Interior Point Algorithms for Integer Programming[1]

John E. Mitchell[2]

Department of Mathematical Sciences

Rensselaer Polytechnic Institute

Troy, NY 12180

mitchj@rpi.edu

R.P.I. Technical Report No. 215

June 15, 1994.

Abstract

Research on using interior point algorithms to solve integer programming problems is surveyed. This paper concentrates on branch and bound and cutting plane methods; a potential function method is also briefly mentioned. 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. Methods 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 a short discussion of theoretical issues.

[1]To appear in the book "Advances in linear and integer programming", Oxford University Press, 1995.

[2]Research partially supported by ONR Grant numbers N00014–90–J–1714 and N00014–94–1–0391.

1 Introduction

An algorithm for solving linear programming problems can be used as a subroutine in methods for more complicated problems. Such methods usually involve solving a sequence of related linear programming problems, where the solution to one linear program is close to the solution to the next, that is, it provides a *warm start* for the next linear program. The *branch and cut* method for solving integer programming problems is of this form: linear programming relaxations are solved until eventually the integer programming problem has been solved. Within the last ten years, interior point methods have become accepted as powerful tools for solving linear programming problems. It appears that interior point methods may well solve large linear programs substantially faster than the simplex method. A natural question, therefore, is whether interior point methods can be successfully used to solve integer programming problems. This requires the ability to exploit a warm start.

Branch and cut methods for integer programming problems are a combination of cutting plane methods and branch and bound methods. In a cutting plane method for solving an integer programming problem, the linear programming relaxation of the integer program is solved. If the optimal solution to the linear program is feasible in the integer program, it also solves the integer program; otherwise, a constraint is added to the linear program which *separates* the optimal solution from the set of feasible solutions to the integer program, the new linear program is solved, and the process is repeated. In a branch and bound method for solving an integer programming problem, the first step is also to solve the linear programming relaxation. If the optimal solution is feasible in the integer program, it solves the integer program. Otherwise, the relaxation is split into two subproblems, usually by fixing a particular variable at zero or one. This is the start of a tree of subproblems. A subproblem is selected and the linear programming relaxation of that subproblem is solved. Four outcomes are possible: the linear programming relaxation is infeasible, in which case the integer subproblem is also infeasible, and the tree can be pruned at this node; the optimal solution to the linear programming relaxation is feasible in the integer subproblem, in which case it also solves the subproblem, and the tree can be pruned at this node; the optimal solution has a worse objective function value than a known integer solution to the original problem, in which case any solution to the integer subproblem is also worse than the known solution, and the tree can be pruned at this node; finally, none of these situations occur, in which case it is necessary to split the node into two further subproblems.

The difficulty with using an interior point method in a cutting plane algorithm is that the solution to one relaxation is usually not a good starting point for an interior point method, because it is close to the boundary of the feasible region. Thus, it is usually necessary to attempt to stop working on the current relaxation before it is solved completely to optimality: the earlier we are able to find good cutting planes, the better the initial solution to the next relaxation. Early termination obviously reduces the number of iterations spent solving the current relaxation; in addition, it reduces the number of iterations spent solving the next relaxation, because the initial point to the next relaxation is more centered. There are two potential disadvantages from trying to find cutting planes early: if the search for cutting planes is unsuccessful, we have wasted time; secondly, it may well be that superfluous constraints are added, with the result that the algorithm requires extra iterations and extra stages of adding cutting planes. We will discuss methods for overcoming these drawbacks.It is also necessary to be able to use a warm start when using an interior point method in a branch and bound algorithm; again, it is possible to use early termination to improve the algorithm. Usually, the only time it is necessary to solve the subproblem to optimality in the branch and bound tree is when the optimal solution to the linear programming relaxation is feasible in the integer subproblem.

We will concentrate on branch and cut algorithms for integer programming in the rest of this paper, with the principal ideas being presented in section 2. A cutting plane method can be regarded as a *column generation method* applied to the dual of the linear programming relaxation, because the addition of a cutting plane, or constraint, to the relaxation adds a column to the dual. There has been research on using interior point methods in column generation algorithms for various problems, and we will discuss some of this work later in the paper, because of its relevance to branch and cut algorithms. For now, we give two examples. First, consider a linear programming problem with many more variables than constraints. If all the variables were included in the working set of columns, the matrix algebra would become impractical. Therefore, a pricing criterion is used to select a subset of the variables, and this subset is updated periodically — see Bixby *et al.* [3], Kaliski and Ye [20], and Mitchell [28], for example. Secondly, column generation methods are also useful for solving nonsmooth optimization problems. The constraints and the objective function are approximated by piecewise linear functions, and the approximation is improved as the algorithm proceeds. The problem solved at each stage is a linear programming problem, and refining the approximation corresponds to adding columns to the linear program. For more details, see, for example, the papers by Goffin and Vial and their coauthors [12, 10].

Another approach for solving integer programming problems using interior point methods has been presented by Kamath *et al.* [23, 21, 22]. We discuss this method in section 3.2.

2 Interior point branch and cut algorithms

In this section, we discuss the use of interior point methods in cutting plane and branch and bound approaches to solving integer programming problems. The two approaches can be combined into a branch and cut algorithm; however, we intend to deal with them separately in order to make the basic ideas clear. Various observations about interior point algorithms will affect the design of efficient algorithms in both the cutting plane and branch and bound approaches. In particular, it is vital to use early termination in order to obtain good performance from these algorithms.

