Solving Real-World Linear Ordering Problems Using a Primal-Dual Interior Point Cutting Plane Method

John E. Mitchell[1]
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180
mitchj@rpi.edu
and
Brian Borchers
Department of Mathematics
New Mexico Tech
Socorro, NM 87801
borchers@nmt.edu

R.P.I. Technical Report No. 207

March 8, 1993
Revised April 28, 1995

Abstract

Cutting plane methods require the solution of a sequence of linear programs, where the solution to one provides a warm start to the next. A cutting plane algorithm for solving the linear ordering problem is described. This algorithm uses the primal-dual interior point method to solve the linear programming relaxations. A point which is a good warm start for a simplex-based cutting plane algorithm is generally not a good starting point for an interior point method. Techniques used to improve the warm start include attempting to identify cutting planes early and storing an old feasible point, which is used to help recenter when cutting planes are added. Computational results are described for some real-world problems; the algorithm appears to be competitive with a simplex-based cutting plane algorithm.

[1]Research partially supported by ONR Grant number N00014–90–J–1714.

1 Introduction

Currently, almost all approaches to solving integer programming problems with linear programming methodology use the simplex method to solve the linear programs. There have been several notable successes with such algorithms; the most successful methods for solving traveling salesman problems are LP-based (see, for example, [1, 2]). Within the last eight years, interior point methods have become accepted as powerful tools for solving linear programming problems. It appears that interior point methods may well solve large linear programs substantially faster than the simplex method (see, for example, [3]). A natural question, therefore, is whether interior point methods can be successfully used to solve integer programming problems.

The linear ordering problem is an NP-hard combinatorial optimization problem which has many real-world applications, including the triangulation of input-output matrices in economics. The polyhedral structure of the problem has been investigated by Grötschel, Jünger, and Reinelt [4, 5, 6]. They described a simplex-based cutting plane algorithm for solving the linear ordering problem and were able to solve sets of European Community and (West) German input-output tables. They showed that almost all of these problems could be solved using just simple bounds and *3-dicycle* inequalities. In this paper, we develop an interior point cutting plane algorithm which we use to solve this real-world test set of linear ordering problems, in times comparable to those obtained by Grötschel *et al.* This is a further development of the work described in [7]. We describe the linear ordering problem in Section 2. Our computational results are contained in Section 5.

Let S be the convex hull of the set of feasible integer points. In a traditional cutting plane method, the linear programming relaxation of the problem is solved to optimality using the simplex algorithm: if the optimal solution $\bar{x}$ is integer, then we are done; otherwise $\bar{x}$ can be *separated* from S, an extra constraint (or *cutting plane*) added to the relaxation and the process repeated. The extra constraint takes the form $a_k^T x \leq b_k$; this constraint is satisfied by all points in S but violated by $\bar{x}$. The recent success of cutting plane methods has come about with the use of *facet-defining inequalities*, which give proper faces of maximal dimension of the convex hull of the set of feasible integer points (see, for example, [8, 9, 1, 4, 10, 2]). We use an interior point method in place of the simplex algorithm. We usually do not solve the relaxation to optimality, but attempt to find cutting planes before reaching optimality. This is usually possible with an interior point method; in fact, it is more attractive with an interior point method than with the simplex method because the interior point method gets close to optimality quickly, whereas the simplex method may pivot a vital element into the basis on the last iteration.

The difficulty with using an interior point method in a cutting plane algorithm is that the solution to one relaxation is usually not a good starting point for an interior point method, because it is close to the boundary of the feasible region. Thus, it is usually necessary to attempt to stop working on the current relaxation before it is solved completely to optimality: the earlier we are able to find good cutting planes, the better the initial solution to the next relaxation. Early termination obviously

reduces the number of iterations spent solving the current relaxation; in addition, it reduces the number of iterations spent solving the next relaxation, because the initial point to the next relaxation is more centered. There are two potential disadvantages from trying to find cutting planes early: if the search for cutting planes is unsuccessful, we have wasted time; secondly, it may well be that superfluous constraints are added, with the result that the algorithm requires extra iterations and extra stages of adding cutting planes. We describe our cutting plane algorithm in detail in Section 4. The primal-dual interior point method which we use to solve the individual relaxations is described in Section 3.

Usually, cutting plane methods are not sufficient by themselves to solve an integer programming problem and are often embedded in branch-and-bound algorithms. They can either be used as a preconditioner to branch-and-bound, or they can be used at each node of the branch-and-bound tree (such a method is called a *branch-and-cut algorithm*). In [11], we have described a branch-and-bound algorithm which uses an interior point algorithm to solve the subproblems at each node of the tree. The computational performance of this algorithm has been encouraging, with run-times comparable to those obtained by OSL [12] on some problems. One of the major difficulties in comparing our "home-built" branch-and-bound code with OSL is that OSL has very sophisticated procedures for determining the next node, so it examines fewer subproblems, it generates better upper bounds, and it can prune nodes by bound more quickly. To embed our cutting plane method in a branch-and-bound algorithm, we could either use the method described in [11], or we could switch to a simplex method for the branch-and-bound aspects, by using some method to round the interior solution to a basic feasible solution (such rounding procedures are discussed in, for example, [13, 14]). For an example of an algorithm which uses a mixture of interior point methods and the simplex method, see Bixby *et al.* [15]. They use a column generation method to solve a large crew-scheduling problem, using interior point methods to solve the first few relaxations and then switching to the simplex method for the final few.

Previous work on using interior point methods to solve integer programming problems and on the related column generation problem includes the following. The first papers in this area were those by Goffin and Vial [16] (see also [17]), who described a column generation method for solving nonsmooth optimization problems, and by Mitchell and Todd [18], who gave a cutting plane algorithm for solving matching problems. These papers described projective algorithms and they demonstrated the viability of the approach by showing that the number of iterations necessary at each stage of the algorithm was indeed less than that required to solve each relaxation completely. Kaliski and Ye [19] describe a potential reduction algorithm for column generation. Their algorithm makes heavy use of column deletion, so the current constraint matrix is usually almost square. They obtained good run times for solving transportation problems and also, in a subsequent paper, for solving crew scheduling problems. Their algorithm is based on an earlier one of Dantzig and Ye [20] which is polynomial in the total number of columns. Resende and Veiga [21] described an application of the dual affine algorithm to network flow problems. They applied the

algorithm to the full set of columns and achieved runtimes comparable with those for the simplex based algorithm *NETFLO* [22]. They used a preconditioned conjugate gradient method to calculate the projections, with a preconditioner based upon the graphical structure of the problem and the current point. The performance of the interior point code, relative to *NETFLO*, improves as problem size increases. Den Hertog *et al.* [23] described a simple primal-dual barrier function column generation algorithm. They showed that if a column is added whenever the corresponding dual slack becomes almost violated, then the algorithm is polynomial in the total number of variables. Atkinson and Vaidya [24] described a primal-dual algorithm for convex programming problems which uses column generation. Their algorithm drops constraints which are no longer important; when it adds constraints, it adds relaxed versions of the constraints: a constraint may well be dropped and later added in a tighter form. One consequence of their work is that the feasibility problem can be solved in polynomial time if the separation problem can. The paper by Lustig *et al.* [25] looks at the related problem of warm starting an interior point method. They experimented on the problem *STAIR* from the Netlib test suite, solving the initial problem, modifying the problem in a manner that makes sense in terms of the original model, and resolving. When warm starting, they increased all variables smaller than a given tolerance. They showed that they were able to exploit the warm start and solve the modified problem in a small number of iterations. Karmarkar *et al.* [26] developed an interior point method to solve some hard integer programming problems in a novel way. They use a potential reduction algorithm which they apply to a non-convex quadratic programming problem that is equivalent to the integer programming problem.

