Solving the *seymour* problem*

Michael C. Ferris†
Gábor Pataki‡
Stefan Schmieta§

July 11, 2001

Optimization problems are at the heart of much of operations research and can vary substantially both in complexity and size. In many problems, the sheer size of the instance makes it very difficult to solve due to time or space limitations. In others, the complexity of the problem (nonlinearities, nonconvexities, or discreteness) can make it difficult or impossible to solve to optimality, even for reasonable sized instances. This note addresses an instance of the latter type of problem, arising as a mixed integer program (MIP) involving discrete variables and linear functions.

Hard problems in MIPLIB

The MIPLIB library of mixed integer programs was created in 1992 ([4]) and most recently updated in 1998 ([5]). Several problems in the library gained some notoriety, for being among the toughest. Some of these are:

- The *danoint* and *dano3mip* problems that arise from network design; the latter of these is unsolved to this date.
- The *markshare* problems, that were created with particular malice to challenge branch-and-bound, and cutting plane algorithms.
- The *seymour* problem: a relatively small setcovering problem with a fascinating origin, and of remarkable difficulty.

A group of researchers, consisting of the authors, of Sebastian Ceria at Columbia University, and Jeff Linderoth at Argonne National Laboratory has recently succeeded in solving the *seymour* problem. In this article, we describe why we found this problem so alluring, what experiments we have done, and eventually, what techniques led us to its solution.

Background on *seymour*

The *seymour* problem is a setcovering problem; i.e. a problem of the form

$$\begin{aligned} \min \quad & e^T x \\ st. \quad & Ax \geq e \\ & x \in \{0,1\}^n \end{aligned}$$

where e denotes a vector of all ones of appropriate dimension, and A is a matrix of zeros and ones. The number of rows in A is 4944 and the number of columns is 1372. The instance was posed by Paul Seymour, as a by-product of a new proof of the Four Color Theorem (FCT) by Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas [12, 13].

An interesting short history of this problem is given by these authors at [9] which we reproduce here verbatim.

The Four Color Problem dates back to 1852 when Francis Guthrie, while trying to color the map of counties of England noticed that four colors sufficed. He asked his brother Frederick if it was true that any map can be colored using four colors in such a way that adjacent regions (i.e. those sharing a common boundary segment, not just a point) receive different colors. Frederick Guthrie then communicated the conjecture to DeMorgan. The first printed reference is due to Cayley in 1878.

A year later the first 'proof' by Kempe appeared; its incorrectness was pointed out by Heawood 11 years later. Another failed proof is due to Tait in 1880; a gap in the argument was pointed out by Petersen in 1891. Both failed proofs did have some value, though. Kempe discovered what became known as Kempe chains, and Tait found an

*This material is based on research supported by the National Science Foundation Grants CDA-9726385, CCR-9972372 and DMS 95-27-124 and the Air Force Office of Scientific Research Grant F49620-01-1-0040

†Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706 (ferris@cs.wisc.edu)

‡Department of Operations Research, Smith Building, University of North Carolina, Chapel Hill, NC 27599 (gabor@unc.edu). This research was done while the author was at the Department of IE/OR, at Columbia University

§Axioma Inc. (sschmieta@axiomainc.com). This research was done while the author was at the Department of IE/OR, at Columbia University *equivalent formulation of the Four Color Theorem in terms of 3-edge-coloring.*

The next major contribution came from Birkhoff whose work allowed Franklin in 1922 to prove that the four color conjecture is true for maps with at most 25 regions. It was also used by other mathematicians to make various forms of progress on the four color problem. We should specifically mention Heesch who developed the two main ingredients needed for the ultimate proof - reducibility and discharging. While the concept of reducibility was studied by other researchers as well, it appears that the idea of discharging, crucial for the unavoidability part of the proof, is due to Heesch, and that it was he who conjectured that a suitable development of this method would solve the Four Color Problem. This was confirmed by Appel and Haken in 1976, when they published their proof of the Four Color Theorem.

The web page also gives an outline of the proof of the theorem, and a longer list of pertinent references.

The *seymour* IP formulates the problem of finding the smallest unavoidable set of configurations that must be “reduced” in order to prove the FCT. Here “reduced” is a technical term meaning that the configuration can be shown not to exist in a minimal counterexample. Seymour has found a solution of value 423, but until this work, we are aware of no one who has been able to reproduce such a solution. The problem actually has many solutions of value 423.

The value of its LP relaxation is 403.84. Therefore, all one must do is raise the lower bound to say 422.0001 (or to better safeguard against numerical errors, to say 422.1) to prove the optimality of Seymour's solution; in fact, for a long time, we were aiming for 423.0001, as the best solution we could find was of value 424.

One may question the value of spending months of research effort trying to solve such hard IP's, which have no particular “realistic” application. We can argue though, that it is the small, and hard problems from which one can learn the most - and the techniques one develops through their study are very much applicable to real-world, difficult problems.

