INTERIOR POINT METHODS FOR COMBINATORIAL OPTIMIZATION*

RPI Mathematical Sciences Report No. 217
September 29, 1995

JOHN E. MITCHELL†
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180

Abstract. Research on using interior point algorithms to solve combinatorial optimization and integer programming problems is surveyed. This paper discusses branch and cut methods for integer programming problems, a potential reduction method based on transforming an integer programming problem to an equivalent nonconvex quadratic programming problem, interior point methods for solving network flow problems, and methods for solving multicommodity flow problems, including an interior point column generation algorithm.

1. Introduction

Research on using interior point algorithms to solve combinatorial optimization and integer programming problems is surveyed. Typically, the problems we consider can be formulated as linear programming problems with the restriction that some of the variables must take integer values. The methods we consider have been used to solve problems such as the linear ordering problem, clustering problems, facility location problems, network flow problems, nonlinear multicommodity network flow problems, and satisfiability problems. This paper discusses four main methodologies, three of which are similar to known approaches using the simplex algorithm, while the fourth method has a different flavor.

Branch and cut methods are considered in section 2. Simplex-based branch and cut methods have been very successful in the last few years, being used to solve both specific problems such as the traveling salesman problem and also generic integer programming problems. The research described in this paper constructs a branch and cut algorithm of the usual type, but then uses an interior point method to solve the linear programming relaxations. The principal difficulty with using an interior point algorithm in a branch and cut method to solve integer programming problems is in warm starting the algorithm efficiently, that is, in using the solution to one relaxation to give a good initial solution to the next relaxation. Methods

* To appear in *"Interior Point Methods in Mathematical Programming"*, edited by Tamás Terlaky, Kluwer Academic Publisher, 1995.
† Research partially supported by ONR Grant number N00014–94–1–0391.

for overcoming this difficulty are described and other features of the algorithms are given. This paper focuses on the techniques necessary to obtain an efficient computational implementation; there is also a discussion of theoretical issues in section 6.1. *Column generation algorithms* have a structural similarity to cutting plane methods, and we describe a column generation algorithm for solving nonlinear **multicommodity network flow problems** in section 5.1.

In section 3, we discuss a method for solving integer programming problems that is based upon reformulating the integer programming problem as an equivalent non-covex quadratic programming problem. The quadratic program is then solved using a **potential reduction method**. The potential function has some nice properties which can be exploited in an efficient algorithm. Care is needed so that the algorithm does not get trapped in a local minimum. We also discuss a related algorithm for solving quadratic integer programming problems, which can be applied to the graph partitioning problem, for example.

Many **network flow problems** can be solved by ignoring the integrality requirement on the variables and solving the linear programming relaxation of the problem, because it is guaranteed that one of the optimal solutions to the linear program will solve the integer programming problem. Typically for these problems, the simplex method can be considerably enhanced by exploiting the structure of the constraint matrix; there are also often very good methods which are not based on linear programming. Thus, the challenge is to design an *efficient* implementation of an interior point method which can compete with the algorithms which are already available. We describe the research in this area in section 4.

Interior point approaches to the **multicommodity network flow problem** are discussed in section 5. These include the column generation algorithm mentioned earlier. These problems can be modelled as linear programming problems which are too large to be solved easily, so it is necessary to use alternative methods to just solving the linear programming problem.

Theoretical issues are discussed in section 6. This includes a discussion of the computational complexity of interior point cutting plane methods and also improved complexity results for various combinatorial optimization problems that have been obtained through the use of interior point methods.

Finally, we offer our conclusions in section 7.

2. Interior point branch and cut algorithms

In this section, we discuss the solution of integer programming problems using cutting plane and branch and bound methods. Before considering the general case, we examine the following example. Consider the integer feasible region S

$$\begin{aligned} 3x_1 + 5x_2 &\geq 9 \\ -2x_1 + 5x_2 &\leq 9 \\ 5x_1 + 2x_2 &\leq 25 \\ 3x_1 - 4x_2 &\leq 7 \\ x_i &\quad \text{integer, } i = 1, 2 \end{aligned}$$

shown in figure 1. The feasible points are shown by dots. The convex hull of the

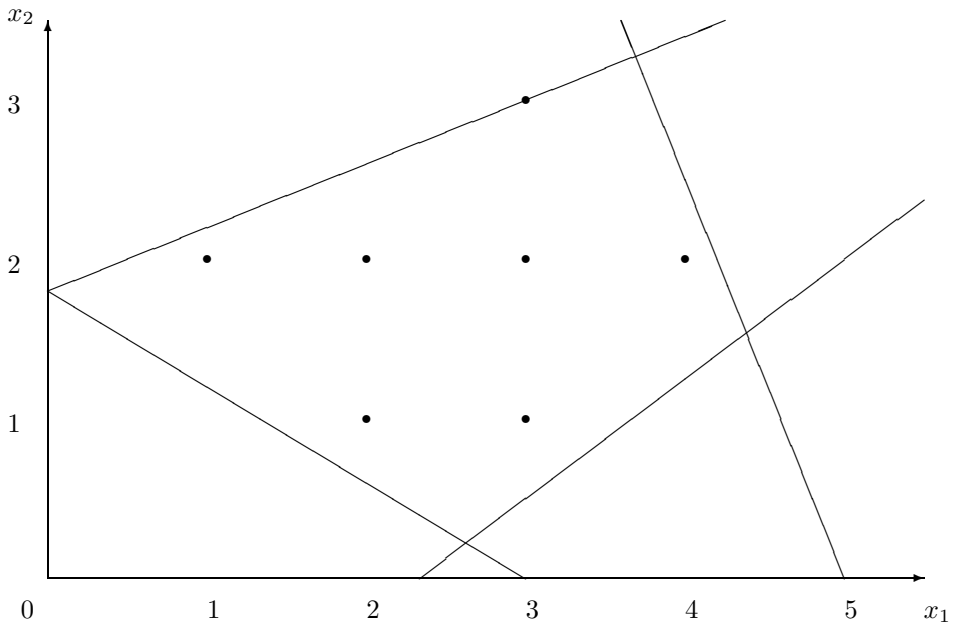

Fig. 1. Feasible region of an integer program

feasible integer points is the set of points

$$P := \{x_1, x_2 : x_1 + x_2 \geq 3, x_2 \geq 1, x_1 - x_2 \leq 2, x_1 + x_2 \leq 6, -x_1 + 2x_2 \leq 3\},$$

which has extreme points (1,2), (2,1), (3,1), (4,2), and (3,3). For a given linear objective function $c^T x := c_1 x_1 + c_2 x_2$, the optimal solution to the integer program $\min\{c^T x : x \in S\}$ will be one of these extreme points. Thus, with the given description of P, we could solve the integer program by solving the *linear* program $\min\{c^T x : x \in P\}$. Of course, in general it is hard to find the polyhedral description P.

Let us take $c_1 = 2$, $c_2 = 3$. The solution to the integer program is then the point $(2, 1)$. A cutting plane method first solves the LP relaxation of the integer program:

$$\begin{array}{lrcl} \min & 2x_1 + 3x_2 & & \\ \text{subject to} & 3x_1 + 5x_2 & \geq & 9 \\ & -2x_1 + 5x_2 & \leq & 9 \\ & 5x_1 + 2x_2 & \leq & 25 \\ & 3x_1 - 4x_2 & \leq & 7 \end{array}$$

This problem has optimal solution $(0, 1.8)$, with value 5.4. We then add an extra constraint (or *cutting plane*) to the LP relaxation that is violated by the point $(0, 1.8)$ but is satisfied by every point in S, and then resolve the LP relaxation. For example, we could add the constraint $4x_1 + x_2 \geq 4$. Modern cutting plane methods attempt to use cutting planes which are facets of the convex hull P of S, so they would add

either $x_1 + x_2 \geq 3$ or $-x_1 + 2x_2 \leq 3$. It is harder to find strong cutting planes like these than a weaker cutting plane such as a Gomory cut.

A branch and bound approach to this problem would examine the solution $(0, 1.8)$ to the LP relaxation and then split the problem into two new problems, one where $x_2 \geq 2$ and one where $x_2 \leq 1$. These new linear programs are then solved and the process is repeated. If the solution to any of the linear programming problems that arise in this process is integer then that point solves the corresponding part of the integer programming problem; if any of the linear programs is infeasible, then the corresponding part of the integer program is also infeasible. The value of the linear program provides a lower bound on the value of the corresponding part of the integer program, and this bound can be used to prune the search space and guide the search.

Cutting plane methods and branch and bound methods can be combined into a branch and cut method, but we will discuss them separately, in order to emphasize their individual features. For a good discussion of simplex-based branch and cut methods, see, for example, the books by Nemhauser and Wolsey [61] and Parker and Rardin [64]. The book [61] is a detailed reference on integer programming and it discusses cutting plane algorithms comprehensively; for a summary of this book, see [62]. The book [64] also discusses cutting plane algorithms, and it discusses branch and bound in more detail than [61]. Jünger *et al.* [35] discuss computational work using branch and cut algorithms to solve a variety of integer programming problems.

As mentioned above, cutting plane and branch and bound methods work by setting up a *linear programming relaxation* of the integer programming problem, solving that relaxation, and then, if necessary, refining the relaxation so that the solution to the relaxation gets closer to the solution to the integer programming problem. These methods have been known for many years (Land and Doig [46], Gomory [26]), and they have achieved very good results in the last few years. Of course, most of these results have been achieved by using the simplex algorithm to solve the linear programming relaxations; the focus in this section is on using an *interior point method* to solve the relaxations. Unfortunately, is is not usually sufficient to simply replace the simplex algorithm with an interior point method, because an interior point method is not as good as the simplex algorithm at exploiting the solution to one relaxation when trying to solve the next relaxation. This relatively poor use of the *warm start* provided by the previous relaxation makes it necessary to only solve the relaxations approximately; the algorithms seem fairly adept at exploiting this approximate solution. Other refinements to a traditional branch and cut approach are also necessary when using an interior point method, but the principal difference is in the use of approximate solutions to the relaxations.

We discuss cutting plane algorithms in section 2.1 and branch and bound algorithms in section 2.2. Adding a constraint to a primal linear programming problem is structurally equivalent to adding a column to the dual problem, so research on column generation algorithms has a strong impact on research on cutting plane algorithms, and vice versa. In section 5.1, we discuss a column generation algorithm for a multicommodity network flow problem. The theoretical performance of cutting plane and column generation algorithms is discussed in section 6.1.

2.1. Interior point cutting plane algorithms

In order to simplify the discussion, we assume that all the variables are constrained to take the values zero or one, and that all the constraints are inequality constraints. We assume we have an integer programming problem of the form

$$\begin{array}{lrcl} \min & \bar{c}^T\bar{x} & & \\ \text{subject to} & \bar{A}\bar{x} & \leq & b \\ & \bar{x}_i & = & 0 \text{ or } 1 \end{array} \qquad (IP)$$

where $\bar{x}$ and $\bar{c}$ are n-vectors, b is an m-vector, and $\bar{A}$ is an $m \times n$ matrix. We assume that c is not in the row space of $\bar{A}$; if this was not the case, every feasible solution would be optimal. We do not make any assumptions regarding the relative magnitudes of m and n, nor do we make any assumptions regarding the matrix $\bar{A}$. Many problems can be cast in this framework. We let Q denote the convex hull of feasible solutions to (IP). The linear programming relaxation (or *LP relaxation*) of (IP) is

$$\begin{array}{lrcl} \min & \bar{c}^T\bar{x} & & \\ \text{subject to} & \bar{A}\bar{x} & \leq & b \\ 0 \leq & \bar{x} & \leq & e \end{array} \qquad (LP^{\leq})$$

where e denotes a vector of ones of the appropriate dimension. (We will use e in this way throughout this paper.) If the optimal solution to $(LP^{\leq})$ is integral then it solves the original problem (IP), because it is feasible in (IP) and it is at least as good as any other feasible point in (IP). If the optimal solution $x^{LP}$to $(LP^{\leq})$ is not integral, then we improve the relaxation by adding an extra constraint or *cutting plane* of the form $a^{0^T}\bar{x} \leq b_0$. This cutting plane is a valid inequality for (IP) but it is violated by the optimal solution x^{LP}. We then solve the modified LP relaxation, and repeat the process.

The recent success of simplex based cutting plane algorithms has been achieved through the use of polyhedral theory and specialized cutting planes; the cutting planes are generally chosen from families of facets of the convex hull of feasible integer points. Traditionally, Gomory cutting planes were derived from the optimal simplex tableau; Mitchell [55] has shown how these same cutting planes can be derived when using an interior point cutting plane algorithm.

