# 1 SOLVING LINEAR ORDERING PROBLEMS WITH A COMBINED INTERIOR POINT/SIMPLEX CUTTING PLANE ALGORITHM[*]

John E. Mitchell

Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, NY 12180 USA.[†]
http://www.math.rpi.edu/~mitchj
mitchj@rpi.edu

Brian Borchers

Mathematics Department
New Mexico Tech
Socorro, NM 87801 USA.
http://www.nmt.edu/~borchers
borchers@nmt.edu

[*]Supported in part by ONR grant N00014–94–1–0391.
[†]Supported in part by a grant from the Dutch NWO and Delft University of Technology for the 1997–98 academic year, while visiting TWI/SSOR at Delft University of Technology.

**Abstract:** We describe a cutting plane algorithm for solving linear ordering problems. The algorithm uses a primal-dual interior point method to solve the first few relaxations and then switches to a simplex method to solve the last few relaxations. The simplex method uses CPLEX 4.0. We compare the algorithm with one that uses only an interior point method and with one that uses only a simplex method. We solve integer programming problems with as many as 31125 binary variables. Computational results show that the combined approach can dramatically outperform the other two methods.

## 1.1 INTRODUCTION

The linear ordering problem has applications in economics, archaeology, scheduling, the social sciences, and aggregation of individual preferences. A cutting plane method provides a way to obtain a provably optimal solution to a linear ordering problem. Such a method requires the solution of a sequence of linear programming problems. It is now possible to solve linear ordering problems of a size where these linear programming problems can be solved more efficiently using an interior point method than by using simplex. In this paper we describe an interior point cutting plane method for the linear ordering problem, we examine combining the interior point method with a simplex cutting plane method, and we present computational results showing that the combined method can dramatically outperform either a pure interior point cutting plane method or a pure simplex cutting plane method.

In §1.2, we define the linear ordering problem and discuss an integer programming model. In §1.3, we describe the polyhedral structure of the linear ordering polytope. In §1.4 and §1.5, we present our cutting plane algorithms for the linear ordering problem. The combination of interior point and simplex cutting plane algorithms is the subject of §1.6. Computational results are presented in §1.7 and conclusions are given in §1.8.

The first authors to consider a cutting plane algorithm for the linear ordering problem were Grötschel et al., 1984a, Grötschel et al., 1984b, Jünger, 1985, and Reinelt, 1985. We have previously discussed interior point cutting plane algorithms for this problem in Mitchell and Borchers, 1992, Mitchell and Borchers, 1996, and Mitchell, 1997. Computational investigations of interior point cutting plane algorithms for other integer programming problems include Mitchell and Todd, 1992, Mitchell, 1997, and Mitchell, 1998. Interior point column generation algorithms implemented in other contexts include Bahn et al., 1995, Goffin et al., 1997, Gondzio, 1998, and Gondzio and Sarkissian, 1996. Many of these references also contain discussions of the theoretical performance of interior point column generation methods.

Christof and Reinelt, 1998, 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, 1997. 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 Christof and Reinelt, 1998, will make it possible to solve large, hard instances.

## 1.2 THE LINEAR ORDERING PROBLEM

### *1.2.1 Applications*

Applications of the linear ordering problem include triangulation of input-output matrices in economics (Grötschel et al., 1984b), archeological seriation, minimizing total weighted completion time in one-machine scheduling, the social sciences (Fishburn, 1992), and aggregation of individual preferences. For more discussion of the linear ordering problem, as well as description of a cutting plane algorithm for solving the problem, see Grötschel et al., 1984a.

As an example of the aggregation of individual preferences, consider a tournament between a number of sports teams, where each team plays every other team. We wish to determine which team is the best, which is second best, and so on. If Team A beats Team B then Team A should finish ahead of Team B in the final ordering. However, it may be that Team B beat Team C, who in turn beat Team A. Therefore, it is not generally a simple matter to determine the final ordering. We could just count the number of victories of each team, but this may not truly represent the relative strength of some teams, and it may well lead to ties in the ordering. Therefore, we usually take the margin of victory into account when determining the final ordering.

An input-output matrix in economics measures the movement of goods from one sector of the economy to another. In advanced economies there will generally be a rotation of goods and capital through the economy, whereas in less advanced economies there will be a more pronounced ordering of the sectors, with goods generally flowing from Sector A to Sector B to Sector C, etc. The objective is to find the ordering of the sectors of the economy that most closely matches the data contained in the input-output matrix. The final solution can be quantified using its *linearity*:

> The linearity of an input-output matrix is the proportion of the total weight in the matrix that agrees with the optimal ordering.