Notation:

The vectors x, s, y, w, z, b, c, and u and the matrix A refer to the current relaxation

$$\begin{array}{lrcll} \min & c^T x & & & \\ \text{s.t.} & Ax & = & b & (P) \\ & x+s & = & u & \\ & x,s & \geq & 0 & \end{array} \qquad \begin{array}{lrcrcrcll} \max & b^T y & & & - & u^T w & & & \\ \text{s.t.} & A^T y & + & z & - & w & = & c & (D) \\ & & & & & z,w & \geq & 0; & \end{array}$$

A is an $m \times n$ matrix, x, s, w, z, c, and u are n-vectors and y and b are m-vectors. We use $\hat{m}$ and $\hat{n}$ to denote upper bounds on m and n. We use e to denote a vector of ones of an appropriate dimension. If a small letter (for example, x) denotes a vector the corresponding capital letter (for example, X) denotes a diagonal matrix whose diagonal entries are the components of the vector.

2 The Linear Ordering Problem

The linear ordering problem is a combinatorial optimization problem with a wide variety of applications, such as triangulation of input-output matrices, archeological seriation, minimizing total weighted completion time in one-machine scheduling,

and aggregation of individual preferences. The linear ordering problem is NP-hard (Karp [27]), and a complete description of its facets is not known. The facet structure of the polytope associated with the linear ordering problem has been investigated by Grötschel, Jünger and Reinelt [4, 5, 6].

Before defining the linear ordering problem, we need to define some terms from graph theory. A *graph* $G = [V, E]$ consists of a finite, nonempty set of *vertices* V together with a set of *edges* E where each element $e \in E$ is defined as an unordered pair of vertices i and j in V — we write $e = ij$; these two vertices are then *adjacent* and are called the endvertices of e. We assume that G is a *simple* graph, that is, it contains no loops of the form $e = ii$ and no parallel edges (two edges connecting the same pair of vertices). A directed graph or *digraph* $D = (V, A)$ consists of a nonempty set of vertices V together with a finite set of arcs A that are ordered pairs of elements of V. If $G = [V, E]$ is a graph, a digraph $D = (V, A)$ is called an *orientation* of G if, whenever $ij \in E$, A contains arc (i, j) or arc (j, i) but not both, and all arcs of A arise in this way.

A set of arcs $P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k)\}$ in $D = (V, A)$ with $v_i \neq v_j$ 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 $(v_k, v_1) \in A$, $C = P \cup \{(v_k, v_1)\}$ is a *dicycle of length* k or *k-dicycle*. A digraph $D = (V, A)$ that contains no dicycle is called *acyclic*.

A digraph D is called *complete* if A contains both of the arcs (i, j) and (j, i) for every pair i, j of vertices in V. Up to isomorphism, there is only one complete digraph on p vertices — let $D_p = (V_p, A_p)$ denote this digraph.

A *tournament* is a digraph $D = (V, A)$ such that for every two vertices i and j, A contains exactly one of the two arcs (i, j) and (j, i) — thus a tournament is an orientation of the complete (undirected) graph K_p. Every acyclic tournament corresponds to a permutation or *ordering* of the vertices of the graph, and vice versa. Solution of the linear ordering problem requires finding the "best" acyclic tournament.

We now define the linear ordering problem. A *linear ordering* (or permutation) of a finite set V with $|V| = p$ is a bijective mapping $\sigma : \{1, 2, \ldots, p\} \to V$. If $i, j \in V$ and $\sigma^{-1}(i) < \sigma^{-1}(j)$, we say that i is *before* j. For every pair of vertices $i, j \in V$ we associate values g_{ij} and g_{ji} which correspond to the costs from having i before j and j before i respectively. Then the total value of a linear ordering is given by

$$\sum_{\sigma^{-1}(i) < \sigma^{-1}(j)} g_{ij}.$$

Given a linear ordering of the vertices V of a digraph, the arc set $\{(i, j) : \sigma^{-1}(i) < \sigma^{-1}(j)\}$ defines an acyclic tournament on the digraph, and similarly any acyclic tournament on a digraph induces a linear ordering on the vertices V. This graph theoretical interpretation can be used to define an instance of the linear ordering problem as follows:

> Given a complete digraph $D_p = (V_p, A_p)$ with arc weights g_{ij} for all edges $(i, j) \in A_p$, find an acyclic tournament (V_p, T) in D_p such that
>
> $$g(T) := \sum_{(i,j) \in T} g_{ij}$$

is minimized.

We now formulate the linear ordering problem in polyhedral terms. First define $\mathcal{T}_p$ to be the set of acyclic tournaments on the complete digraph $D_p = (V_p, A_p)$ of size p. Then the linear ordering problem can be stated as $\min \{g(T) : T \in \mathcal{T}_p\}$.

Let q be the number of edges in A_p, so $q := p(p-1)$. We define a variable $x \in \Re^q$ with each component of x corresponding to a unique arc $(i,j) \in A_p$. We write x_{ij} for the component of x corresponding to the arc (i,j). Consider an arc set $A \subseteq A_p$. The incidence vector $x(A) \in \Re^q$ is defined in the following way:

$$x_{ij}(A) = \begin{cases} 1 & \text{if } (i,j) \in A \\ 0 & \text{if } (i,j) \notin A \end{cases}$$

We define the *linear ordering polytope* Q^p_{LO} to be the convex hull of all incidence vectors of arc sets of acyclic tournaments in D_p. Thus

$$Q^p_{LO} = \text{conv}\{x(T) \in \Re^m : T \in \mathcal{T}_p\}. \tag{1}$$

By definition, every acyclic tournament corresponds to a vertex of Q^p_{LO}, and vice versa. Therefore, solving the linear programming problem $\min \{g^T x, x \in Q^p_{LO}\}$ solves the linear ordering problem. However, to be able to use this approach we need to have a description of Q^p_{LO} in terms of its facets — whereas it is defined in terms of its $p!$ extreme points.

Grötschel, Jünger and Reinelt have found several classes of facets which appear to give a fairly good approximation to the linear ordering polytope. The following theorem is due to those authors.

Theorem 1 *Let $D_p = (V_p, A_p)$ be the complete digraph of order p. Let $p \geq 3$. Then the solution set of the system of equations*

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

is equal to the affine hull of Q^p_{LO}. The trivial inequalities

$$x_{ij} \geq 0 \text{ and } x_{ij} \leq 1 \tag{3}$$

define facets of Q^p_{LO} for all $(i,j) \in A_p$. Each inequality $x_{ij} \geq 0$ is equivalent to the inequality $x_{ji} \leq 1$ (and vice versa) and to no other trivial inequality. The triangle inequalities

$$e^T x(C) \leq 2 \text{ for all 3-dicycles } C \subseteq A_p \tag{4}$$

define facets of Q^p_{LO}. None of these inequalities is equivalent to any of the inequalities (3).