The standard form integer programming problem we consider has the form

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & \geq & b \\ & x & \in & \{0,1\} \end{array} \qquad (IP)$$

where x and c are n-vectors, b is an m-vector and A is an $m \times n$ matrix. Note that we restrict attention to problems where the integer variables are constrained to be binary. We define the polyhedron Q to be the convex hull of feasible solutions to the problem (IP). The problem (IP) can be solved by solving the linear programming problem $\min\{c^T x : x \in Q\}$. (If this linear program has multiple optimal solutions, an interior point method may well return a point which is non-integral, but this can be rounded to give an integer optimal extreme point of Q using the methods of, for example, Megiddo [26].) The difficulty with this approach, obviously, is that in general a closed form expression for Q is not known. In a cutting plane method, we consider *linear programming relaxations* of (IP), where the feasible region gradually becomes a better approximation to Q, at least in the neighbourhood of the optimal solution. The linear programming relaxation of (IP) is

$$\begin{array}{llcl} \min & c^T x & & \\ \text{subject to} & Ax & \geq & b \qquad (LPP) \\ & 0 \leq x & \leq & e \end{array}$$

where e denotes a vector of ones of the appropriate dimension. We denote the feasible region of this problem by $Q^{LPP} := \{x \in \Re^n : Ax \geq b, 0 \leq x \leq e\}$. The dual of this problem is

$$\begin{array}{lrcrcl} \max & b^T y & - & e^T w & & \\ \text{subject to} & A^T y & - & w & \leq & c \qquad (LPD) \\ & & & y, w & \geq & 0 \end{array}$$

where y is an m-vector and w is an n-vector. When necessary, we will let $s = Ax - b$ denote the slack variables for the primal problem (LPP) and $z = c - A^T y + w$ denote the slack variables for the dual problem (LPD). If x denotes an n-vector then X denotes the diagonal $n \times n$ matrix with $X_{ii} = x_i$. The diagonal matrices Y, W, Z, and S are defined similarly. A positive iterate is considered *centered* if

$$XZe - \mu e = 0, \quad SYe - \mu e = 0, \quad (I - X)We - \mu e = 0 \tag{1}$$

for some appropriate scalar μ; the iterate is *approximately centered* if the norm of each of these vectors is small. The scalar μ is called the *barrier parameter*.

2.1 Interior point cutting plane algorithms

Consider solving (IP) using a cutting plane approach. The LP-relaxation (LPP) is solved, giving an optimal point $\hat{x}$, and an optimal dual solution $\hat{y}$, $\hat{z}$ for (LPD). If $\hat{x}$ is feasible in (IP) then it solves (IP). Otherwise, we use a *separation routine* to find a cutting plane which separates $\hat{x}$ from the feasible region for (IP). The separation routine depends on the problem being solved. For linear ordering problems, it is usually sufficient to examine the triangle inequalities (see Mitchell and Borchers [29]). Far more complicated separation routines have been developed for other problems, including the travelling salesman problem (see, for example, Padberg and Rinaldi [34]). Returning to the general case, we denote the cutting plane by

$$a_0^T x = b_0 \tag{2}$$

where a_0 is an n-vector and b_0 is a scalar. The point $\hat{x}$ satisfies

$$a_0^T \hat{x} < b_0 \tag{3}$$

and every feasible point x for (IP) satisfies

$$a_0^T x \geq b_0. \tag{4}$$

The constraint $a_0^T x \geq b_0$ is added to (LPP) giving a new relaxation

$$\begin{array}{lrcll}
\min & c^T x & & & \\
\text{subject to} & Ax & \geq & b & \qquad (LPP0) \\
 & a_0^T x & \geq & b_0 & \\
 & 0 \leq x & \leq & e &
\end{array}$$

with a corresponding dual problem

$$\begin{array}{lrcccccl}
\max & b^T y & + & b_0 y_0 & - & e^T w & & \\
\text{subject to} & A^T y & + & a_0 y_0 & - & w & \leq & c \qquad (LPD0) \\
 & & & y, y_0, w & \geq & 0 & &
\end{array}$$

where y_0 is a scalar. Notice that the current primal iterate $\hat{x}$ is not feasible in $(LPP0)$, but $\hat{y}$, $\hat{w}$ is feasible in $(LPD0)$, if we set $y_0 = 0$. A simplex-based cutting plane algorithm would reoptimize using the dual simplex algorithm. It is not so easy to restart an interior point method, because we no longer have an *interior* dual feasible point, that is, a point where the dual variables and the dual slacks are all strictly positive. This difficulty can be overcome by setting y_0 to a small positive value, although the dual iterate is then not feasible. An infeasible interior point method can be used to solve $(LPP0)$ and $(LPD0)$ — see, for example, Lustig et al [24] or Zhang [41]. A more serious problem is that the primal and dual iterates are both extreme points (provided the LP-relaxation is nondegenerate). Interior point methods tend to have poor performance when started from close to a nonoptimal extreme point. Typically, several iterations are spent moving towards the center of the feasible region before the algorithm moves towards the optimal point. One way to attempt to overcome this problem is to use *early termination*, that is, the problem (LPP) is not solved to optimality, but we attempt to find cutting planes at a non-optimal point. This should ensure a better (ie, more centered) starting iterate for problems $(LPP0)$ and $(LPD0)$.

A simple, conceptual interior point cutting plane algorithm could be written as

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.

We will give a more formal algorithm later.

There are many different interior point methods and each one would be implemented in a slightly different manner in a cutting plane algorithm. Currently, the best algorithm for linear programming appears to be the primal-dual predictor-corrector barrier method (see Lustig *et al.* [24] and Mehrotra [27]), so, in this section, 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; for a discussion of algorithms which only require either the primal or the dual iterate to be strictly positive, see section 3.1.

Early termination

It is very desirable not to have to solve the current relaxation to optimality, but to attempt to add cutting planes early. Adding constraints early has two principal advantages:

- The initial iterates for the next subproblem are more centered, so fewer iterations are required to solve the next problem.
- Fewer iterations are required on the current problem.

The principal disadvantages of looking for constraints before solving to optimality are

- We may not be able to find any cutting planes, although we would be able to find cuts if we solved to optimality. Thus, time is wasted by looking early.
- Conversely, it may be that the optimal solution to (LPP) solves (IP), but the early iterates are not in the convex hull of feasible solutions, so the separation routines return cutting planes that are not necessary.

Attempting to find constraints early also has other, more subtle, advantages. First, the early iterates are closer than the optimal solution to the center of the feasible region for (LPP), so it may be that the cuts which are found are *deeper* cuts, that is, they cut off more of the polyhedron Q^{LPP}. Secondly, it may also be that some of the cuts found by searching early would not have been found until the next relaxation if the relaxation had been solved to optimality. Consider, for example, the perfect matching problem.

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, and a *perfect matching* is a matching which contains exactly $\mid V \mid /2$ edges, where $\mid V \mid$ denotes the cardinality of V. Given a set of weights w_e associated with each edge, the perfect matching problem is to find the perfect matching M with smallest weight $w(M) := \sum_{e \in M} w_e$.