For an advanced economy, the linearity can be as low as 70%; for less advanced economies the linearity can be as high as 90%.

In archeological seriation, we have samples from different sites of different artifacts belonging to various time periods. If object A is closer to the surface than object B then the time period for object A was probably more recent than that of object B. The objective is to aggregate the data of this form from different sites and determine the ordering of the time periods.

### *1.2.2 Modeling the problem*

In a general linear ordering problem, we have $p$ objects to place in order. If we place $i$ before $j$, we pay a cost of $g(i,j)$. Conversely, if we place $i$ after $j$, we pay a cost $g(j,i)$. The objective is to choose the ordering that minimizes the total cost. This problem is $NP$-hard (Karp, 1972). Throughout this paper, we will use $p$ to refer to the number of objects.

A linear ordering problem with $p$ objects can be considered as a problem on the complete directed graph with $p$ vertices. For each pair of vertices $i$ and $j$, we want to pick exactly one of the two arcs $(i,j)$ and $(j,i)$. Further, there should be no directed cycles in the resulting directed subgraph. Such an acyclic digraph is called a *tournament*.

The linear ordering problem can be modelled as an integer programming problem in the following manner. We define indicator variables $x_{ij}$ for each ordered pair of objects $i$ and $j$, to indicate whether $i$ is before $j$:

$$x(i,j) = \begin{cases} 1 & \text{if } i \text{ before } j \\ 0 & \text{otherwise.} \end{cases}$$

We can then model the linear ordering problem as:

$$\begin{array}{ll} \min & \sum_{i=1}^{p} \sum_{\substack{j=1 \\ j \neq i}}^{p} g_{ij} x_{ij} \\ \text{subject to} & x \text{ is the incidence vector of a tournament.} \end{array}$$

Notice that in any feasible solution, we must have $x(i,j) + x(j,i) = 1$ for each pair $1 \leq i < j \leq p$. We can use this observation to eliminate the variables $x(j,i)$, $j > i$. With this modification, we modify the objective function:

$x(i,j)$, $i < j$, has cost coefficient $c(i,j) := g(i,j) - g(j,i)$.

The linear ordering problem can be restated

$$\begin{array}{ll} \min & \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} c_{ij} x_{ij} \\ \text{subject to} & x \text{ is the incidence vector of a tournament.} \quad (LO) \end{array}$$

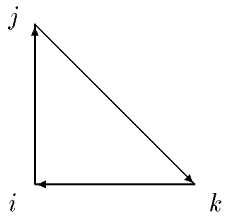
## 1.3 THE POLYHEDRAL STRUCTURE OF THE LINEAR ORDERING POLYTOPE

Grötschel et al., 1984a, Jünger, 1985, and Reinelt, 1985, have investigated the polyhedral combinatorics of the linear ordering problem, and we recap their results. They have shown that the convex hull of the set of feasible solutions to $(LO)$ is full dimensional and that the simple bounds $0 \leq x_{ij} \leq 1$ define facets of this polyhedron.

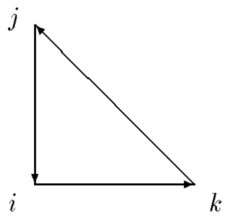
In order to get a tournament, it is necessary to prevent solutions of the form:

$i$ before $j$ before $k$ before $i$,
or, equivalently: $x(i,j) = x(j,k) = x(k,i) = 1$.

This can be prevented by the inequality $x(i,j) + x(j,k) + x(k,i) \leq 2$, which must be satisfied by any linear ordering. For the formulation in $(LO)$, we get two forms of this inequality when $1 \leq i < j < k \leq p$:



Prevented by $x(i,j)+x(j,k)-x(i,k) \leq 1$.



Prevented by $-x(i,j)-x(j,k)+x(i,k) \leq 0$.

Thus, we get the two sets of *triangle inequalities*:

$$\begin{aligned} x(i,j)+x(j,k)-x(i,k) &\leq 1 \quad (1.1)\\ -x(i,j)-x(j,k)+x(i,k) &\leq 0. \quad (1.2)\end{aligned}$$

Every incidence vector of a linear ordering satisfies these inequalities for all $1 \leq i < j < k \leq p$. We call two such inequalities *arc-disjoint* if they involve two non-intersecting sets of objects, $\{i_1, j_1, k_1\}$ and $\{i_2, j_2, k_2\}$.

