1}^n c_{i_1 j}$ is maximum is taken first, node $i_2$ for which $\sum_{j=1, i_1 \neq j \neq i_2}^n c_{i_2 j}$ is maximum is taken next, and so on) and then tries to improve this solution by making successive shifting operations of the form $\sigma = \langle 1, 2, \cdots, i-1, i, i+1, \cdots, j-1, j, j+1, \cdots, n-1, n\rangle \to \langle 1, 2, \cdots, i-1, i, j, i+1, \cdots, j-1, j+1, \cdots, n-1, n\rangle$ or $\sigma \to \langle 1, 2,$

$\cdots, i-1, i+1, \cdots, j-1, j, i, j+1, \cdots, n-1, n\rangle$, improving the objective function value until a certain local optimum is obtained. The incidence vector of the linear ordering found this way is denoted by $x^F$.

*Box 2.* Our first linear program is the trivial program $\max\{\tilde{c}^T x \mid 0 \leq x_{ij} \leq 1, 1 \leq i < j \leq n\}$. We initialize all data structures necessary for the

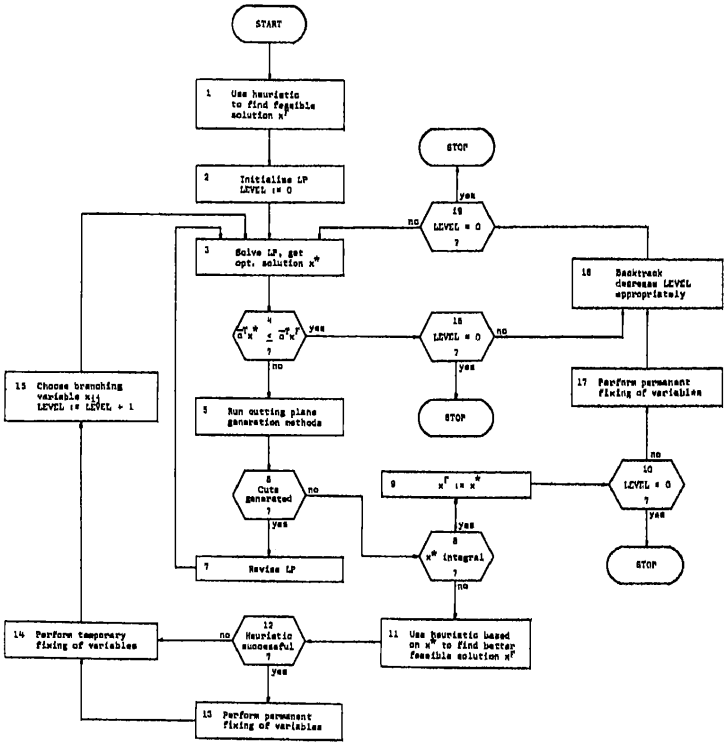


**Figure 1.** Flowchart of algorithm.

use of IBM's MPSX/370. LEVEL indicates the depth of the branch and bound tree we are going to grow. At LEVEL = 0 we are at the root of this tree and start the initial cutting plane phase.

*Box 3.* The first linear program is solved by inspection. In all later steps, we have a current LP and an optimum basis of the previous LP



which we keep in memory. This basis (usually infeasible for the new LP) and the data of the current LP are handed over to MPSX/370, and the routine DUAL is called which tries to revise the basis to find a feasible solution to the new LP. PRIMAL is then called to obtain an optimum solution $x^*$. Our experiences show that in almost all cases the feasible solution found by DUAL was in fact optimal. This way of starting the LP turned out to result in much better running times than starting the LP from scratch or using $x^F$ as a starting basis. For the time being, we describe only the first cutting plane procedure (at LEVEL = 0, no variables are fixed) and forget about the branch and bound part of the algorithm. Since no variables are fixed, we always have $\tilde{c}^T x^* \geq \tilde{c}^T x^F$. If $\tilde{c}^T x^* = \tilde{c}^T x^F$, we go from Box 4 to Box 16 and finish, since $x^F$ has been proved to be optimum. If $\tilde{c}^T x^* > \tilde{c}^T x^F$, then we go to Box 5.

*Box 5.* This is the essential part of the algorithm using the polyhedral results discussed in Section 1. In designing cutting plane generation methods, one usually has to decide whether one uses exact methods (i.e. methods that guarantee that all inequalities in a certain class are checked), if such methods are available, or whether only (fast) heuristics are used. Moreover, one has to decide—if there is a choice—how many cutting planes are added at every step. Our computational experience with the linear ordering and other combinatorial optimization problems indicates that no a priori decisions of this type are possible. One must conduct many experiments to make a good choice.

In our case, it turned out that bounding the number of cuts added definitely reduces the LP-sizes that have to be handled and may reduce the overall running time of the algorithm (see Table VI). We will comment on this observation further in Section 3. Our algorithm contains a parameter MAXCUT that determines how many cutting planes are at most added in one call of Box 5. We now explain how we try to find facet defining inequalities for $P^n_{LO}$ (respectively $P^e_{LO}$) that are violated by $x^*$.