We prefer to write the linear programming relaxation as a problem with equality constraints; thus we include slack variables to get the relaxation

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & = & b \\ 0 \leq & x & \leq & u \end{array} \qquad (LP)$$

where u is a vector of upper bounds on the variables, so $u_i = 1$ for the original integer variables, and u_i takes an appropriate value for the remaining variables. The dual problem to (LP) is

$$\begin{array}{lrcl} \max & b^T y - u^T w & & \\ \text{subject to} & A^T y - w + z & = & c \\ & w, z & \geq & 0 \end{array} \qquad (LD)$$

1. **Initialize:** Pick initial x, y, w and primal and dual slacks.
2. **Approximately solve relaxation:** Solve the current relaxation to the desired degree of accuracy using an interior point algorithm. If the current iterate is a sufficiently accurate solution to the original problem (IP), STOP.
3. **Add cutting planes:** See if the current iterate violates any constraints. If not, tighten the desired degree of accuracy and return to Step 2; otherwise, add a subset of the violated constraints and go to Step 4.
4. **Update the relaxation and restart:** Update the variables appropriately. Return to Step 2.

Fig. 2. A prototype interior point cutting plane algorithm

When we add a cutting plane, we will obtain the new relaxation

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & = & b \\ & a^{0^T} x + x_0 & = & b_0 \qquad (LPnew) \\ & 0 \leq x & \leq & u \\ & 0 \leq x_0 & \leq & u_0 \end{array}$$

for some appropriate upper bound u_0 on the new slack variable x_0. The corresponding new dual problem is

$$\begin{array}{lrcl} \max & b^T y - u^T w - u_0 w_0 & & \\ \text{subject to} & A^T y + a_0 y_0 - w + z & = & c \\ & y_0 - w_0 + z_0 & = & 0 \qquad (LDnew) \\ & w, z & \geq & 0 \\ & w_0, z_0 & \geq & 0 \end{array}$$

Note that if we know feasible solutions $\hat{x} > 0$ and $\hat{y}$, $\hat{w} > 0$, $\hat{z} > 0$ to (LP) and (LD) respectively, then, after the addition of the cutting plane, we can obtain a new feasible solution to $(LDnew)$ by taking $y = \hat{y}$, $w = \hat{w}$, $z = \hat{z}$, $y_0 = 0$ and $w_0 = z_0$. If we pick $w_0 = z_0$ to be strictly positive then all the nonnegativity constraints will be satisfied strictly. It is not so simple to obtain a feasible solution to $(LPnew)$ because we have $a^{0^T} \hat{x} > b_0$ if the new constraint was a cutting plane. Nonetheless, if the old solution was close to optimal to (LP) and (LD) then we can hope that it should also be close to the solution to $(LPnew)$ and $(LDnew)$, so it provides a *warm start* for solution of the new problem.

In this section, we discuss how an interior point method can be used in this setting. A simple, conceptual interior point cutting plane algorithm could be written as in figure 2. We will give a more formal algorithm later. Currently, the best algorithm for linear programming appears to be the primal-dual predictor-corrector barrier method (see Lustig *et al.* [49, 50] and Mehrotra [52]), so we consider modifying this algorithm for use in a cutting plane algorithm. Other interior point algorithms which maintain strictly positive primal and dual iterates can be modified in a similar manner. We will also briefly discuss using a dual algorithm.

With a primal-dual algorithm, we always have *interior* primal and dual iterates, that is, $0 < x < u$, $w > 0$ and $z > 0$. We also have a barrier parameter μ and we refer to an iterate as *centered* if we have

$$x_i z_i = \mu \text{ and } (u_i - x_i)w_i = \mu,\ i = 1, \ldots, n. \tag{1}$$

When $\mu = 0$, these conditions are the complementary slackness conditions. Interior point methods tend to work better when they can use iterates that are close to being centered. The importance of having centered iterates is a theme which will recur in this paper.

We first motivate the discussion by describing two integer programming problems.

2.1.1. *Two example problems*

The *perfect matching problem* can be solved by using a cutting plane algorithm — see Grötschel and Holland [27] and Mitchell and Todd [59].

The perfect matching problem: Given a graph $G = (V, E)$ with vertices V and edges E, a *matching* is a subset M of the edges such that no two edges in M share an end vertex. A *perfect matching* is a matching which contains exactly $|\,V\,|\,/2$ edges, where $|\,V\,|$ denotes the cardinality of V. Given a set of weights w_e associated with the edges e in E, the perfect matching problem is to find the perfect matching M with smallest weight $w(M) := \sum_{e \in M} w_e$.

Edmonds [15, 16] showed that the perfect matching problem can be solved in polynomial time. He also gave a complete polyhedral description of the perfect matching problem. He showed that the optimal solution to a perfect matching problem is one of the solutions to the linear programming problem

$$\begin{array}{lll}
\min & \sum_{e \in E} w_e x_e & \\
\text{subject to} & \sum_{e \in \delta(v)} x_e \ = 1 \text{ for all } v \in V & (2) \\
& \sum_{e \in E(U)} x_e \ \le (|\,U\,| - 1)/2 \text{ for all } U \subseteq V \text{ with } |\,U\,| \text{ odd} & (3) \\
& x_e \qquad \ge 0 \text{ for all } e \in E & (4)
\end{array}$$

where $\delta(v)$ denotes the set of edges in E which are incident to vertex v and $E(U)$ denotes the set of edges in E which have both endvertices in U, where U is a subset of V. Equations (2) are the *degree constraints* and equations (3) are the *odd set constraints*. The number of odd set constraints is exponential in the number of vertices, so it is impracticable to solve the linear programming problem as expressed. Thus, in a cutting plane method, the initial relaxation consists of the degree constraints together with the nonnegativity constraints (4), and the odd set constraints are added as cutting planes. Consider, for example, the graph given in figure 3. Here, the edge weights are the Euclidean lengths of the edges. The optimal matching has $M = \{(v_2, v_3), (v_1, v_4), (v_5, v_6)\}$. The LP relaxation consisting of the degree constraints and the nonnegativity constraints has optimal solution

$$x_e = \begin{cases} 0.5 & \text{if } e \text{ is one of the edges } (v_1, v_2), (v_2, v_3), (v_1, v_3), (v_4, v_5), (v_4, v_6), (v_5, v_6) \\ 0 & \text{otherwise} \end{cases}$$

This solution violates the odd set constraint with $U = \{v_1, v_2, v_3\}$:

$$x_{(v_1,v_2)} + x_{(v_1,v_3)} + x_{(v_2,v_3)} \le 1$$

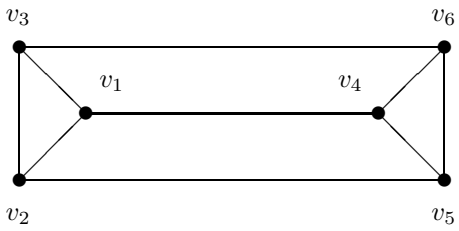

Fig. 3. The effect of an odd set constraint

If this constraint is added to the relaxation, the optimal solution to the linear program is the optimal matching given above.

Another problem that can be solved by a cutting plane algorithm is the *linear ordering problem* — see, for example, Grötschel, Jünger and Reinelt [28] or Mitchell and Borchers [57].

The linear ordering problem: Given a complete directed graph $G = (V, A)$, with costs c_{ij} on the arcs, define the cost of a permutation σ of the vertices to be $c(\sigma) := \sum_{(i,j):\sigma(i)<\sigma(j)} c_{ij}$. The linear ordering problem is to find the permutation with the smallest cost.

The linear ordering problem is NP-Hard [44]. This problem can be expressed as an integer linear programming problem:

$$\begin{array}{lrll} \min & \sum_{i,j} c_{ij} x_{ij} & & \\ \text{subject to} & x_{ij} + x_{ji} & = 1 \text{ for } 1 \le i < j \le |V| & (5) \\ & x_{ij} + x_{jk} + x_{ki} & \le 2 \text{ for } 1 \le i < j < k \le |V| & (6) \\ & x_{ij} & = 0 \text{ or } 1 \text{ for } 1 \le i < j \le |V| & (7) \end{array}$$

Grötschel *et al.* [28] have found several classes of valid inequalities for the linear ordering problem which can be used as cutting planes. However, for many real world problems, the solution to the linear programming relaxation given above solves the linear ordering problem. The equations (6) are known as the *3-dicycle constraints*; notice that there are $\binom{n}{3}$ of them. In both [28] and [57], the initial relaxation consists just of the equations (5) together with the simple bounds $0 \le x_{ij} \le 1$ for each edge; the 3-dicycle constraints are added as cutting planes as needed. In these implementations, the solutions to the relaxations can be integral but not feasible in the linear ordering problem; in this case cutting planes are used to cut off infeasible integral points. Thus, this approach to solving the linear ordering problem does not quite fit in the framework we discussed earlier with relation to the problems (IP) and (LP), but that framework can be extended in an obvious manner to include this approach to the linear ordering problem. The traveling salesman problem is also usually formulated so that solutions to the LP relaxations can be integral but infeasible; the *subtour elimination constraints* are used to cut off these infeasible integral points. (For a good discussion of the traveling salesman problem see the

book edited by Lawler *et al.* [47]; for a recent description of an implementation, see Applegate *et al.* [5].)

2.1.2. *Early termination*

The optimal solution to (LP) is not an interior point. Therefore, if we solve (LP) to optimality then it is necessary to perturb the solution slightly to obtain an interior point before we can even start solving $(LPnew)$ using an interior point method. Typically, if an interior point method is started from close to the boundary, it will move towards the center of the feasible region before starting to move towards the optimal solution. Thus, the optimal solution to (LP) is not a very good starting point for trying to solve $(LPnew)$. A very successful method to try to avoid this difficulty is to terminate solution of (LP) early. We will then have an interior point when we start solving $(LPnew)$. In addition, we will not spend as many iterations returning towards the center of the polyhedron and we will start moving towards the optimal solution to $(LPnew)$ more quickly. Thus, we will spend fewer iterations solving (LP) because we only solve it approximately, and we will also spend fewer iterations solving $(LPnew)$ because we start off with an iterate which is more centered.

We can terminate solution of (LP) early if we can find cutting planes which are violated by the current solution. In fact, if we can find cutting planes which are violated by this current iterate, they may well be *deeper cuts* and cut off more of the feasible region, because the iterate is closer than the optimal solution to the center of the polyhedron. We may also be able to find more good cutting planes at this early iterate.

Consider, for example, the perfect matching problem on the graph in figure 4, where edge weights are the Euclidean lengths. The optimal matching in this graph

Fig. 4. An illustration of the phenomenon of nested odd sets

uses edges (v_1, v_{10}), (v_2, v_3), (v_4, v_5), (v_6, v_7), and (v_8, v_9). The optimal solution to the LP relaxation consisting of just the degree constraints and nonnegativity has

$$x_e = \begin{cases} 1 & \text{for edges } (v_2, v_3), (v_8, v_9) \\ 0.5 & \text{for edges } (v_1, v_4), (v_1, v_5), (v_4, v_5), (v_6, v_7), (v_6, v_{10}), (v_7, v_{10}) \\ 0 & \text{otherwise} \end{cases}$$

The separation routine for detecting violated odd set constraints involves finding connected components in the graph that only has edges where $x_e > \tau$ for some threshold $\tau \geq 0$. Thus, it would find the violated constraints for the oddsets

$\{v_1, v_4, v_5\}$ and $\{v_6, v_7, v_{10}\}$. If we search at an early iterate, we may well have $x_e > 0$ on edges (v_2, v_5) and (v_3, v_4), and in addition the values x_e on these edges are discernibly larger than those on the edges (v_4, v_6), (v_5, v_7) and (v_1, v_{10}). Thus, for appropriately chosen values of τ, we would find the violated odd set constraints given above and also the constraints corresponding to the odd sets $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_6, v_7, v_8, v_9, v_{10}\}$. Without these constraints, the solution to the relaxation is fractional; thus, these constraints are necessary, and the ability of the interior point method to find these constraints at an earlier stage means that one fewer LP relaxation has to be solved.

There are two disadvantages to looking for cuts before solving the current relaxation to optimality. Firstly, we may be unable to find any cuts, so the search is a waste of time. Secondly, the search may return cuts which are violated by the current iterate, but which are not violated by the optimal solution, so we may end up solving additional relaxations. The second disadvantage can mitigated by moving towards the optimal solution from the center of the polyhedron, making it unlikely that we will violate unnecessary cutting planes. One method for reducing the impact of the first disadvantage is to use a *dynamically altered tolerance* for deciding when to search for violated cutting planes. We only search when the duality gap drops below this tolerance. If we find a large number of violated constraints, we increase the tolerance, because we probably did not need to solve the relaxation to such a high degree of accuracy. If we only find a small number of violated constraints, we decrease the tolerance — we should solve the relaxations more accurately to obtain a better set of cutting planes as the relaxation becomes a better approximation to the convex hull of feasible integer points. As the number of violated cutting planes drops, it should also take fewer iterations to solve the next relaxation because the two relaxations should be close to each other.