Grötschel et al., 1984a, showed that if $x$ is integral and satisfies all the triangle inequalities then it is the incidence vector of a linear ordering. Therefore, the linear ordering problem can be written

$$\begin{array}{llcll} \min & \sum_{i=1}^{p-1}\sum_{j=i+1}^{p} c_{ij}x_{ij} & & & \\ \text{subject to} & x(i,j)+x(j,k)-x(i,k) & \leq & 1, & 1 \leq i < j < k \leq p \quad (IPLO)\\ & -x(i,j)-x(j,k)+x(i,k) & \leq & 0, & 1 \leq i < j < k \leq p \\ & x & = & 0 \text{ or } 1, & 1 \leq i < j \leq p. \end{array}$$

The $2\begin{pmatrix} p \\ 3 \end{pmatrix}$ triangle inequalities are facets of the convex hull of incidence vectors of linear orderings. However, they do not give a complete description of the linear ordering polytope, and other families of inequalities are known. In fact, since the linear ordering problem is $NP$-hard, there must be exponentially many other facet defining inequalities, unless $P = NP$. Such inequalities have been investigated by Leung and Lee, 1994, and Christof and Reinelt, 1997, among others.

## 1.4 A BASIC CUTTING PLANE APPROACH

In a cutting plane approach, a sequence of linear programming relaxations of the linear ordering problem are solved, and these relaxations are improved until they give a sufficiently good approximation to the convex hull of incidence vectors of linear orderings in the neighbourhood of the optimal ordering. Because there is a large number of triangle inequalities, these are only included as necessary. We were able to solve most of our test problems using only triangle inequalities; for other linear ordering problems it may well be necessary to use other inequalities or to use branch and bound.

Thus, our relaxation always takes the form:

$$\begin{array}{lll} \min & \sum_{i=1}^{p-1}\sum_{j=i+1}^{p} c_{ij}x_{ij} & \\ \text{subject to} & x \text{ satisfies some of the triangle inequalities} & (LPLO) \\ & 0 \leq x(i,j) \leq 1, \quad 1 \leq i < j \leq p. & \end{array}$$

This relaxation has approximately $p^2/2$ variables; for $p = 100$ we get 4950 variables and for $p = 250$ we have 31125 variables. The initial relaxation includes none of the triangle inequalities. Notice that the optimal value of $(LPLO)$ gives a lower bound on the optimal value of $(IPLO)$.

If the solution to $(LPLO)$ is integral and if it satisfies all of the triangle inequalities then it is optimal for the linear ordering problem. If the optimal solution violates some triangle inequalities, then a subset of the violated inequalities is added to the relaxation, the modified relaxation is solved and the process repeated. If the solution is fractional, but it does not violate any triangle inequalities, then the cutting plane approach is halted; in principle, the method could be extended to this case by searching for other cutting planes, or by using branch and bound. Our interior point solver does not do this, but the CPLEX simplex solver does use branch and bound if the cutting plane approach results in a fractional solution. Only 2 of our 30 test problems had such a fractional solution.

Our complete algorithm takes the following form (the algorithm is discussed in more detail in §1.5):

1. **Initialize**: We assume the data is integral.

2. **Solve current relaxation**, using either a primal-dual interior point method, or the simplex method. This gives a lower bound on the optimal value of the linear ordering problem.

3. **Separation:** Check all triangle inequalities. Bucket sort resulting violated inequalities by violation and add a subset of arc-disjoint constraints to the relaxation. Drop any constraints that no longer appear important.

4. **Primal heuristic:** Look for the incidence vector of a linear ordering close to the solution to the current relaxation. Store the resulting solution if it is better than the best ordering found previously.

5. **Check for termination**: If the difference between the lower bound and the value of the best ordering found so far is less than one, STOP with optimality. If no violated triangle inequalities are found and if the difference is greater than one, use branch and bound to complete the solution.

6. **Loop:** return to step 2

## 1.5 REFINEMENTS WITH THE INTERIOR POINT CUTTING PLANE METHOD

The algorithm given above works well when solving the relaxations using the simplex method: the initial relaxation is solved using the primal simplex algorithm and subsequent relaxations using the dual simplex algorithm. In order to get the algorithm to perform well when an interior point method is used to solve the relaxations, several refinements are required, and we discuss those in this subsection. We also discuss our primal heuristic and our bucket sort.

The most important modification when using an interior point algorithm is to only solve the relaxations approximately. Only the final relaxation needs to be solved accurately, and, since the data are integral, it even suffices to solve the final relaxation to give a gap of less than one between the dual value and the value of the best known linear ordering. This saves time on the current relaxation. It also means that the separation heuristics try to find violated constraints at a more central point.