A complete polyhedral description for the perfect matching polytope was given by Edmonds [8]. He showed that the perfect matching problem is equivalent to the linear program

$$\begin{array}{rll} \min & \sum_{e\in E} w_e x_e & \\ \text{subject to} & \sum_{e\in\delta(v)} x_e & = 1 \text{ for all } v \in V \qquad (5) \\ & \sum_{e\in E(U)} x_e & \leq (\mid U \mid -1)/2 \text{ for all } U \subseteq V, \text{ with } \mid U \mid \text{ odd} \qquad (6) \\ & x_e & \geq 0 \text{ for all } e \in E \qquad (7) \end{array}$$

where $\delta(v)$ denotes the set of edges in E which are incident to v and $E(U)$ denotes the set of edges with both endpoints in the subset U of V. In a cutting plane approach to this problem, the initial relaxation consists of the degree constraints (5) and the nonnegativity constraints (7). The odd set constraints (6) are added as cutting planes. The odd set constraints prevent the existence of circuits with each edge e in the circuit having value $x_e = 0.5$. For details of a separation routine to find violated cutting planes, see Grötschel and Holland [13] and Mitchell and Todd [31]. Consider the graph shown in figure 1. Here,

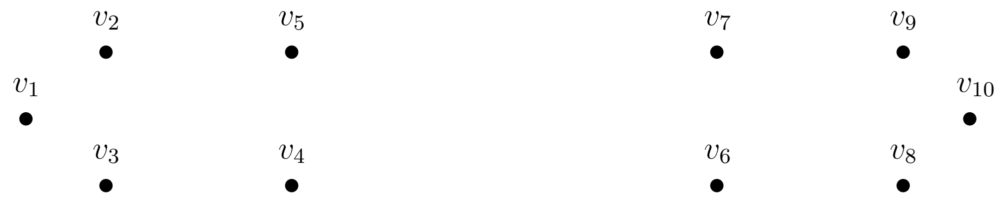

Figure 1: An illustration of the phenomenon of nested odd sets

the edge weights w_e are the euclidean distances between vertices. An optimal perfect matching must contain one of the two edges (v_4, v_6) and (v_5, v_7). The optimal solution to the relaxation has $x_e = 0.5$ for the edges (v_1, v_2), (v_2, v_3), (v_1, v_3), (v_8, v_9), (v_8, v_{10}), and (v_9, v_{10}) and $x_e = 1$ for (v_4, v_5) and (v_6, v_7). The odd set constraints corresponding to the sets of vertices $\{v_1, v_2, v_3\}$ and $\{v_8, v_9, v_{10}\}$ are violated by this solution. After adding these constraints, the solution to the new relaxation has $x_e = 0.5$ for the edges (v_1, v_2), (v_2, v_5), (v_4, v_5), (v_3, v_4), (v_1, v_3), (v_6, v_7), (v_7, v_9), (v_9, v_{10}), (v_8, v_{10}), and (v_6, v_8). This violates the the odd set constraints corresponding to the sets of vertices $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_6, v_7, v_8, v_9, v_{10}\}$. However, if an interior point method is used and the search for violated constraints is conducted early, it may be possible to identify both of these sets of constraints at once, because the values of x_e on the edges (v_2, v_5) and (v_3, v_4) tend to zero less quickly than the values on the edges (v_5, v_7) and (v_4, v_6). Thus, one fewer LP-relaxation needs to be solved.

We have argued that it is necessary and beneficial to add cutting planes early when using an interior point cutting plane method. On the other hand, if we look for cutting planes when we are far from optimality, we may find unnecessary cuts. Thus, it is necessary to find a balance between looking for cuts too soon and waiting too long to look. This can be achieved by using a dynamically altered tolerance on the duality gap: we only look for cutting planes if the duality gap for the current relaxation falls below this tolerance. We are happy with a large tolerance if the current relaxation does not provide a good approximation to (IP), because we do not want to spend a lot of time on poor relaxations. Conversely, once the relaxation becomes a good approximation, the tolerance should become small, because we want to be selective about adding additional cutting planes, and the relaxation is not likely to change dramatically, so the solution to the new relaxation should be close to the solution to the old relaxation. Additionally, the tolerance should be small when the relaxation provides a good approximation to (IP) because we should make sure that the solution to the relaxation does not solve (IP), in order to avoid adding unnecessary cutting planes. One way to modify the tolerance to achieve these aims is to increase it if we find a large number of violated constraints and decrease it if we find only a small number of (or no) violated constraints. This should have the effect of increasing the tolerance if the relaxation is a poor approximation to (IP) and decreasing it if the relaxation is a good approximation to (IP). For details on a particular way to modify this tolerance, see [29].

It is necessary to identify violated constraints early in an interior point cutting plane method in order for it to compete with a simplex cutting plane algorithm. It might be conjectured that a simplex cutting plane algorithm could be improved by using early identification. Such an approach is unlikely to be successful for several reasons. In the first place, the simplex algorithm moves from extreme point to extreme point, and thus it is likely to have relatively few fractional variables, and vital elements may be pivoted into the basis in the last few iterations. Complementary to this concern, it may be that the solution to the current relaxation solves the integer problem, but one of the vital basis elements only enters the basis on the last iteration, so the earlier iterates may violate constraints even if the optimal solution does not; it might then happen that unnecessary cutting planes are added to the relaxation. A third concern relates to the amount of time spent searching for violated cutting planes. Separation routines are generally far more expensive than simplex iterations, so it would be uneconomical to search every iteration; it may be possible to search every K iterations for some number K.

Restarting the algorithm and regaining primal feasibility

Assume we have just added a cutting plane, so the relaxation now has the form of problem $(LPP0)$, with corresponding dual problem $(LPD0)$. We know a primal point $\hat{x}$ which satisfies $Ax \geq b$, but which does not satisfy $a_0^T x \geq b_0$. The current dual point $\hat{y}$, $\hat{w}$ is feasible in $(LPD0)$ provided we set $y_0 = 0$. Since we used an interior point method to solve (LPP) and since we terminated early, we have $\hat{x} > 0$, $\hat{y} > 0$, and the primal and dual slacks strictly positive (except for the new primal constraint). We will use an infeasible interior point method to solve $(LPP0)$, so we need an interior point and so we need positive values for y_0 and s_0. The simplest way to achieve this is to set these variables equal to some small positive amount ϵ; it is also beneficial to reset any other variables which are smaller than ϵ to ϵ. This works reasonably well in practice — the number of iterations required to solve the problem from this starting point is typically one third to one half of the number required to solve the relaxation from a cold start. Setting y_0 equal to ϵ makes the dual iterate infeasible; thus it is necessary to regain both primal and dual feasibility. This requires the careful setting of parameters to control the emphasis on regaining feasibility versus achieving optimality. Without care, the algorithm can drift far from optimality while it searches for feasibility. The predictor-corrector algorithm selects the barrier parameter μ automatically to achieve an appropriate balance between optimality and centering; if a different primal-dual method is used, it appears to be beneficial to take a pure centering step before starting to work towards optimality and feasibility.