PROBLEM 3. *Identification of 3-dicycle inequalities (4).*

The 3-dicycle inequalities (4) are handled by brute-force. We enumerate all $2\binom{n}{3}$ 3-dicycles and find those that are violated by $x^*$. If the number of violated 3-dicycle inequalities exceeds MAXCUT, then we choose the MAXCUT ones of these that are most violated. To avoid expensive sorting, we use a bucket sort. That is, we partition the interval [0, 1] (these are the possible violations) into small intervals $I_1, \cdots, I_k$; for each violated inequality $a^T x \leq b$, we determine the violation $s = a^T x^* - b$ and sort the inequality into the interval $I_j$ with $s \in I_j$. Then, the cutting planes are chosen from the "highest" intervals in the obvious way. Presently, in another version of the code we are trying different strategies to choose cuts taking information about the objective function into

account (the idea is that, although there is only a small violation in an inequality, there may be a considerable "local" decrease in the objective function value by adding this cut, if all edges of the 3-dicycle have a high weight). Our experience with this procedure is limited at present, and we cannot yet compare this technique with the one described above.

In any case, whenever violated 3-dicycle inequalities are found, we directly go to Box 6 of the flowchart, and we do not try to find other violated inequalities. The heuristics for the other inequalities are called only if the enumeration method concludes that no 3-dicycle inequality is violated by $x^*$. The main reason for this strategy is that we want to maintain the largest possible sparsity of our LP (which—as is well-known—yields considerable numerical and other advantages). The 3-dicycle inequalities have only three nonzero coefficients while all other inequalities (5) and (6) have many more of these.

Our enumeration technique—of course—guarantees that at the end of our initial cutting plane procedure (not taking the branch and bound part into account) a point $x^*$ is found contained in $P_2^n$.

PROBLEM 4. *Identification of k-fence inequalities* (5).

As mentioned in Section 1, we do not know how to solve the separation problem for $k$-fence inequalities in polynomial time. Of course, for $l$ fixed we can check the $k$-fence inequalities for all $k \leq l$ by enumeration. This enumeration technique requires about $O(n^{2l})$ time. Although this bound is polynomial (since $l$ is independent of $n$), it is infeasible for practical purposes, even for the simplest case of 3-fences.

The design of the heuristics that we have tried was guided by the structure of the fractional solutions that are obtained by maximizing $c^Tx$, where $c^Tx \leq k^2 - k + 1$ is a $k$-fence inequality, over the polytope $P_2^n$. An optimum (fractional) solution with value $k^2 - k/2$ is obtained by setting $x_{ij} = 1$ for all pickets $(i, j) \in A$ and $x_{ij} = \frac{1}{2}$ for all pales $(i, j) \in A$. It is obvious how to set the remaining variables. Such a solution for a 3-fence inequality looks as is shown in Figure 2 where solid lines correspond to variables with $x_{ij} = 1$ and broken lines to variables with value $x_{ij} = \frac{1}{2}$.

More generally, suppose we maximize $c^Tx$, where $c^Tx \leq k^2 - k + 1$ is a $k$-fence inequality with $k \geq 4$ over the polytope

$$P_2^n \cap \{x \in \mathbb{R}^m \mid x \text{ satisfies all } l\text{-fence inequalities } l < k\}.$$

Then, we obtain an optimum vertex of this polytope by setting $x_{ij} = 1$ for all pickets $(i, j)$ and $x_{ij} = 1/(k-1)$ for all pales $(i, j)$ (and appropriately choosing the other variables). The optimum value is $k^2 - k + k/(k-1)$.

Our strategy was to first search for violated 3-fence inequalities, and if no violated 3-fence inequality was found, to search for 4-fence inequalities, and so forth. However, it turned out that in our practical



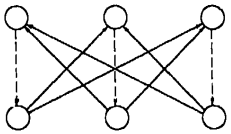

**Figure 2.** Solution for 3-fence inequality.

applications we never reached the latter phase. So the present implementation of our code only searches for 3-fences.

The heuristic for finding violated 3-fence inequalities works as follows. Suppose $y \in \mathbb{Q}^m$ is a point in $P_2^n$. First, we search (in a straightforward enumerative way) for three arcs $(i_1, j_1)$, $(i_2, j_2)$, $(i_3, j_3)$ having no endnode in common such that $\frac{1}{2} - \epsilon \leq y_{i_k j_k} \leq \frac{1}{2} + \epsilon$ for $k = 1, 2, 3$, and $\epsilon > 0$ sufficiently small (we have used $\epsilon = 0.1$), then we check whether the inequalities corresponding to the 3-fence having these three arcs as pales is violated. All violated 3-fence inequalities found this way are stored and finally added to the current LP just as described for 3-dicycles.

PROBLEM 5. *Identification of Möbius-ladder inequalities* (6).

The Möbius ladders are even more difficult to handle than $k$-fences, and we do not even know how to handle large subclasses of Möbius ladders heuristically in a reasonable way. Our code contains three heuristics for three special Möbius ladder inequalities. We describe only one of them here, the others are modeled in a similar way. To describe the philosophy of this heuristic, consider the Möbius ladder $M$ consisting of four 3-dicycles and one 4-dicycle shown in Figure 3.

If we optimize $x(M)$ over $P_2^6$ we obtain the fractional solution $y$ shown in Figure 4, where solid (broken) lines correspond to arcs $(i, j)$ with $y_{ij} = 1$ ($y_{ij} = \frac{1}{2}$), arcs $(i, j)$ with $y_{ij} = 0$ are not shown.