The accuracy to which we solve the relaxations is controlled by a dynamically altered tolerance on the duality gap: if many constraints are added, the tolerance is increased because we probably don't need to solve the relaxations as accurately, and if only a few violated constraints are found then the tolerance is decreased. The change in the tolerance also depends on the size of the largest violation. We also insist that the primal value should be less than the value of the best known linear ordering before looking for separating hyperplanes, and we require that the average of the primal and dual values should be at least one less than the value of the best known linear ordering. These two criteria are designed to reduce the likelihood that we search for cutting planes when instead solving the current relaxation to optimality would solve the linear ordering problem.

All triangle inequalities are checked for feasibility. The prospective constraints are then bucket sorted by violation. We then go through the buckets in order, selecting arc-disjoint inequalities, until we reach an upper limit on the number of constraints. There are two advantages to choosing arc-disjoint inequalities: the inequalities are then orthogonal to one another, and the sparsity of the Cholesky factor is not affected as much as if the constraints shared arcs (see, for example, Mitchell, 1997). The criteria used to decide which buckets to examine and how many cuts to add differ between the simplex implementation and the interior point implementation, because the simplex method returns an extreme point and the interior point method returns an interior point, making the nature of the violations differ between the two algorithms (we return to this point in §1.8).

Once cutting planes are added, the current solution is dual feasible but primal infeasible. When using the simplex method, we would resolve using the dual simplex algorithm. One option for restarting when using an interior point method is to use an infeasible interior point algorithm, but we found computationally that such a method was not very effective: it would often concentrate on regaining feasibility, by which time it had moved far from the original iterate. Therefore, we restart by moving towards the vector $0.5e$, where $e$ denotes the vector of ones. This point is always an interior point in $(LPLO)$. Our restart point is an interior point which is a convex combination of this point and the final iterate. We restart the dual problem with an earlier dual iterate, so that the primal-dual pair are more centered. Gondzio, 1998, and Gondzio and Sarkissian, 1996, have investigated other methods for restarting interior point methods when constraints are added, which can be used in the general case when there is not a good restart point available; these restart methods perform no better than our methods for our problem because we can exploit a known good restart point and we restart before getting too close to the optimal face of the current relaxation.

Our primal heuristic is a modification of that in Grötschel et al., 1984a. We try to round the fractional interior point to the incidence vector of a linear ordering, and then we use a local search technique to improve the solution.

## 1.6 COMBINING THE TWO SOLVERS

We investigated three different cutting plane algorithms:

1. Use the **interior point method exclusively** to solve the relaxations.

2. Use the **simplex method exclusively** to solve the relaxations.

3. **Combine** the two methods: use the interior point method to solve the first few relaxations and use the simplex method to solve the remaining relaxations.

The rationale for a crossover method is that we observed experimentally that the number of iterations that an interior point method requires for each relaxation remains approximately constant, while the number required by a simplex cutting plane algorithm drops dramatically. Often, the simplex method may require only a handful of iterations on the last few stages. It appeared that the interior point method was faster in the earlier stages and the simplex method was faster in the later stages.

We experimented with several different rules for crossing over from one algorithm to the other, and different rules worked better on different classes of linear ordering problems. Generally, the problems require between 10 and 20 stages. The rules we tested included:

- Crossover after two stages.

- Crossover after three stages.

- Crossover when less than $3p$ constraints are added.

- Crossover when less than $p$ constraints are added.

As we move down the list, the number of stages solved using the interior point algorithm increases. As will be seen in §1.7, each one of these rules was superior for at least one class of linear ordering problems; it is an interesting open problem to determine a rule that works well for all linear ordering problems.

## 1.7 COMPUTATIONAL RESULTS

We solved randomly generated problems. There are some real-world linear ordering problems available over the web from LOLIB:

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

We describe computational experience with an interior point method for these problems in Mitchell, 1997. The largest of these problems has 79 objects, which is smaller than the size of problems we wish to investigate. Therefore, we randomly generated linear ordering problems with between 100 and 250 objects. Our generator and all the instances discussed in this paper are available at the web site:

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

Each instance was generated as follows:

- For $i < j$, generate $g(i, j)$ uniformly between 0 and 99.
- For $j < i$, generate $g(i, j)$ uniformly between 0 and 39.
- Randomly permute so it is not easy to guess a very good solution.
- Zero out a percentage of the entries.