The optimal solution $\bar{x}$ to $(LPP0)$ will satisfy the added constraint at equality, if the constraint was a necessary cutting plane, so $\bar{y_0}$ will be strictly positive, since most interior point methods give solutions that satisfy strict complementary slackness (see Güler and Ye [16]). Thus, the algorithm must increase y_0 from ϵ to a nontrivial positive number, so it must move away from the face $y_0 = 0$.

Several papers (eg, Anstreicher [1], Mizuno *et al.* [32], Zhang [41]) have 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 (LPP), the primal infeasibility is $s_0 + \mid b_0 - a_0^T \hat{x} \mid$. The total complementary slackness is $\hat{x}^T \hat{z} + \hat{s}^T \hat{y} + (e - \hat{x})^T \hat{w} + s_0 y_0$. To ensure that the ratio between complementary slackness and infeasibility remains reasonably large, it is necessary to set $s_0 y_0$ to a large value. This is going to result in an iterate which is far from centered, because it will be far from satisfying the condition of equation (1), with the 0th component of SYe being far larger than the others.

An alternative approach that we have experimented with is to store and update a point x^{FEAS} which is the relative interior of the polyhedron Q. When a cutting plane is added, the algorithm is restarted from this point, which is strictly feasible in $(LPP0)$. The dual iterate is still infeasible, but only because y_0 must be chosen to be strictly positive. Thus, infeasibility is only a minor concern and it is possible to concentrate on regaining optimality. Initially, x^{FEAS} can be set equal to a convex combination of feasible integer points. This also gives a good initial iterate for (LPP). If the separation routines fail to find any cutting planes, the current iterate is in Q (provided the separation routines are exact) so x^{FEAS} can be updated to the current iterate. Even if the current iterate can be separated from Q, it is often possible to update x^{FEAS} by moving towards the current iterate. Often, this extra work requires only one extra call to the separation routines to determine the feasibility of x^{FEAS} exactly, and then a ratio test. For more details, see Mitchell and Borchers [29].

To the best of our knowledge, none of the interior point column generation algorithms proposed in the literature generate infeasible iterates. They either shift constraints so that the current iterate is still an interior point (for example, [2],[28]), or they back up to a known feasible point (for example, [17, 18],[20],[31]).

Adding many constraints at once

Usually, the separation routines will find several violated constraints. After adding these constraints, the problem has the form

$$\begin{array}{lrcll} \min & c^Tx & & & \\ \text{subject to} & Ax & \geq & b & \quad (LPPA0) \\ & A_0^Tx & \geq & b_0 & \\ & 0 \leq x & \leq & e & \end{array}$$

with a corresponding dual problem

$$\begin{array}{lrcrcrcll} \max & b^Ty & + & b_0y_0 & - & e^Tw & & & \\ \text{subject to} & A^Ty & + & A_0y_0 & - & w & \leq & c & \quad (LPDA0) \\ & & & y, y_0, w & \geq & 0 & & & \end{array}$$

where A_0 is a matrix and b_0 and y_0 are vectors. Again, we can construct a dual feasible solution by taking $y_0 = 0$, and we set each component of y_0 to be a small positive number in order to obtain a strictly positive but slightly infeasible point. In the primal problem, the simplest way to restart is to set the slacks corresponding to the additional constraints to a small positive value, giving an infeasible iterate. Alternatively, as for the single constraint case, we can update to a point x^{FEAS} in the relative interior of Q, if possible.

In many problems, the separation routines return a very large number of constraints. For example, the separation routines for a typical linear ordering problem often find thousands of violated constraints. It is inefficient to add all these constraints at once for a couple of reasons. First, it will take many iterations to solve $(LPPA0)$ because the linear programming relaxation has been dramatically modified, so the algorithm will take several iterations attempting to regain feasibility and recenter before trying to achieve optimality. Secondly, the large constraint matrix will make the linear algebra required at each of these iterations expensive. Thus, it is advisable to add only a subset of the constraints found by the separation routines. Perhaps the simplest way to decide between constraints is to see how badly they are violated by the current iterate — only constraints that are violated by more than some threshold are added. This is generally the method used in simplex based cutting plane algorithms, although refinements have been used.

Consider for example the linear ordering problem, where the cuts correspond to directed cycles.

> **The linear ordering problem:** Given a complete directed graph $G = (V, A)$ with weights c_{ij} on the arcs, find a minimum weight *acyclic tournament*, that is, a subset of the arcs which contains exactly one of the arcs (i, j) and (j, i) for each pair i and j, and which contains no directed cycles.

This problem has applications in economics and archaeology, as well as other areas. It can be formulated as

$$\begin{array}{rll} \min & \sum_{(i,j)\in A} c_{ij} & \\ \text{subject to} & x_{ij} + x_{jk} + x_{ki} & \leq 2 \text{ for each subset } \{i, j, k\} \text{ of } V \qquad (8) \\ & x_{ij} + x_{ji} & = 1 \text{ for each pair } \{i, j\} \text{ in } V \qquad (9) \end{array}$$

$$x_{ij} \qquad = 0,1 \text{ for each arc } (i,j) \tag{10}$$

Grötschel *et al.* [14] developed a simplex based cutting plane algorithm where the initial relaxation consists of the simple bounds $0 \leq x_{ij} \leq 1$ and the equalities $x_{ij}+x_{ji}=1$, and the 3-dicycle constraints (8) are added as cutting planes. They showed that these inequalities were sufficient to solve most of a set of real world problems. They experimented with two alternatives for deciding which constraints to add in their simplex based cutting plane algorithm: firstly, the policy mentioned above of adding all constraints with violation greater than some threshold, and secondly, only adding a subset of the violated constraints with violation greater than some (smaller) threshold, where the cycles corresponding to two constraints in the subset are *arc-disjoint*. It should be noted that constraints added at different times may well share arcs. They found that implementations using the two policies required about the same amount of time, because the second policy resulted in the addition of fewer constraints at each relaxation but, by way of compensation, the solution of several more relaxations. For an interior point cutting plane algorithm, the benefit of adding arc-disjoint constraints is far more pronounced, because the projections required at each iteration can be calculated more efficiently due of the structure of the constraint matrix. This is because the columns of A correspond to the arcs, so if two constraints share an arc then the inner product between the two rows is nonzero; thus, if two constraints correspond to arc-disjoint dicycles then the corresponding entry of AA^T will be zero. It follows that if all the constraints are arc-disjoint dicycles then AA^T is a diagonal matrix, so the Cholesky factor is also diagonal. Of course, constraints added at different times may share arcs, so there will be some fill-in in the Cholesky factor, but ensuring that each set of added constraints corresponds to arc-disjoint constraints does considerably reduce the fill-in.