We see that the nodes 2, 3, 5, 6 form a 4-dicycle where all arcs (in both directions) have value $\frac{1}{2}$. Our heuristic for the kind of Möbius ladders shown in Figure 3 (and the ones obtained by reversing all directions of arcs) therefore first searches for a 4-dicycle where all arcs have value

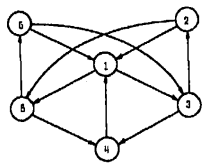


**Figure 3.** Möbius ladder of four 3-dicycles and one 4-dicycle.

$\frac{1}{2} \pm \epsilon$ (we have used $\epsilon = 0.1$). Then, we check whether there are two nodes whose addition to the 4-dicycle creates a structure like that shown in Figure 4, and we check whether the inequality associated with the corresponding Möbius ladder (cf. Figure 3) is violated. The handling of violated Möbius ladder inequalities is done as described for 3-dicycles above.

Our present code uses the following strategy for calling our methods for cutting plane generation. First, we search for violated 3-dicycle inequalities. If some have been found, we go to Box 6. Otherwise, we search for violated 3-fence inequalities. If some have been found, we go to Box 6. If no violated 3-dicycle and no violated 3-fence inequality could be identified, then we search for violated Möbius ladder inequalities, and finally go to Box 6. (Of course, we have implemented "switching parameters" with which we can choose the sequence of heuristics, e.g., we may also decide not to call the $k$-fence and Möbius-ladder heuristics.)

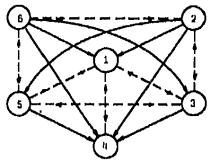


**Figure 4.** Fractional solution of Problem 6.

The first iterations of the cutting plane generation procedure usually found quite a number of facet defining inequalities for $P^1_{LO}$ that are violated by $x^*$. So we continue from Box 6 to Box 7.

*Box 7.* Here, we revise our current LP to start the next solution run in Box 3. To keep the size of the LP "small," we first throw away all inequalities (except for upper and lower bounds) that have a positive slack at the present optimum LP-solution $x^*$. This feature considerably improved our running times although it might result in one or two more iterations of the cutting plane generation process, and some inequalities might be added and removed a few times. However, we delete inequalities only at LEVEL = 0. The branch and bound phases with LEVEL $\geq$ 1 add inequalities but do not remove them.

Next we add all the cutting planes generated (as described above) in Box 5, and we go to Box 3 to continue the cutting plane phase. In case no cutting planes have been found in Box 5 we go from Box 6 to Box 8.

*Box 8.* If we arrive here, then we know that our point $x^*$ satisfies all 3-dicycle inequalities. So if $x^*$ is integral (and we are at LEVEL = 0 of the branch and bound tree), we can conclude that $x^*$ is an optimum solution of our linear ordering problem. If we are not at LEVEL = 0, we



still have made an improvement by finding a feasible solution of the linear ordering problem whose value is at least as good as the present best known one. So, in Box 9, we store the present best solution and go to Box 10, and leave from there, since LEVEL > 0, to Box 17.

This completes the description of the LP-cutting plane part of the algorithm. In case our last solution $x^*$ is fractional, we have to enter the branch and bound phase which we shall describe next.

As a matter of fact, all the 44 real-world applications whose solution will be presented in Section 3 could be solved in the initial LP-phase. In no case did the heuristic of Box 1 find an optimum solution. We have also run these problems using only 3-dicycle inequalities (i.e., the heuristics searching for 3-fences and Möbius ladders were not used). In this version, only 3 of 44 problems happened to have no integral solution after the first cutting plane phase. Thus, for 41 of these problems it was not necessary to enter the branch and bound stage. This indicates that generation of 3-dicycle inequalities might be sufficient in many practical cases.

Our experience shows that the optimum value, let us denote it by $U$ from now on, obtained from the LP-relaxation discussed above is almost equal to the optimum value of the linear ordering problem. In the three real-world applications that were not solved after the first stage, the LP-values $U$ after the initial phase were 0.0004%, 0.002%, and 0.005% larger than the optimum value of the integer program.

After completing the initial cutting plane phase, we start the branch and bound part of the algorithm by going from Box 8 (the last $x^*$ is not integral) to Box 11. To bound the optimum value tightly from below, we first try to improve our heuristic solution by a further heuristic that takes the structure of the LP-solution $x^*$ into account.

*Box 11.* For every node $i \in V$ we calculate

$$s(i) = \sum_{k<i} c_{ik}(1 - x^*_{ki}) + \sum_{i<j} c_{ij} x^*_{ij}$$

and sort the nodes such that $s(i_1) \geq s(i_2) \geq \cdots \geq s(i_n)$. Starting with the linear ordering $i_1, i_2, \cdots, i_n$, we then try to make improvements by local interchanges as in the heuristic described in Box 1. If the interchange heuristic concludes that no more local improvements from the present linear ordering are possible, we compare in Box 12 the solution found with the present best linear ordering. If the new one is better, or if we reach Box 12 for the first time, we go to Box 13, and afterward to Box 14; otherwise we go directly to Box 14. Let us denote the value of the best lower bound found so far by $L$. Of course, we can stop if $L = U$ (or if $L$ equals the round down of $U$ in case the data are integral) and go to Box 16. The flowchart does not indicate this possibility.