Following Grötschel, Jünger and Reinelt, we define

$$\begin{aligned} Q^p_1 &:= \{x \in \Re^q : x \text{ satisfies (2) and (3)}\}, \\ Q^p_2 &:= \{x \in \Re^q : x \text{ satisfies (2), (3), and (4)}\}. \end{aligned}$$

Then $Q^p_{LO} \subseteq Q^p_2 \subseteq Q^p_1$. It can be shown that

$$Q^p_{LO} = \text{conv}\{x \in Q^p_2 : x \text{ is integral}\}.$$

Therefore, if the optimal solution to the linear program $\min\{g^T x : x \in Q^p_2\}$ is integral, it is also the optimal solution to the linear ordering problem $\min\{g^T x : x \in Q^p_{LO}\}$.

Grötschel, Jünger and Reinelt have discovered several other classes of facets in addition to those mentioned above. The experience of these authors indicates that it is likely that the relatively simple facet defining inequalities given above are sufficient to solve many real-world problems. Thus it is often true that

$$\min\{g^T x : x \in Q^p_{LO}\} \;=\; \min\{g^T x : x \in Q^p_2\}.$$

For all the problems we solve in this paper, the inequalities given above suffice. However, there are some linear ordering problems for which the inequalities given above are not enough.

We use the equalities (2) to perform the substitution

$$x_{ji} = 1 - x_{ij}, \text{ for } 1 \le i < j \le p,$$

so we replace x_{ij} for $j < i$ in any inequalities in which it occurs by $1 - x_{ji}$. Thus we reduce the number of variables to $\binom{p}{2}$. Let $\bar{q} := \binom{p}{2}$. The above substitution defines a projection of $\Re^q$ onto $\Re^{\bar{q}}$. For a polyhedron $Q^p \subseteq \Re^q$ we denote the image of Q^p under this projection by $\bar{Q}^p$. It should be noted that $\bar{Q}^p$ is also a polyhedron.

Given a linear program

$$\min\{g^T x \;:\; x \in Q^p,\; Q^p \subseteq \text{ affine hull of } Q^p_{LO}\},$$

there exists an equivalent linear program

$$\min\{c^T x : x \in \bar{Q}^p\},$$

where c is given by $c_{ij} = g_{ij} - g_{ji},\ 1 \le i < j \le p$. Clearly, there exists a bijection between the points in Q^p and the points in $\bar{Q}^p$, and the value of a point in Q^p differs from the value of the corresponding point in $\bar{Q}^p$ by the constant $G := \sum_{i>j} g_{ij}$.

The systems of facets of Q^p_{LO} given by inequalities (3) are equivalent to the systems of facets of $\bar{Q}^p_{LO}$ given by

$$0 \le x_{ij} \le 1, \text{ for } 1 \le i < j \le p. \tag{5}$$

For each triple $\{i, j, k\}$ with $1 \le i < j < k \le p$ there are two facets of Q^p_{LO} given by triangle inequalities. These two facets are

$$\begin{aligned} x_{ij} + x_{jk} + x_{ki} &\le 2 \\ \text{and } x_{ji} + x_{kj} + x_{ik} &\le 2. \end{aligned}$$

These two facets are equivalent to the two facets

$$\begin{aligned} x_{ij} + x_{jk} - x_{ik} &\leq 1 \qquad (6) \\ \text{and } -x_{ij} - x_{jk} + x_{ik} &\leq 0 \qquad (7) \end{aligned}$$

of $\bar{Q}^p_{LO}$. The first of these facets corresponds to the directed cycle $\{i, j, k\}$ and the second to the directed cycle $\{k, j, i\}$.

$\bar{Q}^p_1$ consists of all $x \in \Re^{\bar{q}}$ which satisfy inequalities (5), that is, $\bar{Q}^p_1$ is the unit hypercube. $\bar{Q}^p_2$ consists of all $x \in \Re^{\bar{q}}$ which satisfy inequalities (5), (6) and (7). Hence the description of $\bar{Q}^p_2$ requires $\binom{p}{2}$ variables and $2\binom{p}{2} + 2\binom{p}{3}$ constraints.

Thus, to solve a linear ordering problem, we solved the linear programming problem $\min \{c^T x : x \in \bar{Q}^p_2\}$. We used a cutting plane procedure. Our initial feasible region for x was given by the polyhedron $\bar{Q}^p_1$. Our cuts were triangle inequalities drawn from the sets of facets defined by inequalities (6) and (7). Upper bounds on the dimensions of the matrix A in the LP relaxation (P) are $\hat{m} = 2\binom{p}{3}$ and $\hat{n} = \hat{m} + \binom{p}{2}$. We refer to p as the number of *sectors* of the linear ordering problem.

3 The primal-dual barrier method

We use the primal-dual barrier method to solve the linear programming relaxations of the linear ordering problem. Our implementation is similar to that of Lustig *et al.* [28].

Consider a linear programming problem in the form

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{s.t.} & Ax & = & b \\ & x + s & = & u \\ & x, s & \geq & 0, \end{array} \qquad (P)$$

where A is an $m \times n$ matrix, x, s, u, and c are n-vectors and b is an m-vector. Introducing a barrier term into the objective gives the problem

$$\begin{array}{lrcl} \min & c^T x - \mu \sum_{i=1}^n \log x_i s_i & & \\ \text{s.t.} & Ax & = & b \\ & x + s & = & u \\ & x, s \geq & 0. & \end{array} \qquad (P(\mu))$$

The first order Karush-Kuhn-Tucker optimality conditions for $(P(\mu))$ can be written

1. *Initialize:* Find an initial point x in $\Re^n$, y in $\Re^m$, z and w in $\Re^n$, Set $s = u - x$. Pick initial μ.

2. *Iterate:* Find the Newton direction for solving the equations (OPT). Move in this direction from the current point (x, s, y, w, z), choosing a steplength to ensure that the new iterate satisfies $x > 0$, $s > 0$, $w > 0$, and $z > 0$.

3. *Check for optimality:* If the duality gap $x^T z + w^T s$ is small enough and if the primal and dual iterates are feasible, STOP.

4. *Loop:* Decrease μ. Return to step 2.

Figure 1: The primal-dual barrier method

$$\begin{array}{rcccccc} & & Ax & & & = & b \\ & & x & + & s & = & u \\ A^T y & + & z & - & w & = & c \\ & & Zx & & & = & \mu e \\ & & & & Ws & = & \mu e, \end{array} \qquad (OPT)$$

together with nonnegativity of x, s, z, and w.

The primal-dual barrier method can be described as in figure 1. We regard the problem as solved in step 3 if the relative duality gap is less than 10^{-8} and if the relative primal and dual infeasibilities are less than 10^{-8}. The relative duality gap is $(x^T z + s^T w)/\max\{1, |\, b^T y - u^T w\,|\}$. The relative primal infeasibility is $\| Ax - b \| \,/\, \| x \|$ and the relative dual infeasibility is $\| A^T y + z - w - c \| \,/ \max\{\| y \|, \| z \|, \| w \|\}$. We follow the procedures given in [28] for finding an initial iterate, choosing a step length, and updating μ. Zhang [29] has shown that a slight variant of this algorithm converges in polynomial time.