It has also proved useful in other problems to consider the impact of a constraint on the nonzero structure of the constraint matrix; we discuss this in more detail later. It has been noted by Lustig *et al* [25] that the runtime of an interior point method for linear programming is increased to a time comparable to that required by the simplex algorithm only when the Cholesky factors of the matrix product AA^T are dense, ie, they contain a large number of nonzeroes. In [29], we first ordered the constraints by violation and then examined them in order to see if they were arc-disjoint, keeping only those that did not share any arcs with constraints with larger violations, and this improved the algorithm considerably.

There may be better measures of the "importance" of a constraint than the size of violation. One possibility was proposed by Atkinson and Vaidya [2]: it considers the violation divided by the size of the constraint in the norm defined by the Hessian of the barrier function. Larger values of this quantity correspond to constraints that will have a larger impact on the barrier function. Computational experiments are needed to determine the importance of this selection criterion.

Dropping constraints

One way to improve the performance of a cutting plane algorithm is by judiciously dropping constraints that no longer appear to be important. This keeps the size of the relaxations reasonable, so less work is needed at each iteration to calculate the projections. A further benefit is that smaller linear programming problems require fewer interior point iterations. The simplest way to decide whether to drop a constraint is to check its current value: if the current iterate easily satisfies it, it is a candidate to be dropped.

Dropping a constraint changes the structure of the constraint matrix and the matrix AA^T, so it becomes necessary to find a new good ordering of the columns of AA^T for performing the Cholesky factorization. For this reason, constraints were only dropped when other constraints were added in [29], with good results.

Other criteria can be used to measure the "importance" of a constraint. In particular, the measure proposed by Atkinson and Vaidya [2] can be used: a constraint is dropped if it is easily satisfied and it has little impact on the barrier function. Numerical experiments are necessary to determine the usefulness of this criterion.

Primal heuristics

Primal heuristics, which generate integer solutions to (IP) from fractional solutions to (LPP), can be very useful in some circumstances. If the heuristics are cheap, they can be used at every iteration. However, it is usually more cost effective to call them only when the separation routines are also called.

One of the major advantages of using a cutting plane approach to solve an integer programming problem is that it provides a guarantee of optimality, and the current dual value provides a lower bound on the optimal value of the integer program. It is occasionally possible to stop the algorithm early by using the primal integer solution generated by the heuristics in conjunction with a good dual solution to (LPD), especially if the entries of the objective function c are integer.

If the interior point cutting plane method is converging to a fractional optimal point in the polyhedron Q, the primal heuristics can give an optimal solution to (IP). This is obviously desirable in order to claim to have solved (IP), and it may also prevent unnecessary calls to the separation routines.

The primal heuristics can also be used to update x^{FEAS}. Recall that x^{FEAS} is supposed to be a good point in Q, so it can be updated by taking a convex combination of the old x^{FEAS} and the good solution returned by the primal heuristics. Care should be taken to ensure that x^{FEAS} does not get too close to a non-optimal integral point, because it would then not be a good restarting point, as argued above.

Multiple optimal solutions and degeneracy

Some integer programming problems have more than one optimal solution. In this case, the set of optimal solutions to the linear programming problem $\min\{c^Tx : x \in Q\}$ is a face of Q of dimension at least one. It is then likely that an interior point cutting plane algorithm will converge to a point in the interior of this optimal face, and thus it will not converge to an integer point. Perhaps the simplest way to find an integer optimal point is to use primal heuristics to convert the fractional point into a nearby integer point. Alternatively, methods have been developed in the linear programming context for converting from a point in the interior of the optimal face to an extreme point of that face — see, for example, Megiddo [26]. Thus, once it has been determined that we have found a point on the optimal face of Q, it should be possible to convert that point into an integer point.

If the process is converging to an optimal fractional point and if the primal heuristics do not produce an optimal integer point then a cutting plane algorithm may call the separation routines several times without success. This may produce a drag on performance of the algorithm, and this is the major cost involved with converging to a fractional optimal point. In such a situation, it may be worthwhile to invest more time in the primal heuristics in an attempt to find an optimal integer point. More computational testing is required to determine the importance of this issue.

In some cases, the fractional optimal solution may actually provide *more* information about the problem than an integer extreme point of the optimal face, because every integer optimal point has nonzero components only where the fractional point has nonzero components. For example, some of the linear ordering problems solved in [29] have multiple optimal solutions. In this case, the fractional optimal solutions provided by the cutting plane method correspond to partial orderings, and all optimal full orderings are implied by these fractional orderings.

Degeneracy of the linear programming relaxation is probably not a disadvantage for an interior point cutting plane algorithm, because interior point methods are not usually bothered by degeneracy. Thus, an interior point cutting plane method may possess a distinct advantage over a simplex one if the relaxation is degenerate. If the problem has multiple optimal solutions then the final few relaxations are probably dual degenerate, so the simplex algorithm may slow down in the final few stages. Again, this is an issue that needs more computational investigation.

Fixing variables

Simplex branch and cut algorithms use reduced costs to fix integral variables at zero or one as follows. Let v_{UB} be the value of the best integer solution found so far, let v_{LB} denote the optimal value of the current relaxation, and let r_i denote the reduced cost for primal nonbasic variable x_i. If x_i has value zero at the optimal solution to the current relaxation, then any feasible primal solution to the current relaxation with $x_i = 1$ must have value at least $v_{LB} + r_i$. Thus, if $v_{LB} + r_i > v_{UB}$ then x_i can be fixed at zero; no integral solution with $x_i = 1$ that is feasible in the current relaxation will be better than the best known integer solution. Similarly, if $x_i = 1$ at the optimal solution to the current relaxation and if $v_{LB} - r_i > v_{UB}$ then x_i can be fixed at one.

