Mathematical Programming 56 (1992) 245-284
North-Holland

Solving combinatorial optimization problems using Karmarkar's algorithm

John E. Mitchell*

Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Michael J. Todd**

School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853, USA

Received 9 March 1989
Revised manuscript received 29 April 1991

We describe a cutting plane algorithm for solving combinatorial optimization problems. The primal projective standard-form variant of Karmarkar's algorithm for linear programming is applied to the duals of a sequence of linear programming relaxations of the combinatorial optimization problem.

Key words: Combinatorial optimization, integer programming, Karmarkar's method.

1. Introduction

A combinatorial optimization problem involves picking the best solution from a finite set of feasible solutions. One way to solve such problems is to use a cutting plane approach. In such a method, a sequence of linear programming relaxations of the original combinatorial problem is examined, with each linear programming problem in the sequence being obtained from the previous relaxation by adding one or more inequalities.

Classically, linear programming problems have been solved using the simplex algorithm due to Dantzig [8]. In 1984, Karmarkar [28] introduced an alternative algorithm for linear programming which has better worst-case performance than the simplex algorithm. This algorithm has led to development of a class of interior point methods for linear programming (see Goldfarb and Todd [19] or den Hertog and Roos [26] for surveys). Computational testing (see, for example, [1, 6, 33, 34])

Computational facilities provided by the Cornell Computational Optimization Project supported by NSF Grant DMS-8706133 and by the Cornell National Supercomputer Facility. The Cornell National Supercomputer Facility is a resource of the Center for Theory and Simulation in Science and Engineering at Cornell Unversity, which is funded in part by the National Science Foundation, New York State, and the IBM Corporation. The research of both authors was partially supported by the U.S. Army Research Office through the Mathematical Sciences Institute of Cornell University.

* Research partially supported by ONR Grant N00014-90-J-1714.

** Research partially supported by NSF Grant ECS-8602534 and by ONR Contract N00014-87-K-0212.

indicates that some interior point methods may outperform the simplex algorithm on many classes of linear programming problems.

In this paper, we present a cutting plane algorithm for solving combinatorial optimization problems which is based on Karmarkar's algorithm. The algorithm is applicable to most integer linear programming problems; we show how it may be used to solve matching problems. Our computational experience with this algorithm on matching problems is described. The results of this paper were announced in Mitchell and Todd [40].

This paper is organized as follows. In Section 2, we describe how we apply an interior method to a given relaxation of our combinatorial optimization problem. In Section 3, we present our algorithm, with some of the details being given in Section 3, and others being presented in Section 4 and Section 5. In Section 6, we discuss some of the implementation details and in Section 7 we present our computational results. Our conclusions are contained in Section 8. Further computational testing is required before definitive conclusions can be made.

At each iteration of our algorithm, we have a linear programming relaxation of the combinatorial optimization problem. The standard way to solve these linear programs is by using the simplex method. Every point which is feasible in the combinatorial optimization problem is also feasible in each relaxation. Generally, the next relaxation is obtained from the current one by adding constraints, and these constraints cut off the current feasible solution, so it is not feasible in the next relaxation. The addition of cutting planes adds variables to the dual linear program; if we give these additional variables value zero then the previous dual point is still feasible. Since dual feasibility is maintained, each revision of the original linear program is solved using the dual simplex method.

We consider using an interior point method instead of the simplex method to solve the linear programs that arise when using a cutting plane approach. We choose to apply Karmarkar's algorithm to the dual of the relaxation of the combinatorial problem, so after adding cutting planes we know a feasible point, but that point is not strictly positive. Therefore, we need to be able to obtain an interior point in the dual when we add cutting planes. We give a direction which produces such a point in Section 4.

We use the primal projective standard-form variant of Karmarkar's algorithm, applying it to the dual of our current relaxation. The projective standard-form variant is known to solve linear programs in polynomial time. The affine variant is not known to be polynomial. Through consideration of *trajectories* (see [3, 4, 35]), Megiddo and Shub [36] have provided evidence that it may take exponential time when started very close to a particular vertex of the polyhedron of feasible solutions to the linear program. These authors exhibited problems where one of the infinitesimal trajectories touches every vertex, so if the sequence of iterates follows this infinitesimal path, an exponential number of iterations will be required.

If the current relaxation of the combinatorial problem is not the final relaxation, the optimal vertex of the current relaxation will not be the optimal vertex of the final one. The sequence of iterates we generate converges to the optimal vertex of the current relaxation. Therefore, we will probably be restarting the algorithm from a point close to a nonoptimal vertex, so some centering may well be necessary in order to obtain an efficient algorithm. For this reason, we chose to use an interior point method with a built-in centering component.

For many combinatorial optimization problems, it is relatively easy to find the optimal solution; it is much harder to actually prove optimality. Usually, proving optimality requires the examination of a dual problem. This observation provides motivation for our decision to apply Karmarkar's algorithm to the *dual* of the relaxation of our combinatorial optimization problem. Another benefit of applying the algorithm to the dual of the relaxation is that branch-and-bound is easier to implement, since fixing a primal variable corresponds to dropping a dual constraint.

Many of the best computational results for interior point methods have been obtained for primal-dual interior point methods; see Lustig et al. [33, 34] and Choi et al. [6]. These methods maintain strictly interior points in both the primal and the dual. Thus, the addition of a cutting plane will introduce dual infeasibility in addition to giving a point on the boundary of the primal linear programming problem, making it necessary to find new interior points in both the primal and the dual problems. One way to get around this potential difficulty is to restart from the last dual iterate which was feasible in the modified problem; the drawback with this approach is that the resulting point may be far from the central trajectory in the new problem.

Goffin et al. [16, 17, 18] have investigated using variants of Karmarkar's algorithm to solve problems in non-differentiable optimization. Their methods involve the addition of cutting planes and we refer the interested reader to the papers cited. Karmarkar et al. [29, 30] have considered using interior-point methods in a different way to solve problems in combinatorial optimization.

Freund [12] has proposed a shifted barrier approach when a "warm start" is known; Mitchell [38] has shown how this method can be used in the column generation setting and has related the resulting direction to the one derived in this paper. Warm starts can be exploited in an algorithm based on an interior point method, as evidenced by Borchers and Mitchell [5], where a branch-and-bound code which solves the subproblems using a dual affine method is presented. The performance of this algorithm has been compared with the IBM package OSL [27] on several facility location problems. The run times for the interior point code are competitive with, and for larger problems often better than, those for OSL. From a more theoretical viewpoint, Ye [47] has investigated the change in a potential value of a polytope when an additional constraint is added. He has used this result to show that a particular column generation algorithm runs in time polynomial in the length of the original and added data. The papers [5, 38, 47] were prepared after the first version of the current paper.

In order to employ linear programming techniques we have to express our combinatorial problem in polyhedral form. We assume that our problem is given in the form $\max\{b^{\mathrm{T}}y : y \in \mathcal{Y}\}$ where $\mathcal{Y}$ is a finite set of points in $\mathbb{R}^m$ and $b \in \mathbb{R}^m$.

Define $\operatorname{conv}(\mathcal{Y})$ to be the convex hull of $\mathcal{Y}$. Then the combinatorial problem is equivalent to the linear program $\max\{b^{\mathrm{T}}y : y \in \operatorname{conv}(\mathcal{Y})\}$. Notice that the constraints are not stated explicitly in this formulation. In using a cutting plane approach, we are looking to represent a sufficient number of these constraints explicitly in the form $B^{\mathrm{T}}y \leq h$ so that the optimal point for the linear program $\max\{b^{\mathrm{T}}y : B^{\mathrm{T}}y \leq h\}$ is in $\operatorname{conv}(\mathcal{Y})$.

The algorithm we develop is employed to solve the perfect matching problem, a classical problem in graph theory. Each feasible solution consists of a subset of the edges of a given graph, so it can be represented by a vector in $\mathbb{R}^p$, namely the incidence vector of the subset. (Here, p denotes the number of edges of the graph.) As was shown by Edmonds [10, 11], the convex hull of these incidence vectors is the set of vectors which satisfy three sets of constraints, known as *nonnegativity constraints*, *degree constraints* and *odd set constraints*. The number of odd set constraints is considerably larger than the number of nonnegativity constraints and degree constraints, so our cutting plane algorithm would use an initial relaxation consisting of all points satisfying the nonnegativity constraints and the degree constraints, and the algorithm would add odd set constraints as necessary in the form of cutting planes. The reason for testing the code on the perfect matching problem is that there are good straightforward separation routines for this problem so it is possible to analyze the fundamental algorithm without getting distracted by the peripheral issue of finding good cutting planes. Grötschel and Holland [24] have developed a cutting plane algorithm for the perfect matching problem which is based on the simplex algorithm. Like those authors, we do not expect to obtain an algorithm competitive with combinatorial codes for this problem.

2. Our standard-form variant

We assume that the current linear programming relaxation of our combinatorial optimization problem is expressed in the form

$$(\hat{\mathrm{D}}) \qquad \begin{aligned} \max \quad & b^{\mathrm{T}}y \\ \text{s.t.} \quad & \hat{A}^{\mathrm{T}}y \leq \hat{c}, \end{aligned}$$

where y and b are m-vectors, $\hat{c}$ is an $(n-1)$-vector, and $\hat{A} \in \mathbb{R}^{m \times (n-1)}$. (We discuss alternative formulations at the end of this section.) The problem $(\hat{\mathrm{D}})$ is equivalent to the following problem:

$$(\mathrm{D}) \qquad \begin{aligned} \max \quad & z \\ \text{s.t.} \quad & A^{\mathrm{T}}y + zg \leq c, \end{aligned}$$

where $A := [\hat{A} \mid b] \in \mathbb{R}^{m \times n}$, $c := (\hat{c}^{\mathrm{T}}, 0)^{\mathrm{T}} \in \mathbb{R}^n$, and $g := (0, 1) \in \mathbb{R}^n$. We assume that A has rank m.

The dual problem to (D) is

$$\text{(P)} \quad \begin{array}{ll} \min & c^{\mathrm{T}}x \\ \text{s.t.} & Ax=0, \\ & g^{\mathrm{T}}x=1, \\ & x\geqslant 0. \end{array}$$

In order to be consistent with the usual terminology, we refer to (P) as the current *primal* linear programming problem, and we refer to (D) as the current *dual* linear programming problem.

The problem (P) is in the standard form for the primal projective standard-form variant of Karmarkar's algorithm. This variant is known to converge to the optimal solution in time that is bounded by a polynomial in the length of the input, provided the feasible region is compact. It has been investigated by Anstreicher [2], Gonzaga [22], Gay [13], Jensen and Steger [44] and Ye and Kojima [48], among others.

The algorithm is given in Figure 1. For a motivation and derivation of this method, see, for example, [22]. The vector of ones is denoted by e and (y^k, z^k) denotes the vector $(y^{k^{\mathrm{T}}}, z^k)^{\mathrm{T}}$. For any matrix M, P_M denotes the projection map onto the null space of M; if M has full row rank, the matrix operator corresponding to P_M is $I-M^{\mathrm{T}}(MM^{\mathrm{T}})^{-1}M$.

```
Initialization: Set $k=0$. Let $x^0>0$ be feasible for (P) and $(y^0, z^0)$ be feasible for (D). Choose a tolerance
  $\varepsilon$ for the duality gap.
While $c^{\mathrm{T}}x^k - z^k > \varepsilon$
    (Update dual solution): Solve the one variable linear program
    (D(z))   max  $z$
             s.t. $A^{\mathrm{T}}y(z)+zg\leqslant c$,
      where $y(z):=(A^kA^{k^{\mathrm{T}}})^{-1}A^k(c^k-zg^k)$. Here, $A^k:=AX^k$, $c^k:=X^kc$, $g^k:=X^kg$, and $X^k$ denotes the
      diagonal matrix whose diagonal entries are the coordinates of the current primal solution $x^k$. Let $\bar{z}$
      be the optimal value of the problem (D(z)). If $\bar{z}>z^k$, set $z^{k+1}\leftarrow\bar{z}$, $y^{k+1}\leftarrow y(\bar{z})$; otherwise, set
      $z^{k+1}\leftarrow z^k$, $y^{k+1}\leftarrow y^k$.
    (Compute search direction): Set $d\leftarrow -P_{A^k}(c^k-z^{k+1}g^k)$.
    (Choose step size): Choose step length $\alpha$ and set $\bar{x}\leftarrow e+\alpha d$, with $\alpha$ chosen so that $\bar{x}>0$.
    (Update primal solution): Set $x^{k+1}\leftarrow(g^{k^{\mathrm{T}}}\bar{x})^{-1}X^k\bar{x}$.
    (Update iteration count): Set $k\leftarrow k+1$.
End While
```

Fig. 1. The projective standard-form variant of Karmarkar's algorithm.