Early termination is the most important technique for improving an interior point cutting plane algorithm. By using a dynamically altered tolerance for determining when to search for cutting planes, the time spent on unnecessary searches can be dramatically reduced.

2.1.3. *Restarting the algorithm and regaining primal feasibility*
When adding a cutting plane, we can obtain a new feasible interior dual iterate by setting $y_0 = 0$ and $w_0 = z_0 = \epsilon$ for some appropriate small positive value of ϵ. It is not straightforward to transform the old primal iterate into a feasible interior iterate in the new problem ($LPnew$). One option is to pick a strictly positive value for x_0 and then use a primal-dual infeasible interior point method, as described in, for example, Zhang [82]. To improve stability and performance, it is useful to also increase any small components of x, w, z and $u - x$ up to ϵ. It is also often useful to take a pure centering step when restarting. This works reasonably well, typically using about one third to one half of the number of iterations required to solve the problem from a cold start. However, the sequence of iterates often tends to move towards the center of the feasible region and away from the optimal solution while attempting to regain feasibility, with the result that it takes several iterations to solve ($LPnew$).

We have found that better performance can be obtained if it is possible to update

the primal iterate to a point that is known to be feasible and interior in ($LPnew$). Any interior point which is a convex combination of feasible integral points will satisfy all cutting planes, so it will be feasible in ($LPnew$). In addition, it will be interior in ($LPnew$) provided it satisfies all the cutting planes strictly. Any point in the relative interior of Q will be feasible and interior in ($LPnew$). It is often straightforward to find an initial point of this type; this point can be updated as the algorithm progresses, either by combining it with integral solutions that are found by heuristics, or by combining it with any iterate which is in the convex hull. The improved performance of the algorithm is because there is no need to balance the search for feasibility with the searches for optimality and centrality, and also because the point in the convex hull is often more centered than the point which is feasible in (LP).

One possible reason for the difficulty with restarting an interior point cutting plane method with an infeasible point can be developed from the work of, for example, Anstreicher [3], Mizuno *et al.* [60], and Zhang [82], who have all discussed interior point algorithms for linear programming which move towards feasibility and complementary slackness simultaneously. A common feature of the analysis of these algorithms is the exploitation of the fact that they move towards feasibility at least as fast as they move towards complementary slackness. When restarting directly from the approximate solution to the previous relaxation (LP), the primal infeasibility is $x_0 + \mid b_0 - a_0^T \hat{x} \mid$. The total complementary slackness is $\hat{x}^T \hat{z} + (u - \hat{x})^T \hat{w} + x_0 z_0 + (u_0 - x_0) w_0$. In order to get an iterate which is approximately centered, we could choose $w_0 = z_0 = 2\mu/u_0$ and $x_0 = u_0/2$. The complementary slackness will then be approximately $(2n+2)\mu$, so the ratio between infeasibility and complementary slackness will be large if μ is small. Other choices for x_0, w_0, and z_0 would require a tradeoff between centrality and this balance between infeasibility and complementary slackness. This may explain why it is hard to get very fast convergence from the infeasible warm start generated in a cutting plane algorithm.

Ye, Mizuno and Todd [81] introduced a skew symmetric self dual algorithm for linear programming. Further investigation of this algorithm is described in, for example, [76, 77]. This algorithm has the property that it is easy to generate a perfectly centered initial iterate. This has the potential to make this algorithm very useful in a cutting plane framework, because we can take the iterate for the previous relaxation, modify it slightly, and obtain an almost centered iterate for the new relaxation. This is an issue that needs more computational investigation.

2.1.4. *Adding many constraints at once*

It is usual in practice to add many constraints at once. If we add many constraints to the relaxation (LP), we obtain the new relaxation

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & = & b \\ & A_0 x + x_0 & = & b_0 \qquad (LPmany) \\ & 0 \leq x & \leq & u \\ & 0 \leq x_0 & \leq & u_0 \end{array}$$

for some appropriate upper bound u_0 on the new slack variables x_0. Note that x_0 and u_0 are now vectors and A_0 is a matrix of the appropriate dimension. The corresponding dual problem is

$$\begin{array}{lrcrcrcl} \max & b^T y & & - & u^T w & - & u_0 w_0 & \\ \text{subject to} & A^T y & + A_0^T y_0 & - & w & + & z & = c \\ & & y_0 & - & w_0 & + & z_0 & = 0 \qquad (LDmany) \\ & & & & w, z & \geq & 0 & \\ & & & & w_0, z_0 & \geq & 0 & \end{array}$$

where y_0, w_0, and z_0 are all vectors with dimension equal to the number of added constraints. If we have an interior feasible solution to (LP) and (LD) then we can get an interior feasible solution to $(LDmany)$ by setting $y_0 = 0$ and $w_0 = z_0 = \epsilon e$ for some small positive constant ϵ. If we set x_0 to some positive vector, we can then restart using an infeasible interior point method. Alternatively, it may be possible to update the primal solution to a point which is feasible in $(LPmany)$. Thus, the algorithm can be restarted when many constraints are added in much the same way that it can be restarted when only one constraint is added.

A very large number of constraints is occasionally generated when solving some problems. If all of these constraints are added to the relaxation at once, the algorithm slows down for several reasons:

- The LP relaxation has been changed dramatically, so the interior point algorithm takes several iterations to approach a new center, and then several iterations to move towards the optimal solution.
- The constraint matrix becomes far larger, so the computational time required at each iteration increases.
- It takes more iterations to solve a linear program with a large number of constraints than one with a small number.

Thus, it is advisable to only add a subset of the constraints.

Perhaps the simplest method for choosing which constraints to add is to add the constraints that are most violated by the approximate solution $\hat{x}$ to (LP). The disadvantages of this method are that it may add a large number of very similar constraints, or that a constraint with a large violation may not actually be that important.

In some situations, it is useful to consider the effect of a constraint on the structure of the nonzeroes in the constraint matrix when deciding whether to add a constraint. It is desirable to keep the Cholesky factors of the product AA^T sparse in order to be able to calculate the projections quickly, and if we add a lot of constraints which all use the same variables then these constraints are going to lead to fill-in in AA^T. This was found to be an important consideration when solving the linear ordering problem [57]. For this problem, it was found to be very advantageous to add only a subset of the violated 3-dicycle constraints that was pairwise arc-disjoint — if the same arc appears in two different 3-dicycles then the inner product between the rows of the constraint matrix representing the two constraints is nonzero, so there is fill-in in AA^T.

An alternative measure for the importance of a constraint has been proposed by Atkinson and Vaidya [6], who considered convex feasibility problems. In their

setting, the current relaxation has the constraints $Gx \leq g, 0 \leq x \leq u$ for some upper bound u, and they consider a possible constraint of the form $h^T x \leq h_0$. They suggest looking at the interplay of the constraint with the Hessian of the barrier function used in an interior point method; in particular, if G has full column rank then the quantity $h^T(S^2 + G^T D^2 G)^{-1} h$ should be large if the constraint is important, where S and D are appropriate diagonal matrices. The cutting plane algorithm described earlier in this section calculates projections using matrices of the form $\tilde{S}^2 + G\tilde{D}^2 G^T$ rather than $S^2 + G^T D^2 G$ for some appropriate diagonal matrices $\tilde{S}$ and $\tilde{D}$; however, these products are related (see, for example, Birge *et al.* [9]), so a similar test could be derived using quantities already available in the algorithm. To the best of our knowledge, nobody has investigated this measure computationally.

2.1.5. *Dropping constraints*

Computationally, it is useful to be able to drop constraints because this will reduce the time required for each iteration by reducing the size of the constraint matrix. An additional benefit of dropping constraints is that smaller linear programs require fewer iterations to solve. The simplest way to decide whether to drop a constraint is to check its slack value — if the current iterate satisfies the constraint easily, then the constraint is a candidate to be dropped. When a constraint is dropped, the structure of the matrix AA^T is changed, so it is necessary to calculate a new ordering of the columns of this matrix for the Cholesky factorization. Because of this work, the algorithm described in [57] only dropped constraints when other constraints were added to the matrix.

The Atkinson-Vaidya measure of the importance of a constraint can be used to decide whether to drop a constraint. To the best of our knowledge, nobody has attempted to use this heuristic, principally because of the work required to calculate this measure.

2.1.6. *Primal heuristics*

Primal heuristics are algorithms which generate integer solutions to (IP) from fractional solutions to (LP). If they are very cheap, it is possible to call them at every iteration; however, it is usually more cost effective to only call them when the separation heuristics are also called.

For many problems, it is usually considerably easier to find the optimal solution than to prove that it is optimal. The primal heuristics may well find the optimal solution, and the cutting plane method can then be used to prove that that solution is optimal. If the objective function vector c is integer then we do not need to proceed any further with the cutting plane algorithm once the lower bound provided by the value of (LD) is within one of the value of the best known feasible solution to (IP) provided by the primal heuristics. Thus, the primal heuristics may well save us work by letting us terminate without having to construct a relaxation which has an optimal solution that is feasible in (IP).

If the interior point method is converging to a point in the interior of the optimal face of Q then the primal heuristics may well provide one of the optimal solutions to (IP), so we can terminate the algorithm, because the value of the relaxation will agree with the value of the integer solution. Without the primal heuristics, we may

search futilely for cutting planes, and be forced to branch. Thus, a good primal heuristic algorithm can save a great deal of time.

Another use for the primal heuristic is to modify the restart point in Q. It can be modified to be slightly closer to the integer point generated by the primal heuristics.

2.1.7. *Multiple optimal solutions and degeneracy*
If the integer programming problem has multiple optimal solutions, then it is likely that the iterates generated by an interior point cutting plane method will converge to a point in the interior of the optimal face of Q. In this case, the primal heuristics can usually be used to find an optimal solution to (IP). Alternatively, Megiddo's approach [51] can be used to find an optimal basic feasible solution in Q which will then solve (IP). Megiddo's algorithm is strongly polynomial.

It is possible that the fractional optimal solution may provide more information than one of the optimal integer solutions. For example, in the linear ordering problem, an optimal fractional solution corresponds to a *partial* ordering of the nodes, and every ordering which agrees with this partial ordering is optimal.

For a survey of the effects of degeneracy on interior point methods for linear programming problems, see Güler *et al.* [30]. Degeneracy does not appear to be as serious a problem for interior point methods as it is for the simplex algorithm. The principal practical effect of degeneracy on an interior point method is to cause possible numerical problems because of numerical instability and ill-conditioning of the projection matrix. Many integer programming problems have highly degenerate relaxations, so an interior point method might be particularly well suited to such problems.

2.1.8. *Fixing variables*
Simplex branch and cut methods can use reduced costs to fix variables at zero or one in the following manner. Let r be the reduced cost of a variable which is currently zero in the solution to the relaxation. Let v^{UB} be the value of the best known feasible solution to (IP) and let v^{LB} be the value of the relaxation. If $r > v^{UB} - v^{LB}$ then this variable must be zero in any optimal solution to (IP). A similar test can be given for fixing a variable at one.

The reduced costs are not available at the current interior solution to the relaxation (LP), but the dual variables are available, and these can be used to fix variables. If z_i is the dual variable corresponding to the primal variable x_i and if $\bar{v}^{LB}$ is the value of the current feasible solution to (LD) then x_i can be fixed at zero if $z_i > v^{UB} - \bar{v}^{LB}$. A similar test can be used to fix variables at one. See Mitchell [55] for more details.

2.1.9. *The complete algorithm*
We summarize the complete algorithm in figure 5.

2.1.10. *Algorithms which only require positive dual iterates*
After adding constraints to the primal problem, the current primal iterate is infeasible and the dual variables corresponding to the additional columns have value zero.