The Newton direction is a combination of three different directions: a direction for feasibility, a centering direction, and an optimality direction. If the iterates are primal and dual feasible then the Newton direction for the conditions (OPT) is exactly the centering direction when $\mu = (x^T z + s^T w)/n$; when $\mu = 0$ the direction is exactly the optimality direction. The usual choice for μ lies between these two extremes. The iterates generated by our algorithm in section 4 are only slightly infeasible, so the direction for feasibility only plays a small part in our algorithm.

4 The Cutting Plane Algorithm

At each stage, our cutting plane algorithm approximately solves a relaxation of the linear ordering problem using the primal-dual barrier method. If any 3-dicycle con-

straint is violated, a subset is added to the relaxation and the relaxation is resolved. We proceed in this way until no violated constraints are found. Throughout, we attempt to convert the current interior point into an ordering by using heuristics. We terminate when we have found an ordering that is within some tolerance of the lower bound provided by the value of the current dual iterate.

This section describes our algorithm in more detail. We first give a formal description of the algorithm in figure 2. Steps 2 through 4 involve the solution of the LP-relaxation. Steps 5 through 10 describe the process we go through when modifying the relaxation. We improve some quantities associated with the linear ordering problem (steps 5 and 6), look for violated constraints and update the relaxation appropriately (steps 7 through 9), and create the warm start for the next relaxation (step 10).

Several parameter and algorithmic choices were made after extensive experimentation. One of the guiding principles is that the initial iterate in the new relaxation should be somewhat *centered*. We will work under the assumption that this means that it is desirable that the components of x, s, w, and z should not become too small; we will relate this characterization to the more usual characterization in section 6. We now explain some of the steps in the algorithm in more detail.

4.1 Initialization

The initial relaxation of the linear ordering problem has the form:

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{s.t.} & x + s & = & e \\ & x, s & \geq & 0, \end{array} \qquad (P_0)$$

where x and s have one entry for each ordered pair of vertices $i < j$, and c is a vector with components c_{ij}, the cost incurred from having i before j. We refer to this linear program as the initial *primal relaxation*. The initial *dual relaxation* is

$$\begin{array}{lrcl} \max & -e^T w & & \\ \text{s.t.} & z - w & = & c \\ & z, w & \geq & 0 \end{array} \qquad (D_0)$$

Here, z and w each have one entry for each ordered pair of vertices $i < j$.

The optimal solutions to (P_0) and (D_0) are given by

$$x_{ij} = \begin{cases} 1 & \text{if } c_{ij} \leq 0 \\ 0 & \text{otherwise,} \end{cases}$$

and

$$z_{ij} = \begin{cases} c_{ij} & \text{if } c_{ij} \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

1. *Initialize:* Set up and solve the initial relaxation. Choose an initial subset of the 3-dicycle constraints, and form A, b, c, and u. Initialize x, s, y, w, z for this relaxation. Initialize $lower = b^T y - u^T w$. Initialize x^{ORDER} to be the incidence vector of the natural ordering, and $upper = c^T x^{ORDER}$. Initialize $x^{FEAS} = 0.5e$. Initialize $\epsilon_1 = 0.3$, $\epsilon_2 = 10^{-6}$, $maxadd = 200$.

2. *LP step:* Take a step of the primal-dual algorithm. Let $value^P = c^T x$, $value^D = b^T y - u^T w$, $dualgap = value^P - value^D$.

3. *Check for convergence:* If the relative dual infeasibility is smaller than 10^{-8}, let $lower = \max\{lower, value^D\}$. If $\frac{upper - lower}{\max\{1, |lower|\}} \leq \epsilon_2$, STOP.

4. *Should we check for violated constraints?* If the relative dual infeasibility is greater than 10^{-8} or if $\frac{dualgap}{\max\{1, |value^D|\}} > \epsilon_1$, return to Step 2.

5. *Update x^{ORDER}:* Use heuristics to generate an ordering from x. If this ordering is better than the best ordering found previously, then update x^{ORDER} and *upper* appropriately.

6. *Update x^{FEAS}:* Let $d = x - x^{FEAS}$. Let λ_{max} be the largest value of λ such that $x^{FEAS} + \lambda d$ satisfies all the 3-dicycle inequalities. If $\lambda_{max} \geq 0.1$ then update $x^{FEAS} \leftarrow x^{FEAS} + 0.9\lambda_{max} d$.

7. *Look for violated constraints:* If x satisfies all the 3-dicyle constraints, decrease ϵ_1 and return to Step 2.

8. *Add violated inequalities:* Bucket sort the 3-dicycle constraints by the size of the violation. Go through the violated constraints, adding only arc-disjoint constraints, adding at most $maxadd$. Update ϵ_1.

9. *Drop some constraints:* Examine the primal and dual solutions to see if any constraints can be dropped.

10. *Update x, s, y, w, z:* Set $x = x^{FEAS}$. Set the additional components of x corresponding to the slack variables of the additional constraints to the appropriate value so that x is primal feasible. Set $s = u - x$. Set additional components of y to 0. If any component of x (or s) is less than 10^{-6}, set it to 10^{-6}; decrease the corresponding component of s (or x) so that $x + s = u$. If any component of z (or w) is smaller than 10^{-6}, set it to 10^{-6}; increase the corresponding component of w (or z) so that dual feasibility is maintained. Take one pure centering step. Update μ. Return to Step 2.

Figure 2: The cutting plane algorithm

with s and w at the appropriate values to ensure feasibility. We then determine which 3-dicycle constraints are violated by the optimal solution to the initial relaxation, and add an arc-disjoint subset. (Two dicycles are arc-disjoint if they do not share an arc.) The primal relaxation then has the standard form

$$\begin{array}{lrcccl} \min & c^T x & & & & \\ \text{s.t.} & Ax & & & = & b \\ & x & + & s & = & u \\ & x, s & \geq & 0. & & \end{array} \qquad (P)$$

Here, A has one row for each added 3-dicycle constraint (equations (6) or (7)). The entries of b are 0 or 1 as appropriate. Each row has four nonzeroes: three for the three arcs in the dicycle and one for a slack variable. Let m be the number of rows in A. Thus, x now has one entry for each ordered pair of vertices $i < j$, and also m entries for the slack variables, so the number of columns in A is $n = m + \binom{p}{2}$; s is configured similarly. The upper bounds $u_i = 1$ if i corresponds to an arc and $u_i = 2$ if i corresponds to a slack variable, since the largest value the slack can take in either equation (6) or (7) is 2. The dual relaxation is

$$\begin{array}{lrcccl} \max & b^T y & & & - & u^T w \\ \text{s.t.} & A^T y & + & z & - & w = c \\ & & & z, w & \geq & 0. \end{array} \qquad (D)$$

The initial iterate x, s, y, w, z is then constructed using the method described in [28].

For bookkeeping purposes, we need an initial ordering, so we take the natural ordering $\sigma(i) = i$, $i = 1, ..., p$. This is stored as an incidence vector in x^{ORDER} and its value is stored in *upper*. We use a point in the interior of $\bar{Q}_2^p$ when recentering after adding cutting planes; this point is initialized as $x^{FEAS} = .5e$.

