Computational experience with an interior point cutting plane algorithm

John E. Mitchell[1]

Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180

February 24, 1997
Revised: March 4, 1999 and December 28, 1999

Abstract

There has been a great deal of success in the last twenty years with the use of cutting plane algorithms to solve specialized integer programming problems. Generally, these algorithms work by solving a sequence of linear programming relaxations of the integer programming problem, and they use the simplex algorithm to solve the relaxations. In this paper, we describe experiments using a predictor-corrector interior point method to solve the relaxations. For some problems, the interior point code requires considerably less time than a simplex based cutting plane algorithm.

Keywords: Interior point methods, integer programming, cutting planes, linear ordering, Ising spin glasses, maxcut.

1 Introduction

Any integer linear programming problem can be written $\min\{c^T x : x \in S, x_i = 0, 1 \, \forall i\}$, where S is a polyhedron. Often, a good solution can be found by heuristic methods such as local search, tabu search, simulated annealing, genetic algorithms, or algorithms specific to the particular problem; this heuristic solution may well be optimal. It is usually harder to prove optimality. Algorithms such as branch and bound, cutting plane approaches, and branch and cut can be used to obtain lower bounds on the optimal value, and if the algorithms are allowed to run for long enough, they will reduce the gap between the upper and lower bounds to zero and thus find the optimal solution. Cutting plane algorithms form a linear programming relaxation of the integer programming problem, solve the relaxation to obtain a lower bound on the optimal value of the integer program, and, if the upper and lower bounds do not agree, improve the relaxation and repeat the process. Cutting plane methods can be incorporated into a branch and bound method to give a branch and cut algorithm.

[1]Research supported in part by ONR grant number N00014-94-0391 and NSF grant number CCR–9901822

Cutting plane and branch and cut algorithms have been successfully used to solve many types of integer linear programming problems, including the traveling salesman problem [1, 20, 39], the linear ordering problem [21], clustering problems [24], and the maximum cut problem [2]. See Jünger *et al.* [26] for a survey. The simplex algorithm was used to solve the linear programming relaxations in all of these references.

Interior point algorithms are now a very good alternative to the simplex method for linear programming problems, and they are superior for large problems where the structure of the nonzeroes in the constraint matrix is not too unfavourable. (See, for example, [30].) It is natural to investigate the use of interior point methods in a cutting plane algorithm. The successful use of an interior point method in this setting requires the ability to exploit a *warm start*: the solution to one relaxation should be close to the solution to the next relaxation in some sense, so it should require relatively few iterations to solve the next relaxation from this warm start as opposed to starting from a cold start that does not exploit this information. The simplex method appears to be fairly adept at exploiting the warm starts provided in a cutting plane algorithm, but equally efficient ways to restart when using an interior point method are not known. A general methodology is proposed by Gondzio [19] with encouraging computational results; as with the results we present in this paper, the principal emphasis in a restart method has to be to restart with an iterate that is centered. The principal technique we use is *early termination*: the relaxations are solved approximately, which results in an initial iterate for the next relaxation that is somewhat centered, leading to better performance.

Mitchell and Todd [38] presented a promising first attempt at using an interior point cutting plane algorithm, solving matching problems. An interior point cutting plane algorithm for the linear ordering problem was described in Mitchell and Borchers [35]. The computational times in that paper were comparable to those obtained by Jünger and Reinelt [21, 43] with a cutting plane algorithm which used the simplex solver CPLEX3.0 [10] to solve the linear programs. Interior point approaches to integer programming problems are surveyed in [37]; this reference includes discussions of the theoretical performance of interior point cutting plane algorithms and of other applications of interior point column generation methods.

In the current paper, we present results on two classical integer programming problems, namely, the MAXCUT problem and the linear ordering problem. The MAXCUT instances arise from finding the ground state of Ising spin glasses, a problem in statistical physics. Our results appear to be considerably better under one distribution of the data than recent results in the literature [13] obtained using the simplex solver in CPLEX3.0. These are the hardest problems considered in this paper, requiring a more conservative choice of parameters than the other problems in order to obtain a robust implementation. We improve somewhat on the results in [35] for real-world linear ordering problems and also look at some larger randomly generated problems, obtaining better runtimes on some of these problems with our interior point method than with a cutting plane algorithm using the simplex solver in CPLEX4.0. Because of extensive experimentation, we are able to be more confident and therefore more specific about our choices of parameters than in [35]. Our algorithm is presented in §2. The results for linear ordering problems and Ising spin glass problems are contained in §3.1 and §3.2, respectively.

Many different integer programming problems can be formulated using the framework (IP) that we present in §2; the great majority of research on polyhedral theory and cutting plane algorithms is on problems that can be written in this form (see, for example, [7, 26, 34]). Of course, not all of these problems are equally amenable to the interior point cutting plane approach that we present in this paper. We return to the issue of determining appropriate problems for the interior point approach in the conclusions §4. One requirement for this investigation is that the linear programming relaxations should be large and yet the integer programming problems are solvable, so we examined problems where the time required to solve the linear programming relaxations is a substantial portion of the total solution time. For this paper we restrict our attention to problems that can be solved at the root node of a branch-and-cut tree, for several reasons, including the following two: interior point branch-and-bound is not well understood (see, for example, [28]), and the time to solve large problems that require branching is impracticable for this investigation.