The linearity of the resulting problems is around 72, which is similar to the real world problems in `LOLIB`. The problems in LOLIB also have various entries equal to zero, and their entries have a larger range than our randomly generated problems.

We generated six different classes of problems, varying by the number of objects and by the percentage of entries zeroed out. Each class contained five problems. For each problem within a particular class, we used the same crossover criterion.

All runs were performed on a Sun SPARC 20/71. All runtimes will be quoted in **seconds**. The interior point code was written in Fortran and the Fortran command `ETIME` was used for timings. We do not use a publically available code such as HOPDM (Gondzio, 1995) or PCx (Czyzyk et al., 1997), because none of these codes make 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. The simplex code was written in C. It uses CPLEX 4.0 to solve the relaxations. The UNIX command `time` was used for timings. For the crossover runs, the interior point code wrote the problem out to files and the simplex code read from the files. The times to write out and to read in the problem are *included* in the runtimes we give. CPLEX has an option of using the point provided by the interior point method as a warm start. We found that this was only a very marginal help because several more stages are still needed when we crossover, and occasionally it led to failure to terminate. Therefore, we report results that do not use this feature.

The runtimes of the three algorithms on the thirty test problems are contained in Table 1.1. Even including all the triangle inequalities in the relaxation $(LPLO)$ does not give an integral solution for the two problems `r100b2` and `r200d1`, so the means for the algorithms omit these problems. The simplex solver has a branch and bound component, so the simplex and crossover codes can solve these two problems, but the interior point code cannot. The interior point code is also unable to solve the problems `r200a1` and `r200e1` because of memory limitations: for each of these problems, the Cholesky factorization of $AA^T$ contained more than our limit of $10^6$ nonzeroes.

In Table 1.2, we give the percentage of the time used by the interior cutting plane code within the combination cutting plane code. It appears to be best to try to split the runtime somewhat evenly between the interior point code and the simplex code.

To give a flavour of the performance of the cutting plane algorithms, Table 1.3 contains more details of three runs for problem `r150a1`: one using just the interior point code, one using just the simplex code, and one using a combination code. The reduction of the number of iterations required by simplex per stage as the algorithm proceeds can be seen in the table. The reduction in the number of simplex iterations when using the crossover code as opposed to the pure simplex code is also interesting.

The relative times required by the interior point and simplex cutting plane algorithms are portrayed in Figure 1.1. The graph uses a linear scale. The four problems that the interior point code was unable to solve are omitted from the graph. Notice that as the problems become more difficult to solve, the times required by the two algorithms become comparable, with the ratio of the runtimes getting close to 1.

The relative times required by the combined and simplex cutting plane algorithms are portrayed in Figure 1.2. The graph uses a linear scale. The two problems for which a branch and bound solver was used are omitted from the graph. It is clear from the graph that the combined code is as much as ten times faster than the simplex code on the harder problems, that is, problems that take the simplex algorithm at least about half an hour.

For the six different classes of problems, we used the following criteria to determine when to switch from the interior point solver to the simplex solver:

| | |
|---|---|
| `r150.0` and `r200.0`: | switch after two stages. |
| `r150.1` and `r250.0`: | switch after three stages. |
| `r200.1`: | switch after add $< 600$ constraints in a stage. (On average, after 7 stages.) |
| `r100.2`: | switch after add $< 100$ constraints in a stage. (On average, after 7 stages.) |

**Table 1.1** Times for the 3 algorithms

| Objects | % zeroes | Name | Interior | Simplex | Crossover |
|---|---|---|---|---|---|
| 150 | 0 | r150a0 | 185 | 89 | 70 |
| | | r150b0 | 201 | 55 | 66 |
| | | r150c0 | 219 | 96 | 72 |
| | | r150d0 | 224 | 75 | 66 |
| | | r150e0 | 203 | 62 | 65 |
| | | Mean | 206 | 75 | 68 |
| 200 | 0 | r200a0 | 565 | 318 | 200 |
| | | r200b0 | 1196 | 408 | 245 |
| | | r200c0 | 610 | 276 | 193 |
| | | r200d0 | 713 | 468 | 198 |
| | | r200e0 | 690 | 455 | 209 |
| | | Mean | 755 | 385 | 209 |
| 250 | 0 | r250a0 | 2382 | 3593 | 683 |
| | | r250b0 | 4685 | 3574 | 548 |
| | | r250c0 | 3924 | 4013 | 574 |
| | | r250d0 | 7486 | 3860 | 524 |
| | | r250e0 | 3983 | 3947 | 631 |
| | | Mean | 4492 | 3797 | 592 |
| 100 | 20 | r100a2 | 1043 | 487 | 185 |
| | | r100b2 | frac | 3886 | 426 |
| | | r100c2 | 2216 | 794 | 214 |
| | | r100d2 | 1040 | 669 | 132 |
| | | r100e2 | 1322 | 645 | 192 |
| | | Mean | 1405 | 649 | 181 |
| 150 | 10 | r150a1 | 1249 | 1496 | 218 |
| | | r150b1 | 1717 | 1511 | 165 |
| | | r150c1 | 2723 | 963 | 221 |
| | | r150d1 | 4229 | 1441 | 237 |
| | | r150e1 | 1316 | 1058 | 200 |
| | | Mean | 2247 | 1294 | 208 |
| 200 | 10 | r200a1 | dnf | 10791 | 1101 |
| | | r200b1 | 9167 | 9317 | 886 |
| | | r200c1 | 8856 | 9142 | 692 |
| | | r200d1 | frac | 33038 | 1667 |
| | | r200e1 | dnf | 10687 | 836 |
| | | Mean | – | 9984 | 879 |