Notice that $y(z)$ is a linear function of z, so the problem (D(z)) defined in Step 2 can be solved by a ratio test. (D(z)) is a constrained version of (D), so $(y(z), z)$ is feasible in (D) if it is feasible in (D(z)). This method of updating the dual solution was first proposed by Todd and Burrell [45].

The step length α defined in Step 4 can be found in any of several different ways. For example, α can be chosen by using a line search on an appropriate potential function, it can be a fixed multiple of $\|d\|^{-1}$, or it can be chosen so that $\min\{\tilde{x}_i : 1 \leq i \leq n\} = \gamma$ for some fixed γ such as 0.1 or 0.01. See references [28], [45] and [46].

Because g is nonnegative and nonzero and $\tilde{x}$ is strictly positive, the update of the primal solution is well defined.

If some of the constraints in $(\hat{\mathrm{D}})$ are equality constraints, some of the variables in the primal problem will be unrestricted in sign. In such a situation, we use the primal projective algorithm described in Mitchell and Todd [39]. This algorithm is based on eliminating unrestricted variables, as is done also in the dual affine algorithm first described in Adler et al. [1].

3. An algorithm for perfect matching problems

3.1. *The perfect matching problem*

The perfect matching problem is one of the fundamental problems of combinatorial optimization. It was shown to be polynomially solvable by Edmonds [11] in a classic paper in which he introduced and formalized the concept of a good algorithm. Edmonds [10] found a polyhedral description for the perfect matching problem, giving a complete and nonredundant description of the facets of this polyhedron. The number of facets is exponential in the number of vertices of the graph, but Edmonds was able to use the structure of the set of facets to obtain an alogirthm which runs in time polynomial in the number of vertices.

In order to define the perfect matching problem, we first 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. A graph G is called *complete* if every pair of vertices is adjacent and if G contains no loops, that is, no edges of the form ii. The complete graph on n vertices is written K_n. If $G = [V, E]$ is a graph and $W \subseteq V$, the set of edges in the subgraph induced by W is defined to be $E(W) := \{ij \in E : i, j \in W\}$, and the cut induced by W is defined to be $\delta(W) := \{ij \in E : i \in W, j \in V \backslash W\}$. We write $\delta(v)$ for $\delta(\{v\})$, the set of edges adjacent to vertex v.

We now define the perfect matching problem. A set of edges $M \subseteq E$ with no two edges sharing a common endvertex is called a *matching*. Let $n = |V|$. Henceforth, we assume n is even. A matching of cardinality $\frac{1}{2}n$ is called a *perfect matching*. It should be noted that if M is a perfect matching then every vertex in V is an endvertex of exactly one edge in M. Let $b : E \rightarrow \mathbb{R}$ be a weight function on the edges of the graph G. For a matching M, the weight of M is given by

$$b(M) := \sum_{e \in M} b(e).$$

Then the perfect matching problem is to find the minimum weight perfect matching in the graph G. Since we are interested in solving the perfect matching problem we assume, without loss of generality, that $b>0$.

3.2. Formulation as a linear programming problem

We now recall Edmonds's polyhedral description of the perfect matching problem.

Let A be a subset of the edges of the graph $G=[V,E]$. Define $m=|E|$, and consider the components of $\mathbb{R}^m$ indexed by E. The *incidence vector* $y(A)$ of A is the $\{0,1\}$-vector in $\mathbb{R}^m$ given by

$$y_e(A)=\begin{cases}1 & \text{if } e\in A,\\ 0 & \text{otherwise.}\end{cases}$$

The *perfect matching polytope* $P(G)\subseteq\mathbb{R}^m$ of the graph G is defined to be the convex hull of the set of incidence vectors of perfect matchings of the graph. Thus $P(G)$ can be written

$$P(G):=\operatorname{conv}\{y(M)\in\mathbb{R}^m\colon M\subseteq E \text{ is a perfect matching}\}.$$

There is a bijection between the vertices of $P(G)$ and the perfect matchings M of the graph G.

Edmonds showed:

Theorem 1. *For every graph $G=[V,E]$, $P(G)$ is the solution set of the following system of equations and inequalities:*

$$y(\delta(v))=1 \quad \textit{for all } v\in V, \tag{1}$$

$$y(E(W))\leqslant\tfrac{1}{2}(|W|-1) \quad \textit{for all } W\subseteq V, |W| \textit{ odd}, \tag{2}$$

$$y(e)\geqslant 0 \quad \textit{for all } e\in E. \qquad \square \tag{3}$$

$P(G)$ can also be described by (1) and (3) together with

$$y(\delta(W))\geqslant 1 \quad \text{for all } W\subseteq V,\ |W| \text{ odd.} \tag{4}$$

We refer to (2) (or (4)) as *odd set* constraints, or alternatively as *clique* constraints. Thus the perfect matching problem can be expressed as

$$\min\{b^{\mathrm{T}}y\colon y\in\mathbb{R}^m, y \text{ satisfies (1), (2), (3)}\},$$

or alternatively as

$$\min\{b^{\mathrm{T}}y\colon y\in\mathbb{R}^m, y \text{ satisfies (1), (4), (3)}\}.$$

This is the *optimization problem* associated with the perfect matching polytope $P(G)$. The *separation problem* for this polytope is:

Separation problem for $P(G)$. For a given $y\in\mathbb{R}^m$, decide whether $y\in P(G)$. If not, exhibit a hyperplane separating y from $P(G)$.

$P(G)$ consists of all points $u\in\mathbb{R}^m$ which satisfy (1), (3) and (4). It is trivial to verify whether a given point satisfies (1) and (3). If we regard the components of a point as capacities on the flow on the corresponding edge of the graph, then a given point $y \in \mathbb{R}^m$ satisfies (4) if and only if the graph has no odd cut of capacity less than one. Padberg and Rao [41] have used this observation to construct an algorithm which solves the separation problem for $P(G)$ by means of solving a sequence of max flow problems. This algorithm is a variant of the Gomory–Hu procedure for finding the minimum cut in a graph (see [21] or [32]) and it runs in $O(n^4)$ time in the worst case. It follows from [25] that the ellipsoid algorithm can then be used in conjunction with this algorithm to give a polynomial time algorithm for the perfect matching problem. This algorithm has a considerably higher worst case running time than Edmonds's algorithm.

The *matching polytope* $P_M(G)$ is defined to be the convex hull of incidence vectors of matchings on the graph G. It has been shown by Edmonds that $P_M(G)$ is given by the set of vectors $y \in \mathbb{R}^m$ which satisfy

$$\begin{aligned}
&y(\delta(v)) \leq 1 \quad \forall v \in V, \\
&y(E(W)) \leq \tfrac{1}{2}(|W|-1) \quad \forall W \subseteq V, |W| \text{ odd}, \\
&y_e \geq 0 \quad \forall e \in E.
\end{aligned}$$

Notice that the degree constraints are inequalities in the description of $P_M(G)$. There is a bijection between vertices of $P_M(G)$ and matchings of the graph G.

3.3. *An overview of the algorithm*

The initial relaxation is comprised of the degree constraints and nonnegativity constraints. Odd set constraints are added as cutting planes as necessary.

As described in Section 2, we refer to the current relaxation as the current *dual* linear programming problem, and we apply the primal projective standard-form variant of Karmarkar's algorithm to the corresponding primal problem. Therefore, cutting planes are added as constraints in the dual problem and as variables in the primal linear program.

It is easy to find an initial dual solution by using heuristics to find a good perfect matching. We attempt to improve upon the best perfect matching found to date by rounding any points which are dual feasible.

When using the simplex algorithm, the current relaxation is solved to optimality and then cutting planes are added. By contrast we try to add cutting planes before solving the current relaxation, so the initial iterate for the next relaxation is not an extreme point. This should aid the solution of the next relaxation. It is possible to attempt to separate any non-integral dual feasible solution (any integral solution is the incidence vector of a perfect matching). The initial dual solution for each relaxation is the incidence vector of the best perfect matching found to date and dual feasibility is always maintained. (In fact, the algorithm is monotonic in the dual, though not strictly monotonic.) As soon as a better dual feasible solution is found, the separation routines can be invoked.

A simplified version of our algorithm is as follows:

- (Initialization). Set up initial relaxation.
- (Loop)

  1. Update the solution to the current relaxation. If a non-integral dual solution is obtained, decide whether to attempt to round it and/or separate it.
  2. If cutting planes are found, decide which (if any) to add to the relaxation. If cutting planes are added, update the primal solution to obtain a strictly feasible point and update the dual solution to the incidence vector of the best perfect matching found to date.
  3. If termination criterion is met, STOP.

The performance of this algorithm can be considerably enhanced by various modifications. The algorithm is discussed in considerably more detail in Section 3.5.

3.4. *An initial solution*

The degree constraints are equality constraints in the perfect matching problem, so the primal variables corresponding to them are unrestricted in sign. Therefore, we could solve the relaxations using the method described in Mitchell and Todd [39]. This method requires the solution at each iteration of a least squares problem where the constraining matrix is stored implicitly using a sparse format, but is itself dense. Such problems can be solved efficiently using a preconditioned conjugate gradient method. However, we use a sparse QR factorization in order to solve least squares problems, and such a method is not well suited to solving problems where the constraining matrix is stored in the form indicated.

Therefore, we use a different approach. We relax the degree constraints so that we solve a matching problem rather than a perfect matching problem. We adjust the weight function b to ensure that the optimal solution is the optimal perfect matching.

Thus, our initial relaxation has the form

$$(\mathrm{D}^0) \qquad \begin{array}{ll} \max & z \\ \text{s.t.} & Sy \leqslant e, \\ & y \geqslant 0, \\ & (b - Be)^{\mathrm{T}} y + z \leqslant 0. \end{array}$$

Here, S denotes the vertex-edge incidence matrix for the graph, with columns only for those edges that are in the relaxation. It follows that S has exactly two nonzero entries in each column, with each nonzero entry having value one. The vector b contains the edge weights, with B being a positive real number which is large enough to ensure that if integrality constraints on y were added to (D^0), the optimal solution would be a *perfect* matching. The vector y is associated with the edges of the graph, with one element for each edge in the relaxation. We refer to (D^0) as the inital *dual* relaxation. The solution of (D^0) consists of matched edges with value one and circuits with value 0.5 on each edge (see [9]).

The initial *primal* relaxation is

$$(\mathrm{P}^0) \qquad \begin{array}{ll} \min & e^{\mathrm{T}} x_v \\ \text{s.t.} & S^{\mathrm{T}} x_v - x_e + (b - Be) x_0 = 0, \\ & x_0 = 1, \\ & x_v \geq 0, \quad x_e \geq 0, \quad x_0 \geq 0. \end{array}$$

The primal variables x_v correspond to the vertices of the graph, with one element for each vertex. The surplus variables x_e correspond to the edges of the graph, with one element for each edge in the relaxation. Then a strictly positive, feasible solution to (P^0) is

$$x_v = \tfrac{1}{2}B \quad \forall v \in V, \qquad x_e = b_e \quad \forall e \in E, \quad \text{and} \quad x_0 = 1.$$

3.5. *The algorithm in detail*

As we proceed, we modify (D^0) by adding cutting planes of the form (2). There are an exponential number of cutting planes of this type, so we cannot add them all if our algorithm is to remain practical. However, we want to add a large enough number so that the optimal solution to our modified linear program is the optimal solution to the problem $\max\{\bar{b}^{\mathrm{T}} t \colon y \in P_M(G)\}$, where $\bar{b} := Be - b$.

Our algorithm is outlined in Figure 2. Preprocessing is done in boxes 1 and 2. The iterations of the interior point algorithm are performed in the loop consisting of boxes 3, 4, 5, 12, and 9. After updating the dual solution, we decide whether to invoke the separation routines. If we do, we update the relaxation and the primal solution appropriately. This is described in boxes 5, 6, 7, 8, and 12. Because of the preprocessing, we have to do some postprocessing, as described in boxes 10, 11, and 12. This postprocessing may lead us back into the heart of the algorithm. We now describe how the algorithm works in detail.

Box 1. In order to save computational time, our initial linear programming relaxation only involves a subset $\bar{E} \subseteq E$ of the edges.

For each vertex $v \in V$, find the NN shortest edges (with respect to the objective function) in the set $\delta(v)$. Then $\bar{E}$ is the union of these subsets. (In order to ensure that there exists a perfect matching in the modified graph, we may modify $\bar{E}$ in Box 2.) We have $1 \leq NN \leq n-1$, and it appears from the work of both Grötschel and Holland and ourselves that the best choice for NN for problems of the size and type we were solving lies in the range $5 \leq NN \leq 10$.