It is not necessary to have the optimal solution in order to fix variables, and it is possible to use the current dual solution to derive a criterion for fixing variables when using an interior point method. Any dual feasible solution provides a lower bound on the value of the relaxation. Let $\bar{x}$ be the current primal solution, let $\bar{y}$, $\bar{w}$ be the current dual solution, let $\bar{z}$ be the corresponding dual slack, and let $\bar{v}_{LB}$ be the dual value. Then $c = A^T\bar{y} - \bar{w} + \bar{z}$, so any primal feasible solution x to the current relaxation (LPP) will have value

$$\begin{aligned} c^T x &= (A^T\bar{y} - \bar{w} + \bar{z})^T x \\ &= \bar{y}^T A x + \bar{z}^T x - \bar{w}^T x \\ &\geq b^T\bar{y} + \bar{z}^T x - \bar{w}^T x \quad \text{since } Ax \geq b \text{ and } \bar{y} \geq 0 \\ &= b^T\bar{y} - e^T\bar{w} + \bar{z}^T x + \bar{w}^T(e - x) \\ &= \bar{v}_{LB} + \bar{z}^T x + \bar{w}^T(e - x). \end{aligned}$$

Thus, any feasible solution to (LPP) with $x_i = 1$ will have value at least $\bar{v}_{LB} + z_i$, so x_i can be fixed at zero if $\bar{v}_{LB} + z_i > v_{UB}$. Similarly, x_i can be fixed at one if $\bar{v}_{LB} + w_i > v_{UB}$. Note that we may be able to fix a variable at zero or one even if it is currently fractional. Fixing some variables using duality may have logical implications for the other variables. For example, if it is possible to fix one of the edges in a matching problem at one, then all other edges adjacent to the end points of that edge must have value zero. When variables are fixed, some constraints may become redundant, and these constraints should be dropped. If we again consider the matching problem with an edge fixed at one, the degree constraints (5) for the endpoints of that edge are now both redundant.

If it is possible to fix a large number of variables, the amount of linear algebra required at each iteration will be reduced; in addition, the number of iterations needed to solve the current relaxation will also probably be reduced. The drawback to fixing variables is that the current solution becomes infeasible; the restarting point x^{FEAS} also has to be modified.

The complete algorithm

The discussion so far leads to a cutting plane algorithm which has the following form:

1. **Initialize:** Set up the initial relaxation. Pick an initial iterate which strictly satisfies the required nonnegativity bounds on all primal and dual variables (including slack variables). A standard method for initializing an interior point linear programming algorithm can be used, for example, the method given in Lustig *et al.* [25]. Set a tolerance τ_1 for optimality. Set a tolerance $\tau_2 > \tau_1$ for searching for cutting planes.

2. **Inner iteration:** Perform an iteration of the linear programming solver being used.

   - If the difference in value between the best integer solution and the best dual solution found so far is smaller than τ_1, STOP with optimality.
   - If the duality gap between the current primal and dual values is smaller than τ_2, go to Step 3.
   - If the duality gap between the current primal and dual values is smaller than τ_1, the process has converged to a fractional point which is on the optimal face of Q, so use a basis refinement technique to obtain an optimal integer solution. Then STOP.

   Repeat this step.

3. **Primal heuristics:** Use heuristics to obtain a good integer solution using the current fractional primal solution. If the heuristics return the best integer solution found so far, store this solution.

4. **Look for cutting planes:** Use separation routines to find violated cutting planes. If no violated cutting planes are found, reduce τ_2 and return to Step 2; otherwise, go to Step 5.

5. **Add cutting planes:** Select and add a subset of the violated constraints to the relaxation. Increase τ_2 if many violated constraints were found; decrease τ_2 if only a few cuts were found.

6. **Fix variables:** If the dual solution implies that variables must take certain values, fix those variables, if desired.

7. **Drop cutting planes:** Drop any constraints which no longer appear to be important.

8. **Modify current iterate:** Increase any small components to some value ϵ. If μ is not selected by the predictor-corrector methodology, perform a centering step and update the centering parameter μ, if appropriate. Return to Step 2.

This algorithm will converge to the optimal solution, provided the separation routines are guaranteed to find a violated cutting plane if one exists. If the separation routines may fail to find a violated constraint even if the current primal point is not in Q, then it may be necessary to use branch and bound to find an optimal solution to the integer program.

If it is possible to maintain and update a point x^{FEAS} in the relative interior of Q, this should be done at Step 5. This point can then be used to update the primal solution in Step 8.

The structure of the constraint matrix

As mentioned above, interior point methods do not work so well if the Cholesky factors of the matrix AA^T become dense. This should be taken into account when deciding which constraints to add. On a more fundamental level, it will also affect the choice of whether to use an interior point or cutting plane interior point method. If the structure of the constraint matrix is unsuitable, a simplex cutting plane algorithm will probably outperform an interior point one.

The paper by Mitchell and Todd [31] describes an interesting manifestation of the problem of decreased performance when the Cholesky factors become dense. In this paper, an interior point method was used to solve matching problems. Two different distributions were used to generate test problems. The first randomly placed points in a unit square. The odd set constraints that are generated tend to correspond to neighbouring points. This results in a matrix AA^T that can be permuted to have small bandwidth, so the Cholesky factors are sparse. The second distribution randomly generated distances between vertices. In this case, the odd set constraints found by the algorithm are not nicely related to one another and the matrix AA^T does not have a good structure, resulting in a lot of fill in the Cholesky factors. The upshot of this was that although problems generated using the first distribution required more iterations, they required considerably less time. When Grötschel and Holland [13] solved matching problems using a simplex-based cutting plane algorithm, they were able to solve problems generated using the second distribution considerably quicker than those generated using the first distribution, illustrating the fact that the structure of the constraint matrix is not so important with a simplex algorithm.

As pointed out by Saunders [37], Vanderbei [40] and others, there are alternative methods for finding projections than calculating Cholesky factors. It may happen that if these alternative methods are used then cutting plane interior point methods will not be so sensitive to fill in of the matrix AA^T and its Cholesky factors.