1. **Initialize.** Set up the initial relaxation. Find initial interior primal and dual points. If possible, find a feasible point in Q. If possible, find a restart point in the relative interior of Q for use in Step 8.
2. **Inner iteration.** Perform one iteration of the primal dual algorithm. While the duality gap is above the tolerance τ, repeat this step.
3. **Primal heuristics.** Use the primal heuristics to try to improve on the current best solution to (IP). If successful, also update the known feasible point in the relative interior of Q.
4. **Look for cutting planes.** Use heuristics and/or exact algorithms to find cutting planes, if any exist.
5. **Add cutting planes.** If any cutting planes were found in Step 4 then add an appropriate subset.
6. **Fix variables.** If possible, fix variables at zero or one.
7. **Drop cutting planes.** If any cutting plane appears to no longer be important, drop it.
8. **Modify current iterate.** Increase any small components of w and z to a small value ϵ. If a feasible point in the relative interior of Q is known, update the primal solution to this point. Otherwise, increase any small components of x and the vector of primal slacks to ϵ. Modify the barrier parameter. If not using the predictor-corrector algorithm, perform one pure centering step to get a better initial point for the next relaxation. Return to Step 2

Fig. 5. An interior point cutting plane algorithm

Some barrier methods, affine methods, and projective methods have been developed for solving problems using either just the primal variables or just the dual variables, and such methods can be used to solve the dual problem. Mitchell and Todd [59] considered using a projective algorithm applied to the dual problem in a cutting plane algorithm. This algorithm does not require primal iterates, and only uses the value of a primal feasible point in calculating the direction at each iteration. Heuristics are used to generate primal solutions. When cutting planes are added to the primal problem, a strictly positive dual iterate is obtained by first moving in a direction which is guaranteed to increase the additional dual variables, and the algorithm is then restarted from this new point. They obtained reasonable computational results on matching problems, in terms of the number of iterations required. They also obtained promising results on linear ordering problems; for details see [54].

Goffin *et al.* [25, 22, 21] have also experimented with algorithms which only require primal iterates in their algorithms for nonsmooth optimization and multicommodity flow problems. For a discussion of their algorithm for multicommodity flow problems see section 5.1.

2.2. Interior point branch and bound methods

Branch-and-bound is a method for solving an integer programming problem (IP) by solving a sequence of linear programming problems. The subproblems can be regarded as forming a tree, rooted at the linear programming relaxation (LP) of

the integer programming problem. As we move down the tree, more and more integer variables are fixed at their values. We provide a very brief description of the technique in order to highlight some aspects which prove important when using an interior point method. For a more detailed discussion of branch-and-bound and the options available, see, for example, Parker and Rardin [64].

As with interior point cutting plane methods, one of the important features of a competitive interior point branch and bound algorithm is that the relaxations are not solved to optimality but are terminated early. This is usually possible, as we now argue. When using branch-and-bound, one of four things can happen at each node of the tree. The subproblem could be infeasible; in an interior point method this can be detected by finding a ray in the dual problem. The subproblem could have optimal value worse than the value of a known integer feasible solution, so the node is fathomed by bounds; in an interior point method, this can usually be detected well before the subproblem is solved to optimality. The optimal solution could be an integer solution with value better than the best known solution; in this case we need to solve the subproblem to optimality, but the node is then fathomed. The final possibility is that the optimal solution to the subproblem has optimal value smaller than the best known solution, but the optimal solution is not feasible in the integer program; in this case, it is possible to use heuristics based upon the basis identification techniques described in El-Bakry *et al.* [17] to determine that one of the integer variables is tending to a fractional value, and therefore that we should probably not solve the relaxation to optimality but should branch early.

It should be noted that in only one case is it necessary to actually solve the relaxation to optimality, and in that case the node is fathomed. When we branch early, one constraint in the dual relaxation (LD) is dropped, so the previous solution to (LD) is still feasible. One variable in (LP) is fixed, so infeasibilities are introduced into (LP). Despite this, it is still possible to solve the child node quickly [10, 11].

A branch and bound interior point algorithm has the form given in figure 6.

Notice that we do not necessarily maintain primal and dual feasibility throughout the algorithm. This means that some of the tests for convergence have to depend upon whether the iterates are feasible.

If the relaxation is infeasible that is detected in Step 3d. If the relaxation has an optimal value that is worse than that of the best known integer solution then that is detected in Step 3c. In these two situations, the solution of the relaxation should not take very many iterations, and the node is then fathomed. If the relaxation has an integer solution which is better than the best known solution, then this relaxation is solved to optimality and the node is fathomed in Step 3b. Here, the solution of the relaxation may take several iterations, because an exact solution is needed, but the node is then fathomed. Of course, it is possible that the rounding heuristics provide the optimal solution to the relaxation early, and this is sufficiently close in value to the dual value that the solution to the relaxation can be terminated early.

The one other possibility for a node of the tree is that the optimal solution to the relaxation is fractional, but it has value smaller than that of the best known integer solution. We discuss this situation further in section 2.2.1.

If it is necessary to branch then two child nodes will be created. It may eventually be necessary to solve the relaxations at these child nodes, so it will be necessary to

1. **Initialize:** Pick an initial relaxation. Choose an initial primal and dual iterate, using, for example, the method of Lustig *et al.* [50]. Set a tolerance τ for optimality. Initialize the branch and bound tree to contain only the initial relaxation.
2. **Pick a node:** Select a node of the branch and bound tree to solve next. Find an initial solution for this node. (We discuss a method for restarting at a node later.)
3. **Perform an interior point iteration:** Perform one iteration of the interior point method for the current node.
   a) Attempt to find a good integer solution by rounding the fractional solution in an appropriate manner. If this gives the best solution found so far, store it, and take its value as an upper bound on the optimal value of (IP).
   b) If the duality gap of the current relaxation is smaller than τ and if the primal value is better than the upper bound on the optimal value of (IP) then
      − If the primal solution is integral, fathom this node, update the upper bound on the optimal value of (IP), and return to Step 2.
      − If the primal value is nonintegral, go to Step 5.
   c) If the dual value of the current node is greater than the best known upper bound, prune this node.
   d) If we can find a dual ray, showing that the primal problem is infeasible, prune this node.
   e) If the current solution is dual feasible and if the relative primal infeasibility is smaller than some tolerance, go to Step 4.

   Repeat this step.
4. **Check for nonintegrality:** Check to see whether the solution to the current node appears to be fractional. If it appears that the solution to this node will be fractional, and if it appears unlikely that this node will be fathomed by bounds, go to Step 5; otherwise return to Step 3.
5. **Branch:** Split the current node into two nodes. Pass the current primal and dual solution onto the child nodes as a warm start. Go to Step 2.

Fig. 6. An interior point branch and bound algorithm

start an interior point method on these relaxations. The simplex method can start directly from the solution to the parent node, using the dual simplex algorithm. It is necessary to modify the solution to the parent slightly before restarting with an interior point method, in order to obtain a slightly more centered point. We discuss restarting in more detail in section 2.2.2.

The linear programming problems generated in a branch-and-bound tree can suffer from considerable degeneracy, which can greatly slow the simplex method. Degeneracy is generally not such a problem for interior point methods, and at least one commercial package has installed a switch to change from simplex to an interior point method within the branch-and-bound tree if difficulties arise. Applegate *et al.* [5] also implemented such a switch. For a discussion of the effects of degeneracy on interior point methods for linear programming, see Güler *et al.* [30].

One cost of using the simplex algorithm in a branch and bound method is that it is necessary to perform an initial basis factorization for each child subproblem. The cost of this is clear when examining, for example, the performance of the branch and bound code for OSL [33] on solving integer programming problems: it often happens that the average time per iteration is about three times larger for subproblems than it is for the root node of the branch and bound tree. This extra time is almost completely attributable to the overhead required at each node to calculate a new basis factorization. A comparable slow down does not happen with interior point methods. One way to avoid this overhead would be to store the basis factorization of each parent node, but this usually requires too much storage and is hence impracticable. Of course, part of the reason that the slow down is so noticeable is that the simplex algorithm requires far fewer iterations to solve subproblems than to solve the root node, because the optimal solution to the parent node does provide a good simplex warm start for the child subproblem. At present, it does not seem possible to get a similar reduction with an interior point method, but the fact that the basis refactorization is so expensive means that it is not necessary to obtain as good a reduction in the number of iterations as enjoyed by the simplex algorithm.

Interior point branch and bound methods are somewhat competitive with simplex based branch and bound algorithms on some problems, including facility location problems [11]. These problems have a large number of continuous variables and a relatively small number of integer variables, so the LP relaxations are large yet the branch and bound tree is small. It is necessary to have large relaxations for an interior point method to compete with a simplex method. In addition, we need the problem to be solvable on current hardware, so the branch and bound tree can not grow too large, so we need to have only a small number of integer variables. Because of the early termination of the solution of the relaxations, not as much information is available at each node, so the pseudocosts [64] used to select the next branching variable and the next node can not be calculated as accurately. This is another reason why an interior point method can currently only be competitive on problems with a small proportion of integer variables, because in this situation the effect of a bad choice of branching variable is not so dramatic. More research is needed to find good, reliable pseudocosts in an interior point branch and bound method.

To conclude this section on interior point branch and bound methods, we discuss restarting the algorithm (subsection 2.2.2) and terminating the solution of the relaxation early when the iterates are tending towards a fractional solution (subsection 2.2.1).

2.2.1. *Terminating the current relaxation early*

In this section, we assume that the optimal solution to the relaxation is fractional, but it has value smaller than that of the best known integer solution. In this situation, we can use basis identification techniques [17], as mentioned in Step 4. This will usually save a few iterations on the solution of the current node, and it will also result in termination at a more centered iterate, which will result in a better initial iterate for each child of this node.

There is a risk associated with attempting to stop solution of the parent subproblem early: the parent may be split into two child subproblems, when it might have

been possible to prune the parent if it had been solved to optimality. This could happen if the parent subproblem has worse objective function value than that of the best known feasible solution to (IP), or if it is infeasible, or if it has an integer optimal solution. (Notice that the last possibility is unlikely if the basis identification techniques are working well.) Therefore, it is wise to include some safeguards to attempt to avoid this situation. Upper and lower bounds on the value of a subproblem are provided by the values of the current primal and dual solutions, respectively, and these can be used to regulate the decision to branch.

There are three tests used in [11] to prevent branching too early: the dual iterate must be feasible, the relative primal infeasibility must be no greater than 10%, and the dual objective function must not be increasing so quickly from iteration to iteration that it is likely that the node will be fathomed by bound within an iteration or two. Dual feasibility can usually be maintained throughout the branch and bound tree so the first criterion is basically just a technicality. Every time a variable is fixed, primal infeasibility is introduced; if the initial iterate for a subproblem is a good warm start, primal infeasibility can be regained in a few iterations. Thus, the important criterion is the third one, regarding the increase in the dual value. This criterion prevents branching if the difference between the dual value and the value of the incumbent integer solution has been reduced by at least half in the last iteration, provided the current primal value is greater than the current integer solution if the current primal iterate is feasible.

2.2.2. *Warm starting at a node of the tree*

The exact method used for restarting at a node of the branch and bound tree depends upon the interior point algorithm used. In this section, we assume that the primal-dual barrier method is being employed (see, for example, Lustig *et al.* [49]). It should be noted that many of the observations we make will also be applicable if other interior point methods are used.

Assume a child problem has been created by fixing the variable x_0 at 0 or 1 in the parent problem

$$\begin{array}{lrcl} \min & c^T x + c_0 x_0 & & \\ \text{subject to} & Ax + a_0 x_0 & = & b \qquad (LPparent) \\ & 0 \le x, x_0 & \le & e, \end{array}$$

where A is an $m \times n$ matrix, a_0 and b are m-vectors, c and x are n-vectors, and c_0 and x_0 are scalars. The child problem has the form

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & = & \bar{b} \qquad (LPchild) \\ & 0 \le x & \le & e, \end{array}$$

where $\bar{b} = b$ if x_0 is fixed at zero, and $\bar{b} = b - a_0$ if x_0 is fixed at one. An approximate solution $x = x^*$, $x_0 = x_0^*$ to $(LPparent)$ is known. Since we created this particular child problem, x_0^* must be fractional, so x^* is probably infeasible in $(LPchild)$. If

we examine the dual problems

$$\begin{array}{lrrrrrrll}
\max & b^T y & - & e^T w & - & w_0 & & & \\
\text{subject to} & A^T y & - & w & + & z & = & c & (LDparent) \\
& a_0^T y & - & w_0 & + & z_0 & = & c_0 & \\
& & & w, w_0, & & z, z_0 & \geq & 0, &
\end{array}$$

and

$$\begin{array}{lrrrrrll}
\max & \bar{b}^T y & - & e^T w & & & & \\
\text{subject to} & A^T y & - & w & + & z & = & c \quad (LDchild) \\
& & & & & w, z & \geq & 0, &
\end{array}$$

we notice that the approximate solution $y = y^*$, $w = w^*$, $z = z^*$ (and $w_0 = w_0^*$, $z_0 = z_0^*$) to ($LDparent$) is still feasible in ($LDchild$) — all that has changed is the objective function value. Therefore, it is possible to restart the algorithm using an infeasible interior point method. It may be that some of the components of x^* are very close to zero or one, and these components should be modified to give a slightly more interior point, with small components being increased and large components being decreased. Similarly, if some components of the dual variables w^* and z^* are smaller than some tolerance, they should be increased, at the cost of making the initial iterate for the subproblem slightly more infeasible. Thus, we can start the interior point method on the child problem if we have stored x^*, y^*, and w^*. It may be beneficial to use a pure centering step first before updating the barrier parameter μ in a standard manner.