Box 2. We use a greedy procedure to find a perfect matching M. If $M \nsubseteq \bar{E}$, we modify $\bar{E} \leftarrow \bar{E} \cup M$. Let y^F be the incidence vector of M. This gives us an upper bound $b^{\mathrm{T}} y^F$ on the value of $\min\{b^{\mathrm{T}} y \colon y \in P(G)\}$. We then set B to be this upper bound and we define the modified objective function $\bar{b}$ by

$$\bar{b}_e := B - b_e.$$

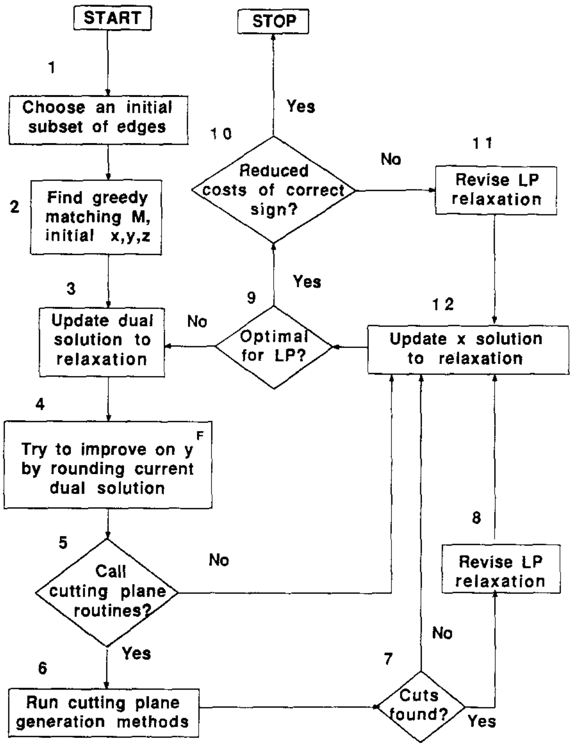
START
1
Choose an initial subset of edges
2
Find greedy matching M, initial x,y,z
3
Update dual solution to relaxation
4
Try to improve on y^F by rounding current dual solution
5
Call cutting plane routines?
No
Yes
6
Run cutting plane generation methods
7
Cuts found?
No
Yes
8
Revise LP relaxation
12
Update x solution to relaxation
9
Optimal for LP?
No
Yes
10
Reduced costs of correct sign?
Yes
No
STOP
11
Revise LP relaxation

Fig. 2. Algorithm for the perfect matching problem.

Then $z^F := \bar{b}^T y^F = \frac{1}{2}B(n-2)$ is a lower bound on the optimal value of

$$\max\{z: z \leq \bar{b}^T y, y \in P_M(G)\}$$

and also on any relaxation of this linear program. (It was shown in Mitchell [37] that this choice of B is large enough to ensure that the optimal solution to the given matching problem is the optimal solution to our original perfect matching problem.)

M is actually the best of several matchings which we inspect. To find each matching requires several steps:

```
1. Choose an ordering $\sigma$ of the vertices $V$.
2. (Label all vertices unmatched.) Set $M \leftarrow \emptyset$ and $i \leftarrow 1$.
3. (Find a greedy matching.) While $i \leq n$
    • If $i$ unmatched then
        - Find $j$ with $b(ij) = \min\{b(ij'): j'$ unmatched$\}$.
        - Add $ij$ to $M$ and label $i$ and $j$ matched.
    • Set $i \leftarrow i+1$.
4. (Use 2-opt (see [31] or [42]) to locally optimize the matching.) While $\exists i, j, k, l \in V$ with $ij, kl \in M$ and $b(ij)+b(kl) > b(ik)+b(jl)$
    • Set $M \leftarrow M \cup (ik \cup jl) \setminus (ij \cup kl)$.
```

In our tests we used two orderings — placing the vertices in numerical order and in reverse numerical order.

At this stage, we also initialize the primal and dual solutions. The initial dual solution is $y \leftarrow y^F$, $z \leftarrow z^F$. The initial primal solution x is defined as described in Section 3.4.

Box 3. We use the Todd–Burrell procedure [45] to update the current dual solution (y, z).

Box 4. If $y \neq y^F$ then we round y to find a perfect matching $\bar{M}$ whose value we compare with z^F. Let $y(\bar{M})$ be the incidence vector of $\bar{M}$. Set $z^{\bar{M}} := \bar{b}^T y(\bar{M})$. If $z^{\bar{M}} > z^F$ then we update y^F to $y(\bar{M})$ and z^F to $z^{\bar{M}}$. *If* $z^{\bar{M}} > z$ then we set $y = y(\bar{M})$ and $z = z^{\bar{M}}$.

We round y as follows.

```
1. Initialize $\bar{M} \leftarrow \emptyset$.
2. For $e \in \bar{E}$
    • If $y_e > 0.5$ update $\bar{M} \leftarrow \bar{M} \cup e$.
3. If $|\bar{M}| < \frac{1}{2}n$ then
    • Pair off unmatched vertices and add resulting edge(s) to $\bar{M}$.
4. While $\exists i, j, k, l \in V$ with $ij, kl \in \bar{M}$ and $b(ij) + b(kl) > b(ik) + b(jl)$
    • Set $\bar{M} \leftarrow \bar{M} \cup (ik \cup jl) \setminus (ij \cup kl)$.
```

Notice that the degree constraints imply that $\bar{M}$ obtained in step 2 is a matching, though not necessarily a perfect matching. If it is necessary to pair off vertices in step 3, it is unlikely that the resulting matching will improve upon the current best matching; therefore, we chose to pair off vertices randomly in step 3.

Box 5. We call the cutting plane routines if $y \neq y^F$, provided Phase I has been completed.

Box 6. Our separation routines are based on those of Grötschel and Holland [24]. To be computationally efficient, it is vital for us to reduce the number of stages of adding constraints. For that reason our heuristics are more extensive than those of Grötschel and Holland. Our hope is that by spending more time searching for violated constraints we will find more such constraints and save iterations later on. This hope has been borne out by our computational experience.

We choose k values $\lambda^1, \ldots, \lambda^k$. (Our standard choice is to take $k = 7$ and use the values 0.01, 0.06, 0.11, 0.16, 0.21, 0.26, and 0.31. All of the results we report used this choice unless otherwise stated.) For $1 \leq i \leq k$ define $E^i := \{e \in \bar{E}: y(e) > \lambda^i\}$. We then use depth first search to find the components of the graph (V, E^i). If any of the odd components W violates the corresponding odd set constraint, add the constraint of the form (2) to the formulation (provided the component was not a component of the graph (V, E^j) for some $j < i$). However, if $|W| > |V \setminus W|$ then we add the constraint corresponding to $V \setminus W$ rather than that corresponding to W, because this constraint is likely to be more sparse and the two constraints are equivalent.

Note that there is no point in setting $\lambda^i = 0$ for some i because Karmarkar's algorithm is an interior point method (and usually, except for one constraint, in the dual), so the components of (V, E^i) will nearly always be the same as the components of $G = (V, \bar{E})$.

We have not implemented the Padberg–Rao procedure, which is guaranteed to find a violated constraint if one exists. Therefore, it is possible that we will be unable to separate y from $P_M(G)$ although $y \notin P_M(G)$. However, it appears that the heuristic routines are sufficiently good that this rarely happens.

Box 7. If cutting planes were found in Box 6 then we go to Box 8; otherwise we go to Box 12.

Box 8. We update our relaxation by adding the constraints found in Box 6. The dual point (y, z) is not feasible after we add the cutting planes, so we update z to z^F and y to y^F.

It is possible to develop criteria to decide when to drop primal variables. For details, see Mitchell [37]. In this algorithm, we never drop any primal variables.

Box 9. We check whether the duality gap is sufficiently small (see Section 6.3). If it is then we move to Box 10 and check reduced costs; if it is not then we return to Box 3.

Box 10. Our initial formulation does not use all the edges of the graph. Therefore, it is necessary to check the reduced costs of these edges in order to confirm optimality of the solution. If some edge has negative reduced cost then we move to Box 11. Otherwise we STOP. It should be noted that it is not guaranteed that our final solution is a matching because we do not use the Padberg–Rao procedure. Define

$$Y := \{y \in \mathbb{R}^m \colon y \text{ satisfies (1) and (3), } y \text{ is not separable from } P_M(G) \text{ using the separation routines of Box 6}\}.$$

Define $P_Y(G) := \operatorname{conv}\{y \in Y\}$. Then our final solution is an optimal solution to the problem $\max\{\bar{b}^{\mathrm{T}}y \colon y \in \bar{P}(G)\}$ for some polytope $\bar{P}(G)$ with $P_M(G) \subseteq \bar{P}(G) \subseteq P_Y(G)$.

Box 11. We choose some subset of the edges with negative reduced cost and add these variables to our formulation. We describe how to do this in Section 5.4, with surplus variables being introduced in each additional primal constraint and an artificial variable with large cost being added in the primal relaxation. If more than fifty edges have negative reduced cost, the edges are bucket sorted by violation, and the (approximately) fifty most violated constraints added. In the resulting Phase I problem, all the additional primal surplus variables have the value of the appropriate reduced cost, with the artificial column being the sum of the vector of ones and the vector of reduced costs. After revising our linear program, we move to Box 12. (Note that all the dual variables are bounded by one, so if we added this constraint to the dual linear program, we could immediately find an interior primal feasible point, as described in Section 5.2. We intend to investigate this option at a later date; for the current paper, we were interested in the performance of the procedure described in Section 5.4.)

Box 12. The primal solution x is updated.

If we came to this box from either Box 5 or Box 7 or Box 11, we have an interior primal feasible point and we use the projective standard-form variant of Karmarkar's algorithm to calculate a new point.

If we entered this box from Box 8, the current primal feasible point is not an interior point. A procedure to find an interior primal feasible point is described in Section 4.5. We set the vector of relative weights for the added variables to be e, unless otherwise stated. The step length we chose is 0.75 of the maximum possible step length.

4. Obtaining an interior point when adding cutting planes

4.1. Introduction

We assume that the linear program

$$\text{(D)}\qquad \begin{array}{ll} \max & z \\ \text{s.t.} & A^{\mathrm{T}}y + zg \leqslant c, \end{array}$$

is the current linear programming relaxation of the combinatorial problem that we are solving. The dual of this linear program is

$$\text{(P)}\qquad \begin{array}{ll} \min & c^{\mathrm{T}}x \\ \text{s.t.} & Ax = 0, \\ & g^{\mathrm{T}}x = 1, \\ & x \geqslant 0. \end{array}$$

We wish to modify (D) by adding the constraint

$$a_0^{\mathrm{T}}y + zg_0 \leqslant c_0, \tag{5}$$

where a_0 is an m-vector and c_0 and g_0 are scalars. Adding this constraint corresponds to adding a variable x_0 to (P), giving the problem

$$(\text{P}_0)\qquad \begin{array}{ll} \min & c^{\mathrm{T}}x + c_0x_0 \\ \text{s.t.} & Ax + a_0x_0 = 0, \\ & g^{\mathrm{T}}x + g_0x_0 = 1, \\ & x \geqslant 0, \quad x_0 \geqslant 0. \end{array}$$

We assume that we have an interior feasible point $\hat{x}$ for the problem (P). Then the point

$$x = \hat{x}, \quad x_0 = 0,$$

is feasible for (P_0), but it is clearly not an interior point. We also assume that g and g_0 are nonnegative.

In this section, we give two methods by which an interior point may be obtained, and we show that these methods are essentially equivalent. One method resembles a Phase I procedure in that it introduces an artificial variable. The other method calculates an appropriate direction more directly. For consideration of other directions, the reader is referred to Mitchell [37].

Before describing the two procedures in more detail, we define some notation and make an observation. We define $\hat{X}$ to be the $n \times n$ diagaonal matrix whose diagonal elements are the entries of $\hat{x}$. Then e, the vector of ones, is feasible for the scaled problem

$$(\tilde{\mathrm{P}}) \qquad \begin{aligned} \min \quad & \tilde{c}^{\mathrm{T}}\tilde{x} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = 0, \\ & \tilde{g}^{\mathrm{T}}\tilde{x} = 1, \\ & \tilde{x} \geqslant 0, \end{aligned}$$

where $\tilde{c} := \hat{X}c$, $\tilde{A} := A\hat{X}$ and $\tilde{g} := \hat{X}g$. The corresponding scaled version of (P_0) is

$$(\tilde{\mathrm{P}}_0) \qquad \begin{aligned} \min \quad & \tilde{c}^{\mathrm{T}}\tilde{x} + c_0x_0 \\ \text{s.t.} \quad & \tilde{A}\tilde{x} + a_0x_0 = 0, \\ & \tilde{g}^{\mathrm{T}}\tilde{x} + g_0x_0 = 1, \\ & \tilde{x} \geqslant 0, \quad x_0 \geqslant 0, \end{aligned}$$

and $\tilde{x} = e$, $x_0 = 0$ is feasible for $(\tilde{\mathrm{P}}_0)$.