It should be noted that each of the criteria resulted in improvement over pure simplex for almost every set of problems.

**Table 1.2** Breakdown of the time required by the combination code

| Objects | % zeroes | Name | Total time | % Interior |
|---|---|---|---|---|
| 150 | 0 | r150a0 | 70 | 56.2 |
| | | r150b0 | 66 | 58.9 |
| | | r150c0 | 72 | 54.4 |
| | | r150d0 | 66 | 56.6 |
| | | r150e0 | 65 | 58.1 |
| | | Mean | 68 | 56.9 |
| 200 | 0 | r200a0 | 200 | 51.1 |
| | | r200b0 | 245 | 36.9 |
| | | r200c0 | 193 | 45.9 |
| | | r200d0 | 198 | 47.5 |
| | | r200e0 | 209 | 44.3 |
| | | Mean | 208 | 45.1 |
| 250 | 0 | r250a0 | 683 | 38.3 |
| | | r250b0 | 548 | 46.1 |
| | | r250c0 | 574 | 44.4 |
| | | r250d0 | 524 | 51.0 |
| | | r250e0 | 631 | 41.0 |
| | | Mean | 592 | 44.2 |
| 100 | 20 | r100a2 | 185 | 89.2 |
| | | r100b2 | 426 | 37.0 |
| | | r100c2 | 214 | 83.8 |
| | | r100d2 | 132 | 72.5 |
| | | r100e2 | 192 | 86.6 |
| | | Mean | 230 | 73.8 |
| 150 | 10 | r150a1 | 218 | 29.5 |
| | | r150b1 | 165 | 39.2 |
| | | r150c1 | 221 | 29.9 |
| | | r150d1 | 237 | 27.7 |
| | | r150e1 | 200 | 32.0 |
| | | Mean | 208 | 31.6 |
| 200 | 10 | r200a1 | 1100 | 34.1 |
| | | r200b1 | 886 | 73.4 |
| | | r200c1 | 692 | 47.1 |
| | | r200d1 | 1666 | 35.8 |
| | | r200e1 | 836 | 52.3 |
| | | Mean | 1036 | 48.6 |

The final number of constraints for the pure interior point code are approximately:

**Table 1.3** A typical run for r150a1

| Stages | Interior | | Simplex | | Crossover | |
|---|---|---|---|---|---|---|
| | Cuts | Itns | Cuts | Itns | Cuts | Itns |
| 1 | 1000 | 3 | 2151 | 2100 | 1000 | 3 |
| 2 | 1948 | 3 | 1365 | 977 | 1948 | 3 |
| 3 | 1001 | 3 | 942 | 718 | 1001 | 3 |
| 4 | 288 | 3 | 740 | 897 | 288 | 3 |
| 5 | 248 | 3 | 726 | 1703 | 248 | 3 |
| 6 | 194 | 4 | 1070 | 3285 | 194 | 4 |
| 7 | 145 | 3 | 1157 | 2363 | 145 | 6041 |
| 8 | 445 | 4 | 203 | 3569 | 143 | 104 |
| 9 | 79 | 4 | 1118 | 2074 | 121 | 132 |
| 10 | 38 | 3 | 132 | 2385 | 126 | 98 |
| 11 | 40 | 4 | 98 | 2762 | 97 | 131 |
| 12 | 30 | 4 | 540 | 903 | 110 | 130 |
| 13 | 34 | 4 | 26 | 880 | 104 | 147 |
| 14 | 23 | 4 | 16 | 345 | 38 | 126 |
| 15 | 23 | 4 | 9 | 236 | 97 | 76 |
| 16 | 55 | 3 | 10 | 127 | 18 | 36 |
| 17 | 9 | 5 | 1 | 58 | 7 | 1 |
| 18 | 17 | 5 | 4 | 75 | 8 | 1 |