2.2 Interior point branch and bound methods

Branch-and-bound is a method of 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 (LPP) 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 [35].

When using branch-and-bound, one of four things can happen at each node of the tree. The subproblem could be unbounded; 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.* [9] to determine that one of the integer variables is tending to a fractional value, and therefore that we 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 (LPD) is dropped, so the previous solution to (LPD) is still feasible. One variable in (LPP) is fixed, so infeasibilities are introduced into (LPP). Despite this, it is still possible to solve the child node quickly [4, 5].

A branch and bound interior point algorithm has the following form:

1. **Initialize:** Pick an initial relaxation. Choose an initial primal and dual iterate, using, for example, the method of Lustig *et al.* [25]. 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.

   - 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).
   - 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.
   - If the dual value of the current node is greater than the best known upper bound, prune this node.
   - If we can find a dual ray, showing that the primal problem is infeasible, prune this node.
   - 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. (We discuss this step in more detail later.)

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.

Terminating the current relaxation early

The solution to a parent subproblem can be used as a warm start for a child subproblem. For this to be a good warm start for an interior point method, it should be somewhat centered, and therefore it is useful to terminate solution of the parent problem early. 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. 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.

The paper by El-Bakry *et al* [9] describes a good indicator for determining which variables are tending towards zero and which ones are tending towards nonzero values. In particular, if x_i^k represents ith component of the kth iterate then

$$\begin{array}{rcll} \lim_{k\to\infty} x_i^{k+1}/x_i^k & = & 1 & \quad \text{if } x_i^k \not\to 0 \\ \lim_{k\to\infty} x_i^{k+1}/x_i^k & = & 0 & \quad \text{if } x_i^k \to 0 \end{array}$$

Similar observations apply to the primal slack variables, including those for the constraints $x \le e$, and also to the dual variables. If a primal variable x_i appears to be tending towards a nonzero value, and if this is confirmed by $(1 - x_i)$ tending to a nonzero value and also the corresponding dual slack and w_i tending towards zero, then it is reasonable to conclude that the optimal solution to the relaxation is fractional. It will follow that it will be necessary to split this subproblem into two children if the optimal value of the subproblem is smaller than that of the incumbent integer feasible solution to (IP).

There are three tests used in [5] 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.

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.* [24]). 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

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

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

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

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

$$\begin{array}{lrcrcrcll}
\max & b^Ty & - & e^Tw & - & w_0 & & & \\
\text{subject to} & A^Ty & - & w & & & \leq & c & \qquad (LDparent) \\
 & a_0^Ty & & & - & w_0 & \leq & c_0 & \\
 & & & y, & w, & w_0 & \geq & 0, &
\end{array}$$

and

$$\begin{array}{lrcrcll}
\min & \bar{b}^Ty & - & e^Tw & & & \\
\text{subject to} & A^Ty & - & w & \leq & c & \qquad (LDchild) \\
 & & & y, w & \geq & 0, &
\end{array}$$

we notice that the approximate solution $y = y^*$, $w = w^*$ (and $w_0 = w_0^*$) 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^* may be 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 y^* and w^* or the primal or dual slacks 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.

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 [19] 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.

Degeneracy

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 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.

3 Extensions

In this section we describe some other aspects of interior point algorithms for integer programming problems. The algorithms discussed in section 2 generally required that both the primal and the dual iterates be strictly feasible; in section 3.1 we consider algorithms which only require that the dual iterates be strictly feasible. In section 3.2 we discuss a different approach to solving integer programming problems using interior point methods; this method works by solving an equivalent non-convex quadratic programming problem. In section 3.3 we briefly consider implementing branch and cut algorithms in parallel. Finally, in section 3.4 we consider theoretical issues.

3.1 Algorithms which only require positive dual iterates

The primal-dual cutting plane algorithm described in section 2.1 uses iterates which are interior in both the primal and the dual feasible region at each iteration. There has been some research on using interior point methods which only require an interior point in the dual problem; a solution to the primal problem is only used to provide an upper bound on the optimal value. These algorithms are generally adaptations of primal potential function methods.

The first experiments in using an interior point method in a cutting plane algorithm were in Mitchell and Todd [31]. They applied the primal potential reduction algorithm to the dual (LPD) of the linear programming relaxation of (IP). The advantage of this approach is that it is not necessary to have an iterate which is feasible in (LPP). The primal solution is updated using the method of Todd and Burrell [38]. Mitchell and Todd found in their experiments on matching problems that it was best to search for cutting planes as soon as the primal solution was updated. When a constraint was added, they updated the solution to the new relaxation $(LPD0)$ by updating y and w by moving in a direction which is a projection of a_0 onto the row space of an appropriately rescaled version of the constraint matrix for (LPD). They used heuristics to round the current solution to (LPP) to a point feasible in (IP). When the relaxation was updated, the best solution to (IP) found so far was used as the new primal iterate. They were able to obtain promising results.

Goffin and Vial [12, 10] used a similar algorithm to solve nonsmooth optimization problems using a column generation algorithm. They obtained results comparable with or superior to other algorithms for these problems.

3.2 Solving an equivalent quadratic programming problem

The integer programming feasibility problem can be stated

$$\begin{array}{ll} \text{Find a point } x \text{ in } \Re^n \text{ satisfying} & (IPfeas) \\ Ax \geq b, x_i \in \{0,1\}, i = 1, \ldots, n, & \end{array}$$

where A is an $m \times n$ matrix and b is an m-vector. This can be solved by solving the nonconvex quadratic programming problem

$$\begin{array}{llcll} \min & x^T(e-x) & & & \\ \text{subject to} & Ax & \geq & b & (QP) \\ & x & \leq & e & \\ & x & \geq & 0. & \end{array}$$

Note that for every binary point, the objective function $x^T(e-x)$ has value zero and for every fractional point this inner product has positive value. Kamath *et al* [23, 21, 22] have investigated using an interior point method to solve this quadratic programming problem, and hence to solve the integer programming problem (IP). Their algorithm attempts to minimize the potential function

$$(m+2n)\log(x^T(e-x)) - \sum_{i=1}^{m} \log s_i - \sum_{i=1}^{n} \log(x_i) - \sum_{i=1}^{n} \log(1-x_i),$$

where $s = Ax - b$. They show that x^* is a global minimizer of this potential function if and only if it is a feasible solution to $(IPfeas)$. The potential function may have local minimizers which are not global minimizers; therefore, the set of constraints is augmented to cut off local minimizers which are not feasible in $(IPfeas)$, if necessary. They used their algorithm to solve satisfiability problems from inductive inference, obtaining promising results compared to an implementation of the Davis-Putnam procedure [6] (an implicit enumeration procedure which is similar to branch and bound).