It may be possible to start the solution of the child problem from an iterate for ($LPparent$) which was found before x^*. This earlier iterate would be further from optimality for ($LPparent$) than x^*, but it may be a better initial solution to ($LPchild$) just because it is more centered, with the nonbasic components being somewhat larger. Preliminary experiments show that this approach may hold some promise, but it needs considerably more investigation.

3. A potential function method

Karmarkar and various coauthors [43, 40] proposed a novel approach to solve integer programming problems. This approach examines a related continuous optimization problem and uses this continuous problem to approach a solution to the original integer program. We outline their approach in this section.

The algorithm transforms the problem to a hard equivalent quadratic programming problem (see section 3.1). At each iteration, given a strictly feasible iterate for the hard QP, it solves an easier nonconvex quadratic programming problem to find a direction in the hard quadratic problem (see section 3.2), and it moves in the direction to obtain a new iterate; it also rounds the new point to an integer point which is then checked for feasibility (see section 3.3). It may happen that the algorithm converges to a noninteger solution; a method for handling this situation is described in section 3.4. Computational experience with the algorithm is discussed in section 3.5.

Finally, in section 3.6, we describe a method for solving quadratic integer programming problems.

3.1. Transforming the problem

The integer programming feasibility problem can be stated as

Find a point $\hat{x}$ in $\Re^n$ satisfying $\hat{A}\hat{x} \leq \hat{b}, \hat{x}_i \in \{0,1\}, i = 1, \ldots, n,$ $(IPfeas)$

where $\hat{A}$ is an $\hat{m} \times n$ matrix and $\hat{b}$ is an $\hat{m}$-vector. This can be scaled to give the equivalent problem

Find a point x in $\Re^n$ satisfying $\bar{A}x \leq \bar{b}, x_i \in \{-1,1\}, i = 1, \ldots, n,$

where $x = 2\hat{x} - e$, $\bar{A} = \hat{A}$, and $\bar{b} = 2\hat{b} - \hat{A}e$. This can be transformed to the nonconvex quadratic programming problem

$$\begin{array}{lrcl} \min & n - x^T x & & \\ \text{subject to} & \bar{A}x & \leq & \bar{b} \\ & -e \leq x & \leq & e \end{array} \qquad (\overline{QP})$$

The problem $(\overline{QP})$ is written equivalently as

$$\begin{array}{lrcl} \min & n - x^T x & & \\ \text{subject to} & Ax & \leq & b \end{array} \qquad (QP)$$

where $A = [\bar{A}^T \ I \ -I]^T$ and $b = [\bar{b}^T \ e^T \ e^T]^T$. We define $m = \bar{m} + 2n$, so A is an $m \times n$ matrix and b is an m-vector. The feasible solutions to $(IPfeas)$ correspond to points in (QP) with value zero. Any other feasible point to (QP) has strictly positive value. The algorithm is initialized with any strictly feasible point for the problem (QP).

3.2. Finding the next iterate

The algorithm uses an interior point method to attempt to minimize the potential function

$$m \log(n - x^T x) - \sum_{i=1}^{m} \log s_i$$

where $s = b - Ax$. The point x^* is a global minimizer of this potential function if and only if it is a feasible solution to $(IPfeas)$.

At each iteration, a quadratic approximation to the potential function is constructed, and a direction is obtained by finding the minimum of this quadratic approximation over an inscribed sphere. The gradient of the potential function at the current iterate x^k is given by

$$h = -(2m/f_0)x^k + A^T S^{-1} e \tag{8}$$

where $f_0 := n - {x^k}^T x^k$ and S is the diagonal matrix with diagonal elements the entries of the vector s. The Hessian of the potential function at the current point is

$$H = -(2m/f_0)I - (4m/f_0^2)x^k {x^k}^T + A^T S^{-2} A. \tag{9}$$

It should be noted that the Hessian is a dense matrix, due to the outer product of the vector x^k with itself. The subproblem which is solved to find the direction Δx

is then

$$\begin{array}{ll} \min & (1/2)\Delta x^T H \Delta x + h^T \Delta x \\ \text{subject to} & \Delta x A^T S^{-2} A \Delta x \leq r^2 \end{array}$$

If $0 < r \leq 1$ then a feasible point Δx in this subproblem leads to a feasible point $x + \Delta x$ in the problem (QP). The solution to this subproblem depends on the eigenvalues of the Hessian matrix H in the norm defined by the matrix $A^T S^{-2} A$; for details, see [43, 40]. This subproblem can be solved in polynomial time. For methods for solving it, see [43, 40] or Ye [79]. Kamath *et al.* originally proposed taking a step of a fixed length in the direction Δx; it was subsequently pointed out by Shi and Vannelli [73] that the algorithm can be considerably enhanced by using a line search to determine a step length.

Van Bentham *et al.* [7, 75] developed a variant of this algorithm to solve the radio link frequency assignment problem. Their refinements to the original algorithm included a method to deal with equality constraints, and the use of a barrier method rather than a potential function method so that the Hessian matrix retains the sparsity structure of AS^2A^T. The structure of their problem is such that all the slack variables in the original problem $(IPfeas)$ must also be binary. They used this observation to develop a quadratic objective function which enabled them to eliminate the inequality constraints.

3.3. Obtaining an integer point

At each iteration k, a new strictly feasible point x^k for (QP) is obtained. This point is then rounded to obtain an integral point $\tilde{x}^k$. The simplest method to obtain $\tilde{x}^k$ is to set

$$\tilde{x}_i^k = \begin{cases} 1 \text{ if } x_i^k \geq 0 \\ -1 \text{ if } x_i^k < 0 \end{cases}$$

Other rounding schemes can be used. For example, we can choose $\tilde{x}_i^k$ by examining whether x_i^k is increasing or decreasing. For some problems, the structure of the problem suggests a natural rounding scheme; for example, Van Bentham *et al.* [7] have suggested several rounding schemes for the radio link frequency assignment problem. If the rounded point $\tilde{x}^k$ is feasible in (QP) then we can terminate the algorithm with success: $\tilde{x}^k$ leads to a feasible point in the original problem $(IPfeas)$.

3.4. Avoiding local minimizers

The nonconvex quadratic problem may have local minimizers which are fractional points and therefore not global minimizers. If this happens, it may be that the rounded solution $\tilde{x}^k$ remains infeasible. In this situation, it is possible to add a constraint

$$\tilde{x}^{k^T} x \leq n - 2 \tag{10}$$

Notice that every $\{-1, 1\}$ point satisfies this constraint except $\tilde{x}^k$. After adding this constraint, the algorithm is restarted from a strictly feasible point. It is best to restart from scratch because the objective function is nonconvex, so we want to generate a sequence of iterates that does not lead to the local minimizer. There is no guarantee that equation (10) will cut off the local minimizer, but Karmarkar claims that the addition of this constraint is usually sufficient to push the sequence

of iterates in a different direction, so that the algorithm terminates at a different point.

3.5. Computational experience

The algorithm was used to solve set covering problems [43]. It was also used to solve satisfiability problems from inductive inference [40], obtaining promising results compared to an implementation of the Davis-Putnam procedure [13] (an implicit enumeration procedure which is similar to branch and bound). Shi and Vannelli [73] improved on these results by incorporating a linesearch.

Van Bentham *et al.* [7, 75] solved radio link frequency assignment problems using a potential reduction method. By cleverly exploiting the structure of their model, they were able to develop a variant of the algorithm which solves problems with several thousand variables and constraints.

The algorithm can be used to solve optimization problems by incorporating constraints of the form $c^T x \leq K$ and resolving with different values of K. It should be noted that this method is essentially a heuristic, in that it can not determine that an infeasible instance is really infeasible. Thus, the algorithm does not lead to a guarantee that the optimal solution has been found when solving optimization problems.

3.6. Quadratic integer programming problems

Kamath *et al.* [37, 38] also investigated using a potential function interior point method to solve quadratic integer programming problems of the form

$$\begin{array}{ll} \max & x^T Q x \\ \text{subject to} & x_i = -1, 1, \quad i = 1, \ldots, n \end{array}$$

where Q is a symmetric $n \times n$ matrix and x is an n-vector. This problem is NP-Hard if Q has at least one positive eigenvalue [63]. The graph partitioning problem can be modeled in this manner [37]. They were able to use their method to obtain upper bounds on the optimal value of the quadratic integer programming problem in polynomial time [38].

This approach encloses the feasible region in an ellipsoid, finds the maximum value of the objective function in the ellipsoid, and then modifies the ellipsoid appropriately. The maximum value of the objective function over the ellipsoid is the largest eigenvalue of an appropriate matrix. By modifying the ellipsoid appropriately, it is possible to obtain a reasonable upper bound on the optimal value of the quadratic problem.

4. Solving network flow problems

4.1. Introduction

Network flow problems arise in the shipment of commodities (for example, oil, telephone calls, or cars) over a network from sources to destinations. For example, oil is shipped from oil fields to refineries to its eventual destination, telephone calls are routed from the caller to the destination over the telephone company's network, and cars are routed through a city between residences, offices, and commercial buildings.

Many of these network flow problems can be modeled as linear programming problems with a constraint matrix with a special structure. Historically, these problems have been solved by using specialized versions of the simplex algorithm designed to exploit the structure of the constraint matrix. Recently, several researchers have experimented with using an interior point algorithm to solve these linear programming problems, with results that are very competitive with the network simplex method. We describe some of the computational results in section 4.3. As is to be expected, the interior point method has to be modified in order to exploit the structure of the constraint matrix fully. The principal modification is in the use of a preconditioned conjugate gradient method to calculate the necessary projections at each iteration. We describe this and other modifications in section 4.2.

An overview of work in this area is contained in Resende and Pardalos [69]. An implementation of the dual affine method is described in Resende and Veiga [70, 71]. Other interior point algorithms are investigated in Portugal *et al.* [65, 66]. For an introduction to network flow problems and applications, see the book by Ahuja *et al.* [1].

We now describe the *minimum cost network flow* problem. Given a directed graph $G = (V, E)$ with m vertices V and n arcs E, the arc from vertex i to vertex j is denoted by (i, j). Flow moves around the network along the directed arcs. If more flow is produced at a node i than is consumed at that node, then the node is called a *source* node. If more flow is consumed at a node i than is produced at that node, then the node is called a *sink* node. Any node which is neither a source node nor a sink node is called a *transshipment* node. Let b_i denote the net required flow out of node i; if $b_i > 0$ then node i is a source, if $b_i < 0$ then node i is a sink, and if $b_i = 0$ then node i is a transshipment node. For a feasible flow to exist, it is necessary that $\sum_{i \in V} b_i = 0$. The flow must satisfy Kirchhoff's Law of flow conservation: the total flow out of node i must equal the sum of b_i and the total flow into node i for each node i. There is a cost c_{ij} for each unit of flow shipped along arc (i, j). We assume without loss of generality that the lower bound on each arc is zero (see [1]), and we denote the upper bound on arc (i, j) by u_{ij}. The minimum cost network flow problem is then to meet the demands at the nodes at minimum cost while satisfying both Kirchhoff's Law and the bounds on the edge capacities. This can be expressed as the following linear programming problem:

$$\min \quad \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{11}$$

$$\text{subject to} \sum_{(i,j) \in E} x_{ij} - \sum_{(j,i) \in E} x_{ji} = b_i \text{ for all } i \in V \tag{12}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \text{for all } (i,j) \in E \tag{13}$$

where x_{ij} denotes the flow on arc (i, j). Usually, the problem data is integer, in which case one of the optimal solutions to this linear program will be integer.