*Box 13.* We keep in memory the reduced costs after the initial cutting plane phase at LEVEL = 0. The $x^*$ found there gives a true upper bound

for the optimum value of the linear ordering problem. Having improved our lower bound $L$, we may be able to fix more variables permanently using a well-known reduced cost criterion. Let $d_{ij}$ denote the reduced cost of a nonbasic variable $x_{ij}^*$ of the optimum (LEVEL = 0) LP-solution $x^*$, then we can do the following

(a) If $x_{ij}^* = 0$ and $U - L \leq -d_{ij}$ then we fix $x_{ij} = 0$ permanently (since every LP-solution with $x_{ij} = 1$ has a value which is not larger than $L$, thus no linear ordering with $i$ before $j$ can be better than the present one).
(b) If $x_{ij}^* = 1$ and $U - L \leq d_{ij}$, then we fix $x_{ij} = 1$ permanently (since every LP-solution with $x_{ij} = 0$ has a value which is not larger than $L$).

Moreover, if we have $x_{ij} = 1$ and $x_{jk} = 1$ from the fixing above (or former permanent fixing), we can fix $x_{ik} = 1$ because of transitivity. Similarly, $x_{ij} = 0$ and $x_{jk} = 0$ implies $x_{ik} = 0$ by transitivity. All these permanent variable fixings are performed here. In fact, if $U - L$ is small, quite a large number of variables can be fixed after the initial cutting plane phase.

*Box 14.* Here we do the same fixing as described in Box 13 above. But instead of using the reduced costs of the initial LP-phase, we use the reduced costs obtained at the optimum of the last LP-phase. This optimum is not a true upper bound for the whole problem, but an upper bound for the linear ordering problem restricted to those linear orderings containing (not containing) the arcs the corresponding variables of which are temporarily fixed to one (zero). Thus, this fixing is made only temporarily. From Box 14, we go to

*Box 15.* We have tried various branching strategies and found the following one to be the most effective. We choose a variable which is as close to ½ as possible (in absolute error). If there is a choice, we use the one with the largest objective function coefficient. If this coefficient is negative, we temporarily set the variable to zero, otherwise we set it to one. (Of course, we also set the transitive implications of this choice.) We increase LEVEL by one and go to Box 3 to solve the next LP.

We still must describe two parts of the algorithm.

In one case, we have found no cuts in Box 6, concluded in Box 8 that $x^*$ is integral, and found in Box 10 that we are not at LEVEL = 0. Thus, we have found a new feasible solution of the linear ordering problem whose value is at least as good as the present best. Therefore, we keep this solution and perform in Box 17 the same permanent fixing as in Box 13. We then proceed to Box 18.

In the second case, we have concluded in Box 4, that $\tilde{c}^T x^* \leq \tilde{c}^T x^{F'}$ and in Box 16 that we are not at LEVEL = 0. Then we go to Box 18.



*Box 18.* Here we check the status of the last stacked variable $x_{ij}$. If it was set in Box 15, we fix it to the opposite value, reset all temporary fixings (transitively) implied by the previous setting and then stack $x_{ij}$ with its new status and all transitively implied temporary fixings.

If, on the other hand, $x_{ij}$ was fixed in this box, we know that the opposite value has been taken care of already and therefore we perform a usual backtracking operation the details of which should be obvious. It either ends with the fixing of some variable set in Box 15 to the opposite value as described above (after appropriate freeing of temporarily fixed variables while "going down the levels of the branch and bound tree") or by reaching the 0-level. Box 19 checks for this possibility. LEVEL = 0 implies optimality of $x^F$ and so we stop the algorithm. Otherwise we go to Box 3 in order to solve the new LP. This completes the outline of our algorithm.

## 3. COMPUTATIONAL RESULTS

In this section we report on the computational experiences we have with the algorithm described in Section 2. As mentioned above, we have used IBM's LP-package MPSX/370 to solve the LP problems that arise during the execution of the algorithm (Box 3). All experiments were run on the IBM 370/168 of the Rechenzentrum der Universität Bonn.

We were mainly interested in solving real-world triangulation problems for input-output matrices. We had 30 (44, 44)-matrices, 11 (56, 56)-matrices, and 3 (60, 60)-matrices. For all of these problems, we were able to find an optimum solution. Standard sizes of input-output matrices are in the range between 10 and 70 sectors. There are a few exceptionally large matrices, but we were not able to get the data of such matrices. (For example, we know that the Deutsches Institut für Wirtschaftsforschung has a (400, 400)-matrix, but these data are not available to the public.) The size range of matrices we have considered covers almost all input-output matrices that have been compiled so far all over the world.

In input-output analysis one is mainly interested in the so-called *degree of linearity* $\lambda(C)$ which is defined as follows. If $C = (c_{ij})$ is an $(n, n)$-input-output matrix and $\sigma$ an optimum linear ordering then

$$\lambda(C) = 100 \cdot \sum_{\sigma^{-1}(i) < \sigma^{-1}(k)} c_{ik} / \sum_{i \neq k} c_{ik}$$

is the degree of linearity of $C$. That is, this number gives the amount of value which (in an optimum triangulation) is above the main diagonal as a percentage of the total sum of the entries of the matrix (without the entries on the main diagonal). For all the matrices we have triangulated we also list this degree of linearity. (For an economic analysis of our results see Grötschel et al. [1982b].)