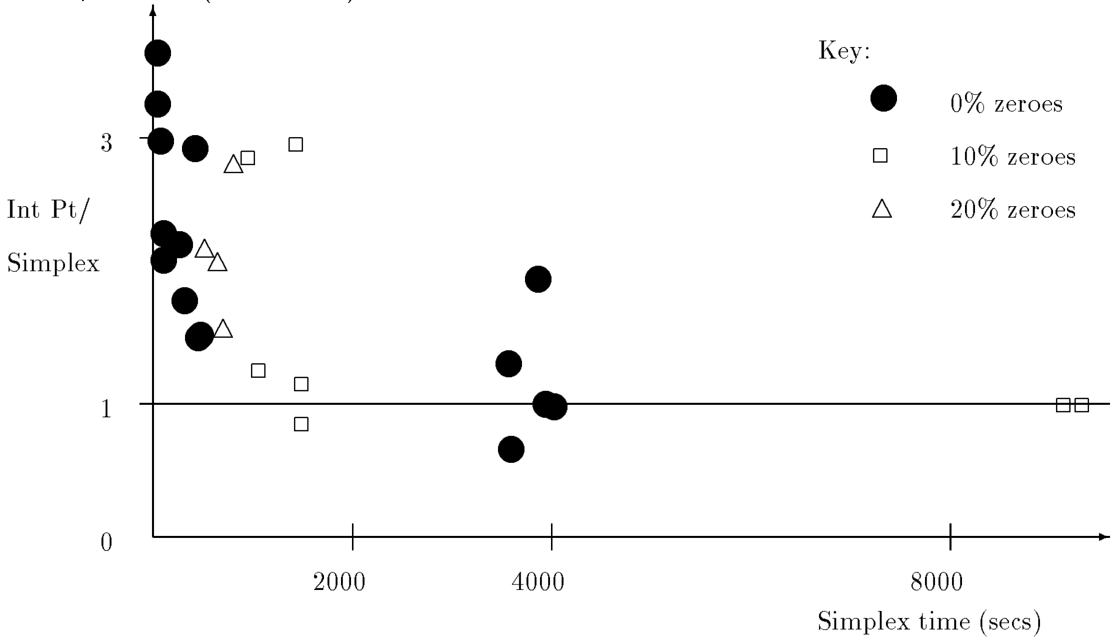
| | |
|---|---|
| `r100.2`: | 3000 constraints. |
| `r150.0`: | 4000 constraints. |
| `r150.1`: | 5000 constraints. |
| `r200.0`: | 6500 constraints. |
| `r200.1`: | 8000 constraints. |
| `r250.0`: | 10000 constraints. |

## 1.8 CONCLUSIONS

For larger problems, the interior point and simplex codes require comparable time. The interior point solver is a research code, and we believe based on our experience in solving standard test problems that this interior point solver is roughly half as fast as current high quality interior point solvers.

For sufficiently hard problems, combining the two codes performs **significantly better** than either code individually. It appears that the interior point method is faster in the early stages and the simplex method is faster in the later stages. Furthermore, it appears that the interior point method is able to add a better set of cutting planes in the early stages, because it is looking for constraints at an interior point. There are at least two reasons why the interior point is useful in the early stages:

**Figure 1.1** Time to solve with the interior point solver versus time to solve with the simplex solver. (No crossover.)



- At the extreme point, most of the violated constraints are violated by exactly one, so it is hard to discriminate between these constraints to find the more important ones. With the interior point solver, there is a greater range of violations, which provides more information about the relative importance of the constraints.