We let A denote the *node-arc* incidence matrix of the graph. Each column of A corresponds to an arc (i, j) and has an entry "1" in row i and an entry "-1" in row j, with all the remaining entries being zero. Notice that the constraint (12) can be written $Ax = b$. The rank of the matrix A is equal to the difference between

1. **Given:** Constraint matrix A, diagonal matrix D, preconditioner M, vector w, tolerance ϵ, want to calculate an approximate solution v to equation (14).
2. **Initialize:** Set $v = 0$, $r_0 = w$, $z_0 = M^{-1}r_0$, $p_0 = z_0$, $k = 0$.
3. **Main loop:** While the stopping criterion is not satisfied, repeat the following steps:
   a) Calculate $q_k = ADA^T p_k$.
   b) Calculate $\alpha_k = z_k^T r_k / p_k^T q_k$.
   c) Calculate $v_{k+1} = v_k + \alpha_k p_k$.
   d) Calculate $r_{k+1} = r_k - \alpha_k q_k$.
   e) Find z_{k+1} by solving $M z_{k+1} = r_{k+1}$.
   f) Calculate $\beta_k = z_{k+1}^T r_{k+1} / z_k^T r_k$.
   g) Calculate $p_{k+1} = z_{k+1} + \beta_k p_k$.
   h) Increase the iteration counter k by one.
4. **Stop:** Final solution is $v = v_k$.

Fig. 7. The preconditioned conjugate gradient algorithm

the number of vertices and the number of connected components of the graph. One redundant row can be eliminated for each connected component. For simplicity of notation we retain the redundant rows, but it should be understood that these rows have been eliminated.

Many combinatorial optimization problems can be formulated as minimum cost network flow problems. Examples include the assignment problem, the transportation problem, the shortest path problem, and the maximum flow problem. For more details, see [1]. The *multicommodity network flow problem* has more than one commodity moving through the network. See section 5 for a discussion of interior point multicommodity network flow algorithms. For background on the multicommodity network flow problem, see, for example, the books by Ahuja [1] and Minoux [53].

4.2. Components of interior point network flow methods

4.2.1. *Calculating the projections by using a preconditioned conjugate gradient method*

In any implementation of an interior point method, it is necessary to find a direction at each iteration by solving a system of equations

$$ADA^T v = w \tag{14}$$

where A is the $m \times n$ constraint matrix, D is a diagonal $m \times m$ matrix, v is an unknown m-vector, and w is a known m-vector. This is usually done by calculating a factorization of the matrix ADA^T. The matrix D and the vector w change from iteration to iteration; it is necessary to solve this system for more than one vector w at each iteration of some algorithms. Resende and Veiga showed that superior performance can be obtained on network flow problems if the system (14) is solved using a preconditioned conjugate gradient method.

A preconditioned conjugate gradient algorithm for solving (14) is given in figure 7. The preconditioner is denoted by M. The matrix M is a positive definite

matrix and it is chosen so that the matrix $M^{-1}(ADA^T)$ is less ill-conditioned than the original matrix ADA^T, and this should then improve the convergence of the conjugate gradient algorithm. Notice that Step 3e of the preconditioned conjugate gradient algorithm requires the solution of a system of equations involving M. The loop in the algorithm will probably be executed at least five to ten times for each calculation of a projection; thus, it is essential that it be considerably easier to solve a system of equations involving M than one involving ADA^T.

The structure of the network flow problem makes it possible to choose a good preconditioner M. The simplest preconditioner is to take M to be the diagonal of the matrix ADA^T. This is simple to compute, it makes the calculation of z_{k+1} trivial, and it can be effective. A more sophisticated preconditioner that exploits the nature of the problem is the *maximum weighted spanning tree (MST) preconditioner*. The edges of the graph are weighted by the corresponding elements of the diagonal matrix D, and a maximum weight spanning tree is then found using either Kruskal's algorithm or Prim's algorithm. (For descriptions of these algorithms for finding a maximum weight spanning forest, see [1].) Let S denote the columns of A corresponding to the edges in the maximum weight forest. The MST preconditioner is then

$$M = S\hat{D}S^T, \tag{15}$$

where $\hat{D}$ is a diagonal matrix containing the entries of D for the edges in the maximum weight spanning forest. The preconditioned residue system solved in Step 3e can be solved in time proportional to the number of vertices because the coefficient matrix S can be permuted into block triangular form.

The diagonal preconditioner appears to be better than the MST preconditioner in the early iterations of the interior point algorithm, in that it requires fewer steps of the preconditioned conjugate gradient algorithm to obtain a direction of sufficient accuracy. The situation reverses in later iterations. Initially, the MST preconditioner is a poor approximation to the matrix ADA^T because it puts too much emphasis on a few edges when it is not really possible to decide which edges are important. Eventually, the MST preconditioner becomes a better approximation to the matrix ADA^T, because it is possible to pick the right subset of the edges. Thus, Resende and Veiga [70, 71] use the diagonal preconditioner initially and switch to the MST preconditioner once the performance of the diagonal preconditioner falls off in their dual affine algorithm.

Portugal *et al.* [65, 66] have proposed a preconditioner based on an incomplete QR factorization of the matrix $D^{1/2}A^T$. This preconditioner appears to behave like the diagonal preconditioner in the early iterations, like the MST preconditioner in the later iterations, and to perform better than either of the other two preconditioners in the intermediate iterations. They have used this preconditioner in a primal-dual interior point algorithm for network flow problems. A preconditioner proposed by Karmarkar and Ramakrishnan [42] is based on selectively zeroing out elements of DA and also of the resulting modified product ADA^T, and then using the incomplete Cholesky factors of the approximation to this matrix as the preconditioner. This preconditioner also performs similarly to the diagonal preconditioner in the early iterations and similarly to the MST preconditioner in the later iterations.

We now discuss the stopping criterion used within the preconditioned conjugate

gradient algorithm. Recall that we want to solve equation (14) and that we use the vectors v_k as successive approximations to v. The check used in the papers discussed in this section examines the vector $ADA^T v_k$: if the angle θ between this vector and the right hand side vector w is close to zero, then we have solved equation (14) approximately. Resende and Veiga use the criterion that the preconditioned conjugate gradient algorithm can be halted if $|\ 1 - \cos\theta\ | < \epsilon_{cos}$, where ϵ_{cos} is 10^{-3} in early iterations of the interior point algorithm and is gradually decreased. The calculation of $\cos\theta$ requires about as much work as one conjugate gradient iteration, so it is only calculated every fifth iteration by Resende and Veiga. Additionally, the conjugate gradient method is halted if the size of the residual r_k becomes very small.

4.2.2. *Recovering the optimal flow*

Since the node arc incidence matrix is totally unimodular, every basic feasible solution to the network flow problem is integral provided b is integral, so every iterate generated by the simplex algorithm corresponds to an integral flow. The basic feasible solutions correspond to forests in the graph, with nonzero flow only on the edges in the forest. An interior point method usually converges to a point in the relative interior of the face of optimal solutions, so, if the optimal solution is not unique, an interior point method will not return an integral solution. We discuss methods used to obtain an integral optimal solution from the iterates generated by an interior point algorithm.

The maximum weight spanning tree found in the preconditioned conjugate routine can be used to guess an optimal solution: if the basic solution corresponding to this forest is feasible and the corresponding dual solution is also feasible then this solution is optimal. This works well if the solution is unique, but unfortunately it usually does not work well in the presence of multiple optimal solutions. If the primal basic solution is not feasible, then the current dual iterate is projected to give a point $\hat{y}$ which is complementary to the primal basic solution. The edges for which the dual slack has small magnitude for this dual vector $\hat{y}$ are then used to define a subgraph of the original graph. The edges in this subgraph are a superset of the edges in the forest. The flow on all edges not in this subgraph are assigned flow either 0 or equal to their upper bound. Resende and Veiga then attempt to find a feasible flow in the original graph by only adjusting flow on the edges in the subgraph. This can be done by solving a maximum flow problem and is guaranteed to give an integral flow if one exists.

As the interior point iterates converge towards optimality, this procedure will eventually give an integral optimal flow, provided the flows on the nonbasic edges are set correctly to 0 or their upper bound. Resende and Veiga examine the dual variable s_i corresponding to the nonbasic variable x_i and the dual variable z_i corresponding to the upper bound constraint on this variable x_i. If $s_i > z_i$ then variable x_i is set to zero; otherwise it is set equal to its upper bound. As the interior point method converges to optimality, this setting will eventually be optimal, and so the procedure outlined above will give an optimal integral solution to the network flow problem. The basis identification method of Megiddo [51] can be used to determine an optimal integral basic feasible solution once the interior point method is close enough.

4.3. Comparison with Network Simplex

Resende and Veiga [70, 71] have compared their code with version 3.0 of CPLEX Netopt [12]. They generated problems of seven different structures and of varying sizes for each structure. Two problem classes were generated using NETGEN [45], and the other problems were generated using various generators contributed to the First DIMACS Algorithm Implementation Challenge [14] (these generators are available from DIMACS at Rutgers University, at FTP site: **dimacs.rutgers.edu**). Both CPLEX Netopt and the code of Resende and Veiga were able to solve all of the generated problems, providing integer flows as output. In all but two classes, the interior point code was faster than CPLEX Netopt on the largest problems. On one of the remaining classes, the difference between the interior point code and the simplex code was decreasing as the problem size increased.

Thus, this work shows that interior point methods can outperform the simplex algorithm even in problem classes which lend themselves to sophisticated implementations of simplex. For an interior point method to be successful, it is necessary to use a preconditioned conjugate gradient method to calculate the projections, and to use various other techniques outlined here and discussed in more detail in [65, 66, 70, 71].

Many of the computational runs of these authors took several hours, and some of the runs with CPLEX Netopt took longer than a day. They used a number of workstations (each solving a separate problem) to obtain their results, and they were able to solve problems which are considered very large. It is on these large problems that the advantages of interior point methods become clear.

5. The multicommodity network flow problem

In this section, we describe two interior point approaches to multicommodity network flow problems. The nonlinear multicommodity network flow problem with separable increasing convex costs can be modelled as a nonlinear programming problem with linear constraints. The problems of interest generally create very large nonlinear programs. They arise in, for example, the areas of telecommunication, transportation, computer networks, and multi-item production planning. For more discussion of the multicommodity network flow problem, see the books by Ahuja [1] and Minoux [53]. (For a description of single commodity linear network flow problems, see section 4.)

5.1. A column generation algorithm for the multicommodity network flow problem

Goffin *et al.* [21] have described an interior point algorithm for solving nonlinear multicommodity network flow problems that has similarities to the Dantzig-Wolfe algorithm. Their algorithm is a column generation method, with new columns added either one at a time or in bunches. It approximately solves the nonlinear program that arises at each stage by using a projective method, specifically the de Ghellinck and Vial [19] variant of Karmarkar's algorithm [41]. The column generation subproblem is formulated as a shortest path problem and is solved using an implementation of Dijkstra's algorithm.

Goffin *et al.* [25, 22] have previously described column generation interior point

algorithms designed to solve nonsmooth optimization problems. The research described in this section is a continuation and extension of the work described in their earlier papers.

We are given a graph $G = (V, E)$ and a set of commodities I. We denote the node arc incidence matrix by A. For each commodity, there are source nodes where flow is produced, sink nodes where flow is consumed, and transshipment nodes, where the flow is in balance. The required net flow out of node v of commodity i is represented by d_v^i. Goffin *et al.* [21] restrict themselves to the case where each commodity has exactly one source node and one sink node. The capacity y_e of each arc e can be selected, with an associated convex cost $f_e(x_e)$; the upper bound on the capacity is denoted by γ_e. Associated with each commodity i and each arc e is a linear cost c_e^i for each unit of commodity i shipped along arc e. The multicommodity flow problem can then be formulated as

$$\begin{array}{lll} \min & \sum_{i\in I}\sum_{e\in E} c_e^i x_e^i + \sum_{e\in E} f_e(y_e) & & (16)\\ \text{subject to} & \sum_{i\in I} x_e^i \le y_e & \forall e \in E & (17)\\ & Ax^i = d^i & \forall i \in I & (18)\\ & x_e^i \ge 0 & \forall i \in I, e \in E & (19)\\ & 0 \le y_e \le \gamma_e & \forall e \in E. & (20) \end{array}$$

Here, x_e^i represents the flow of commodity i on arc e and y_e represents the total flow on arc e. We assume that the cost function $f_e(y_e)$ is strictly increasing and convex and that the costs c_e are nonnegative. The standard linear multicommodity flow problem corresponds to $f_e = 0$ for every arc e. Equation (17) is called the *coupling* constraint and equation (18) is the *flow conservation* constraint. Without equation (17), the problem would be separable. This equation is dualized in the Lagrangian relaxation developed for this problem. The Lagrangian multipliers for these constraints are nonnegative because of the structure of the objective function; with the use of an interior point cutting plane algorithm, the multipliers are actually always positive.