4.2 Identifying constraints early

It is necessary to attempt to avoid solving the current relaxation to optimality. If this can be accomplished successfully, it will obviously reduce the number of iterations spent on the current relaxation; in addition, it will result in an initial iterate for the next relaxation which is more centered, which means that fewer iterations will be needed on that relaxation as well. It is possible to look for violated inequalities at every iteration, but we found that this resulted in the addition of too many constraints and it was hard for the algorithm to make progress. Thus, we introduced a requirement that the duality gap should fall below some tolerance before looking for cutting planes. Rather than using a fixed tolerance throughout the progress of the algorithm, we found it beneficial to use a dynamically altered tolerance ϵ_1. If the duality gap falls below ϵ_1 then the algorithm stops work on the current relaxation,

at least temporarily. Before looking for violated 3-dicycle inequalities, the algorithm performs two other tasks, in Steps 5 and 6.

4.2.1 Finding an ordering

The incidence vector of the best ordering found so far is stored in x^{ORDER} and its value is stored in *upper*. We attempt to improve upon x^{ORDER} by rounding the current primal iterate as follows. We set up a precedence table. If $x_{ij} > .6$ then the precedence table states that i is before j; if $x_{ij} < .4$ then the precedence table states that j is before i (these parameters were chosen after extensive computational testing). We then attempted to construct an ordering which was consistent with the precedence table, using a similar procedure to that described in Grötschel *et al.* [4]. If this procedure resulted in an ordering which was better than the best ordering found previously then we update x^{ORDER} and *upper* appropriately.

It is necessary to break ties occasionally when trying to construct an ordering, because of a lack of a preference between certain pairs. This was done randomly. For some of the problems we solved, there were several orderings that were very close to optimal, and our heuristics needed to be called several times, with ties being broken randomly, before they found the optimal ordering.

Note that we do not declare our solution optimal until we find an ordering which is within a relative duality gap of 10^{-8}. There may be several optimal solutions to the linear ordering problem. The primal iterate x will converge to a convex combination of these solutions, but our heuristics will choose one of the orderings.

4.2.2 Updating a point in $\bar{Q}_2^p$.

The vector x^{FEAS} is a point in the interior of $\bar{Q}_2^p$. It is updated by stepping towards the new iterate x. Let $d = x - x^{FEAS}$. We find the largest value of λ such that $x^{FEAS} + \lambda d$ is in $\bar{Q}_2^p$. If this value $\lambda_{\max}$ is at least 0.1, we update x^{FEAS} to $x^{FEAS} + 0.9\lambda_{\max} d$. It should be noted that $\lambda_{\max}$ can be found efficiently if we look for it at the same time as we find all cutting planes violated by x.

4.2.3 Searching for cutting planes

The 3-dicycle constraints violated by x are found by complete enumeration. They are then bucket sorted according to the size of the violation. We then go through the buckets in order and select an arc-disjoint subset of up to about two hundred of the constraints.

An alternative to adding constraints corresponding to arc-disjoint constraints is to just add the two hundred (say) most violated constraints. Adding arc-disjoint constraints has a very beneficial effect on the structure of nonzeroes in the constraint matrix A, in that at most one nonzero is added to each column. This considerably reduces the density of the Cholesky factors of the product AA^T, which in turn reduces the time spent calculating the projections and the time spent on each iteration. The drawback to adding only arc-disjoint constraints is that the number of stages of adding

cutting planes is increased, with a corresponding increase in the number of iterations; we found that this disadvantage was considerably outweighed by the benefit of reduced time per iteration.

The parameter ϵ_1 is modified depending on the maximum violation and the number of violated constraints. If the maximum violation is close to one and a large number of constraints are violated then ϵ_1 is increased; conversely, if the maximum violation is close to zero and only a few constraints are violated then ϵ_1 is decreased. This means that ϵ_1 is increased if it appears that we are close to the optimal solution to the current relaxation, so we then will not solve the new relaxation to such a high degree of accuracy. Similarly, ϵ_1 is decreased if it appears that we are not sufficiently close to solving the current relaxation, so that we will solve the new relaxation more accurately. This also has the consequence that as we approach optimality to the linear ordering problem we will not call the separation routines as often.

4.3 Restarting from a warm start

Usually, many cutting planes are added at once. To simplify the exposition, we assume that just one constraint has been added; the procedures discussed can be extended straightforwardly when many constraints are added simultaneously.

Immediately after adding a cutting plane $a_0^T x \leq b_0$, the relaxation has the form

$$
\begin{array}{lllllllllll}
\min & c^T x & & & & & & & & & \\
\text{s.t.} & Ax & & & & & & & = & b & \\
 & a_0^T x & + & x_0 & & & & & = & b_0 & \\
 & x & & & + & s & & & = & u & \\
 & & & x_0 & & & + & s_0 & = & u_0 & \\
 & x, s & \geq & 0, & & x_0, s_0 & \geq & 0, & & &
\end{array}
\qquad (P)
$$

and the dual is

$$
\begin{array}{lllllllllllll}
\max & b^T y & + & b_0 y_0 & & & - & u^T w & - & u_0 w_0 & & & \\
\text{s.t.} & A^T y & + & a_0 y_0 & + & z & - & w & & & = & c & \\
 & & & y_0 & + & z_0 & & & - & w_0 & = & 0 & \\
 & & & z, w & \geq & 0, & & z_0, w_0 & \geq & 0 & & &
\end{array}
\qquad (D)
$$

The primal problem has an extra constraint and the old primal iterate is not feasible. In our initial experiments, we tried restarting directly from this point. This is possible with the primal dual barrier method, and it worked well for randomly generated problems, but we found that it was not good for the real world problems, because of the nature of the data (see section 5). The algorithm took several iterations to get feasible, by which time it was far from optimality. Thus, it required about as many iterations to solve the relaxation from the warm start as it would have taken from a cold start. We were able to considerably improve the warm start by using the point x^{FEAS} in $\bar{Q}_2^p$. Namely, we update the first $\binom{p}{2}$ components of x to their values

in x^{FEAS}, and then x_0, s_0, the slack elements of x and the entries of s are chosen to ensure that the new iterate is primal feasible. Finally, all very small components of x (s) are increased, and the corresponding components of s (x) are decreased by the same amount. This may cause a (numerically small) loss of primal feasibility, but it improves the numerical stability of the algorithm as well as giving an iterate which is slightly more centered.

The dual problem has an extra column and one extra constraint corresponding to the additional primal slack variable x_0. The old dual solution is still feasible in this problem, with $y_0 = 0$ and $w_0 = z_0$; in order to get an interior point, we set $w_0 = z_0 = 0.1$. In addition, all very small components of w (z) are increased, with the corresponding components of z (w) increased by the same amount, in order to maintain dual feasibility. Note that we expect the new constraint to be active at the solution to the new relaxation, so the optimal value of x_0 will be zero. If the optimal solution is nondegenerate, the optimal value of z_0 will be positive and that of w_0 will be zero.

We also found it beneficial to do one pure centering iteration immediately after adding cutting planes. Thus, μ is set equal to $(x^T z + s^T w)/n$, where all quantities are the updated quantities, and one primal-dual step is taken. The barrier parameter is then updated in the usual way and we return to Step 2 in order to approximately solve the new relaxation.