4.2. *A Phase I procedure*

We first describe the Phase I procedure. We define the linear program

$$(\tilde{\mathrm{P}}_{-1}) \qquad \begin{aligned} \min \quad & x_{-1} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} + a_0x_0 - a_0x_{-1} = 0, \\ & \tilde{g}^{\mathrm{T}}\tilde{x} + g_0x_0 - g_0x_{-1} = 1, \\ & \tilde{x} \geqslant 0, \quad x_0 \geqslant 0, \quad x_{-1} \geqslant 0. \end{aligned}$$

It should be noted that $\tilde{x} = e$, $x_0 = 1$, $x_{-1} = 1$ is a strictly positive feasible solution to the problem $(\tilde{\mathrm{P}}_{-1})$, and $\tilde{x} = e$, $x_0 = 0$, $x_{-1} = 0$ is an optimal solution. Indeed, any optimal solution to $(\tilde{\mathrm{P}}_{-1})$ leads naturally to a feasible solution to (P_0). We propose to solve (P_0) using Karmarkar's algorithm for linear programming. Therefore we consider the direction

$$P_{[\tilde{A}\; a_0\; -a_0]}\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} =: \begin{pmatrix} d \\ d_0 \\ d_{-1} \end{pmatrix},$$

where P_M denotes projection onto the null space of the matrix M.

In general a Phase I procedure can take several iterations; however, in this case, the direction (d, d_0, d_{-1}) defined above is such that the Phase I procedure can be completed in one iteration with a suitable choice of step length. In addition, a strictly positive feasible point for (P_0) is returned, provided $\hat{x}$ is strictly positive. In order to show this we first prove several lemmas. The first lemma gives an explicit expression for the direction (d, d_0, d_{-1}).

Lemma 1. *The direction* (d, d_0, d_{-1}) *given above is proportional to*

$$-\begin{pmatrix} \tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 \\ \theta \\ 1+\theta \end{pmatrix},$$

where

$$\theta := a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0. \tag{6}$$

Proof. We use the Sherman–Morrison–Woodbury formula (see, e.g., [20]) to calculate

$$\begin{aligned}
\begin{pmatrix} d \\ d_0 \\ d_{-1} \end{pmatrix} &= P_{[\tilde{A}\; a_0\; -a_0]}\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} - \begin{bmatrix} \tilde{A}^{\mathrm{T}} \\ a_0^{\mathrm{T}} \\ -a_0^{\mathrm{T}} \end{bmatrix}[\tilde{A}\tilde{A}^{\mathrm{T}}+2a_0a_0^{\mathrm{T}}]^{-1}[\tilde{A}\; a_0\; -a_0]\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} - \begin{bmatrix} \tilde{A}^{\mathrm{T}} \\ a_0^{\mathrm{T}} \\ -a_0^{\mathrm{T}} \end{bmatrix}[(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1} - \frac{2}{1+2\theta}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}]a_0 \\
&= \frac{-1}{1+2\theta}\begin{pmatrix} \tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 \\ \theta \\ 1+\theta \end{pmatrix}. \qquad \square
\end{aligned}$$

Notice that $\theta > 0$ so we can drop the scaling term $1/(1+2\theta)$ and redefine the direction to be

$$\begin{pmatrix} d \\ d_0 \\ d_{-1} \end{pmatrix} := -\begin{pmatrix} \tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 \\ \theta \\ 1+\theta \end{pmatrix}. \tag{7}$$

The following lemma discusses the relative sizes of the components of the direction (d, d_0, d_{-1}).

Lemma 2. *The smallest component of the direction defined above is* d_{-1}, *and it is strictly smaller than any other component. In addition,* d_{-1} *is negative.*

Proof. Since $\theta > 0$ it is clear that $d_{-1} < d_0$ and $d_{-1} < 0$. In addition $d^{\mathrm{T}}d = \theta$ so every component d_i, $1 \leqslant i \leqslant n$, of d satisfies $d_i \geqslant -\sqrt{\theta}$. We have two cases depending on the size of θ:

1. If $\theta \leqslant 1$ then $d_i \geqslant -1 > d_{-1}$, $1 \leqslant i \leqslant n$.
2. If $\theta > 1$ then $d_i \geqslant -\theta > d_{-1}$, $1 \leqslant i \leqslant n$.

Hence d_{-1} is the smallest component of the direction defined above. $\square$

Lemma 3. *The largest step length α such that the point*

$$\begin{pmatrix} \tilde{x} \\ x_0(\alpha) \\ x_{-1}(\alpha) \end{pmatrix} := \begin{pmatrix} e \\ 1 \\ 1 \end{pmatrix} + \alpha \begin{pmatrix} d \\ d_0 \\ d_{-1} \end{pmatrix}$$

is nonnegative is $\tilde{\alpha} := -1/d_{-1}$. Then $x_{-1}(\tilde{\alpha}) = 0$, $x_0(\tilde{\alpha}) > 0$ and $\tilde{x}(\tilde{\alpha}) > 0$.

Proof. This follows directly from the previous lemma. □

The standard Karmarkar step, as defined in, for example, the work of Gonzaga [22], involves moving in the direction which is the negative of the projection of the (scaled) objective function onto the null space of the (scaled) matrix of subspace constraints and then normalizing to satisfy the normalizing constraint. In this case, the projected direction is the direction (d, d_0, d_{-1}) defined above.

Theorem 2. *The problem $(\tilde{\mathrm{P}}_{-1})$ can be solved in one step using Karmarkar's algorithm by moving an appropriate amount in the projected direction and then normalizing to satisfy the normalizing constraint. Moreover, the resulting solution gives a strictly positive feasible solution to the problem $(\tilde{\mathrm{P}}_0)$ and thus a strictly positive feasible solution to (P_0)*

Proof. The step length $\tilde{\alpha}$ defined in the previous lemma gives a strictly positive point $(\tilde{x}(\tilde{\alpha}), x_0(\tilde{\alpha}))$ which is feasible in the subspace constraints of the problem $(\tilde{\mathrm{P}}_0)$. Define $\xi := \tilde{g}^{\mathrm{T}}\tilde{x}(\tilde{\alpha}) + g_0 x_0(\tilde{\alpha})$. It should be noted that $\xi > 0$ since $\tilde{x}(\tilde{\alpha}) > 0$, $x_0(\tilde{\alpha}) > 0$ and (g, g_0) is nonnegative and not identically zero. Then $(1/\xi)(\tilde{x}(\tilde{\alpha}), x_0(\tilde{\alpha}))$ is strictly positive feasible point in $(\tilde{\mathrm{P}}_0)$ and $(1/\xi)(\hat{X}\tilde{x}(\tilde{\alpha}), x_0(\tilde{\alpha}))$ is a strictly positive point which is feasible in (P_0). □

Now

$$x_0(\tilde{\alpha})) = 1 - \frac{d_0}{d_{-1}} = \frac{1}{1+\theta}, \tag{8}$$

and

$$\tilde{x}(\tilde{\alpha}) = e - \frac{1}{d_{-1}} d = e - \frac{1}{1+\theta} \tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1} a_0. \tag{9}$$

Therefore, when using the Phase I procedure detailed above the following strictly positive feasible point for (P_0) is obtained:

$$x_0 = \psi, \tag{10}$$

$$x = (1+\theta)\psi\hat{x} - \psi\hat{X}^2 A(A\hat{X}^2 A^{\mathrm{T}})^{-1} a_0, \tag{11}$$

where

$$\psi := \frac{1}{1 + \theta + g_0 - g^{\mathrm{T}}\hat{X}^2 A(A\hat{X}^2 A^{\mathrm{T}})^{-1} a_0}, \tag{12}$$

and $\theta = a_0^{\mathrm{T}}(A\hat{X}^2 A^{\mathrm{T}})^{-1} a_0$, as in equation (6).

4.3. Working directly in $(\tilde{\mathrm{P}}_0)$

We now describe a more direct method of obtaining a direction in the problem (P_0). Moving in this direction from the point $x=\hat{x}$, $x_0=0$ gives a strictly positive feasible point for (P_0). In order to derive this direction we first define the direction

$$\begin{pmatrix} d \\ d_0 \end{pmatrix} := P_{[\tilde{A}\, a_0]}\begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{13}$$

We are going to consider moving in this direction from the point $\tilde{x}=e$, $x_0=0$ in the problem $(\tilde{\mathrm{P}}_0)$.

As mentioned in Section 3, the speed of convergence of Karmarkar's algorithm is adversely affected if the sequence of iterates approaches a nonoptimal vertex too closely. For that reason, it is important to bring the added variable x_0 in at as large a value as possible without greatly increasing the objective function value. In addition, from considerations of complementary slackness, it is probable that the larger the value of x_0 the smaller the slack in the corresponding dual constraint when the dual solution is updated. This is desirable because it is likely that the added dual constraint will hold at equality at the optimal solution to the current relaxation.

The direction (d, d_0) is the direction of steepest ascent for the problem

$$\begin{aligned} \max\quad & x_0 \\ \text{s.t.}\quad & \tilde{A}\tilde{x}+a_0x_0=0, \\ & \tilde{x}\geqslant 0,\quad x_0\geqslant 0. \end{aligned}$$

We work in the scaled problem because in this problem the initial point is $\tilde{x}=e$, so a step of length at least one can be taken without violating the nonnegativity constraints.

Lemma 4. *The direction* (d, d_0) *is proportional to*

$$\begin{pmatrix} -\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 \\ 1 \end{pmatrix},$$

and $d_0>0$.

Proof. This lemma is proved by direct calculation, using the Sherman–Morrison–Woodbury formula (see, e.g., [20]). Recall $\theta=a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0$ from (6). We have

$$\begin{aligned} P_{[\tilde{A}\, a_0]}\begin{pmatrix} 0 \\ 1 \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} \tilde{A}^{\mathrm{T}} \\ a_0^{\mathrm{T}} \end{pmatrix}[\tilde{A}\tilde{A}^{\mathrm{T}}+a_0a_0^{\mathrm{T}}]^{-1}\mathrm{a}_0 \\ &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} \tilde{A}^{\mathrm{T}} \\ a_0^{\mathrm{T}} \end{pmatrix}[(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}-\frac{1}{1+\theta}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}]a_0 \\ &= \frac{1}{1+\theta}\begin{pmatrix} -\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 \\ 1 \end{pmatrix}. \end{aligned}$$

Thus $d_0=1/(1+\theta)$, which is clearly positive. □

Thus we are led to consider the set of points

$$\tilde{\mathscr{X}} := \{(\tilde{x}(\alpha), x_0(\alpha)) \colon \tilde{x}(\alpha) = e - \alpha\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0, x_0(\alpha) = \alpha, \alpha > 0\}. \tag{14}$$

Every point in this set satisfies the subspace constraints for the problem $(\tilde{\mathrm{P}}_0)$.

Define

$$\alpha_{\max} := \frac{-1}{\min_i\{\tilde{a}_i^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0\}}, \tag{15}$$

where $\tilde{a}_i$ denotes the ith column of $\tilde{A}$.

Lemma 5. *$\alpha_{\max}$ is well-defined and positive.*

Proof. Recall that e is in the null space of $\tilde{A}$ so

$$e^{\mathrm{T}}\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0 = 0.$$

Also, a_0 is not identically zero so $\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0$ is not identically zero. Therefore $\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0$ has a negative component. $\square$

Lemma 6. *The point $(\tilde{x}(\alpha), x_0(\alpha))$ is strictly positive and satisfies the subspace constraints for the problem $(\tilde{\mathrm{P}}_0)$ if and only if $0 < \alpha < \alpha_{\max}$.*

Proof. The given point satisfies the subspace constraints by definition. The value of $x_0(\alpha)$ is positive if and only if $0 < \alpha$. From the definition of $\alpha_{\max}$, for $\alpha > 0$, $\tilde{x}(\alpha)$ is positive if and only if $\alpha < \alpha_{\max}$. $\square$

Lemma 7. *The value of $\alpha_{\max}$ is bounded below by $\theta^{-1/2}$ and it is strictly bounded below by $(1+\theta)^{-1}$. (Recall, from* (6), *$\theta = a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0$.)*

Proof. This follows from the fact that the 2-norm of the vector $\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}a_0$ is the square root of θ and that $\sqrt{w} < 1 + w$ for any real positive w. $\square$

Define

$$\xi(\alpha) := \tilde{g}^{\mathrm{T}}\tilde{x}(\alpha) + g_0x_0(\alpha). \tag{16}$$

Then the point $((1/\xi(\alpha))\hat{X}\tilde{x}(\alpha), (1/\xi(\alpha))x_0(\alpha))$ is feasible for the problem (P_0) provided $0 \leqslant \alpha \leqslant \alpha_{\max}$, and it is an interior point if the inequalities hold strictly. We denote this point by $(x(\alpha), x_0(\alpha))$. Then

$$x_0(\alpha) = \psi, \tag{17}$$

$$x(\alpha) = \psi\alpha^{-1}\hat{x} - \psi\hat{X}^2A^{\mathrm{T}}(A\hat{X}^2A^{\mathrm{T}})^{-1}a_0, \tag{18}$$

where

$$\psi := \frac{1}{\alpha^{-1} + g_0 - g^{\mathrm{T}}\hat{X}^2A^{\mathrm{T}}(A\hat{X}^2A^{\mathrm{T}})^{-1}a_0}. \tag{19}$$

Theorem 3. *There exists a step length $\tilde{\alpha}$ for which the point $(x(\tilde{\alpha}), x_0(\tilde{\alpha}))$ is the point found using the Phase* I *approach defined in Section* 4.2.