3.3 Parallel implementations

Compared to the simplex algorithm, interior point methods typically require only a few iterations to solve a linear programming problem, with each iteration requiring a considerable amount of time. The linear algebra required at each iteration of an interior point method can be implemented in parallel, with the savings in time increasing as the number of processors increases. By contrast, the potential time savings from implementing the simplex algorithm in parallel are more limited.

This potential superiority of interior point methods for linear programming can be usefully exploited in a cutting plane algorithm. Furthermore, the separation routines can be run in parallel with the linear programming solver, making it possible to carefully choose when to add cutting planes. Separation routines typically require less time than the linear programming parts of an implementation, so it may be possible to implement more sophisticated separation routines if one or more processors are dedicated to them. A simplex cutting plane algorithm generally needs the optimal fractional solution to the relaxation before calling the separation routines, because the basic feasible solution can change dramatically in the last few iterations; thus, it is harder to run the separation routines and the linear programming solver in parallel when using the simplex algorithm.

Branch and bound can be implemented efficiently in parallel, even on machines with a large number of processors; see, for example, Eckstein [7]. Thus, the difficulty with parallelizing the simplex algorithm need not be a drawback for a branch and bound algorithm: once there are many nodes in the tree, the algorithm can work on them simultaneously.

3.4 Theoretical behaviour of column generation algorithms

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 polynomial time 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.* [15]. 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 [39]. This algorithm uses the *volumetric center*, so its analysis differs from that of more standard interior point methods.

Atkinson and Vaidya [2] 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 [30] developed a barrier function cutting plane algorithm using some of the ideas from [2]. They showed some links between the notion of a point being centered (see, for example, Roos and Vial [36]) and the criteria for a constraint to be added or dropped in [2]. 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.* [11] 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. Their paper used a result of Nesterov [33] which placed a bound on an important quantity in the analysis of column generation interior point methods.

4 Conclusions

We have concentrated on discussing the possibility of incorporating interior point methods into cutting plane and branch and bound algorithms for linear programming. In order to do this successfully, it is necessary to be able to use a warm start somewhat efficiently. 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. Other techniques and potential pitfalls are discussed in section 2.

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.

References

[1] K. M. Anstreicher. A combined phase I – phase II scaled potential algorithm for linear programming. *Mathematical Programming*, 52:429–439, 1991.

[2] D. S. Atkinson and P. M. Vaidya. An analytic center based cutting plane algorithm for convex programming. Technical report, Department of Mathematics, University of

Illinois at Urbana-Champaign, June 1992.

[3] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno. Very large–scale linear programming : A case study in combining interior point and simplex methods. *Operations Research*, 40:885–897, 1992.

[4] B. Borchers. *Improved branch and bound algorithms for integer programming.* PhD thesis, Rensselaer Polytechnic Institute, Mathematical Sciences, Troy, NY, 1992.

[5] B. Borchers and J. E. Mitchell. Using an interior point method in a branch and bound algorithm for integer programming. Technical Report 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1991. Revised July 7, 1992.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 7:201–215, 1960.

[7] J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the CM–5. Technical Report TMC–257, Mathematical Sciences Research Group, Thinking Machines Corporation, 245 First Street, Cambridge MA 02142, September 1993.

[8] J. Edmonds. Maximum matching and a polyhedron with 0, 1 vertices. *Journal of Research National Bureau of Standards*, 69B:125–130, 1965.

[9] A. S. El–Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior–point methods. Technical Report TR–91–15, Dept. of Mathematical Sciences, Rice University, Houston, TX 77251, USA, 1991. Revised December 1992. To appear in *Siam Review.*

[10] J.-L. Goffin, A. Haurie, and J.-P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.

[11] 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.

[12] J. L. Goffin and J. P. Vial. Cutting planes and column generation techniques with the projective algorithm. *Journal of Optimization Theory and Applications*, 65:409–429, 1990.

[13] M. Grötschel and O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985.

[14] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[15] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, Germany, 1988.

[16] O. Güler and Y. Ye. Convergence behavior of interior point algorithms. *Mathematical Programming*, 60:215–228, 1993.

[17] 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.

[18] 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.

[19] IBM. *IBM Optimization Subroutine Library Guide and Reference*, August 1990. Publication number SC23–0519–1.

[20] J. A. Kaliski and Y. Ye. A decomposition variant of the potential reduction algorithm for linear programming. Technical Report 91–11, Department of Management Sciences, University of Iowa, Iowa City, Iowa 52242, June 1991.

[21] A. P. Kamath and N. K. Karmarkar. A continuous method 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*, pages 125–140. Princeton University Press, Princeton NJ, 1992.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] N. Megiddo. On finding primal– and dual–optimal bases. *ORSA Journal on Computing*, 3:63–65, 1991.

[27] S. Mehrotra and J. Sun. On the implementation of a (primal–dual) interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[28] J. E. Mitchell. An interior point column generation method for linear programming using shifted barriers. Technical Report 191, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, November 1990. Revised August, 1992. To appear in *SIAM Journal on Optimization.*

[29] 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.

[30] 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.

[31] J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, 56:245–284, 1992.

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

[33] Y. Nesterov. Cutting plane algorithms from analytic centers: efficiency estimates. Technical report, University of Geneva, Geneva, Switzerland, December 1992.

[34] M. W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[35] R. G. Parker and R. L. Rardin. *Discrete Optimization*. Academic Press, San Diego, CA 92101, 1988.

[36] C. Roos and J. P. Vial. A polynomial method of approximate centers for linear programming. *Mathematical Programming*, 54:295–305, 1992.

[37] M. A. Saunders. Major Cholesky would feel proud. *ORSA Journal on Computing*, 6(1):23–27, 1994.

[38] M. J. Todd and B. P. Burrell. An extension of Karmarkar's algorithm for linear programming using dual variables. *Algorithmica*, 1:409–424, 1986.

[39] 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*.

[40] R. J. Vanderbei. Interior-point methods: algorithms and formulations. *ORSA Journal on Computing*, 6(1):32–34, 1994.

[41] 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.