Dualizing the coupling constraints (17) gives the Lagrangian

$$L(x, y; u) := \sum_{i\in I}\sum_{e\in E} c_e^i x_e^i + \sum_{e\in E} f_e(y_e) + \sum_{e\in E} u_e(-y_e + \sum_{i\in I} x_e^i) \tag{21}$$

where u is the vector of Lagrange multipliers for the coupling constraints. Since the multicommodity flow problem is convex, it can be solved by solving the Lagrangian dual problem

$$\begin{array}{ll} \max & L^D(u)\\ \text{subject to} & u \ge 0 \end{array}$$

where the Lagrangian dual function $L^D(u)$ is given by

$$L^D(u) := \min\{L(x, y; u) : Ax^i = d^i, x^i \ge 0, i \in I,\ 0 \le y \le \gamma\} \tag{22}$$

The Lagrangian dual function $L^D(u)$ is a nonsmooth concave function. The Lagrangian dual problem can be solved by obtaining a polyhedral approximation

1. Select a point $(\bar{z}, \bar{u})$ in the localization set LOC.
2. Compute $L^D(\bar{u})$ and find a subgradient ξ of $L^D(\bar{u})$ at $\bar{u}$.
3. Add the inequality
$$z - \xi^T u \leq L^D(\bar{u}) - \xi^T \bar{u}$$
to the definition of the localization set LOC. If $L^D(\bar{u}) > \theta^\kappa_{\rm inf}$, then update $\theta^\kappa_{\rm inf}$ to $L^D(\bar{u})$.
4. Repeat the process until the termination criterion is satisfied.

Fig. 8. Column generation algorithm for the multicommodity flow problem

to the dual function using *supergradients* ξ. If $L^D(u)$ is differentiable at the point $\bar{u}$ then the only supergradient at that point is the gradient itself. In general, a supergradient ξ at $\bar{u}$ satisfies

$$L^D(u) \leq L^D(\bar{u}) + \xi^T(u - \bar{u}) \tag{23}$$

for all $u \geq 0$. Given points $u^k \geq 0$ and associated supergradients ξ^k for $k = 1, \ldots, \kappa$, the optimal value to the linear programming problem

$$\begin{array}{ll} \max & z \\ \text{subject to} & z - (\xi^k)^T u \leq L^D(u^k) - (\xi^k)^T u^k \quad \text{for } k = 1, \ldots, \kappa \end{array}$$

provides an upper bound $\theta^\kappa_{\rm sup}$ on the optimal value of the Lagrangian dual. It can be shown that if κ is large enough, then the solution to this linear program will solve the Lagrangian dual. The maximum of $L^D(u^k)$ for $k = 1, \ldots, \kappa$ provides a lower bound $\theta^\kappa_{\rm inf}$ on the optimal value of the dual, and any optimal solution lies in the *localization set*

$$LOC_\kappa = \{(z, u) : z \geq \theta^\kappa_{\rm inf}, z - (\xi^k)^T u \leq L^D(u^k) - (\xi^k)^T u^k, k = 1, \ldots, \kappa\}. \tag{24}$$

At each stage, the algorithm generates a point in the localization set. If this point is feasible in the Lagrangian dual, then we can update the lower bound $\theta^\kappa_{\rm inf}$. If the point is not feasible, then we can generate a new subgradient ξ and add the corresponding constraint to the localization set. In either case, the localization set is updated, so we then find a new point in this set and repeat the process until the gap between $\theta^\kappa_{\rm inf}$ and $\theta^\kappa_{\rm sup}$ is sufficiently small. We summarize this in the prototypical algorithm given in figure 8, dropping the iteration counter k to simplify the notation.

Step 1 of this process is usually called the *Master Problem*. Classically, it has been solved using the simplex algorithm, and then the whole process resembles Dantzig-Wolfe decomposition. Goffin *et al.* use an interior point method to solve the Master Problem. They apply the de Ghellinck and Vial variant [19] of Karmarkar's projective algorithm [41] to the dual of Master Problem to calculate the analytic center of the localization set. The localization set is modified by the addition of constraints so columns are added to the dual of this problem. An interior point is generated in the dual by using the technique of Mitchell and Todd [59]. The method

used by Goffin *et al.* generates primal and dual iterates at each approximate solution to the master problem, so an approximate solution to the Lagrangian dual can be converted to an approximate solution to the multicommodity flow problem.

Step 2 of the prototype algorithm is called the subproblem or oracle. There are choices available in the solution of this problem for a multicommodity flow problem, depending upon the level of disaggregation of the constraints. The constraints for the subproblem are separable by commodity. It is then possible to generate one subgradient for the whole problem, or to generate subgradients corresponding to each commodity. Goffin *et al.* obtained better results by disaggregating the constraints and generating separate subgradients for each commodity; this is in agreement with other work in the literature which used different algorithms to solve the Master Problem (see Jones *et al.* [34]).

Goffin *et al.* give computational results for random problems with up to 500 nodes, 1000 arcs, and 4000 commodities, and for some smaller problems from the literature. (In their formulation, the largest problems could have up to 8×10^6 primal variables x_e^i.) They compared their algorithm with an implementation of Dantzig-Wolfe decomposition, and the interior point algorithm was clearly superior for the problems discussed.

5.2. Other interior point methods to solve the multicommodity network flow problem

Kamath *et al.* [36, 39] have described several interior point methods for the multicommodity flow problem. One approach solves the problem as a linear programming problem using a dual projective interior point method. They obtained computational results comparable with CPLEX Netopt [12]. A second approach places the network flow constraints in a convex quadratic objective function and solves a minimization problem with this objective subject to the capacity constraints. This algorithm has good theoretical complexity for approximately solving the multicommodity network flow problem.

6. Computational complexity results

6.1. Theoretical behaviour of cutting plane algorithms

It is usually straightforward to show that an interior point cutting plane (or column generation) algorithm runs in time polynomial in the total number of constraints (or columns) generated during the algorithm — see, for example, Mitchell [56] or den Hertog *et al.* [31, 32]. A harder problem is to show that such an algorithm runs in time polynomial in the size of the original description of the problem.

Given an integer programming problem, a separation routine either confirms that a point is in the convex hull of the set of feasible integer points, or it provides a cutting plane which separates the point from the convex hull. If the separation routine runs in time polynomial in the size of the problem, then the ellipsoid algorithm can be used to solve the integer programming problem in polynomial time — see Grötschel *et al.* [29]. It is not necessary to drop any constraints when using this method. For the rest of this subsection, we assume that the separation routines require polynomial time.

To date, the only interior point algorithm which solves the integer program in polynomial time and which does not drop constraints is due to Vaidya [74]. This algorithm uses the *volumetric center*, so its analysis differs from that of more standard interior point methods. Vaidya's analysis of his algorithm shows that only a polynomial number of constraints are generated, even though an infinite number of possible constraints exists. This is a crucial point in proving the polynomial complexity of his algorithm, and indeed of any cutting plane or column generation algorithm. For an alternative analysis of this algorithm, see Anstreicher [4]. Anstreicher was able to greatly reduce the constants involved in the complexity analysis of Vaidya's algorithm, making the algorithm considerably more attractive for implementation. For example, Anstreicher reduced the number of Newton steps by a factor of 1.8 million and he reduced the maximum number of constraints used by a factor of 10^4. Vaidya's algorithm is a short step algorithm, in the sense that the reduction in the duality gap at an iteration is dependent on the dimension of the problem. Ramaswamy and Mitchell [68] have developed a long step variant of Vaidya's algorithm that has polynomial convergence. Their algorithm reduces the duality gap by a fixed ratio at any iteration where it is not necessary to add or drop constraints.

Atkinson and Vaidya [6] developed a polynomial time cutting plane algorithm which used the analytic center. This algorithm drops constraints that become unimportant, and this is essential in their complexity analysis. Previous algorithms were often shown to be polynomial in the number of additional constraints, but without a proof that the number of added constraints is polynomial. Atkinson and Vaidya's algorithm finds a feasible point for a set of convex inequalities by finding an analytic center for a subset of the inequalities and using an oracle to test whether that point satisfies all the inequalities. If the oracle returns a violated inequality, a shifted linear constraint is added so that the analytic center remains feasible and close to the new analytic center.

Mitchell and Ramaswamy [58] developed a barrier function cutting plane algorithm using some of the ideas from [6]. This algorithm is a long step algorithm, unlike the algorithm in [6]: if it is not necessary to add or drop constraints, then they reduce the duality gap by a constant fraction. They showed some links between the notion of a point being centered (see, for example, Roos and Vial [72]) and the criteria for a constraint to be added or dropped in [6]. Barrier function methods for linear programming have shown excellent computational performance and they can be constructed to have superlinear and quadratic convergence. It would thus appear desirable to employ these methods in a column generation algorithm.

Goffin *et al.* [23, 24] presented a pseudopolynomial column generation algorithm which does not need to drop any columns. The number of iterations required to get the objective function value to within ϵ of optimality is polynomial in ϵ, but this algorithm does not obtain a solution within 2^{-L} of optimality in time polynomial in L, where L is the size of the data.

There have been several papers recently analyzing algorithms that add many cuts at once (see, for example, Luo [48], Ramaswamy and Mitchell [67], and Ye [80]). These papers generally show that the complexity of an algorithm is not harmed if many cuts are added at once, although there do have to be some bounds on the number of constraints added simultaneously.

The earlier theoretical papers on interior point cutting plane algorithms generally added the constraints far from the current center, so that the center of the new system is close to the center of the old system. The paper by Goffin *et al.* [24] shows that it is possible to add a cutting plane right through the current analytic center without changing the complexity of their algorithm [23]. Ye [80] extended this analysis to the case where multiple cuts are placed right through the analytic center. Ramaswamy and Mitchell [67] describe an algorithm which adds multiple cuts through the analytic center, and they show that the new analytic center can be regained in $O(\sqrt{p}\log(p))$ iterations, where p is the number of added cuts.

6.2. Improved complexity results for selected combinatorial optimization problems

There has been some research on using interior point methods within algorithms to solve some combinatorial optimization problems that can be solved in polynomial time. This has led to improved complexity results for some problems.

The research on interior point methods for positive semi-definite programming has lead to improved algorithms for various problems in combinatorial optimization. For example, see the the chapter in this book by Alizadeh, or the papers by Goemans *et al.* [18, 20], Alizadeh [2] or chapter 9 of the book [29].

Bertsimas and Orlin [8] use the interior point algorithm for convex programming given by Vaidya [74] to obtain algorithms with superior theoretical complexity for several combinatorial optimization problems, principally by giving a new method for solving the Lagrangean dual of a problem. This leads to improved complexity for lower bounding procedures for the traveling salesman problem (particularly, the Held and Karp method), the Steiner tree problem, the 2-connected problem, vehicle routing problems, multicommodity flow problems, facility location problems, and others.

Xue and Ye [78] have described an interior point algorithm for solving the problem of minimizing a sum of Euclidean norms. This algorithm can be used to solve problems related to Steiner trees with better theoretical complexity than the previously best known algorithm.

7. Conclusions

We have discussed the ways in which interior point methods have been used to solve combinatorial problems. The methods discussed include algorithms where the simplex method has been replaced by an interior point method as well as a new method which appears unrelated to previous simplex-based algorithms.

We discussed incorporating interior point methods into cutting plane and branch and bound algorithms for integer programming in section 2. In order to do this successfully, it is necessary to be able to use a warm start somewhat efficiently. The effective use of a warm start in an interior point method is an active area of research; if a warm start could be exploited successfully by an interior point method then the performance of interior point cutting plane and branch and bound algorithms would be considerably enhanced. In the research to date, the most important technique appears to be early termination: the current relaxation is only solved to within

some tolerance of optimality before we attempt to refine the relaxation. Currently, interior point cutting plane methods do appear to be somewhat competitive with simplex cutting plane algorithms, at least for some problems. Interior point branch and bound algorithms still appear weaker than simplex based algorithms, at least for the size of problems which can currently be solved. For linear programming, interior point methods start to outperform simplex for large problems, so a branch and bound interior point method would only be advantageous for large problems (thousands of variables and constraints). Pure integer problems of this size are currently generally intractable. Thus, interior point branch and bound methods are currently only useful for problems with a small number of integer variables, but a large number of continuous variables. As hardware improves, it will become possible to solve larger problems, and interior point branch and bound methods will become more attractive. Additionally, if a warm start could be exploited more efficiently then an interior point method would become attractive even for smaller problems.

We described a potential reduction algorithm that transforms an integer programming problem into an equivalent quadratic program in section 3. This algorithm appears to have reasonable computational performance, and it could solve large problems that were previously unsolved.

We described the use of interior point methods to solve network flow problems in section 4. These problems can be solved by solving a single linear program. The computational results with an interior point method are better than those with a specialized simplex method for large problems in several classes.