4.4 Dropping constraints

If constraints are never dropped, the constraint matrix becomes large and the Cholesky factors of AA^T become dense, so the time per iteration becomes somewhat prohibitive. Therefore, it is necessary to drop constraints that appear to no longer be necessary. If the primal slack variable x_i corresponding to a constraint is bigger than 0.4 then it appears that the 3-dicycle constraint is not important, so we drop it. Intuitively, it seems like the condition for deciding whether to drop a constraint should depend on the dual variables as well as the primal variables. We tried several different criteria using the dual variables, but none of them worked as well as the simple primal-only criterion we have described.

It is also possible to develop a different criterion for dropping constraints using the basis identification techniques described in El-Bakry *et al.* [30]. Because of the structure of the constraint matrix, we found this unnecessary; the variables themselves served as good indicators of which variables are basic.

Dropping a constraint does not affect primal feasibility, but it will make the dual iterate slightly infeasible. However, it appears that the loss of dual feasibility is small and the algorithm quickly recovers.

There is a risk that the algorithm will repeatedly add and drop the same constraints, and we occasionally saw this behaviour in some of our early experiments. To reduce the possibility of this happening, we do not drop a constraint in the stage immediately following the one where it was added. This safeguard prevented cycling in our experiments.

5 Computational results

Table 1 contains the results of our algorithm on 43 real-world linear ordering problems. All were obtained on a SUN SPARC 10/30, with code written in FORTRAN and compiled using the Sun f77 compiler, and the CPU time was measured using the command ETIME. These times include the time to read in the problem.

All of the problems come from input-output tables. Except for the more recent problem *usa79*, they all date from 1954 to 1975. The entries in an input-output matrix measure the number of deliveries from one sector of the country to another. For an economic analysis of these results, see Grötschel *et al.* [31]. Problems whose names start with *t* are 44 by 44 tables available from EUROSTAT in the European Community. The second and third characters in the name give the year of the table and the fourth character identifies the country: b is Belgium, d is (West) Germany, e is Spain, f is France, i is Italy, k is Denmark, l is Luxembourg, n is the Netherlands, r is Ireland, u is the United Kingdom, w is the compilation of the data for the six original members of the European Community, and x is the compilation of the data for the first nine members of the E.C. t70d11b is one of two tables available from EUROSTAT for Germany for the year 1970. There are two additional tables available from EUROSTAT: t59b11 and t70d11a; neither of these problems can be solved by using only 3-dicycle inequalities. Problems whose names start with *diw* are 56 by 56 tables compiled by the Deutches Institut für Wirtschaftforschung for (West) Germany for the years 1954 to 1972. The last two characters indicate the year; the character before these two indicates whether prices are current prices or 1962 prices. The problems whose names start with *sbm* are compiled by the Statistisches Bundesamt for the years 1974 and 1975 and again are for (West) Germany. There is also a table available for 1970 but this again can not be solved only using 3-dicycle inequalities. The problem *usa79* is a more recent 79 by 79 input-output table containing data for the U.S.A. The coefficients g_{ij} in all these tables are distributed between 0 and approximately 30000, but they are not uniformly distributed: there are a number of entries which are zero and also a number of entries which are small but positive. The zero entries lead to the presence of alternative optima. Interior point methods tend to converge to the interior of the optimal face; therefore, they converge to a solution which corresponds to a *partial* ordering, with every ordering that satisfies the partial ordering being optimal.

The columns in Table 1 have the following meanings. The first column contains the name of the problem. The second column contains the number of 3-dicycle inequalities in the initial relaxation. The third column contains the total number of 3-dicycle inequalities added subsequently. The fourth column contains the number of times cutting planes are added. The fifth column contains the number of 3-dicycle inequalities in the final relaxation. The sixth column contains the number of iterations of the primal-dual method which were required. The final column contains the total run time, in seconds. The total number of constraints dropped is the number of constraints in the initial relaxation together with the number of constraints added minus the number of constraints in the final relaxation. The table contains the mean

Problem	Initial m	Cuts added	Stages	Final m	Iterations	Time (seconds)
t59d11	67	294	17	298	78	23.5
t59f11	76	223	13	247	65	16.2
t59i11	80	349	15	353	73	19.7
t59n11	73	318	19	306	86	24.3
t65b11	93	333	15	339	87	23.8
t65d11	105	248	13	282	63	17.3
t65f11	95	258	12	295	63	16.1
t65i11	97	345	16	380	77	22.0
t65l11	77	182	11	194	52	13.4
t65n11	104	393	21	422	101	42.3
t65w11	101	275	12	328	63	16.7
t69r11	75	337	20	309	98	25.7
t70b11	86	276	11	285	61	16.1
t70d11b	132	303	10	362	51	15.7
t70f11	104	355	14	399	69	20.5
t70i11	85	383	18	385	96	28.0
t70k11	73	380	23	362	105	33.5
t70l11	75	417	22	346	93	24.1
t70n11	94	303	11	328	65	16.9
t70u11	63	159	9	177	44	11.2
t70w11	110	234	11	303	56	14.3
t70x11	105	248	12	299	59	14.9
t74d11	125	387	21	395	85	28.4
t75d11	124	324	12	367	65	20.6
t75e11	100	391	13	415	80	22.9
t75i11	88	373	16	380	89	25.2
t75k11	81	361	17	352	80	23.3
t75n11	86	316	19	340	92	26.8
t75u11	92	264	16	298	71	18.8
Mean	92	311	15	329	75	21.4
diw56n54	210	753	22	703	109	87.7
diw56n58	206	628	15	627	68	44.7
diw56n62	205	627	17	646	84	67.5
diw56n66	204	670	14	681	62	43.1
diw56n67	211	776	19	726	106	110.1
diw56n72	213	716	18	693	91	70.0
diw56r54	211	681	18	681	89	70.0
diw56r58	207	633	13	629	65	38.9
diw56r66	206	648	15	684	68	48.9
diw56r67	217	777	20	772	97	131.7
diw56r72	206	768	18	738	96	121.2
Mean	209	698	17	689	85	75.8
sbm1974	225	1006	19	879	109	180.6
sbm1975	231	901	18	892	98	173.0
Mean	228	954	19	886	104	176.8
usa79	447	2104	25	1793	187	3208.2

Table 1: Results on 43 input-output matrices

values for each of the first three sets of problems. The dimension of the primal variable x in the final relaxation is equal to $n = m + \binom{p}{2}$, where p is the number of sectors and m is the number of 3-dicycle inequalities in the final relaxation. Thus, the mean numbers of variables in the final relaxation for the 44, 56, 60, and 79 sector problems are 1275, 2229, 2656, and 4874, respectively.

The number of stages required is similar to the number reported by Grötschel *et al.* [4] when they added arc-disjoint inequalities; thus, the attempt to identify violated constraints early appears to be successful, without many superfluous constraints being added. The number of iterations per stage is about five, considerably smaller than would be required if each stage was solved to optimality and then the next stage started from scratch.

The fact that the input-output tables contain a number of small positive entries means that there are several orderings which are close to optimal. It was because of the presence of these alternative orderings that we had to introduce a randomization aspect into our heuristics for generating an ordering from an interior point. We found these real world problems harder than randomly generated problems with the same degree of linearity; we conjecture that this is due to the presence of the small positive entries, because there is then a large face which is almost orthogonal to the objective function. This makes it more difficult to identify the correct set of important constraints.