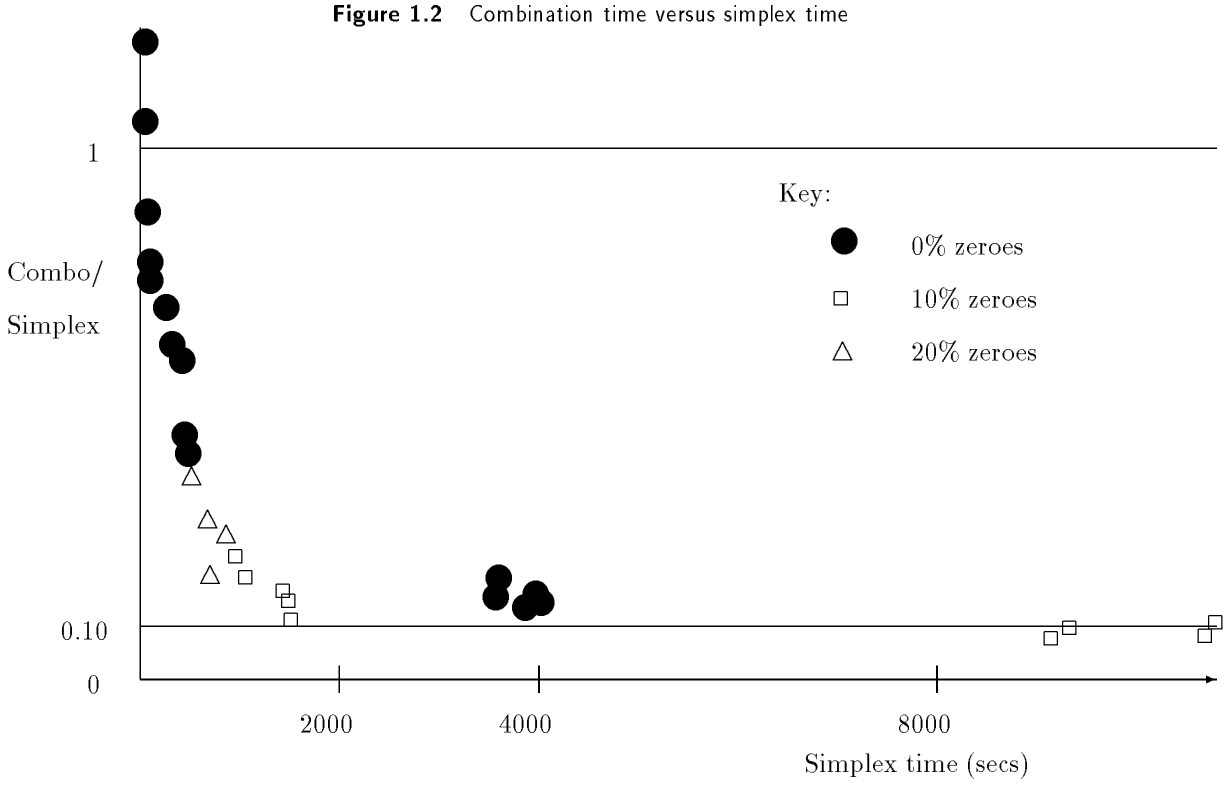
- Initially, the problem may have primal and dual degeneracy. This hurts the simplex method by forcing it to take more iterations. Further, the interior point method finds an interior point close to the middle of the optimal face, so it will find cuts that are useful over much of the optimal face, rather than just at one vertex of this face.

## References

[Bahn et al., 1995] Bahn, O., Merle, O. D., Goffin, J. L., and Vial, J. P. (1995). A cutting plane method from analytic centers for stochastic programming. *Mathematical Programming*, 69:45–73.

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

**Figure 1.2** Combination time versus simplex time



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

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

[Fishburn, 1992] Fishburn, P. C. (1992). Induced binary probabilities and the linear ordering polytope: A status report. *Mathematical Social Sciences*, 23:67–80.

[Goffin et al., 1997] Goffin, J.-L., Gondzio, J., Sarkissian, R., and Vial, J.-P. (1997). Solving nonlinear multicommodity network flow problems by the analytic center cutting plane method. *Mathematical Programming*, 76:131–154.

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

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

[Gondzio and Sarkissian, 1996] Gondzio, J. and Sarkissian, R. (1996). Column generation with a primal-dual method. Technical report, Logilab, HEC Geneva, Section of Management Sciences, University of Geneva, 102 Bd Carl Vogt, CH-1211 Geneva 4, Switzerland.

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

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

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

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York.

[Leung and Lee, 1994] Leung, J. and Lee, J. (1994). More facets from fences for linear ordering and acyclic subgraph polytopes. *Discrete Applied Mathematics*, 50:185–200.

[Mitchell, 1997] Mitchell, J. E. (1997). Computational experience with an interior point cutting plane algorithm. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590. Revised: April 1997.

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

[Mitchell and Borchers, 1992] Mitchell, J. E. and Borchers, B. (1992). A primal-dual interior point cutting plane method for the linear ordering problem. *COAL Bulletin*, 21:13–18.

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

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

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