The European Community compiles (44, 44)-input-output matrices for all of its members. 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 on tape from "Office Statistique des Communautes Europeennes, EUROSTAT, Boite postale 1907, Kirchberg, Luxembourg" free of charge. For our experiments, we have chosen the (most important) matrices which are the (44, 44)-matrices of intermediate consumption at current prices for the years 1959, 1965, 1970, and 1975. EUROSTAT has 30 of these matrices. Table I contains a list of the matrices together with their degree of linearity, and the respective running times. Every matrix is given by its country name, the year, and a code number that is used by EUROSTAT to identify the matrices uniquely. The running times in Table I include the total time used by our code on the machine including input and output of data, paging, and so forth. The version of the code used for these matrices did not use the heuristics for 3-fences and Möbius ladders. That is, after the initial LP-phase, the optimum over $P_2^{44}$ was found. Moreover, in every cutting plane generation step (calls of Box 5) all cutting planes that were found were added to the LP. In 28 of the 30 matrices, we obtained an integral solution after the initial phase, and only for the matrices BELGIUM 59 and GERMANY 70 did we have to enter the branch and bound stage. Both problems could be solved with only one branching operation. So the branch and bound tree consisted of three nodes (including the root).

The second class of matrices are those compiled by the Deutsches Institut für Wirtschaftforschung (DIW), Berlin, for West Germany for the years 1954–1972. The matrices can be found in "Beiträge zur Strukturforschung, Heft 27/1973, Heft 38/1975, Vierteljahreshefte zur Wirtschaftsforschung, Heft 1/1974," but can also be obtained from the DIW on tape. The degrees of linearity and the running times are listed in Table II. The code name DIW56XYZ is to be interpreted as follows. The first part DIW56 states that these are (56, 56)-tables compiled by DIW, X indicates whether the matrix is in current prices (the code is N for nominal) or in prices of 1962 (the code is R for real), YZ denotes the year. The version of the code we used was the same as for the EUROSTAT-matrices. All the nine triangulation problems could be solved without entering the branch and bound stage.

Finally, we tried the same version of the code on three (60, 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. We used the three tables "Input-Output Tabelle zu Ab-Werk-Preisen-Inländische Produktion 70, 74, 75" which are available from the Statistisches Bundesamt. The results



can be found in Table III. 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.

TABLE I
EUROSTAT MATRICES[a]

| Input-Output Table | | Degree of Linearity | IBM 370/168 CPU Time (min : sec) |
|---|---|---|---|
| T59I11XX.B | (ITALY 59) | 88.443 | 1 : 02.10 |
| T59D11XX.B | (GERMANY 59) | 88.148 | 0 : 45.84 |
| T59F11XX.B | (FRANCE 59) | 86.516 | 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)[b] | 87.862 | 0 : 56.85 |
| T70K11XX.B | (DENMARK 70) | 86.893 | 0 : 54.56 |
| T70I11XX.B | (ITALY 70) | 86.613 | 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)[b] | 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)[b] | 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 |

[a] Minimum CPU time, 0 : 28.95; maximum CPU time, 1 : 36.69; and average CPU time, 1 : 04.09.

[b] There are two matrices GERMANY 70 available from EUROSTAT, denoted by 70a and 70b above. GERMANY 70b is a revised version of GERMANY 70a, but GERMANY 70a was used for the compilation of EUR-6 70 and EUR-9 70. Also, IRELAND 69 was used instead of IRELAND 70.

To give an impression of the sizes of the LPs that have to be solved and the number of times cutting planes have to be added (in the initial phase), we have compiled appropriate data in Tables IV and V. In all

TABLE II
DIW MATRICES[a]

| Input-Output Table | Degree of Linearity | IBM 370/168 CPU Time (min : sec) |
|---|---|---|
| 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 |

[a]Minimum CPU time, 4 : 24.14; maximum CPU time, 5 : 10.75; and average CPU time, 4 : 52.16.

cases of the (44, 44)- and (56, 56)-matrices, the algorithm required at most 4 cutting plane generation steps (calls of Box 5). We have listed the average (minimum, maximum) number of cuts that were added in such a step and the number of inequalities eliminated, as well as the average LP-size after the $k$th cutting plane generation step, $k = 1, \cdots, 4$. (Here "min" means minimum positive number and the average is taken over the nonzero numbers.)

The LP-sizes are considerable, so we have tried to make them smaller using the parameter MAXCUT discussed in the description of Box 5. Table VI shows one example of the influence of this parameter on the number of cutting plane generation steps, the final LP-size, and the running time. The data (here for the I/O table T75D11XX.B) clearly show that it is profitable to bound the number of cuts added, but we do not have a good "formula" yet to determine an "optimal" MAXCUT.

Recent experiments have shown that another modification of the cutting plane procedure, namely the insertion of arc-disjoint cuts in each

TABLE III
STATISTISCHES BUNDESAMT MATRICES

| Input Output Table | Degree of Linearity | IBM 370/168 CPU Time (min : sec) |
|---|---|---|
| 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 |



TABLE IV
SIZE OF LPS AND CUTTING PLANES

| Cutting Plane Phase | No. of Eliminated Cuts | | | No. of Inserted Cuts | | | LP Size | | |
|---|---|---|---|---|---|---|---|---|---|
| | Avg | Min | Max | Avg | Min | Max | Avg | Min | Max |
| 1 | — | — | — | 1346 | 786 | 1787 | 1346 | 786 | 1787 |
| 2 | 312 | 112 | 519 | 267 | 58 | 475 | 1292 | 742 | 1695 |
| 3 | 88 | 10 | 188 | 30 | 3 | 108 | 1218 | 742 | 1623 |
| 4 | 31 | 3 | 102 | 9 | 2 | 20 | 1160 | 837 | 1486 |

cutting plane phase, can considerably decrease the size of the final LP and could therefore be valuable when solving large problems. The results we obtained so far show that this version causes a slight increase of the running time (due to the larger number of cutting plane phases), but gives much smaller final LPs; e.g., the I/O table T75D11XX.B could be triangulated using arc-disjoint cuts within 17 cutting plane phases, and the final LP consisted of only 480 rows.