2 An Interior Point Cutting Plane Algorithm

We assume we have an integer programming problem of the form

$$\begin{array}{lrcll} \min & & c^T x & & \\ \text{subject to} & & Ax & = & b \qquad\qquad (IP) \\ & 0 \leq & x & \leq & u \\ & & x_i & = & 0 \text{ or } 1 \text{ for } i \in I \\ & & x & & \text{satisfies some additional conditions} \end{array}$$

where x, c, and u are n-vectors, b is an m-vector, A is an $m \times n$ matrix of rank m, and I is the set of integer variables. We assume $u_i = 1$ for $i \in I$. We assume the additional conditions can be modelled as a (possibly exponential) set of linear constraints. Many problems can be cast in this framework; for example, the traveling salesman problem can be represented in this form, with the additional conditions being the subtour elimination constraints [20, 39] and the conditions $Ax = b$ representing the degree constraints that the tour must enter and leave each vertex exactly once. Some problems do not need additional conditions, and we regard such problems as also falling in our general framework. We let Q denote the convex hull of feasible solutions to (IP). We assume that the dimension of Q is $n - m$. The linear programming relaxation (or *LP relaxation*) of (IP) is

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

with dual

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

where y is an m vector and w and z are n-vectors. The value of any feasible solution to (LD) provides a lower bound on the optimal value of (IP). We solve (LP) and (LD) using

a predictor-corrector primal-dual interior point method similar to those described in Lustig *et al.*. [30] and Mehrotra [31]. This algorithm keeps x, w, z, and the primal slacks $s := u - x$ strictly positive. We call such a point an *interior point*. The method is a barrier method, finding a sequence of approximate analytic centers in order to approach the optimal solution, where an analytic center is a solution to $\min\{c^T x - \mu \sum_i \ln(x_i(u_i - x_i)) : Ax = b\}$ for some positive scalar μ. All iterates generated by the algorithm will satisfy $Ax = b$, as described later.

If the optimal solution to (LP) is feasible in (IP) then we can stop with optimality. If the optimal basic feasible solution x^{LP} to (LP) is not in Q then we cut off x^{LP} by adding an extra constraint or *cutting plane* of the form $a^{0^T} x \leq b_0$. If the integer programming problem is NP-hard then it is also NP-hard to find a violated cutting plane [23], so heuristics are usually used to generate cuts. This gives the relaxation

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

where x_0 is a new fractional variable giving the slack in the added constraint. The cutting plane is a valid inequality for (IP) but it is violated by the optimal solution x^{LP}. We then solve $(LPnew)$, and repeat the process. In this paper, the cutting planes we add are generally facets of Q, and we use specialized routines to find the cutting planes. The dual problem to $(LPnew)$ is

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

Every iterate $\hat{x}, \hat{y}, \hat{w}, \hat{z}$ generated by an interior point method before reaching optimality will satisfy $0 < \hat{x} < u$ and $\hat{w} > 0$, $\hat{z} > 0$. These can be used to 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.

It has been observed that 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 $(LPnew)$, so we search for cutting planes violated by $\hat{x}$ before reaching optimality. Such cutting planes may well be *deeper cuts* and cut off more of the part of the feasible region that is close to the optimal solution to (LP), because the iterate is further than the optimal solution from the boundary of the polyhedron.

The two principal disadvantages of looking for cuts before solving the current relaxation to optimality are, first, we may be unable to find any cuts, so the search is a waste of time,

and second, 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 be minimized by moving towards the optimal solution from the center of the polyhedron, reducing the likelihood of violating cutting planes that are satisfied by the optimal solution to (LP). To reduce the impact of the first disadvantage, we use a *dynamically altered tolerance* τ for deciding when to search for violated cutting planes, searching only when the duality gap drops below this tolerance. This tolerance is increased if we find a large number of violated constraints, and decreased if we find only a few violated constraints.

As mentioned earlier, we can obtain a new feasible interior iterate for $(LDnew)$ by setting $y_0 = 0$ and $w_0 = z_0 = \epsilon_D$ for some appropriate small positive value of ϵ_D. We chose $\epsilon_D = 10^{-3}$, which is considerably larger than the 10^{-6} used in [35]. To improve stability and performance, it is useful to also increase any small components of w and z up to ϵ_D.

We update the primal iterate using 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 $(LPnew)$. We used the vector of all halves as an initial point of this type for both problem classes considered in this paper. This point is updated as the algorithm progresses, by combining it with any iterate which is in the convex hull. We can restart either at this feasible point or at an appropriate convex combination of this point and the previous iterate. To improve stability and performance, it is useful to also increase any small components of x and s to $\epsilon_P := 10^{-5}$.

In practice, many constraints are added at once. The same procedures for finding initial solutions to the new primal and dual relaxations can still be used.

Cutting plane algorithms are useful for proving optimality by generating lower bounds on the optimal value of (IP). Fractional primal points x can also be used to generate new feasible solutions to (IP) by using problem-specific rounding heuristics. 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 good primal heuristics, the algorithm may search in vain for cutting planes, and be forced to branch, resulting in longer run times.

It is useful to drop constraints that no longer appear important. This has the advantage of shrinking the size of the relaxation, with the principal benefit of reducing the time required for each iteration, and the marginal benefit of very slightly reducing the number of iterations to solve a relaxation. Generally, we do not discard a constraint for several stages, and we drop the constraint if its slack variable is large — see §3 for more details. Note that if the slack variable is large then the corresponding dual variable y will be close to zero. More sophisticated tests are available, but the costs of these outweigh the benefits of the reduction in the size of the relaxations.