It is impossible to really compare our run times with those of Grötschel *et al.* [4], who were using an IBM 370/168 in 1982. For the 44 by 44 tables, their run times are about one minute, for the 56 by 56 tables they are about four and a half to five minutes, and for the 60 by 60 problems they are almost ten minutes. Thus as the problem size increases, the ratio of their run times to ours increases, which is the standard pattern for simplex results versus interior point results. Their machine is not listed in Dongarra's list of LINPACK benchmarks [32]. However, the *IBM370/168 Fast Mult* has a Mflop/s rate of 1.2, which compares with a rate of 9.3 Mflop/s for the SUN SPARC 10/30. Thus, we conjecture that our machine is about eight times faster than the machine used by Grötschel *et al.* [4], and so their code would take (all things being equal) approximately 8 seconds, 40 seconds, and 80 seconds for the three problem sizes, if it was run on our machine. Therefore, our runtimes are very competitive with those of Grötschel *et al.* [4], at least for the larger problems. Note that the simplex method has been improved considerably since 1982 — see for example Bixby *et al.* [15]. We have corresponded recently with Gerhard Reinelt, and he now has an implementation of his linear ordering code which uses CPLEX 2.2 [33], which he has run on a SUN SPARC 10/20. From information provided by Sun Microsystems, we can conclude that the SUN SPARC 10/20 is about 90% as fast as the SUN SPARC 10/30 (the SUN SPARC 10/20 is not listed in [32]). Reinelt is not willing to publish these results in detail yet because they are still preliminary, and they may not be with the optimal parameter set. The average runtimes that he obtained were about 3 seconds for the 44 sector problems, about 20 seconds for the 56 sector problems, about 50 seconds for the 60 sector problems, and 1944 seconds for *usa79*. We summarize these runtime averages in Table 2. Thus, we are about

	Our results	Grötschel *et al.*	
	SPARC 10/30	SPARC 10/20	IBM 370/168
44 sector	21.4	3	63
56 sector	75.8	20	292
60 sector	176.8	50	585
79 sector	3208.2	1944	Not available

Table 2: Comparing average runtimes (in seconds)

eight times worse for the small 44 sector problems, about four times worse for the medium sized 56 and 60 sector problems, and only two times worse for the 79 sector problem. Clearly, the difference between the two algorithms decreases as the problem size increases, and we would expect to see superior performance for the interior point algorithm for slightly larger problems, perhaps around 100 sectors. The size of the linear programming relaxations are around the size where interior point and simplex methods require about the same amount of time; thus, it is reasonable that we would need to look at larger problems before we would see the benefit of using an interior point cutting plane algorithm.

Our runtimes could also be improved by using linear algebra routines designed specifically for interior point methods. For example, the use of *supernodes* when calculating the Cholesky factorizations can speed up an interior point method considerably — see Lustig *et al.* [34], for example. For the 44, 56, 60, and 79 sector problems, the time to calculate the projections was about 60%, 70%, 80%, and 97% respectively of the total run time, increasing as the size of the problems increased. (Reinelt states that over 99% of his runtime is spent in the linear programming routines for *usa79*.)

6 Theoretical concerns

In this section, we show how some of our algorithmic choices were driven by theoretical concerns. Theoretically, it is known that keeping the iterates centered results in better algorithms for linear programming. We demonstrate how early termination and the use of x^{FEAS} both help to produce iterates which are more centered. We show the importance of attempting to restart from feasible points. We also show how our algorithm could be made polynomial in the number of added constraints.

As in section 4.3, we assume in sections 6.1 and 6.2 that only one constraint is added at a time. The arguments presented in these sections can easily be extended to the general case.

6.1 Early termination produces more-centered iterates

Roos and Vial [35] showed that if the norm of the vector

$$v := n(Xz + Sw)/(x^T z + s^T w) - e \tag{8}$$

is small then, with an appropriate choice of μ, the iterates converge quadratically to the solution to $(P(\mu))$; the norm gives a good measure of centrality. Instead of using this standard measure of centrality, we used the observation that it is advisable to avoid letting the iterates approach the boundary of the feasible region too closely. We want to show how the two criteria are related.

The quantity $x^T z + s^T w$ is the duality gap. If the current relaxation is solved almost to optimality and then a cutting plane is added with slack variable x_0, the new duality gap will be approximately $x_0 z_0 + s_0 w_0$, provided x, s, z, and w are left unchanged. Thus, the vector v will have one term which is approximately n and n terms which are approximately -1, so the standard measure of centrality will be large, and it will take several iterations to get approximately centered.

For numerical reasons, it is useful to increase small terms when restarting. When we increase very small components of x, s, z, and w, we will dramatically change the corresponding values of $x_i z_i$ and $s_i w_i$. This will again result in an iterate that is far from centered. This is an effect that can be mitigated somewhat by terminating early, because the values of the variables are then not quite so small.

One of the methods we used to improve the performance of the algorithm was to restart from a primal point which is feasible in the whole problem. This changes the whole of the vector x, so it may well change the vector v considerably; however, it does increase the duality gap, so the imbalance between the added variable and the old variables in the vector v is greatly reduced. In addition, the relative changes between the old components of v will be reduced somewhat if the variables have not been driven too close to zero. Notice that in the algorithm, we take a pure centering step immediately after adding constraints; this is precisely because the norm of the vector v has usually increased considerably.

6.2 The importance of restarting from feasible iterates

Several papers (eg, Anstreicher [36], Mizuno *et al.* [37], Zhang [29]) have discussed algorithms which move towards feasibility and optimality simultaneously. A common feature of the analysis of these algorithms is the exploitation of the fact that they move towards feasibility at least as fast as they move towards optimality. In our cutting plane algorithm, we restart with the primal point reset to x^{FEAS}. If instead we attempted to restart directly from the approximate solution to the previous relaxation, the primal infeasibility would be $x_0 + |\, b_0 - a_0^T x \,|$. To ensure that the ratio between the duality gap $x^T z + s^T w + x_0 z_0 + s_0 w_0$ and the infeasibility remained reasonably large, it would be necessary to set $x_0 z_0 + s_0 w_0$ to a large value. As argued above, this is going to result in an iterate which is far from centered.

To the best of our knowledge, none of the interior point column generation algorithms proposed in the literature generate infeasible iterates. They either shift constraints so that the current iterate is still an interior point (for example, [24],[38]), or they back up to a known feasible point (for example, [39, 23],[19],[18]). The algorithm in this paper is related to the second class, in that we back up to the point x^{FEAS}.

6.3 Total number of iterations

It is straightforward to modify the algorithm presented to obtain one which requires time polynomial in the total number of constraints added. (Of course, this is not going to give a polynomial time algorithm for the linear ordering problem, just a polynomial time algorithm for solving the problem $\min\{g^T x : x \in Q_2\}$.)

Because of the way we modify the iterate when adding constraints, the initial iterate for each relaxation gives a vector v with norm $O(nL)$, where v is defined in equation (8), n is the number of constraints in the current relaxation, and L is the size of the data. Thus, from the analysis of Zhang [29], the algorithm can be modified slightly so that only a polynomial number of iterations are required at each stage. If we impose an upper bound K on the number of times any one constraint can be dropped, then the number of stages is no more than $2K\binom{p}{3}$, since $2\binom{p}{3}$ is the number of constraints for $\bar{Q}_2^p$. Thus the total number of iterations required by the slightly modified algorithm is polynomial, and each of these iterations can be performed in polynomial time.