TABLE V
SIZE OF LPS AND CUTTING PLANES

| Cutting Plane Phase | No. of Eliminated Cuts | | | No. of Inserted Cuts | | | LP Size | | |
|---|---|---|---|---|---|---|---|---|---|
| | Avg | Min | Max | Avg | Min | Max | Avg | Min | Max |
| 1 | — | — | — | 3395 | 3293 | 3528 | 3395 | 3293 | 3528 |
| 2 | 1212 | 1017 | 1311 | 957 | 753 | 1383 | 3140 | 2895 | 3495 |
| 3 | 376 | 262 | 511 | 192 | 34 | 454 | 2957 | 2814 | 3226 |
| 4 | 50 | 29 | 66 | 6 | 5 | 8 | 2907 | 2780 | 2998 |

Up to now we have reported on the version of our code that uses only 3-dicycle inequalities. Since we never call the other heuristics if violated 3-dicycle inequalities are found (see description of Box 3), the only cases where the code, using the other heuristics differs from this version, are the input-output tables for which the branch and bound phase had to be called.

TABLE VI
MAXCUT EFFECT ON CUTTING PLANES, LP SIZE, AND RUNNING TIME

| MAXCUT | No. of Cutting Plane Phases | Size of Final LP | IBM 370/168 CPU Time (min : sec) |
|---|---|---|---|
| 50 | 26 | 685 | 1 : 23.67 |
| 100 | 13 | 791 | 1 : 11.93 |
| 300 | 6 | 939 | 0 : 52.51 |
| 500 | 6 | 1244 | 0 : 53.87 |
| ∞ | 3 | 1529 | 1 : 13.87 |

TABLE VII
GERMANY 1970 (44, 44)-MATRIX

| Cuts Inserted | No. of Cuts Eliminated | LP Size | LP Value | Difference |
|---|---|---|---|---|
| — | — | — | 455,077 | — |
| 500 3-dicycles | — | 500 | 450,164 | 4913 |
| 500 3-dicycles | 68 | 932 | 441,014 | 9150 |
| 370 3-dicycles | 204 | 1098 | 438,470 | 2544 |
| 43 3-dicycles | 154 | 987 | 438,291 | 179 |
| 44 3-dicycles | 74 | 957 | 438,243 | 48 |
| 1 Möbius | 28 | 930 | 438,242 | 1 |
| 16 3-dicycles | 40 | 906 | 438,235 | 7 |

Tables VII, VIII, and IX give a full description of the performance of our code in those three cases. Table VII reports on the 1970 (44, 44)-matrix of Germany where the number of cuts added is restricted by MAXCUT = 500. The first LP-value 455,077 is determined by inspection from the unit hypercube. Then, five steps generated 3-dicycle inequalities. In the sixth step, no such inequality was found, but the Möbius ladder heuristic found one violated Möbius ladder (which is shown in Figure 3) which was added to the LP. A seventh step added 16 more 3-dicycle inequalities and the LP terminated with the incidence vector of a linear ordering.

Table VIII contains the analogous data for the (44, 44)-matrix of Belgium of the year 1959. Here also, only one Möbius ladder inequality had to be added.

In Table IX we report on the solution of the (60, 60)-matrix of West Germany, 1970. MAXCUT was set to 2000. Again, the algorithm found only one violated Möbius ladder which finally led the LP to terminate with an integral solution.

TABLE VIII
BELGIUM 1959 (44, 44)-MATRIX

| Cuts Inserted | No. of Cuts Elimi-nated | LP Size | LP Value | Difference |
|---|---|---|---|---|
| — | — | — | 256,765.0 | — |
| 500 3-dicycles | — | 500 | 252,250.0 | 4515.0 |
| 500 3-dicycles | 75 | 925 | 247,110.0 | 5140.0 |
| 223 3-dicycles | 219 | 929 | 245,960.0 | 1150.0 |
| 58 3-dicycles | 90 | 897 | 245,770.0 | 190.0 |
| 4 3-dicycles | 32 | 869 | 245,762.5 | 7.5 |
| 1 Möbius | 14 | 856 | 245,755.0 | 7.5 |
| 3 3-dicycles | 4 | 855 | 245,750.0 | 5.0 |
| 2 3-dicycles | 3 | 854 | 245,750.0 | 0 |



TABLE IX
WEST GERMANY 1970 (60, 60)-MATRIX

| Cuts Inserted | No. of Cuts Elim-inated | LP Size | LP Value | Difference |
|---|---|---|---|---|
| — | — | — | 447,824.0 | — |
| 2000 3-dicycles | — | 2000 | 433,974.0 | 13850.0 |
| 2000 3-dicycles | 742 | 3258 | 426,221.0 | 7753.0 |
| 2000 3-dicycles | 743 | 4515 | 422,534.0 | 3687.0 |
| 591 3-dicycles | 1134 | 3972 | 422,182.0 | 352.0 |
| 104 3-dicycles | 375 | 3701 | 422,094.0 | 88.0 |
| 4 3-dicycles | 87 | 3618 | 422,093.0 | 1.0 |
| 19 3-dicycles | 27 | 3610 | 422,089.5 | 3.5 |
| 1 Möbius | 6 | 3605 | 422,088.0 | 1.5 |
| 3 3-dicycles | 12 | 3596 | 422,088.0 | 0 |

So, in all our real-world applications we were able to solve the linear ordering problem purely by LP-cutting plane-techniques without resorting to branch and bound.