Simplex branch and cut methods can use reduced costs to fix variables at zero or 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, as described

in [32]. Fixing variables has the practical disadvantage of making the old restart point for ($LPnew$) no longer feasible, because this restart point is interior. Fixing some variables may impose logical constraints on other variables, so the restart point usually has to be modified and these additional logical constraints sometimes have to be added to the model. We did not find it necessary to fix variables for problems with integral objective function coefficients.

We summarize the complete algorithm in Figure 1. Note that more details can be found in §3 for the two problem classes considered in this paper. We say that we have completed a *stage* every time we enter Step 10. We complete the final stage when we enter Step 5 for the last time. The set of appropriate constraints in Step 7 is usually obtained using a bucket sort. The results in this paper represent an improvement over those obtained with a similar algorithm for linear ordering problems in [35], with a reduction in the number of iterations as well as the runtime. The principal differences are the use of larger restart parameters ϵ_P and ϵ_D in Step 10, keeping constraints for more stages before allowing them to be dropped in Step 8, slightly changing the method for updating τ in Step 7, and using different parameters to choose the appropriate subset, including adding a larger number of constraints.

3 Computational results

We have used this algorithm to solve several different problems in combinatorial optimization. In this section, we describe the modifications made to the basic algorithm and give computational results for each problem. The computer code was written in FORTRAN 77. We have a framework where the majority of the code remains the same for each problem, and we use problem specific subroutines for initializing the problem, finding primal integral solutions using heuristics, finding cutting planes, and modifying the relaxation by adding and dropping constraints. All the computational testing was performed on a Sun SPARC 20/71 UNIX workstation. All runtimes are reported in seconds.

We use the Yale Sparse Matrix Package [16] to calculate the projections, using the routine `mmd` due to Liu [29] to find an ordering of the columns of ADA^T for the Cholesky factorization of this matrix, where D is an appropriate diagonal matrix. Our interior point linear programming solver could be improved. It is probably about two to three times slower than commercial solvers such as CPLEX [10]. In particular, some of the linear algebra routines could be improved. We do not use a publically available code such as HOPDM ([18]) or PCx ([11]), because none of these codes makes it easy to access the current solution after each iteration, stop the process when desired, suggest a new starting point, and not preprocess each relaxation, which are all required features of our algorithm.

3.1 The linear ordering problem

3.1.1 Definition of the problem

The linear ordering problem is a combinatorial optimization problem with a wide variety of applications, such as triangulation of input-output matrices, archeological seriation, minimizing total weighted completion time in one-machine scheduling, and aggregation of individual preferences. It is NP-hard (Karp [27]), and a complete description of the facets of its con-

1. **Initialize.** Set up the initial relaxation. Find initial interior primal and dual points. Find a feasible point in Q. Find a restart point x^{FEAS} in the relative interior of Q for use in Step 10.
2. **Inner iteration.** Perform one iteration of the primal dual algorithm.
3. **Check for early termination.** If the relative duality gap is larger than the tolerance τ, return to Step 2.
4. **Primal heuristics.** Use the primal heuristics to try to improve on the current best solution to (IP).
5. **Check for optimality.** The current dual solution provides a lower bound and the value of the best known feasible point provides an upper bound. If the difference between these two is sufficiently small, **Stop** with optimality.
6. **Look for cutting planes.** If possible, also update the known feasible point x^{FEAS}.
7. **Add cutting planes.** If any cutting planes were found in Step 6 then add an appropriate subset; otherwise, reduce τ and return to Step 2.
8. **Drop cutting planes.** If any cutting plane appears to no longer be important, drop it.
9. **Fix variables.** If possible, fix variables at zero or one.
10. **Modify current iterate.** Increase any small components of w and z to a small value ϵ_D. If necessary, increase appropriate components of w and/or z to regain dual feasibility. Update the primal solution to a convex combination of the current iterate and x^{FEAS}, giving a point which is interior in the new relaxation. Increase any small components of x and the vector of primal slacks to ϵ_P. Modify the tolerance τ. Return to Step 2.

Figure 1: An interior point cutting plane algorithm

vex hull is not known. The polyhedral structure of the linear ordering problem has been investigated by Grötschel, Jünger and Reinelt [21, 25, 42].

The problem requires placing p sectors (or objects) in order, where there is a cost g_{ij} for placing sector i before sector j. It was shown by Grötschel *et al.* [21] that the linear ordering problem with p sectors is equivalent to the following integer programming problem:

$$\begin{array}{rll} \min & \sum_{1\le i<j\le p} c_{ij}x_{ij} & \\ \text{subject to} & x_{ij}+x_{jk}-x_{ik} & \le 1 \text{ for } 1\le i<j<k\le p \qquad (LO) \qquad (1)\\ & -x_{ij}-x_{jk}+x_{ik} & \le 0 \text{ for } 1\le i<j<k\le p \qquad (2)\\ & x_{ij} & = 0 \text{ or } 1 \text{ for } 1\le i<j\le p \end{array}$$

where $c_{ij} = g_{ij} - g_{ji}$ for $1 \le i < j \le p$. Here, we obtain $x_{ij} = 1$ if i is before j in the ordering, and $x_{ij} = 0$ otherwise. Equations (1) and (2) are called *triangle inequalities*; they prevent solutions x which correspond to, for example, sector i before sector j, sector j before sector k and sector k before sector i.

3.1.2 Details of the algorithm

The initial linear programming relaxation of (LO) is $\min\{c^T x : 0 \le x \le e\}$, where c and x are $p(p-1)/2$ vectors, and e is the $p(p-1)/2$ vector of ones. (Throughout, we use e to denote the vector of ones of an appropriate dimension.)