Proof. Define $\tilde{\alpha} := (1+\theta)^{-1}$. Then $(x(\tilde{\alpha}), x_0(\tilde{\alpha}))$ is exactly the point obtained using the Phase I method. See equations (10)-(12). □

In general we do not want to be limited to choosing $\alpha = \tilde{\alpha}$. We can either choose to go a certain fraction of the maximum possible step $\alpha_{\max}$ or we can use a line search on a potential function similar to Karmarkar's potential function. Care is needed if this second approach is used because the potential function is not defined on the boundary of the feasible region, so in particular it is not defined at $(e, 0)$, the inital point of the step.

4.4. *The objective function value*

We consider here the change in the primal objective function value when adding a dual constraint. When moving in the direction found above, we obtain a point in the set $\tilde{\mathscr{X}}$, defined in equation (14).

Lemma 8. *Assume the dual values y and z have been updated in the standard way given in Todd and Burrell* [45] *and that (y, z) violates the added constraint defined in equation* (5). *Then every point in $\tilde{\mathscr{X}}$ has smaller value with respect to the objective function $(\tilde{c} - z\tilde{g}, c_0 - zg_0)$ than the point* $(e, 0)$.

Proof. By assumption we have

$$y = (\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\tilde{A}(\tilde{c} - z\tilde{g}),$$

and so for $(\tilde{x}(\alpha), x_0(\alpha)) \in \tilde{\mathscr{X}}$ we have

$$\begin{aligned}(\tilde{c} - z\tilde{g})^{\mathrm{T}}\tilde{x}(\alpha) + (c_0 - zg_0)x_0(\alpha) - (\tilde{c} - z\tilde{g})^{\mathrm{T}}e \\ = \alpha[c_0 - zg_0 - a_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\tilde{A}(\tilde{c} - z\tilde{g})] \\ = \alpha[c_0 - zg_0 - a_0^{\mathrm{T}}y].\end{aligned}$$

Since y violates the added constraint, this quantity must be negative. □

4.5. *Adding many constraints*

We now consider adding many constraints to the problem (P). Thus we modify (D) by adding the p constraints

$$A_0^{\mathrm{T}}y + zg_0 \leq c_0, \tag{20}$$

where $A_0 \in \mathbb{R}^{m\times p}$ and c_0 and g_0 are both real p-vectors, g_0 being nonnegative. Adding these constraints corresponds to adding the variables x_0 to (P), giving the problem

$$\begin{aligned}
(\mathrm{P}_{00}) \quad & \min \quad c^{\mathrm{T}}x + c_0^{\mathrm{T}}x_0 \\
& \text{s.t.} \quad Ax + A_0x_0 = 0, \\
& \qquad\quad g^{\mathrm{T}}x + g_0^{\mathrm{T}}x_0 = 1, \\
& \qquad\quad x \geq 0, \quad x_0 \geq 0,
\end{aligned}$$

where x_0 is now a p-vector. By analogy with Section 4.3, we condier the direction

$$\begin{pmatrix} d \\ d_{00} \end{pmatrix} := P_{[\tilde{A}\,A_0]}\begin{pmatrix} 0 \\ e \end{pmatrix}. \tag{21}$$

Lemma 9. *The direction d_{00} is given by $(I+\Theta)^{-1}e$, where we define*

$$\Theta := A_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}A_0. \tag{22}$$

Proof. This can be shown by direct calculation using the Sherman-Morrison-Woodbury formula (see, e.g., [20]). □

A drawback to finding an interior feasible point for (P_{00}) using the direction defined in (21) is that it is not guaranteed that the direction d_{00} is strictly positive. Since d_{00} solves the system

$$(I+\Theta)d_{00} = e,$$

standard error analysis shows that d_{00} will be close to e (and thus positive) if the norm of the matrix Θ defined in (22) is sufficiently small. The scaling of the columns of A_0 is arbitrary, so it is possible to scale them so that the norm of $A_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}A_0$ is small and $d_{00} > 0$. However, such an approach is liable to produce badly scaled constraint matrices and for that reason we develop a different approach.

This method involves replacing the extra variables with one variable. We then use the methods defined above for dealing with problems with one extra variable. Therefore, we define

$$\breve{a}_0 := A_0 w, \tag{23}$$

$$\breve{c}_0 := c_0^{\mathrm{T}} w, \tag{24}$$

$$\breve{g}_0 := g_0^{\mathrm{T}} w, \tag{25}$$

where w is a strictly positive p-vector, and consider the linear program

$$\begin{aligned}
(\breve{\mathrm{P}}_{00}) \quad & \min \quad \tilde{c}^{\mathrm{T}}\tilde{x} + \breve{c}_0\breve{x}_0 \\
& \text{s.t.} \quad \tilde{A}\tilde{x} + \breve{a}_0\breve{x}_0 = 0, \\
& \qquad\quad \tilde{g}^{\mathrm{T}}\tilde{x} + \breve{g}_0\breve{x}_0 = 1, \\
& \qquad\quad \tilde{x} \geq 0, \quad \breve{x}_0 \geq 0.
\end{aligned}$$

Here $\check{x}_0$, $\check{c}_0$ and $\check{g}_0$ are scalars, $\check{g}_0$ being nonnegative, and $\check{a}_0$ is an m-vector. Notice that $\tilde{x}=e$, $\check{x}_0=0$ is feasible for the problem $(\check{\mathrm{P}}_{00})$.

The simplest choice for w is to set it equal to e. We have also experimented with two other choices for w, namely, setting the ith component of w to be the amount by which the current solution y violates the ith added constraint, and setting the ith component of w to be the reciprocal of the 2-norm of the normal to the ith added constraint.

Lemma 10. *If* $(\tilde{x}, \check{x}_0)$ *is feasible for* $(\check{\mathrm{P}}_{00})$ *then* $(\hat{X}\tilde{x}, \check{x}_0 w)$ *is feasible for* (P_{00}) *and has the same objective function value.*

Proof. This lemma is proved by substitution. Notice that $\check{x}_0 A_0 w=\check{x}_0\check{a}_0$ and similarly $\check{x}_0 g_0^{\mathrm{T}} w=\check{x}_0\check{g}_0$ and $\check{x}_0 c_0^{\mathrm{T}} w=\check{x}_0\check{c}_0$. □

It follows from this lemma that if $(\tilde{x}, \check{x}_0)$ is a strictly positive feasible point for $(\check{\mathrm{P}}_{00})$ then $(\hat{X}\tilde{x}, \check{x}_0 w)$ is a strictly feasible point for (P_{00}) with the same objective function value. Thus we propose to find an interior feasible point for the problem (P_{00}) by first finding such a point for $(\check{\mathrm{P}}_{00})$ using the methodology of Section 4.3. Equations (17)-(19) imply that we obtain the following point, which is an interior feasible point for $(\check{\mathrm{P}}_{00})$ provided $0<\check{\alpha}<\check{\alpha}_{\max}$:

$$\check{x}_0(\alpha)=\psi,$$
$$\tilde{x}(\alpha)=\psi\alpha^{-1}e-\psi\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\check{a}_0.$$

where

$$\psi:=\frac{1}{\alpha^{-1}+\check{g}_0-\tilde{g}^{\mathrm{T}}\tilde{A}^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\check{a}_0},$$
$$\check{\alpha}_{\max}:=\frac{-1}{\min_i\{\tilde{a}_i^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\check{a}_0\}}. \tag{26}$$

Thus we obtain the interior feasible point

$$x_0(\alpha)=\psi w, \tag{27}$$
$$x(\alpha)=\psi\alpha^{-1}\hat{x}-\psi\hat{X}^2A^{\mathrm{T}}(A\hat{X}^2A^{\mathrm{T}})^{-1}A_0 w, \tag{28}$$

for the problem (P_{00}). Note that it follows from Lemma 7 that $\check{\alpha}_{\max}$ is bounded below by the reciprocal of $\sqrt{\check{a}_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}\check{a}_0}=\sqrt{w^{\mathrm{T}}A_0^{\mathrm{T}}(\tilde{A}\tilde{A}^{\mathrm{T}})^{-1}A_0 W}=\sqrt{w^{\mathrm{T}}\Theta w}$.

5. Adding variables

5.1. Introduction

We assume that the linear program

$$\begin{aligned}(\mathrm{D})\qquad &\max\quad z\\ &\text{s.t.}\quad A^{\mathrm{T}}y+zg\le c,\end{aligned}$$

is the current linear programming relaxation of the combinatorial problem that we are solving. The dual of this linear program is

$$\text{(P)} \quad \begin{aligned} \min \quad & c^{\mathrm{T}}x \\ \text{s.t.} \quad & Ax = 0, \\ & g^{\mathrm{T}}x = 1, \\ & x \geqslant 0. \end{aligned}$$

In this section we consider adding variables to the problem (D). The reason we develop the methodology to do this is as follows. We are interested in solving combinatorial optimization problems. For some of these problems, it is desirable to omit several variables from the initial formulation if the size of the relaxation is to remain manageable and if a cutting plane approach is to be efficient. For example, when solving perfect matching problems only a small subset of the edges appear in the initial formulation, because it is expected that edges connecting widely separated vertices are unlikely to appear in the optimal perfect matching. Using the cutting plane approach defined in Section 3, the relaxation of the combinatorial problem will have the same form as the problem (D). In order to show that the optimal solution to the relaxation is the optimal solution to the combinatorial problem, it is necessary to check that none of the omitted variables should appear in the relaxation. This check involves looking at the reduced costs of the omitted variables. If any reduced cost is of the wrong sign then it is necessary to add the corresponding variable to the relaxation, and the appropriate constraint to the dual of the relaxation. Therefore, for many problems, a cutting plane approach based on Karmarkar's algorithm can be competitive only if it has the ability to add variables to problems of the form (D).

In what follows, we assume that we have either a simple upper bound or a simple lower bound on each variable of the combinatorial problem. Thus, each possible extra constraint for the problem (P) is an inequality constraint.

We assume that the problem (D) is modified to the following problem when the variable y_0 is added to the relaxation:

$$(\mathrm{D}_0) \quad \begin{aligned} \max \quad & z \\ \text{s.t.} \quad & A^{\mathrm{T}}y + a_0 y_0 + zg \leqslant c, \\ & y_0 \leqslant u_0, \\ & -y_0 \leqslant -l_0. \end{aligned}$$

Here, a_0 is an n-vector, and y_0, u_0 and l_0 are scalars. Thus u_0 and l_0 are upper and lower bounds respectively on y_0. We do not require that both u_0 and l_0 be finite, merely that one of them is. We assume that $l_0 \leqslant 0 \leqslant u_0$, so if (y, z) is feasible for the problem (D) then setting $y_0 = 0$ gives a feasible point for (D_0).

The dual problem to (D_0) is

$$(P_0) \quad \begin{aligned} \min \quad & c^T x + u_0 x_u - l_0 x_l \\ \text{s.t.} \quad & Ax = 0, \\ & a_0^T x + x_u - x_l = 0. \\ & g^T x = 1, \\ & x \geq 0, \quad x_u \geq 0, \quad x_l \geq 0, \end{aligned}$$

where x_u and x_l are scalars.

Recall from Section 3 that we regard the current linear programming relaxation as solved when the duality gap is less than some threshold ε. We refer to a primal feasible point x and a dual feasible point (y, z) as an *ε-optimal pair of solutions* if $c^T x - z < \varepsilon$; we also say x is ε-optimal for (P) and (y, z) is ε-optimal for (D) in this case.

Assume that $\hat{x}$ and $(\hat{y}, \hat{z})$ constitute an ε-optimal pair for the dual pair of problems (P) and (D) and that $\hat{x}$ was obtained using an interior point method, so that it is strictly positive. Define $\eta := a_0^T \hat{x}$, the reduced cost of the variable y_0. We divide the analysis into four cases:

1. $\eta = 0$: Then $x = \hat{x}$, $x_u = x_l = 0$ and $y = \hat{y}$, $z = \hat{z}$, $y_0 = 0$ are an ε-optimal pair of solutions for (P_0) and (D_0).

2. $\eta \geq 0$ and $l_0 = 0$: Then $x = \hat{x}$, $x_u = 0$, $x_l = \eta$ and $y = \hat{y}$, $z = \hat{z}$, $y_0 = 0$ are an ε-optimal pair of solutions for (P_0) and (D_0).