As another application (for pure fun), we have triangulated the final table of the German soccer-championship of 1981/82. Each of the 18 teams plays each other team twice, an entry $C = (c_{ij})$ of the objective function then is the sum of the goals team $i$ scored against team $j$ in these two games. Table X compares the official result (using the standard lexicographic technique) with the ranking by optimum linear ordering. The champion remains the same, but otherwise there are considerable

TABLE X
GERMAN SOCCER CHAMPIONSHIP 1981/82

| Official Result | Our Result |
|---|---|
| 1. Hamburger SV | 1. Hamburger SV |
| 2. 1.FC Köln | 2. 1.FC K'lautern |
| 3. Bayern München | 3. Bor. Dortmund |
| 4. 1.FC K'lautern | 4. Eintr. Frankfurt |
| 5. Werder Bremen | 5. 1.FC Köln |
| 6. Bor. Dortmund | 6. Werder Bremen |
| 7. Bor. M'gladbach | 7. Bor. M'gladbach |
| 8. Eintr. Frankfurt | 8. Bayern München |
| 9. VfB Stuttgart | 9. VfB Stuttgart |
| 10. VfL Bochum | 10. VfL Bochum |
| 11. Eintr. Braunschweig | 11. Eintr. Braunschweig |
| 12. Arm. Bielefeld | 12. Fort. Düsseldorf |
| 13. 1.FC Nürnberg | 13. Bayer Leverkusen |
| 14. Karlsruher SC | 14. Arm. Bielefeld |
| 15. Fort. Düsseldorf | 15. Karlsruher SC |
| 16. Bayer Leverkusen | 16. Darmstadt 98 |
| 17. Darmstadt 98 | 17. MSV Duisburg |
| 18. MSV Duisburg | 18. 1.FC Nürnberg |

differences. In particular 1. FC Nürnberg being 13th officially would have had to leave the Bundesliga. We want to point out that the optimum linear ordering is not unique, we could find two further optimum rankings (changes occur in positions 2 to 5).

The success of the code on the real-world problems reported above does not carry over to random problems. Particularly bad are uniform distributions. For instance, we encountered one problem of size (50, 50) which the code was unable to solve (within the time available to us). Of course, uniform distribution means that almost every solution is nearly optimal, and these types of problems are usually extremely hard to solve.

On the other hand, we successfully solved an (80, 80)-problem which was randomly generated so that its "structure" roughly corresponds to a real-world input-output table. This experience indicates that 60 sectors are not the limit for our code when applied to real-world input-output tables.

## 4. CONCLUSIONS AND COMPARISONS

Many codes for the solution of linear ordering problems have been developed in the recent years. We give a brief overview on the sizes they can handle. A more thorough discussion and description of the various techniques used can be found in Kaas [1981].

The code of Korte and Oberhofer [1968, 1969] was able to handle random (13, 13)-problems. It solved real-world input-output tables up to 18 sectors. In 1981 Wessels [1981] used this code to triangulate (25, 25)-matrices optimally. According to Kaas [1981], an improvement of the Korte and Oberhofer method by Lenstra [1973] could handle (17, 17)-matrices. The code of Kaas solves (34, 34)-input-output matrices and random (25, 25)-matrices.

Linear programming techniques have been suggested by Krelle [1964], Korte and Oberhofer [1969], and de Cani [1969], none of whom implemented the method in a code. Later Marcotorchino and Michaud [1979] and Boenchendorf [1982], produced such implementations. Both use all 3-dicycle inequalities explicitly.

Boenchendorf [1982] solves these linear programs in a primal fashion and was able to run problems up to $n = 30$. He encountered only integral solutions of the linear program and does not discuss how to handle nonintegral solutions. (Boenchendorf [1982] is a further source for the description of other approaches to the linear ordering problem.)

Marcotorchino and Michaud [1979] have written an entire book on the linear ordering problem. Their main solution technique is linear programming and they use MPSX/370 as we do. They take the dual linear program to $\max\{c^T x \mid x \in P_n^2\}$ (to overcome the MPSX-restriction of 16,000 constraints). With one exception, their problems have size at most 36. They report on the solution of one (72, 72)-problem, but they do not say what kind of problem it is (we have unsuccessfully tried to obtain the data to make comparisons). Moreover, Marcotorchino and Michaud never describe what they do if the linear program has a nonintegral solution.



We have not discussed the running times of the codes mentioned above. Different computers were used and it is quite hard to compare the performance of the various algorithms. Anyway, it seems that the problem sizes our code can handle are far beyond the capabilities of the other existing codes. Moreover, our code still has potential improvements. We have not optimized all the data structures we use and waste a substantial amount of time by setting up the REVISE data in MPSX-format. We consider our code as an experimental one leaving much room for modifications to gain practical experience. Basically, it consists of exchangeable modules roughly corresponding to the boxes in Figure 1. These modules are coded in ECL, an extension of PL/I, and work on global as well as local data structures. This organization allows us to implement new heuristics and new cutting plane generation methods quickly and to quickly compare various versions and strategies. A special purpose code without this flexibility could easily be designed to improve the algorithm's time and space requirements. In any case, we believe that the results reported show that our approach is a promising solution strategy when confronted with linear ordering problems arising in practice.