The only cutting planes we add are triangle inequalities of the form given in equations (1) and (2) — these were sufficient to solve most of the problems in our test set. We first called the separation routines when the relative duality gap (the duality gap divided by the larger of the absolute value of the dual value and 1) was below $\tau = 0.3$. When cutting planes were found, this tolerance τ was multiplied by 1.4^k, where $k = \lfloor 10(MAXVIOL + 0.1)\rfloor - 9$ and $MAXVIOL$ is the maximum cutting plane violation.

The separation routine comprised complete enumeration of all the triangle inequalities. These were bucket sorted by violation. We only add constraints that have violation at least $0.5MAXVIOL$. The algorithm proceeds through the inequalities in order of decreasing violation until an edge-disjoint set of at most 500 constraints has been found, which is then added to the relaxation. (We say several constraints are *edge-disjoint* if they use distinct sets of variables.) Adding an edge-disjoint subset has the beneficial effect of reducing the amount of fill-in in the matrix product AA^T, and thus reducing the linear algebra required to calculate projections when finding the next interior point iterate. Note that if we chose to translate the cutting planes so that they are satisfied at equality then it is easy to find a restart direction if the cuts are orthogonal [41, 17], as they are if they are edge-disjoint.

Our primal heuristics are similar to those suggested in Grötschel *et al.* [21]. We round the current iterate. An ordering is constructed from this rounded solution using a greedy heuristic: at step k it picks the kth element in the ordering, breaking ties arbitrarily. A local optimization routine is then applied to this greedy ordering, where each sector is examined in a different position in the ordering.

We dropped any constraint which had been in the relaxation for at least five stages and which still had a slack of at least 0.4.

Sectors	44	50	56	60	79
Number	29	3	11	2	1
Iterations	53	64	62	67	104
Time (seconds)	9.1	21.1	32.1	52.9	487.4
Stages	14	17	17	18	24
Cuts added	322	539	732	891	1985
Cuts dropped	77	140	225	269	646

Table 1: Results on real-world input-output matrices

We initialized the restart point to be $x_{ij}^{FEAS} = 0.5$. This was updated at each iteration to x if x did not violate any of the cutting planes. If x violated any triangle constraint then we updated x^{FEAS} by taking a step of length α from x^{FEAS} in the direction towards x, where α is 90% of the distance to the closest triangle inequality.

Christof and Reinelt [9] have developed a simplex-based branch-and-cut algorithm for hard instances of the linear ordering problem where the cutting planes come from small-dimensional versions of the problem, as in Christof and Reinelt [8]. The instances we examine in this paper are larger, but they do not generally require branching or extensive separation routines to find violated cutting planes. We are interested in large instances because they have large linear programming relaxations, so the amount of time spent solving the relaxations will be a significant proportion of the total solution time. We expect that the methods described in this paper, in conjunction with the methods described in [9], will make it possible to solve large, hard instances.

3.1.3 Real world problems

Table 1 contains the results of our algorithm on 46 real-world linear ordering problems. All the problems come from input-output tables in economics; except for the 79-sector problem *usa79*, they are all available from `LOLIB` at the URL

`http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html`

For a discussion of the origins of these problems, see Grötschel *et al.* [21] or Mitchell and Borchers [35]; for a discussion of the economic interpretation of the results see Grötschel *et al.* [22]. All the problems in Table 1 except for those with 50 sectors were attacked using the algorithm discussed in [35]. The costs in all of these input-output tables are integral, so we terminated when the gap between our upper and lower bounds was smaller than one.

The rows of the tables convey the following information. The first row gives the number of sectors and the second row the number of instances of that particular size that were solved. The rows labelled *Iterations*, *Time (seconds)*, and *Stages* give the means of, respectively, the total number of primal-dual predictor corrector iterations required to solve the integer programming problem, the total time in seconds required to solve the problems, and how often the LP relaxation was modified so the total number of LP relaxations formed for a

particular problem is one more than the number of stages. The row labelled *Cuts added* and *Cuts dropped* give, respectively, the total number of cutting planes added to the relaxations and the number of these cuts that were subsequently dropped. The numbers are rounded to the number of digits shown.

As can be seen, all these problems can be solved easily with our code. The algorithm only requires around four iterations per stage; as would be expected, the number of iterations required on a stage increases as the algorithm proceeds, so the last stage may well require about ten iterations. Of course, this last stage is the only one that has to be solved exactly. The proportion of time spent actually solving the linear programming relaxations increases as the problem size increases, accounting for over 90% of the time on the largest problem *usa79*. The number of stages is larger than in some simplex based implementations because we add a set of edge-disjoint constraints at each stage, which keeps the Cholesky factor from becoming too dense.

The iteration counts and the number of stages are better than those contained in [35]. The SUN SPARC 20/71 used in the experiments in this paper is about twice as fast as the SUN SPARC 10/30 used for the experiments in [35]. After adjusting for this, the runtimes for the 44 and 56 sector problems in Table 1 are similar to those in the earlier paper, but the runtimes for the larger problems are two to three times better than those in [35]. It was argued in [35] that the runtimes in that paper were comparable to those obtained by the simplex based cutting plane algorithm due to Jünger and Reinelt [21, 43] — they were somewhat worse, but the difference was shrinking as the problem size increased. Thus, the new results give a runtime that is very similar to that in [43] for the largest problem *usa79*.