Research on the multicommodity network flow problem was discussed in section 5. A column generation algorithm which appears to outperform classical Dantzig-Wolfe decomposition on these problems was described.

With all of these methods, the relative performance of the interior pont method to other methods improves as the problem size increases. This is typical of computational results with interior point methods for linear programming and other problems. Interior point methods will probably not be the method of choice for small or medium sized problems, but they may become the preferred method for larger problems once computational hardware improves sufficiently to make it possible to routinely solve problems which are currently impracticably large. The increasing use of parallel computers and networks of workstations is leading to the solution of ever larger problems. Of course, improvements in simplex may keep it the method of choice even for large problems, but we expect that there will be at least some classes of problems where an interior point method is superior for large instances. Research on most of the algorithms discussed in this paper is ongoing, and the researchers involved are attempting to solve larger problems, in an effort to determine the best algorithm for large hard problems.

We discussed theoretical issues concerning cutting plane and column generation algorithms in section 6.1. There are polynomial time interior point cutting plane algorithms. However, to date there is no polynomial time interior point cutting plane algorithm that is based upon the analytic center and which does not drop constraints. Whether such an algorithm exists is an interesting open problem. The discussion in section 6.2 of improved complexity results for various combinatorial optimization problems is a starting point for what will probably be an active research area in the

next few years.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.
2. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.
3. K. M. Anstreicher. A combined phase I – phase II scaled potential algorithm for linear programming. *Mathematical Programming*, 52:429–439, 1991.
4. K. M. Anstreicher. On Vaidya's volumetric cutting plane method for convex programming. Technical report, Department of Management Sciences, University of Iowa, Iowa City, Iowa 52242, September 1994.
5. D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP (a preliminary report). Technical report, Mathematics, AT&T Bell Laboratories, Murray Hill, NJ, 1994.
6. D. S. Atkinson and P. M. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69:1–43, 1995.
7. H. van Bentham, A. Hipolito, B. Jansen, C. Roos, T. Terlaky, and J. Warners. Radio link frequency assignment project, Technical annex T–2.3.2: Potential reduction methods. Technical report, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.
8. D. Bertsimas and J. B. Orlin. A technique for speeding up the solution of the Lagrangean dual. *Mathematical Programming*, 63:23–45, 1994.
9. J. R. Birge, R. M. Freund, and R. J. Vanderbei. Prior reduced fill–in in solving equations in interior point algorithms. *Operations Research Letters*, 11:195–198, 1992.
10. B. Borchers. *Improved branch and bound algorithms for integer programming*. PhD thesis, Rensselaer Polytechnic Institute, Mathematical Sciences, Troy, NY, 1992.
11. B. Borchers and J. E. Mitchell. Using an interior point method in a branch and bound algorithm for integer programming. Technical Report 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1991. Revised July 7, 1992.
12. CPLEX Optimization Inc. CPLEX Linear Optimizer and Mixed Integer Optimizer. Suite 279, 930 Tahoe Blvd. Bldg 802, Incline Village, NV 89541.
13. M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.
14. DIMACS. The first DIMACS international implementation challenge: The benchmark experiments. Technical report, DIMACS, RUTCOR, Rutgers University, New Brunswick, NJ, 1991.
15. J. Edmonds. Maximum matching and a polyhedron with 0, 1 vertices. *Journal of Research National Bureau of Standards*, 69B:125–130, 1965.
16. J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
17. A. S. El–Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior–point methods. *SIAM Review*, 36:45–72, 1994.
18. Uriel Feige and Michel X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, 1995.
19. G. de Ghellinck and J.-P. Vial. A polynomial Newton method for linear programming. *Algorithmica*, 1:425–453, 1986.
20. Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. Assoc. Comput. Mach.*, 1994. (To appear). A preliminary version appeared in Proc. 26th Annual ACM Symposium on Theory of Computing.
21. J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-P. Vial. Solving nonlinear multicommodity network flow problems by the analytic center cutting plane method. Technical report, GERAD, Faculty of Management, McGill University, Montreal, Quebec, Canada H3A 1G5, October 1994.
22. J.-L. Goffin, A. Haurie, and J.-P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.

23. J.-L. Goffin, Z.-Q. Luo, and Y. Ye. On the complexity of a column generation algorithm for convex or quasiconvex problems. In *Large Scale Optimization: The State of the Art*. Kluwer Academic Publishers, 1993.
24. J.-L. Goffin, Z.-Q. Luo, and Y. Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. Technical report, Faculty of Management, McGill University, Montréal, Québec, Canada, June 1994.
25. J.-L. Goffin and J.-P. Vial. Cutting planes and column generation techniques with the projective algorithm. *Journal of Optimization Theory and Applications*, 65(3):409–429, 1990.
26. R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.
27. M. Grötschel and O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985.
28. M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.
29. M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, Germany, 1988.
30. O. Güler, D. den Hertog, C. Roos, T. Terlaky, and T. Tsuchiya. Degeneracy in interior point methods for linear programming: A survey. *Annals of Operations Research*, 46:107–138, 1993.
31. D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming, Algorithms and Complexity*. PhD thesis, Faculty of Mathematics and Informatics, TU Delft, NL–2628 BL Delft, The Netherlands, September 1992.
32. D. den Hertog, C. Roos, and T. Terlaky. A build–up variant of the path–following method for LP. *Operations Research Letters*, 12:181–186, 1992.
33. IBM. *IBM Optimization Subroutine Library Guide and Reference*, August 1990. Publication number SC23–0519–1.
34. K. L. Jones, I. J. Lustig, J. M. Farvolden, and W. B. Powell. Multicommodity network flows — the impact of formulation on decomposition. *Mathematical Programming*, 62:95–117, 1993.
35. M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. Technical Report 94.156, Institut für Informatik, Universität zu Köln, Pohligstraße 1, D-50969 Köln, Germany, March 1994.
36. A. P. Kamath. *Efficient Continuous Algorithms for Combinatorial Optimization*. PhD thesis, Department of Computer Science, Stanford University, Palo Alto, CA, February 1995.
37. A. P. Kamath and N. K. Karmarkar. A continuous approach to compute upper bounds in quadratic maximization problems with integer constraints. In C. A. Floudas and P. M. Pardalos, editors, *Recent Advances in Global Optimization*, Princeton Series in Computer Science, pages 125–140. Princeton University Press, Princeton, NJ, USA, 1992.
38. A. P. Kamath and N. K. Karmarkar. An $O(nL)$ iteration algorithm for computing bounds in quadratic optimization problems. In P. M. Pardalos, editor, *Complexity in Numerical Optimization*, pages 254–268. World Scientific Publishing Company, Singapore (USA address: River Edge, NJ 07661), 1993.
39. A. P. Kamath, N. K. Karmarkar, and K. G. Ramakrishnan. Computational and complexity results for an interior point algorithm on multi-commodity flow problem. Technical report, Department of Computer Science, Stanford University, Palo Alto, CA, 1993.
40. A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.
41. N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
42. N. K. Karmarkar and K. G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Mathematical Programming*, 52:555–586, 1991.
43. N. K. Karmarkar, M. G. C. Resende, and K. G. Ramakrishnan. An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming*, 52:597–618, 1991.
44. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
45. D. Klingman, A. Napier, and J. Stutz. Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost network flow problems. *Management*

Science, 20:814–821, 1974.
46. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.
47. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley, New York, 1985.
48. Z.-Q. Luo. Analysis of a cutting plane method that uses weighted analytic center and multiple cuts. Technical report, Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ontario, L8S 4L7, Canada, September 1994.
49. I. J. Lustig, R. E. Marsten, and D. F. Shanno. On implementing Mehrotra's predictor–corrector interior point method for linear programming. *SIAM Journal on Optimization*, 2:435–449, 1992.
50. I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, 1994. See also the following commentaries and rejoinder.
51. N. Megiddo. On finding primal– and dual–optimal bases. *ORSA Journal on Computing*, 3:63–65, 1991.
52. S. Mehrotra. On the implementation of a (primal–dual) interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.
53. M. Minoux. *Mathematical Programming: Theory and Algorithms*. Wiley, New York, 1986.
54. J. E. Mitchell. *Karmarkar's Algorithm and Combinatorial Optimization Problems*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1988.
55. J. E. Mitchell. Fixing variables and generating classical cutting planes when using an interior point branch and cut method to solve integer programming problems. Technical Report 216, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, October 1994.
56. J. E. Mitchell. An interior point column generation method for linear programming using shifted barriers. *SIAM Journal on Optimization*, 4:423–440, May 1994.
57. J. E. Mitchell and B. Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. Technical Report 207, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, March 1993. To appear in *Annals of OR*.
58. J. E. Mitchell and S. Ramaswamy. An extension of Atkinson and Vaidya's algorithm that uses the central trajectory. Technical Report 37–93–387, DSES, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, August 1993.
59. J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, 56:245–284, 1992.
60. S. Mizuno, M. Kojima, and M. J. Todd. Infeasible–interior–point primal–dual potential–reduction algorithms for linear programming. *SIAM Journal on Optimization*, 5:52–67, 1995.
61. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York, 1988.
62. G. L. Nemhauser and L. A. Wolsey. Integer programming. In G. L. Nemhauser et al., editor, *Optimization*, chapter 6, pages 447–527. North-Holland, 1989.
63. P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1:15–23, 1991.
64. R. G. Parker and R. L. Rardin. *Discrete Optimization*. Academic Press, San Diego, CA 92101, 1988.
65. L. Portugal, F. Bastos, J. Júdice, J. Paix ao, and T. Terlaky. An investigation of interior point algorithms for the linear transportation problem. Technical report, Department of Mathematics, University of Coimbra, Coimbra, Portugal, 1993. To appear in *SIAM J. Sci. Computing*.
66. L. Portugal, M. Resende, G. Veiga, and J. Júdice. A truncated primal-infeasible dual-feasible network interior point method. Technical report, AT&T Bell Laboratories, Murray Hill, Jew Jersey, 1994.
67. S. Ramaswamy and J. E. Mitchell. On updating the analytic center after the addition of multiple cuts. Technical Report 37–94–423, Dept. of Decision Sciences and Engg. Systems, Rensselaer Polytechnic Institute, Troy, NY 12180, October 1994.
68. S. Ramaswamy and J. E. Mitchell. A long step cutting plane algorithm that uses the volumetric barrier. Technical report, Dept. of Decision Sciences and Engg. Systems, Rensselaer Polytechnic Institute, Troy, NY 12180, June 1995.

69. M. G. C. Resende and P. M. Pardalos. Interior point algorithms for network flow problems. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey 07974–2070, 1994. To appear in *Advances in Linear and Integer Programming*, J. E. Beasley, ed., Oxford University Press, 1995.
70. M. G. C. Resende and G. Veiga. An efficient implementation of a network interior point method. In D.S. Johnson and C.C. McGeogh, editors, *Network Flows and Matching: First DIMACS Implementation Challenge,*, pages 299–348. American Mathematical Society, 1993. DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 12.
71. M. G. C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. *SIAM Journal on Optimization*, 3:516–537, 1993.
72. C. Roos and J. P. Vial. A polynomial method of approximate centers for linear programming. *Mathematical Programming*, 54:295–305, 1992.
73. C.-J. Shi, A. Vannelli, and J. Vlach. An improvement on Karmarkar's algorithm for integer programming. *COAL Bulletin*, 21:23–28, November 1992.
74. P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 338–343, Los Alamitos, CA, 1989. IEEE Computer Press. To appear in *Mathematical Programming*.
75. J. P. Warners. A potential reduction approach to the radio link frequency assignment problem. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.
76. X. Xu, P. F. Hung, and Y. Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. Technical report, College of Business Administration, The University of Iowa, Iowa City, Iowa 52242, September 1993.
77. X. Xu and Y. Ye. A generalized homogeneous and self-dual algorithm for linear programming. *Operations Research Letters*, 17:181–190, 1995.
78. G. Xue and Y. Ye. An efficient algorithm for minimizing a sum of Euclidean norms with applications. Technical report, Department of Computer Science and Electrical Engineering, University of Vermont, Burlington, VT 05405–0156, June 1995.
79. Y. Ye. On an affine scaling algorithm for nonconvex quadratic programming. *Mathematical Programming*, 56:285–300, 1992.
80. Y. Ye. Complexity analysis of the analytic center cutting plane method that uses multiple cuts. Technical report, Department of Management Sciences, The University of Iowa, Iowa City, Iowa 52242, September 1994.
81. Y. Ye, M. J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.
82. Y. Zhang. On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem. *SIAM Journal on Optimization*, 4(1):208–227, 1994.