3. $\eta \leq 0$ and $u_0 = 0$: Then $x = \hat{x}$, $x_u = -\eta$, $x_l = 0$ and $y = \hat{y}$, $z = \hat{z}$, $y_0 = 0$ are an ε-optimal pair of solutions for (P_0) and (D_0).

4. Otherwise: It is necessary to add the variable y_0 to (D) and the corresponding constraint to (P) and reoptimize.

From now on, we assume that we are in the last of these cases.

5.2. *Both bounds are finite*

If l_0 and u_0 are both finite consider the following procedure:

1. Set $x_l \leftarrow \begin{cases} \eta + \beta & \text{if } \eta \geq 0, \\ \beta & \text{otherwise.} \end{cases}$
2. Set $x_u \leftarrow x_l - \eta$.

This gives an interior feasible point for (P_0) for any positive β. Notice that one of x_u and x_l takes the value $|\eta| + \beta$, with the other taking the value β. The adjustment to the objective function value is

$$\begin{aligned} & \beta(u_0 - l_0) - \eta l_0 \quad \text{if } \eta \geq 0, \\ & \beta(u_0 - l_0) - \eta u_0 \quad \text{otherwise.} \end{aligned}$$

Thus the greater the value of β, the greater the increase in the objective function value. The drawback to choosing a small value for β is that a variable in the problem

(P_0) has that small value, so if this variable is nonzero at the optimal solution, Karmarkar's algorithm will take many iterations to converge.

If η is an arithmetically large positive (negative) number, it appears likely that x_u (x_l) will be zero at the optimal solution and therefore it is reasonable to choose a small value for β. If the absolute value of η is small then it is not clear which one of the variables x_u and x_l will be basic at the optimal solution and therefore a large value of β should be chosen. We suggest choosing β to be a constant, independent of the value of η. Notice that this choice satisfies the requirements for values of η which have either large or small absolute value: if η has small absolute value, x_u and x_l will have approximately the same value; if η has large absolute value, one of x_u and x_l will have approximately that value, with the other variable being small relative to it.

5.3. One bound is infinite

We assume that the upper bound on y_0 is infinite and the lower bound finite but nonpositive. Usually the lower bound is zero; however, we consider a more general situation. The procedure we develop can be modified straightforwardly for the case when the lower bound is infinite and the upper bound finite.

Thus we are interested in finding a strictly positive feasible point for the problem

$$
\begin{array}{lll}
(P_0^{\infty}) & \min & c^{T}x - l_0 x_l \\
& \text{s.t.} & Ax = 0, \\
& & a_0^{T}x - x_l = 0, \\
& & g^{T}x = 1, \\
& & x \geq 0, \quad x_l \geq 0.
\end{array}
$$

The point $\hat{x}$ is ε-optimal for (P) and η is defined to be $a_0^{T}\hat{x}$. If $\eta > 0$, setting $x_l = \eta$ gives an interior feasible point for (P_0^{∞}), and in fact this gives an ε-optimal solution to (P_0^{∞}) if $l_0 = 0$. Therefore, for the rest of this section we assume that $\eta < 0$. Consider the Phase I problem

$$
\begin{array}{lll}
(P_{-1}) & \min & x_{-1} \\
& \text{s.t.} & Ax = 0, \\
& & a_0^{T}x - x_l + (\beta - \eta)x_{-1} = 0, \\
& & g^{T}x = 1, \\
& & x \geq 0, \quad x_l \geq 0, \quad x_{-1} \geq 0,
\end{array}
$$

where β is some positive number. Notice that setting $x = \hat{x}$, $x_l = \beta$ and $x_{-1} = 1$ gives a strictly positive feasible point for (P_{-1}). This Phase I problem can be obtained as a scaling of the limiting version of the problem (P_0) as the upper bound goes to infinity.

If the optimal value to (P_{-1}) is greater than zero, the problem (P_0^∞) is infeasible and therefore the current relaxation of the original combinatorial problem is unbounded. If the optimal value to (P_{-1}) is zero, the optimal solution leads naturally to a feasible point for (P_0^∞); the question remains as to whether this point will be strictly positive. In practice, we use a big-M method, so we use the objective function

$$c^{\mathrm{T}}x - l_0 x_l + M x_{-1},$$

where M is some large positive number, and we use Karmarkar's algorithm. Provided M is large enough and (P_0^∞) is feasible, x_{-1} will be zero at the optimal solution. If the direction we move in is such that choosing a step length of an appropriate size would give an interior feasible point for (P_0^∞), we choose that step length and drop the artificial variable from the formulation. It follows that we always have a strictly positive feasible point for either (P_{-1}) or (P_0^∞).

The scaled version of (P_{-1}) is

$$(\tilde{\mathrm{P}}_{-1}) \quad \begin{aligned} \min \quad & x_{-1} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = 0, \\ & \tilde{a}_0^{\mathrm{T}}\tilde{x} - \beta\tilde{x}_l + (\beta - \eta)x_{-1} = 0, \\ & \tilde{g}^{\mathrm{T}}\tilde{x} = 1, \\ & \tilde{x} \geqslant 0, \quad \tilde{x}_l \geqslant 0, \quad x_{-1} \geqslant 0, \end{aligned}$$

where $\tilde{A} := A\hat{X}$, $\tilde{a}_0 := \hat{X}a_0$, $\tilde{g} := \hat{X}g$ and $\hat{X}$ is the diagonal matrix whose entries are the components of the vector $\hat{x}$. Thus the point $\tilde{x} = e$, $\tilde{x}_l = 1$ and $x_{-1} = 1$ is feasible for $(\tilde{\mathrm{P}}_{-1})$.

Lemma 11. *The direction of steepest descent in the null space of the subspace constraints for the problem* $(\tilde{\mathrm{P}}_{-1})$ *is*

$$\begin{pmatrix} d \\ d_0 \\ d_{-1} \end{pmatrix} := \begin{pmatrix} (\beta - \eta)P_{\tilde{A}}\tilde{a}_0 \\ -(\beta - \eta)\beta \\ -\tilde{a}_0^{\mathrm{T}}P_{\tilde{A}}\tilde{a}_0 - \beta^2 \end{pmatrix}. \tag{29}$$

Proof. This result can be shown by direct calculation, using the fact that

$$\begin{bmatrix} H & w \\ w^{\mathrm{T}} & v \end{bmatrix}^{-1} = \begin{bmatrix} H^{-1} + \rho^{-1}H^{-1}ww^{\mathrm{T}}H^{-1} & -\rho^{-1}H^{-1}w \\ -\rho^{-1}w^{\mathrm{T}}H^{-1} & \rho^{-1} \end{bmatrix},$$

where $\rho = v - w^{\mathrm{T}}H^{-1}w$, H is a symmetric positive definite matrix, w is a vector, and v is a scalar. $\square$

We now give a bound on the reduced cost η of the omitted variable y_0.

Lemma 12. *The size of $|\eta|$ is bounded above by the 1-norm of the vector $P_{\tilde{A}}\tilde{a}_0$. In fact, $\eta = e^{\mathrm{T}}P_{\tilde{A}}\tilde{a}_0$.*

Proof. By definition,

$$\eta = a_0^{\mathrm{T}}\hat{x} = \tilde{a}_0^{\mathrm{T}}e = e^{\mathrm{T}}P_{\tilde{A}}\tilde{a}_0,$$

the last equality holding because e is in the null space of $\tilde{\mathrm{A}}$. □

In the previous section we were able to give a Phase I procedure which converged in one step. The Phase I procedure defined in this section will not in general converge so quickly, although if the size of the reduced cost is sufficiently small we do have the following lemma.

Lemma 13. *If*

$$|\eta|\,\|P_{\tilde{A}}\tilde{a}_0\|_\infty \leqslant \|P_{\tilde{A}}\tilde{a}_0\|_2^2$$

then the smallest component of the direction defined in Lemma 11 is that for the artificial variable, provided β is chosen appropriately.

Proof. Set $\beta = \|P_{\tilde{A}}\tilde{a}_0\|_\infty$. The lemma then follows from equation (29). □

It follows that if

$$|\eta|\,\|P_{\tilde{A}}\tilde{a}_0\|_\infty \leqslant \|P_{\tilde{A}}\tilde{a}_0\|_2^2,$$

the Phase I procedure converges in one step with an appropriate choice of step length. Therefore, using Lemma 12, the procedure converges in one step if

$$|e^{\mathrm{T}}P_{\tilde{A}}\tilde{a}_0|\,\|P_{\tilde{A}}\tilde{a}_0\|_\infty \leqslant \|P_{\tilde{A}}\tilde{a}_0\|_2^2. \tag{30}$$

If the Phase I procedure converges quickly, it is likely that the feasible point we find for (P_0^∞) will be close to $\hat{x}$, so convergence to optimality in this problem should be rapid. We now give two cases in which the condition for convergence in one step holds. First we state the following technical lemma. (For a proof, see [37].)

Lemma 14. *For any n-vector a,*

$$\|a\|_2^2 \geqslant \tau\|a\|_1\|a\|_\infty,$$

where

$$\tau = \frac{2}{1+\sqrt{n}}. \qquad \Box$$

Theorem 4. *If*

(i) $|e^{\mathrm{T}} P_{\tilde{A}} \tilde{a}_0| \leqslant \|P_{\tilde{A}} \tilde{a}_0\|_2$,

or

(ii) $|e^{\mathrm{T}} P_{\tilde{A}} \tilde{a}_0| \leqslant \tau \|P_{\tilde{A}} \tilde{a}_0\|_1$, *where* $\tau = \dfrac{2}{1+\sqrt{n}}$,

then the Phase I *procedure converges in one step.*

Proof. In case (i), the result follows from equation (30), since $\|a\|_\infty \leqslant \|a\|_2$ for any vector a. Case (ii) follows from Lemma 14 and equation (30). □

5.4. Adding many variables

We may wish to add many variables to (D) at once. We can use the procedure given in Section 5.2 for each of these variables that have finite upper and lower bounds in order to obtain a point in the modified version of (P) where the additional slack and surplus variables have positive value. Therefore, we assume that all p added variables have infinite upper bound. Thus we wish to find a strictly positive feasible point to the following problem:

$$
\begin{array}{lll}
(\mathrm{P}_0^\infty) & \min & c^{\mathrm{T}} x - l_0^{\mathrm{T}} x_l \\
& \text{s.t.} & Ax = 0, \\
& & A_0^{\mathrm{T}} x - x_l = 0, \\
& & g^{\mathrm{T}} x = 1, \\
& & x \geqslant 0, \quad x_l \geqslant 0,
\end{array}
$$

where now x_l and l_0 are p-vectors and A_0 is an $n \times p$ real matrix. Define the vector $\eta := A_0^{\mathrm{T}} \hat{x}$. If the value of any component of η is positive the appropriate component of x_l can be set equal to that value. Thus we assume that $\eta \leqslant 0$.

Let β be a strictly positive p-vector. Then, by analogy with the method given in Section 5.3, consider the Phase I problem

$$
\begin{array}{lll}
(\mathrm{P}_{-1}) & \min & x_{-1} \\
& \text{s.t.} & Ax = 0, \\
& & A_0^{\mathrm{T}} x - x_l + (\beta - \eta) x_{-1} = 0, \\
& & g^{\mathrm{T}} x = 1, \\
& & x \geqslant 0, \quad x_l \geqslant 0, \quad x_{-1} \geqslant 0,
\end{array}
$$

where x_{-1} is a scalar. Notice that $x = \hat{x}$, $x_l = \beta$, $x_{-1} = 1$ is a strictly positive feasible point for this problem.

If the optimal value of this problem is zero, the optimal solution leads to a feasible point for (P_0^∞); if the optimal value of (P_{-1}) is strictly greater than zero then (P_0^∞) is infeasible. In practice, we use a big-M method, as in the case when we only add one variable. Thus, our Phase I objective function is

$$c^T x - l_0^T x_0 + M x_{-1},$$

where M is some large positive number. Provided M is large enough and (P_0^∞) is feasible, x_{-1} will be zero at the optimal solution.

6. Discussion of the implementations

6.1. Technical details

The algorithm was run on an IBM 3090 at Cornell University. Our computer code was written in FORTRAN 77. We compiled the code using FORTVS, with an optimization level of three. We used the GOSTMT option. All other options were set to their default values, except as indicated in the next paragraph. We did not use the vector option available on the IBM 3090.

The method we used to perform the projections necessary in an implementation of Karmarkar's algorithm requires large amounts of virtual memory when running larger problems. The operating system VM/CMS is limited to 16 megabytes of virtual memory (24 bit addressing). In order to use more than 16 megabytes of memory on the IBM 3090, it is necessary to use the Extended Data Array Capability (EDAC) of VM/XA. This requires placing large arrays into dynamic common blocks and then compiling the code with a special option. When this facility is used, VM/XA provides extended addressing up to 999 megabytes (31 bit addressing).