Our results can also be compared with a simplex based cutting plane algorithm for these problems [6], which is written in C and uses the simplex solver in CPLEX4.0 to solve the relaxations. It adds all the violated constraints to the relaxation and resolves. We obtained a copy of this code and used it to solve the problems in our test set. Most of them only required two or three stages, and the runtimes are better than those obtained with the interior point code — the ratio decreases as the problem size increases, but the runtimes are still perhaps three times better for the problem *usa79*. This is still a good result for the interior point code, since it was all written "in-house" whereas CPLEX4.0 is an excellent commercial code. We are also comparing runtimes of codes written in different languages, so it is hard to draw definitive conclusions. For these problems, CPLEX4.0 uses devex pricing in the dual and it introduces perturbations in the data; these choices aid the solution procedure considerably.

3.1.4 Random problems

We also solved some larger randomly generated problems, and in addition some of these problems were solved using the code described in [6]. We generated these problems by first setting $pz\%$ of the entries g_{ij} to zero and generating a random permutation τ; the remaining entries were then uniformly distributed integers between 0 and 99 if $\tau(i) < \tau(j)$ or between 0 and 39 if $\tau(i) > \tau(j)$. The problems become harder as pz increases. Many of the real world problems contain a number of zeroes in the g_{ij} entries. The generated problems all had linearity between 70.9 and 74.2 — the linearity measures the proportion of the total weight accounted for by the ordering. The extreme cases are, first, that every entry in the

pz	Sectors	Interior point				Simplex
		Time	Iters	Stages	Added	Time
0	50	6.0	26.0	7.0	236.0	3.3
0	75	20.2	30.2	8.2	543.2	13.8
0	100	51.1	33.6	9.2	1003.2	98.4
0	150	206.4	44.8	12.2	2919.0	—
0	200	754.8	46.2	12.4	6406.4	—
10	50	10.1	35.8	9.6	362.2	6.1
10	75	50.8	47.5	13.0	871.5	73.9
10	100	155.6	53.0	13.8	1510.0	280.9
10	150	2071.9	72.4	12.4	6406.4	—
20	50	19.7	50.8	12.6	500.2	10.4
20	75	240.7	90.5	17.3	1247.5	119.6
20	100	1405.4	89.5	18.5	2313.8	—
30	50	70.1	73.6	15.2	732.6	29.5
30	75	771.3	102.3	17.8	1588.0	251.9

Table 2: Results for random linear ordering problems

matrix takes the same value, when the linearity would be 50, and second, when there are no nonzero entries below the diagonal, in which case the linearity is 100. The randomly generated problems had similar linearity to the real world problems.

The results are contained in Table 2. We let pz take the values 0, 10, 20, and 30, and the number of sectors was set to 50, 75, 100, 150, and 200; five problems were generated with each combination. The table contains the mean results for each set of problems. Because of memory limitations, we were unable to solve problems with more than 100 sectors using the simplex code, and we were also unable to solve problems with 150 sectors and $pz \geq 20$ or with 200 sectors and $pz \geq 10$ using the interior point code. In addition, again because of memory limitations, we could not solve problems with 100 sectors and $pz = 30$ with either code, and we were only able to solve one problem with 100 sectors and $pz = 20$ using the simplex code — on the remaining problems, the code ran for roughly 1000 seconds before running out of memory. The triangle inequalities were not sufficient to solve four of the problems, one each with 75 sectors and pz equal to 10, 20 and 30, and one with 100 sectors and $pz = 20$; we have omitted these problems from the tables. It appears that the simplex code spends well over 90% of its time within CPLEX, at least for the harder problems. The columns in Table 2 contain the same information as the rows in Table 1, with the addition that the last column contains the runtimes with the cutting plane code that uses the simplex solver in CPLEX 4.0. Runtimes are quoted in seconds.

As can be seen, the interior point code outperforms the simplex based code for problems with at least 100 sectors where pz is no bigger than 10. Furthermore, it can be seen that the rate of increase in the runtimes as the problem size increases is far smaller for the interior point code than for the simplex code. When pz is as big as 30, the Cholesky factors become

dense and the simplex code outperforms the interior point code. For $pz = 20$, the simplex code outperforms the interior point code for 50 and 75 sector problems, but it appears that the codes would take similar times for 100 sector problems, were it not for memory limitations.

As the proportion of zeroes pz increases, the linear ordering problems should become more dual degenerate, with multiple optimal solutions. For linear programming problems, degeneracy is normally favourable for an interior point method. However, for these problems, the degeneracy results in the addition of many cutting planes that use the same variables so the constraint matrix A eventually contains several dense columns and there is considerable fill in the Cholesky factor of the matrix AA^T. This increases the time for one iteration of the interior point method, and thus the simplex code outperforms the interior point code when $pz = 30$. One possible remedy for this problem is to use a preconditioned conjugate gradient algorithm to calculate the directions in the interior point method; this is a subject for future research.

We have recently investigated combining an interior point cutting plane method with a simplex cutting plane method [36], with results that appear to be superior to using either method on its own. The random problems used in both [36] and this paper are available at the URL

`http://www.math.rpi.edu/~mitchj/generators`

We also examined a formulation of a clustering problem proposed by Grötschel and Wakabayashi [24]. This problem can be written in a manner similar to the linear ordering problem, with triangle inequalities, although the triangle inequalities have a different structure. The computational results were similar to those for the linear ordering problem, in that they were comparable to the results obtained with a simplex method, and the relative performance of the interior point code improved as the problems increased in size. The algorithm appears to perform worse than one described by Palubeckis [40], at least for smaller problems. As the problem sizes increased, the gap between the algorithms decreased. The random instances of both this clustering problem and also the Ising spin glass problem used in this paper are also available from the URL given above.