7 Conclusions

We have presented a description of an implementation of a cutting plane algorithm for the linear ordering problem which uses an interior point method to solve the linear programming relaxations. This algorithm appears to require time comparable with that required by a simplex implementation. As the problem size increases, the time required by the interior point code does not increase as quickly as that required by the simplex code. To achieve this performance, it was necessary to implement the algorithm carefully in order to exploit the warm start provided by the solution to the linear programming relaxation. We have described several techniques which help in the exploitation of the warm start, and it should be possible to extend these techniques to cutting plane algorithms for other integer linear programming problems such as the traveling salesman problem.

Acknowledgement

We are very grateful to both Michael Jünger and Gerhard Reinelt for supplying us with the input-output matrices analyzed in this paper and for discussing their recent results with us. We are also grateful to Eberhard Kranich for his bibliography on interior point methods.

References

[1] M. Grötschel and O. Holland, Solution of large-scale travelling salesman problems, Mathematical Programming 51(1991)141.

[2] M. W. Padberg and G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33(1991)60.

[3] I. J. Lustig, R. E. Marsten, and D. F. Shanno, On implementing Mehrotra's predictor-corrector interior point method for linear programming, SIAM Journal on Optimization 2(1992)435.

[4] M. Grötschel, M. Jünger, and G. Reinelt, A cutting plane algorithm for the linear ordering problem, Operations Research 32(1984)1195.

[5] M. Jünger, *Polyhedral Combinatorics and the Acyclic Subdigraph Problem* (Heldermann, Berlin, 1985).

[6] G. Reinelt, *The Linear Ordering Problem: Algorithms and Applications* (Heldermann, Berlin, 1985).

[7] J. E. Mitchell and B. Borchers, A primal-dual interior point cutting plane method for the linear ordering problem, COAL Bulletin 21(1992)13.

[8] S. Chopra, E. R. Gorres, and M. R. Rao, Solving the Steiner tree problem on a graph using branch and cut, ORSA Journal on Computing 4(1992)320.

[9] M. Grötschel and O. Holland, Solving matching problems with linear programming, Mathematical Programming 33(1985)243.

[10] M. Grötschel, C. L. Monma, and M. Stoer, Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints, Operations Research 40(1992)309.

[11] B. Borchers and J. E. Mitchell, Using an interior point method in a branch and bound algorithm for integer programming, Technical Report 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180 (1991).

[12] IBM, IBM Optimization Subroutine Library Guide and Reference, Publication number SC23–0519–1, New York (1990).

[13] R. E. Bixby and M. J. Saltzman, Recovering an optimal LP basis from an interior point solution, Operations Research Letters 15(1994)169.

[14] N. Megiddo, On finding primal– and dual–optimal bases, ORSA Journal on Computing 3(1991)63.

[15] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno, Very large-scale linear programming: a case study in combining interior point and simplex methods, Operations Research 40(1992)885.

[16] J. L. Goffin and J. P. Vial, Cutting planes and column generation techniques with the projective algorithm, Journal of Optimization Theory and Applications 65(1990)409.

[17] J. L. Goffin, A. Haurie, and J. P. Vial, Decomposition and nondifferentiable optimization with the projective algorithm, Management Science 38(1992)284.

[18] J. E. Mitchell and M. J. Todd, Solving combinatorial optimization problems using Karmarkar's algorithm, Mathematical Programming 56(1992)245.

[19] J. A. Kaliski and Y. Ye, A decomposition variant of the potential reduction algorithm for linear programming, Management Science 39(1993)757.

[20] G. B. Dantzig and Y. Ye, A build–up interior method for linear programming : Affine scaling form, Technical Report SOL 90–4, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, Stanford, CA 94305, USA (1990).

[21] M. G. C. Resende and G. Veiga, An efficient implementation of a network interior point method, in: *Network Flows and Matching: First DIMACS Implementation Challenge,* ed. D.S. Johnson and C.C. McGeogh, (DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 1993).

[22] R. V. Helgason and J. L. Kennington, NETFLO, Computer Program, Operations Research Department, Southern Methodist University, Dallas, Texas (1985).

[23] D. den Hertog, C. Roos, and T. Terlaky, A build–up variant of the path–following method for LP, Operations Research Letters 12(1992)181.

[24] D. S. Atkinson and P. M. Vaidya, An analytic center based cutting plane algorithm for convex programming, Technical report, Department of Mathematics, University of Illinois at Urbana-Champaign (1992).

[25] I. J. Lustig, R. E. Marsten, and D. F. Shanno, Starting and restarting the primal–dual interior point method, Technical Report SOR 90–14, School of Engineering and Applied Science, Dept. of Civil Engineering and Operations Research, Princeton University, Princeton, NJ 08544, USA (1990).

[26] N. K. Karmarkar, M. G. C. Resende, and K. G. Ramakrishnan, An interior point algorithm to solve computationally difficult set covering problems, Mathematical Programming 52(1991)597.

[27] R. M. Karp, Reducibility among combinatorial problems, in: *Complexity of Computer Computations*, ed. R. E. Miller and J. W. Thatcher (Plenum Press, 1972).

[28] I. J. Lustig, R. E. Marsten, and D. F. Shanno, Computational experience with a primal–dual interior point method for linear programming, Linear Algebra and Its Applications 152(1991)191.

[29] Y. Zhang, On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem, SIAM Journal on Optimization 4(1994)208.

[30] A. S. El–Bakry, R. A. Tapia, and Y. Zhang, A study of indicators for identifying zero variables in interior–point methods, SIAM Review 36(1994)45.

[31] M. Grötschel, M. Jünger, and G. Reinelt, Optimal triangulation of large real-world input-output matrices, Statistiche Hefte 25(1984)261.

[32] J. J. Dongarra, Performance of various computers using standard linear equations software, Technical Report CS – 89 – 85, Computer Science Department, University of Tennessee, Knoxville, TN 37996–1301 (1995).

[33] CPLEX Optimization Inc, Using the CPLEX Callable Library, Suite 279, 930 Tahoe Blvd. Bldg 802, Incline Village, NV 89541 (1994).

[34] I. J. Lustig, R. E. Marsten, and D. F. Shanno, Interior point methods for linear programming: Computational state of the art, ORSA Journal on Computing 6(1994)1.

[35] C. Roos and J. P. Vial, A polynomial method of approximate centers for linear programming, Mathematical Programming 54(1992)295.

[36] K. M. Anstreicher, A combined phase I – phase II scaled potential algorithm for linear programming, Mathematical Programming 52(1991)429.

[37] S. Mizuno, M. Kojima, and M. J. Todd, Infeasible–interior–point primal–dual potential–reduction algorithms for linear programming, Technical Report 1023, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853–3801, USA (1992).

[38] J. E. Mitchell, An interior point column generation method for linear programming using shifted barriers, SIAM Journal on Optimization 4(1994)423.

[39] D. den Hertog, Interior Point Approach to Linear, Quadratic and Convex Programming, Algorithms and Complexity, PhD thesis, Faculty of Mathematics and Informatics, TU Delft, NL–2628 BL Delft, The Netherlands (1992).