6.2. Performing the projections

When using Karmarkar's algorithm to solve linear programming problems, the major work at each iteration is performing a projection onto the null space of a scaled version of the current constraint matrix. There are several methods available for calculating such a projection; we chose to use one based on forming a *QR factorization* of the constraint matrix. This method is very numerically stable, but in practice it is not as fast as other methods based on Cholesky factorization or preconditioned conjugate gradients.

For a description of QR factorizations of matrices, the reader is referred to Golub and Van Loan [20]. The articles by George and Heath [14] and George and Ng [15] discuss solving *sparse* least squares problems using QR factorizations.

Having calculated the projection matrix, we calculate the projections of the objective function c and the normalizing constraint g separately, and once we have a new primal point we project that point in order to ensure primal feasibility. Thus, at each iteration we perform three projections, with the third of these projections not being strictly necessary, merely useful for obtaining greater numerical accuracy. In fact, it was shown by Gay [13] that the new primal point can be calculated using just one projection whenever it is not possible to update the lower bound using the method of Todd and Burrell [45]. In our implementation, most of the run time is spent on calculating the factorization, with a relatively small amount spent on the projections. It follows that the improvement in run time which could be obtained from using Gay's work would not be large. However, if a different method was used for calculating the projections, the benefit of calculating fewer projections could be far greater.

In our current implementation, we calculate the projection matrix anew every time rows or columns are added to the constraint matrix. The run time could be reduced if we exploited information about the projection matrix used during the previous relaxation.

6.3. Convergence criterion

We regarded the current linear programming relaxation as solved if the duality gap was smaller than 10^{-8}. Usually, a convergence criterion depends to some degree on the absolute value of the objective function; however, because of the structure of our set of test problems, the order of magnitude of the optimal value is known in advance, so we used the convergence criterion indicated.

7. Computational results for perfect matching problems

In the tables in this section and the following section, the columns labelled itns, time, cliques, stages, and gap contain, respectively, the following information:

- Iterations: Number of iterations of Karmarkar's algorithm to solve the problem.
- Time: Run time in seconds.
- Cliques: Number of cutting planes added.
- Stages: Number of stages of adding cutting planes.
- Gap: Number of iterations of Karmarkar's algorithm between the time cutting planes are first added and either (a) the next time the separation routines are called or (b) the time the problem is solved, whichever is less.

Note that, for a given perfect matching problem, the total number of relaxations examined is

1 + number of stages of adding cutting planes + number of stages of adding variables.

For more detailed results for the perfect matching problem and also results for the linear ordering problem, the reader is referred to Mitchell [37].

7.1. Generating problems

It is rare to find real world instances of the matching problem in its pure form (see Grötschel and Holland [24]). However, the matching problem does arise in certain heuristics designed to solve other combinatorial optimization problems. For example, the Christofides heuristic (see [7] or [42]) for the traveling salesman problem involves the solution of a perfect matching problem.

Therefore, it was suggested by Grötschel [23] that we generate random perfect matching problems where the edge weights could be those of traveling salesman problem. For that reason, we generated problems as follows:

- Generate k points uniformly in the unit square.
- Look at the complete graph on those vertices.
- Take the Euclidean distance between two vertices as the length of the corresponding edge.

Following [24], we also generated some problems as follows:

- Look at the complete graph on k vertices.
- Generate edge lengths uniformly in the range $[0, 1]$.

Unless otherwise stated, the results we give are for problems generated using the first of these procedures.

7.2. Investigating the procedures used when adding cutting planes

The performance of our algorithm depends upon the choice of a number of parameters. In this subsection we examine the effect of some of the parameters which determine how the relaxation is updated when the separation routines are called.

7.2.1. When to call the separation routines

As described in Section 3, the algorithm calls the separation routines as soon as a non-integral dual solution is found. We found that calling the separation routines later harmed the performance of the algorithm.

7.2.2. Different search levels

In our standard choice of parameters for the algorithm, the separation routines involve searching for connected components at seven different prescribed levels. In this subsection we investigate different choices for the search levels.

In the results that we give, the standard strategy is referred to as "search strategy B". The main alternative we consider is to search at just two levels: 0.01 and 0.31. This is referred to as "search strategy A". A second alternative strategy we considered was to only look for connected components in the graph given by dropping all edges of size less than 0.01. Since we are using an interior point method, no edge will have value exactly zero. Therefore, this strategy closely corresponds to that of Grötschel and Holland [24] of looking for connected components in the graph of all edges with nonzero value. This strategy was considerably inferior to the other two strategies, so we do not report the results for it.

Table 1 gives the results for seven sixty node problems when run with strategies A and B. The number of nearest neighbours, NN, was set equal to 7. Table 2 gives

Table 1

Comparing search strategies for sixty node problems

Problem	Search strategy	Itns	Time	Cliques	Stages	Gap
60i	A	16	19.5	11	2	4
	B	18	23.6	15	3	5
60ii	A	11	11.5	9	1	4
	B	12	12.6	13	1	5
60iii	A	15	10.4	10	2	3
	B	14	12.0	18	3	3
60iv	A	15	19.7	7	2	8
	B	15	19.7	7	2	8
60v	A	10	8.6	2	1	5
	B	10	8.6	2	1	5
60vi	A	20	29.2	17	3	4
	B	21	31.9	22	3	7
60vii	A	31	50.5	18	7	2
	B	31	47.6	29	5	5
average	A	16.9	21.3	10.6	2.6	4.3
	B	17.3	22.3	15.1	2.6	5.4

Table 2

Comparing search strategies for 202 node problems

Problem	Search strategy	Itns	Time	Cliques	Stages	Gap
202i	A	45	329.8	38	8	6
	B	39	267.8	54	6	6
202ii	A	75 [b]	527.4	54	16	4
	B	35	278.1	69	7	5
202iii	A	20	125.0	26	3	5
	B	24	162.2	41	4	6
202iv	A	52 [b]	367.2	49	8	3
	B	74	536.4	66	10	4
202v	A	33	246.0	35	5	5
	B	27	198.3	49	3	7
average [a]	A	32.7	233.6	33.0	5.3	5.3
	B	30.0	209.4	48.0	4.3	6.3

[a] Average is mean of problems 202i, 202ii, and 202v.

[b] This run finished with a non-integral solution.

the results for five 202 node perfect matching problems. For these problems, *NN* was set to 10.

The following conclusions can be drawn from these experiments:

• Using strategy A results in the addition of fewer cliques and in a smaller gap than when strategy B is used. This is to be expected: by its very nature, strategy A will result in the addition of fewer constraints at each stage, and since fewer cliques are added on the first stage, the gap will be smaller.

- For the easier problems (that is, most of the 60 node problems and problem 202iii), strategy A is better than the standard strategy in terms of number of iterations and run time. In these runs, the two strategies required approximately the same number of stages, so the benefit of adding fewer constraints at each stage shows up in the number of iterations because it is then slightly faster to reoptimize.
- For the harder problems (60vii and the remaining 202 node problems), the standard strategy is better. Strategy A failed to solve a couple of these problems, and it took more stages on the others, resulting in a higher number of iterations and a greater run time.

From these last two conclusions, it would appear that it is only beneficial to use the more extensive search routines for harder problems. The harder problems are more likely to contain *nested odd sets*, where cutting planes found at different stages are related in that one of the corresponding cliques is a subset of the other. For example, consider the graph given in Figure 3. The distances in this graph are Euclidean distances. The solution to the initial relaxation of the perfect matching problem on this problem violates the odd set constraints corresponding to the cliques $\{v_1, v_2, v_3\}$ and $\{v_8, v_9, v_{10}\}$. When these constraints are added and the new relaxation solved, the solution violates the odd set constraints corresponding to the cliques $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_6, v_7, v_8, v_9, v_{10}\}$. Because our algorithm is looking for odd set constraints violated by an interior point and because it searches at different levels, it may well discover both of these sets of constraints simultaneously. For a graph containing nested odd sets, the more extensive search routines are likely to lead to a more efficient algorithm than the simpler routines.

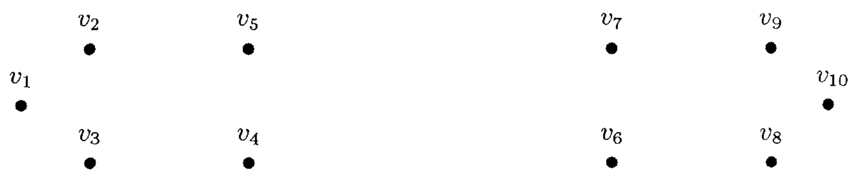

Fig. 3. An illustration of the phenomenon of nested odd sets.

7.2.3. *Different choices for direction*

Having chosen which cutting planes to add, we need to determine their relative weightings.

Recall from Section 4.5 that the direction we move in when adding cutting planes depends on a positive set of weights for the variables corresponding to the added cutting planes. We experimented with three different choices for the weightings:

1. All weights are equal.

2. Each weight is equal to the amount by which the current dual solution y violates the corresponding cutting plane.

3. Each weight is equal to the reciprocal of the 2-norm of the normal to the corresponding cutting plane.

These last two choices are motivated by considerations of dual ellipsoids. In [37], we developed a criterion for deciding when to add a particular cutting plane, and this criterion depended upon the current violation and the 2-norm of the inequality. Therefore we hypothesized that an inequality is more important if the current violation of it is large or if its norm is small.

We found that scaling by violation is very slightly better than the other two options, but it is not significantly better. The algorithm appears to be robust, in that the performance is only slightly affected by varying the choice of weighting.

7.3. Investigating the procedures used when adding variables

Once the duality gap for the current relaxation is less than some tolerance, and if the current dual solution y cannot be cut off using our separation routines, we check the reduced costs of all the edges in the graph. If the reduced cost of edge e is negative, edge e should be added to the relaxation. We discussed how to do this in Section 5.4.

We investigated two different choices for the vector β described in that section. The first choice is to set every component of β to be one (the standard choice); the second is to set β to be the vector of the negatives of the reduced costs. We found no significant difference in the performance of the algorithm when β was varied. Therefore, we only report some of our results for the standard choice of β.

Notice that it may happen that none of the added edges have nonzero value in the optimal matching, with the optimal matching being the best matching found before these edges are added. This is due to two things: primal degeneracy, and the fact that our termination criterion is not exact but depends on the duality gap being smaller than some value.

For $NN < 10$, the sixty node problem 60viii requires the addition of at least one additional edge. In order to see how the performance of the algorithm is affected by the choice of NN, we solved this problem for all choices of NN from 2 to 9. These results are presented in Table 3. The number of edges in the initial relaxation is approximately $\frac{1}{2}(1+NN)$ times the number of vertices. Only for $NN=2$ does there exist a stage at which we have to invoke the bucket sort in order to limit the number of edges added. We repeated the runs for $NN=2$, adding *all* edges with negative reduced cost; the results were substantially the same as those reported here.

As can be seen from the table, the performance of the algorithm does depend upon the choice of NN:

- Iterations: Setting NN to be five or six resulted in a significantly lower number of iterations than other choices. These values result in fewer iterations than the lower values of NN because they result in fewer stages of adding variables; they

Table 3

Problem 60viii with different choices for *NN*

NN	Itns	Time	Cliques	Stages	Added edges*			
					Total	Stages	Stage by stage	
							Added	Phase I
2	51	22.1	24	3	84	3	22	5
							49	3
							13	12
3	38	5.8	18	3	15	2	13	10
							2	5
4	33	9.5	22	3	8	1	8	11
5	22	11.9	17	3	4	1	4	5
6	17	13.5	15	2	2	1	2	4
7	36	31.4	21	4	1	1	1	9
8	37	42.0	19	4	1	1	1	9
9	37	38.0	21	4	1	1	1	9

* The last four columns contain the following information:
- Total: Total number of added edges.
- Stages: Number of stages of adding edges.
- Stage by stage: Breakdown of stages of adding edges.
  - Added: Number of edges added on a particular stage.
  - Phase 1: Number of Phase I iterations.

require fewer iterations than the larger choices of *NN* because the linear programming problems are smaller.

• Run time: Setting *NN* to be 3, 4, 5, or 6 gives significantly lower run times than other choices. The choices 5 and 6 do well because they require few iterations; the choices of 3 and 4 do well because the linear programs are relatively small, making the time per iteration small.

• Number of edges added and number of stages of adding variables: As is expected, the mean values of both of these categories decreased as *NN* increased, with $NN = 2$ resulting in significantly higher values than other choices.

We also analyzed the number of iterations required to solve Phase I after adding edges. This value did not appear to be correlated with either *NN*, or the number of edges added at that stage, or the number of edges already in the relaxation. (The data was analyzed using Duncan's multiple range test (see [43]).)

7.4. Problems with randomly generated edge weights

All the results we have reported so far have been on problems where the *vertices* are uniformly distributed within the unit square and the edge lengths are taken to be the Euclidean distances between the vertices. In this section, we consider generating problems where the *edge lengths* are generated randomly.