3.2 The ground states of Ising spin glasses

3.2.1 Definition of the problem

Finding the ground states of Ising spin glasses is an important problem in physics. We examine two-dimensional Ising spin glasses. This problem was originally discussed in the operations research literature by Barahona *et al.* [3], who modelled the problem as a max-cut problem and developed a simplex-based cutting plane algorithm to solve the problem. Recently, some of these authors and other colleagues have returned to this problem, and have improved their computational results considerably [44, 12, 13]. We have previously sketched our experience on a smaller set of these problems in [33]. Facets of the cut polytope are described in [4, 14, 15].

We are given a collection of points, and we know the interaction between the points; we want to determine which points have a positive charge and which points have a negative

charge. Our model places vertices at points of an $L \times L$ grid on a torus. Each vertex has four neighbours: to the left, to the right, above and below. There are weights on the edges joining a vertex to its neighbours which correspond to the bonds or interactions between the vertices. We generate edge weights using two different distributions, and we report results for problems with grids of size up to 100×100. We assume there is no external field — it was shown by Barahona *et al.* [3] that an external field can be modeled by including an extra vertex; the resulting problem appears to be easier to solve than a problem with no external field, at least when the edge weights have a Gaussian distribution.

The problem can be modelled on an undirected graph $G = (V, E)$ as

$$\begin{array}{ll} \min & \sum_{i=1}^{p} \sum_{j>i,(i,j)\in G} c_{ij} x_{ij} \\ \text{subject to} & x \text{ is the incidence vector of a cut} \end{array}$$

where p is the number of vertices, there is a variable x_{ij} for each edge, and the cost c_{ij} of each edge is derived from the interaction between the vertices. Each vertex has four neighbours, so a $k \times k$ grid will have k^2 vertices and $2k^2$ edges.

Cutting planes can be derived by using the observation that every cycle and every cut intersect in an even number of edges. Every subset F of odd cardinality of every chordless cycle C gives the facet-defining inequality

$$x(F) - x(C \setminus F) \leq |F| - 1 \tag{3}$$

where $x(S)$ denotes $\sum_{(i,j)\in S} x_{ij}$ for any subset $S \subseteq E$. The cycles of length four (the *squares*) in the graph are chordless cycles, and there are many other chordless cycles. There are other families of facet defining inequalities; we only searched for facets of the form (3).

3.2.2 Details of the algorithm

The initial relaxation is $\min\{c^T x : 0 \leq x \leq e\}$. All the cutting planes are of the form (3).

The separation routine consists of three parts. We first search for cutting planes corresponding to the squares in the graph using complete enumeration. The violated constraints are bucket sorted by violation and the most violated constraints are added. We are prepared to add constraints that correspond to squares that share edges. We add at most 500 square constraints; further, if $k < 500$ constraints are violated by at least 0.1 then we add at most $\max\{L - k, 0\}$ constraints with violation less than 0.1. If this does not return at least L constraints which are violated by cutting planes or if the largest violation of a square constraint is no more than 0.2, we then use a heuristic procedure similar to that described in Barahona *et al.* [3] to find longer chordless cycles with violated constraints. The heuristic is restricted to add at most 100 violated constraints; further, we restrict it so that it adds at most L^2 nonzeroes to the constraint matrix A (excluding the columns corresponding to the slack variables).

If the heuristic was called and it did not find at least 20 cutting planes, we use an implementation of the exact algorithm due to Barahona and Mahjoub, which has complexity $O(p^3)$ (p is the number of nodes), and is guaranteed to find a violated cycle inequality, if one exists. We place an upper limit of L on the number of these constraints that we will add, and we only add a constraint if it has a violation that is at least half of the violation of the

most violated constraint found by this exact procedure on this stage. The routine looks for cycles starting from each vertex in the graph; to limit the time spent on this, we start from a maximum of 50 further vertices after finding a constraint with violation at least 0.05. We insist that the set of added constraints arising from longer cycles be edge-disjoint at each stage. The non-square constraints usually contain many more than four edges. We found it advantageous to scale an added constraint with $\mid C \mid$ edges, normalizing so that the L_1-norm of the constraint was $4/\sqrt{\mid C \mid}$.

We solved every tenth LP relaxation to a relative duality gap of 10^{-8}. Several of the problems took a large number of stages, and solving the relaxations accurately is a way to limit the number of stages, at a cost of an increased number of iterations. This approach reduces the variability of the runtimes.

Adding the longer cycles makes it hard to update the restart point: the restart point found in one stage may well be infeasible at a future stage because we do not check every possible constraint. Thus, we restarted in Step 10 of the algorithm by moving towards the point $0.5e$. We move so that the restart point is 5% of the way from the boundary of the feasible region of the new relaxation towards $0.5e$. The dual iterate was updated to an earlier dual iterate, namely the last point where the relative duality gap was at least 10%.

Our primal heuristic used the primal point x to generate the incidence vectors of several cuts. Edges with x_{ij} smaller than 0.01 or greater than 0.99 forced vertices onto the same side or opposite sides of the cut and then unassigned vertices were assigned in a greedy manner. In order to get several cuts, the order in which initially unassigned vertices were examined was randomized. The number of cuts generated at stage k is $(1 + (k/6))$. Once an incidence vector has been generated, it is modified using a local improvement process. The local improvement process looks for paths of vertices — all vertices on a path are moved to the other side of the cut if this results in improvement. We start off looking for paths consisting of just a single vertex, and eventually we look for paths containing up to ten vertices. We use each vertex in turn as the starting vertex. We use a breadth first search to explore all paths starting from the vertex; if a path results in an increase in the size of the cut of at least 2.5 then we stop searching along this branch and backtrack. If we are unable to find an improving path starting from any vertex, we look for paths that do not hurt the solution. If we are then still unable to find improving paths, we terminate the local improvement process. This idea of looking for paths was proposed by Berry and Goldberg [5].