## ACKNOWLEDGMENT

Thanks are due to M. PECCI-BORIANI of EUROSTAT for making available important information about the EUROSTAT-matrices for the year 1970.

This work was partially supported by Sonderforschungsbereich 21 (DFG), Institut für Operations Research, Universität Bonn, West Germany.

## REFERENCES

BOENCHENDORF, K. 1982. Reihenfolgenprobleme/Mean-Flow-Time Sequencing. Mathematical Systems in Economics 74, Verlagsgruppe Athenäum/Hain/Scriptor/Hanstein.

DE CANI, J. S. 1969. Maximum Likelihood Paired Comparison Ranking by Linear Programming. *Biometrika* **56**, 537–545.

GLOVER, F., T. KLASTORIN AND D. KLINGMAN. 1974. Optimal Weighted Ancestry Relationships. *Mgmt. Sci.* **20**, B1190–B1193.

GRÖTSCHEL, M., L. LOVÁSZ AND A. SCHRIJVER. 1981. The Ellipsoid Method and Its Consequences in Combinatorial Optimization. *Combinatorica* **1**, 169–197.

GRÖTSCHEL, M., M. JÜNGER AND G. REINELT. 1982a. Facets of the Linear

Ordering Polytope. WP 82217-OR, Institut für Operations Research, Universität Bonn (to appear in *Mathematical Programming*).

GRÖTSCHEL, M., M. JÜNGER AND G. REINELT. 1982b. Optimal Triangulation of Large Real-Word Input-Output-Matrices. WP 82228-OR, Institut für Operations Research, Universität Bonn (to appear in *Statistische Hefte*).

KAAS, R. 1981. A Branch and Bound Algorithm for the Acyclic Subgraph Problem. *Eur. J. Opnl. Res.* **8**, 355–362.

KORTE, B., AND W. OBERHOFER. 1968. Zwei Algorithmen zur Lösung eines komplexen Reihenfolgeproblems. *Unternehmensforschung* **12**, 217–231.

KORTE, B., AND W. OBERHOFER. 1969. Zur Triangulation von Input-Output-Matrizen. *Jahrbuch. Nationaloekon. Statist.* **182**, 398–433.

KRELLE, W. 1964. In F. Neumark (Hrg.): Strukturwandlungen einer wachsenden Wirtschaft. Schriften des Vereins für Socialpolitik, NF Band 30/II, pp. 1059–1063.

LENSTRA, H. W. JR. 1973. The Acyclic Subgraph Problem. Report BW26, Mathematisch Centrum, Amsterdam.

LENSTRA, J. K. 1977. Sequencing by Enumerative Methods. Mathematical Centre Tracts 69, Mathematisch Centrum, Amsterdam.

MARCOTORCHINO, J. F., AND P. MICHAUD. 1979. *Optimisation en Analyse Ordinale des Données*. Masson, Paris.

SLATER, P. 1961. Inconsistencies in a Schedule of Paired Comparisons. *Biometrika* **48**, 303–312.

WESSELS, H. 1981. Triangulation and Blocktriangulation von Input-Output-Tabellen. Deutsches Institut für Wirtschaftsforschung: Beiträge zur Strukturforschung, Heft 63, Berlin.

YOUNG, H. P. 1978. On Permutations and Permutation Polytopes. *Math. Program. Study* **8**, 128–140.

# Confidence Intervals for Steady-State Simulations: I. A Survey of Fixed Sample Size Procedures

AVERILL M. LAW
*University of Arizona, Tucson, Arizona*

W. DAVID KELTON
*The University of Michigan, Ann Arbor, Michigan*

(Received May 1979; accepted August 1983)

We consider the problem of constructing a confidence interval for the steady-state mean of a stochastic process by means of simulation, and study the five main methods which have been proposed (replication, batch means, autoregressive representation, spectrum analysis, and regeneration cycles) for the case when the length of the simulation is fixed in advance. Comparing the performances of these methods on stochastic models with known steady-state means, we find that the simulator should exercise caution in interpreting the results from a simulation of fixed length, and that the length of a simulation adequate for acceptable performance is highly model-dependent. We also investigate possible sources of error for the methods, and conclude that variance estimator bias is more important than point estimator bias in confidence interval coverage degradation.

ONE OF THE most important but difficult problems encountered in a real-world simulation study is that of constructing a confidence interval (c.i.) for the steady-state mean $\mu$ of a stochastic process. The information contained in such a c.i. provides the decision-maker with a measure of how precisely $\mu$ is known. Constructing the c.i., however, is difficult because the output data from a simulation are in general nonstationary and autocorrelated, so that *direct* application of the techniques of classical statistics is precluded.

This problem has received considerable attention in the literature, and many methodologies have been proposed. Each of them, however, involves some basic set of assumptions on the process being simulated which cannot be made, in general, by a simulator facing a real-world system. The effects of these violations on the c.i. are impossible to determine in practice, but nevertheless may seriously impair the quality of decisions based on the simulation results. In particular, a c.i. which

*Subject classification:* 767 statistical analysis of simulation.



**Operations Research**
Vol. 32, No. 6, November–December 1984

0030-364X/84/3206-1221 $01.25