Table 4

Perfect matching problems with the edge weights distributed uniformly

Problem	Itns	Time	Cliques	Stages	Gap
E60i	12	10.2	0	0	—
E100i	14	42.9	1	1	1
E100ii	10	20.5	0	0	—
E202i	15	285.4	3	2	1
E202ii	15	281.1	2	1	4

In Table 4, we give the results for five problems where the edge weights were independently generated from a uniform distribution on the interval [0, 1]. Problem E60i is a sixty node problems, problems E100i and E100ii are 100 node problems, and problems E202i and E202ii are 202 node problems. For all of these problems, the number of nearest neighbours NN was set to five.

As can be seen, these problems require the addition of very few cutting planes and correspondingly few iterations. However, the low iteration count is not reflected in the run time, when compared with problems generated using the other method. This is because of the structure of the constraint matrix. When the vertices are uniformly distributed, the edge lengths satisfy the triangle inequality. It follows that if vertex i is close to vertex j and vertex j is close to vertex k, vertex i will be close to vertex k. Therefore, vertices that are close together will have similar sets of nearest neighbours. This means that the adjacency matrix for the initial relaxation can probably be permuted so that AA^T has most of its nonzero elements close to the main diagonal (A is the initial constraint matrix). This beneficial numerical feature is unlikely to be a property of the constraint matrix when the edge lengths are randomly generated.

8. Summary

With the standard choice of parameters, the algorithm we proposed produces the optimal perfect matching on most problems, showing that the algorithm is robust and that the separation heuristics are efficient. (If the algorithm does not finish with an integral solution, the Padberg-Rao procedure [41] can be used to find a violated clique inequality. We did not implement this routine.)

In Table 5, we give the results for eight sixty node problems and eight 202 node problems which were solved using our algorithm. For the sixty node problems, $NN = 7$; for the 202 node problems, $NN = 10$. Most of these results were given in the previous section; they are repeated here for convenience. For the sixty node problems, the initial constraint matrices have approximately 240 rows and 300

Table 5

Several 60 and 202 node problems

Problem	Optimal value	Itns	Time	Cliques	Stages	Gap
60i	2.5692	18	23.6	15	3	5
60ii	2.7601	12	12.6	13	1	5
60iii	2.8377	14	12.0	18	3	3
60iv	2.7939	15	19.7	7	2	8
60v	2.2073	10	8.6	2	1	5
60vi	2.5609	21	31.9	22	3	7
60vii	2.7714	31	47.6	29	5	5
60viii	2.6279	36[b]	31.4	21	4	7
202i	4.7793	39	267.8	54	6	6
202ii	4.8593	35	278.1	69	7	5
202iii	4.6887	24	162.2	41	4	6
202iv	4.4972	74	536.4	66	10	4
202v	4.6585	27	198.3	49	3	7
202vi	4.4679	28[b]	180.4	30	2	5
202vii	4.4000	31[a]	253.3	65	6	5
202viii	4.7317	55[b]	427.7	57	7	6

[a] This run finished with a non-integral solution.
[b] This run required the addition of one extra edge which had reduced cost of the wrong sign.

columns, with the final constraint matrices having approximately 15 more columns. For the 202 node problems, the initial constraint matrices have approximately 1100 rows and 1300 columns, with the final matrices having approximately 50 more columns.

We believe that these results give some justification for optimism regarding the performance of our algorithm. This optimism is based on consideration of both the number of iterations and the number of stages.

Our optimism regarding the number of iterations is based purely on consideration of interior point methods. Using Karmarkar's algorithm to solve the final linear programming relaxation starting from scratch would take about half as many iterations as the number reported in the table. This is because we call the separation routines before optimality is reached, so the sequence of iterates does not approach the vertices of the polytope too closely, and thus the number of iterations in each stage is reasonably small. Hence, after a cutting plane is added, we have a fairly central "warm start" which can be exploited. The value of "Gap" in the tables further illustrates this point.

The number of stages of adding cliques using our algorithm is smaller than they would be if a cutting plane algorithm based on the simplex method were used. One reason for this is that, at a given stage, our separation heuristics often find violated odd set constraints that correspond to nested odd sets. This phenomenon occurs because we use an interior point method and because we do not solve the current relaxation to optimality. If each linear program was solved to optimality, it would frequently be the case that the nested odd set constraints would be added at successive stages.

The run times we report are not good, principally because of the method we use to calculate the projections. Forming a QR-factorization is a very numerically stable method of calculating a projection, but it is not fast. The lack of speed is especially noticeable when Householder transformations are used to form the QR-factorization, because this leads to considerable fill in, in the intermediate matrices, and thus in the representation of the orthogonal matrix Q. Another drawback to the method of calculating projections that we used is the need to perform a new symbolic factorization of the matrix every time cutting planes are added. Possible remedies for this problem include calculating the projections using a preconditioned conjugate gradient method (the same preconditioner can be used on successive stages) or using low rank updates to the projection matrices, such as Schur complements (see Choi et al. [6], for example).

It is not clear that a warm start can be exploited as efficiently in a cutting plane method based on an interior point algorithm as in one based on the simplex algorithm. Thus, for an interior point-based method to be competitive, it is necessary to limit the number of iterations performed at each stage and to reduce the number of stages by designing separation routines which exploit the nature of the solutions generated by an interior point method. The results we have presented indicate that this may well be possible.

References

[1] I. Adler, M.G.C. Resende, G. Veiga and N.K. Karmarkar, "An implementation of Karmarkar's algorithm for linear programming," *Mathematical Programming* 44 (1989) 297-335.

[2] K.M. Anstreicher, "A monotonic projection algorithm for fractional linear programming," *Algorithmica* 1 (1986) 483-498.

[3] D.A. Bayer and J.C. Lagarias, "The nonlinear geometry of linear programming I. Affine and projective scaling trajectories," *Transactions of the American Mathematical Society* 314 (1989) 499-526.

[4] D.A. Bayer and J.C. Lagarias, "The nonlinear geometry of linear programming II. Legendre transform coordinates and central trajectories," *Transactions of the American Mathematical Society* 314 (1989) 527-581.

[5] 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, 1991).

[6] In Chan Choi, C.L. Monma and D.F. Shanno, "Further development of a primal-dual interior point method," *ORSA Journal on Computing* 2 (1990) 304-311.

[7] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," Technical report, GSIA, Carnegie-Mellon University (Pittsburgh, 1976).

[8] G.B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in: Tj.C. Koopmans, ed., *Activity Analysis of Production and Allocation* (Wiley, New York, 1951) pp. 339-347.

[9] U. Derigs and A. Metz, "On the use of optimal fractional matchings for solving the (integer) matching problem," *Computing* 36 (1986) 263-270.

[10] J. Edmonds, "Maximum matching and a polyhedron with 0, 1 vertices," *Journal of Research National Bureau of Standards* 69B (1965) 125-130.

[11] J. Edmonds, "Paths, trees and flowers," *Canadian Journal of Mathematics* 17 (1965) 449-467.

[12] R.M. Freund, "A potential-function reduction algorithm for solving a linear program directly from an infeasible "warm start"," *Mathematical Programming (Series B)* 52 (1991) 441-466.

[13] D.M. Gay, "A variant of Karmarkar's linear programming algorithm for problems in standard form," *Mathematical Programming* 37 (1987) 81-90.

[14] J.A. George and M.T. Heath, "Solution of sparse linear least squares problems using Givens rotations," *Linear Algebra and its Applications* 34 (1980) 69-83.

[15] J.A. George and E.G.Y. Ng, "Orthogonal reduction of sparse matrices to upper triangular form using Householder transformations," *SIAM Journal on Scientific and Statistical Computing* 7 (1986) 460-472.

[16] J.-L. Goffin, "Affine and projective transformations in non-differentiable optimization," in: K.-H. Hoffmann, J.-B. Hiriart-Urruty, C. Lemarechal and J. Zowe, eds., *Trends in Mathematical Optimization*, International Series of Numerical Mathematics No. 84 (Birkhauser, Basel-Boston, 1988) pp. 79-91.

[17] J.-L. Goffin, A. Haurie and J.-P. Vial, "Decomposition and nondifferentiable optimization with the projective algorithm," Technical Report 91-01-17, Faculty of Management, McGill University (Montreal, Que., 1990).

[18] 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) 409-429.

[19] D. Goldfarb and M.J. Todd, "Linear programming," in G.L. Nemhauser et al., eds., *Optimization* (North-Holland, Amsterdam, 1989) Chapter 2, pp. 73-170.

[20] G.H. Golub and C.F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, MD, 1983).

[21] R.E. Gomory and T.C. Hu, "Multi terminal network flows," *Journal of the Society for Industrial and Applied Mathematics* 9 (1961) 551-571.

[22] C.C. Gonzaga, "A conical projection algorithm for linear programming," *Mathematical Programming* 43 (1989) 151-174.

[23] M. Grötschel, Private Communication, 1987.

[24] M. Grötschel and O. Holland, "Solving matching problems with linear programming," *Mathematical Programming* 33 (1985) 243-259.

[25] M. Grötschel, L. Lovasz and A. Schrijver, "The ellipsoid method and its conseqeunces in combinatorial optimization," *Combinatorica* 1 (1981) 169-197.

[26] D. den Hertog and C. Roos, "A survey of search directions in interior point methods for linear programming," *Mathematical Programming (Series B)* 52 (1991) 481-509.

[27] IBM. *IBM Optimization Subroutine Library Guide and Reference*, Publication number SC23-0519-1 (1990).

[28] N.K. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4 (1984) 373-395.

[29] N.K. Karmarkar, "An interior-point approach to NP-complete problems (Part I)," in: J.C. Lagarias and M.J. Todd, eds., *Mathematical Developments Arising from Linear Programming*, Contemporary Mathematics No. 114 (American Mathematical Society, Providence, RI, 1991) pp. 297-308.

[30] N.K. Karmarkar, M.G.C. Resende and K.G. Ramakrishnan, "An interior point algorithm to solve computationally difficult set covering problems," *Mathematical Programming (Series B)* 52 (1991) 597-618.

[31] S. Lin, "Computer solutions to the traveling salesman problem," *Bell System Technical Journal* 44 (1965) 2245-2269.

[32] L. Lovász and M.D. Plummer, *Matching Theory* (Akadémiai Kiadó, Budapest, 1986).

[33] 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) 191-222.

[34] I.J. Lustig, R.E. Marsten and D.F. Shanno, "On implementating Mehrotra's predictor-corrector interior point method for linear programming," to appear in: *SIAM Journal on Optimization.*

[35] N. Megiddo, "Pathways to the optimal set in linear programming," in: N. Megiddo, ed., *Progress in Mathematical Programming* (Springer, New York, 1989) pp. 131-158.

[36] N. Megiddo and M. Shub, "Boundary behaviour of interior point algorithms in linear programming," *Mathematics of Operations Research* 14 (1989) 97-146.

[37] J.E. Mitchell, *Karmarkar's Algorithm and Combinatorial Optimization Problems*, Ph.D. thesis, School of Operations Research and Industrial Engineering, Cornell University (Ithaca, NY, 1988).

[38] J.E. Mitchell, "An interior point column generation method for linear programming using shifted barriers," Technical Report 191, Mathematical Sciences, Rensselaer Polytechnic Institute (Troy, NY, 1990).

[39] J.E. Mitchell and M.J. Todd, "A variant of Karmarkar's linear programming algorithm for problems with some unrestricted variables," *SIAM Journal on Matrix Analysis and Applications* 10 (1989) 30-38.

[40] J.E. Mitchell and M.J. Todd, "Solving matching problems using Karmarkar's algorithm," in: J.C. Lagarias and M.J. Todd, eds., *Mathematical Developments Arising from Linear Programming*, Contemporary Mathematics No. 114 (American Mathematical Society, Providence, RI, 1991) pp. 309-318.

[41] M.W. Padberg and M.R. Rao, "Odd minimum cut-sets and *b*-matchings," *Mathematics of Operations Research* 7 (1982) 67-80.

[42] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[43] SAS Institute, Inc, Cary, North Carolina, *SAS User's Guide, Version 5* (1985).

[44] A.E. Steger, "An extension of Karmarkar's algorithm for bounded linear programming problems," Master's thesis, State University of New York at Stony Brook (Stony Brook, NY, 1985).

[45] M.J. Todd and B.P. Burrell, "An extension of Karmarkar's algorithm for linear programming using dual variables," *Algorithmica* 1 (1986) 409-424.

[46] R.J. Vanderbei, M.S. Meketon and B.A. Freedman, "A modification of Karmarkar's linear programming algorithm," *Algorithmica* 1 (1986) 395-407.

[47] Y. Ye, "A potential reduction algorithm allowing column generation," *SIAM Journal on Optimization* 2 (1992) 7-20.

[48] Y. Ye and M. Kojima, "Recovering optimal dual solutions in Karmarkar's polynomial algorithm for linear programming," *Mathematical Programming* 39 (1987) 305-317.