Each edge only appears in eight of the possible cycle constraints of length 4, so the columns of the constraint matrix did not become dense. Therefore, we only dropped a constraint if none of the corresponding edge variables remained unfixed.

3.2.3 Computational results

We generated random problems using two different probability distributions. First, we generated random edge weights with a Gaussian distribution with mean zero and standard deviation 1. Second, we generated edge weights of ± 1, with 1 or -1 equally likely. Our results were far better for the second class. The principal properties of real spin glasses (for example, amorphous alloys) are represented well by the ± 1 spin glass model on a rectangular lattice. The results in [12] are far better than our results with the first distribution, so we

L	N_L	Time	Iters	Stages	Added	Energy
10	1946	0.47	9	2.0	69.0	-1.3895
20	1946	4.79	21	4.0	327.7	-1.3985
30	1546	24.38	39	6.7	809.4	-1.4003
40	1200	93.08	64	9.8	1550.6	-1.4001
50	720	290.75	99	13.8	2502.9	-1.4005
60	440	772.37	154	19.5	3670.0	-1.4019
70	384	2245.92	216	25.8	5294.2	-1.4012
80	310	5787.28	310	34.5	7219.3	-1.4012
90	280	11320.24	400	42.8	9501.6	-1.4017
100	229	11873.59	391	44.1	10975.0	-1.4023

Table 3: Results for Ising spin glass problems

do not report these results in detail.

The results from problems where the edge weights were ± 1 are contained in Table 3. We give the number N_L of problems of each size solved, the number of primal-dual iterations required, the total CPU time to solve the problems, the number of stages, the number of cuts added, and the average ground state energy. Typically, over 40% of the total CPU time for the larger instances was spent on the primal heuristics. In addition, we were unable to solve 2, 18, 10, and 7 instances with $L = 70, 80, 90, 100$ respectively using just cutting planes of type (3), so these instances are omitted from the table.

When solving these problems, we exploited the fact that every cut will have even value, so we can terminate the algorithm with the optimal solution when the gap between the upper and lower bounds falls below two. With this termination criterion, we found that we were rarely able to fix any edges in Step 9 of the algorithm. (This contrasts markedly with our experience with problems with a Gaussian distribution of edge weights, where fixing variables made it possible to solve problems which were otherwise beyond the reach of our implementation due to memory requirements.)

These results compare very favourably with those in De Simone *et al.* [13], who used the simplex solver in CPLEX3.0 in a branch and cut algorithm for problems with the same distribution, on a Sun SPARC 10, which is approximately half as fast as our machine. They report results for problems of size up to 70×70. Problems of size 50×50 took them roughly an hour, problems of size 60×60 took roughly two to three hours, and problems of size 70×70 required on the order of fifteen hours.

One reason for the better results for the problems with ± 1 edge weights than with Gaussian edge weights is that the problems do not have to be solved as accurately: we can terminate if the gap becomes less than two. An interior point method is good at getting close to an optimal solution, but it may take a while in the cutting plane setting to get a relative gap of, say, 10^{-6}. Our primal heuristic worked well for the ± 1 problems, almost always finding the optimal solution at least one or two stages before it was possible to prove optimality; this was not the case for the Gaussian problems, with the optimal solution often

not discovered until the final stage.

The number of stages and iterations for problems with either distribution are sensitive to slight changes in the parameters of the algorithm. We found a slight change may well halve the number of iterations required to solve one problem but double the number of iterations required to solve another. The table contains the results with a set of parameters that appeared to produce reasonable results, producing some of the better runs for most problems and respectable results for most of the remaining problems.

4 Conclusions

We have presented cutting plane algorithms for several integer programming problems. These algorithms use a predictor-corrector interior point method to solve the LP relaxations. For some MAXCUT problems and linear ordering problems, we have obtained runtimes that are comparable with or better than those obtained using a cutting plane method that employs the simplex solver in CPLEX to solve the relaxations.

It appears from the results detailed in the current paper and from other experiments, that the most suitable problems are those where:

- The linear programming relaxations are large, with the number of variables and/or constraints numbering in the thousands. This is because of the well-documented observation that the performance of interior point methods relative to simplex methods for linear programs improves as the problem size increases.

- The objective function coefficients are integer. It then suffices to reduce the duality gap to be less than one in order to prove optimality. This is useful for an interior point cutting plane method because such an approach can typically get close to optimality quickly but then may take a long time to reduce the duality gap to, for example, 10^{-6}. Problems with integral coefficients are more likely to suffer from primal or dual degeneracy, which is more harmful to the performance of a simplex cutting plane algorithm than an interior point cutting plane algorithm. When the objective function coefficients are fractional, an appropriate method may be to use an interior point cutting plane algorithm initially and switch over to a simplex cutting plane algorithm as optimality is approached.

- It should be possible to find a strictly feasible point in the convex hull of feasible integral points efficiently, because such a point can then be used to restart the algorithm after cutting planes have been added. If it is not possible to restart in this manner, the method proposed by Gondzio [19] can be used.

We have recently experimented with combining interior point cutting plane algorithms with dual simplex cutting plane algorithms, using the interior point solver for the early stages and the simplex solver for the later stages. The performance of this algorithm has been outstanding for linear ordering problems [36]. It may well be that such a hybrid cutting plane method is an appropriate choice for a wide variety of integer programming problems.

Acknowledgements: I am grateful to Brian Borchers for providing the cutting plane code for linear ordering problems that uses CPLEX4.0. I am also grateful to Ulrich Dorndorf and Gerd Reinelt for sending me the clustering problems and some of the linear ordering problems, respectively. Further, I thank the two referees and the Associate Editor for their thorough reading of the paper and useful comments.

References

[1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. The traveling salesman problem. Technical report, DIMACS, Rutgers University, New Brunswick, NJ, 1994.

[2] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[3] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0-1 programming. *Mathematical Programming*, 44(2):127–137, 1989.

[4] F. Barahona and A. R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[5] J. Berry and M. Goldberg. Path optimization for graph partitioning problems. *Discrete Applied Mathematics*, 90:27–50, 1999.

[6] Brian Borchers. Private communication: Project of R. M. Cooke, 1996.

[7] A. Caprara and M. Fischetti. Branch and cut algorithms. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 4. John Wiley, 1997.

[8] T. Christof and G. Reinelt. Low-dimensional linear ordering polytopes. Technical report, IWR Heidelberg, Germany, 1997.

[9] T. Christof and G. Reinelt. Algorithmic aspects of using small instance relaxations in parallel branch-and-cut. Technical report, IWR Heidelberg, Germany, 1998.

[10] CPLEX Optimization Inc. CPLEX Linear Optimizer and Mixed Integer Optimizer. Suite 279, 930 Tahoe Blvd. Bldg 802, Incline Village, NV 89541.

[11] J. Czyzyk, S. Mehrotra, M. Wagner, and S. J. Wright. PCx user guide (version 1.1). Technical report, Optimization Technology Center, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439, November 1997.

[12] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *Journal of Statistical Physics*, 80:487–496, 1995.

[13] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84:1363–1371, 1996.

[14] M. Deza and M. Laurent. Facets for the cut cone I. *Mathematical Programming*, 56:121–160, 1992.

[15] M. Deza and M. Laurent. Facets for the cut cone II: Clique-web inequalities. *Mathematical Programming*, 56:161–188, 1992.

[16] S. C. Eisenstat, M. C. Gurshy, M. H. Schultz, and A. H. Sherman. The Yale Sparse Matrix Package, I. The symmetric codes. *International Journal for Numerical Methods in Engineering*, 18:1145–1151, 1982.

[17] J.-L. Goffin and J.-P. Vial. Multiple cuts in the analytic center cutting plane method. Technical Report Logilab Technical Report 98.10, Logilab, Management Studies, University of Geneva, Geneva, Switzerland, June 1998.

[18] J. Gondzio. HOPDM (ver 2.12) — A fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225, 1995.

[19] J. Gondzio. Warm start of the primal-dual method applied in the cutting plane scheme. *Mathematical Programming*, 83:125–143, 1998.

[20] M. Grötschel and O. Holland. Solution of large-scale travelling salesman problems. *Mathematical Programming*, 51(2):141–202, 1991.

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

[22] M. Grötschel, M. Jünger, and G. Reinelt. Optimal triangulation of large real-world input-output matrices. *Statistiche Hefte*, 25:261–295, 1984.

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

[24] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989.

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

[26] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 20, pages 111–152. AMS, 1995.

[27] 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.

[28] E. K. Lee and J. E. Mitchell. Computational experience of an interior point SQP algorithm in a parallel branch-and-bound framework. In H. Frenk *et al.*, editor, *High Performance Optimization*, chapter 13, pages 329–347. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[29] J. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 33:1–19, 1989.

[30] 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.

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

[32] J. E. Mitchell. Fixing variables and generating classical cutting planes when using an interior point branch and cut method to solve integer programming problems. *European Journal of Operational Research*, 97:139–148, 1997.

[33] J. E. Mitchell. An interior point cutting plane algorithm for Ising spin glass problems. In P. Kischka and H.-W. Lorenz, editors, *Operations Research Proceedings, SOR 1997, Jena, Germany*, pages 114–119. Springer-Verlag, 1998.

[34] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, April 1999. Accepted for publication in the *Handbook of Applied Optimization*, 2000.

[35] J. E. Mitchell and B. Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, 62:253–276, 1996.

[36] J. E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In H. L. Frenk *et al.*, editor, *High Performance Optimization*, chapter 14, pages 349–366. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[37] J. E. Mitchell, P. P. Pardalos, and M. G. C. Resende. Interior point methods for combinatorial optimization. In D.-Z. Du and P. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 189–297. Kluwer Academic Publishers, 1998.

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

[39] M. 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.

[40] G. Palubeckis. A branch-and-bound approach using polyhedral results for a clustering problem. *INFORMS Journal on Computing*, 9:30–42, 1997.

[41] S. Ramaswamy and J. E. Mitchell. On updating the analytic center after the addition of multiple cuts. Technical Report 37–94–423, DSES, Rensselaer Polytechnic Institute, Troy, NY 12180, October 1994. Substantially revised: August, 1998.

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

[43] G. Reinelt. Private communication, 1995.

[44] H. Rieger, L. Santen, U. Blasum, M. Diehl, M. Jünger, and G. Rinaldi. The critical exponents of the two-dimensional Ising spin glass revisited: Exact ground state calculations and Monte Carlo simulations. *Journal of Physics A: Mathematical and General*, 29:3939–3950, 1996.