First attempts: branch-and-bound

The first attempt to solve *seymour* was done in 1996 by Greg Astfalk running CPLEX 4.0 with default settings on an HP SPP2000 with 16 processors, each processor having 180 MHz frequency, and 720 Mflops peak performance, for the total of approximately 58 hours, enumerating about 1,275,000 nodes, and using approximately 1360 Mbytes of memory. In this run, CPLEX did not even close 9 units of the gap; remember that we must close a bit more than 18.16 units.

We can do quite a bit better, just by using CPLEX, with the variable selection rule of *strong branching*. Strong branching (SB) was developed by Applegate, Bixby, Chvatal and Cook in their work on the TSP, and it is an available setting in several commercial MIP solvers now. At every node of the branch-and-bound tree, SB tests several variables as a candidate to branch on (by partially reoptimizing on both branches with a limited number of dual simplex pivots), and picks the most promising one.

Figure 1 shows what lower bound the CPLEX 6.0 branch-and-bound code has achieved after enumerating a hundred thousand nodes by using default branching variable selection vs. SB variable selection (all other settings were default). On the horizontal axis one mark means 10 thousand branch-and-bound nodes. The run with SB closed nearly 9 units of the gap, and took about a week on a 337 MHz speed machine.

Cutting

Disjunctive cuts were introduced by Balas in the seventies [1], then rediscovered from a different viewpoint in the nineties [11, 14, 2, 3]. They were termed lift-and-project cuts and computationally tested in the nineties by Balas, Ceria and Cornuéjols in [2, 3]. For our experiments, we used the more recent implementation described in [6].

Here we only give a description of disjunctive cuts in a nutshell. Given P, the linear programming relaxation of a 0-1 program, and a variable x_i, the inequality $\alpha x \geq \beta$ is a 0-1

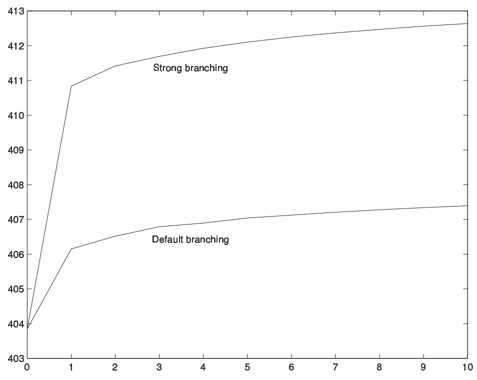

Figure 1: Strong branching vs. default branching on *seymour*

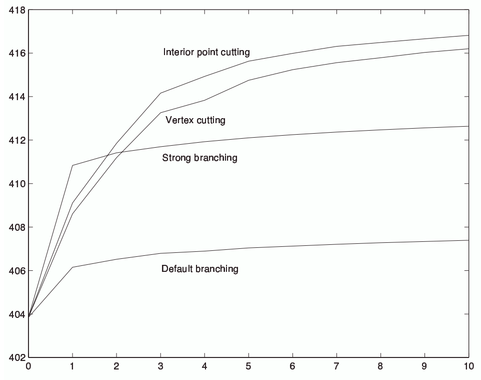

Figure 2: Cutting with 2 options vs. branch-and-bound with 2 options

disjunctive cut from the disjunction $x_i = 0 \vee x_i = 1$, if it is valid for both of the sets $P \cap \{x | x_i = 0\}$ and $P \cap \{x | x_i = 1\}$. From among all such inequalities, following the method developed in [2], one generates the cut which is violated by the current LP optimum $\bar{x}$ by the largest possible amount, i.e. $\beta - \alpha\bar{x}$ is maximized, with respect to some normalization constraint. Generating such a cut is done by solving an LP. Disjunctive cuts (as all cuts in MIP) are added to the LP formulation in rounds; i.e. one adds a batch of cuts, reoptimizes the LP, drops all nonbinding cuts, then repeats the procedure.

According to our experience, it is easy to tell, whether it is worth applying disjunctive cuts on a particular IP, just by looking at the result of two branch-and-bound runs. Disjunctive cuts work (i.e. adding them significantly raises the LP lower bound) if (and one can say, only if) strong branching works! The informal explanation is that both techniques attempt to enhance the effect of the branching operation. Strong branching does this by selecting the best variable to branch on. Generating disjunctive cuts from say 50 variables mimics the effect that can be gained from branching on those variables (of course, adding these 50 cuts will not result in a lower bound as good as the one from a 50 level deep branch-and-bound tree).

In our first experiment, we generated 10 rounds of 100 cuts, by selecting the 100 variables that were the most fractional in the current LP optimum. This run took about 6 hours, and closed about 9.45 units of the gap!

After some experimentation, we produced a formulation called *Formulation 1*, that we thought was worth trying to finish off with branch-and-bound. The setup was as follows:

- In each cutting iteration we generated cuts from *all* the fractional variables; there were typically about 600 of these.
- We sorted the cuts, by putting the one first from which the euclidean distance of the LP optimal solution is the largest, and so on. The distance of the hyperplane $\{x \mid \alpha x = \beta\}$ from the point $\bar{x}$ is
$$\frac{\beta - \alpha\bar{x}}{\|\alpha\|}.$$
Then, assuming that we have c cuts, we perform the following step for $i = 1, \ldots, c$:
  – If the cosine of the angle of i^{th} cut hyperplane with any one of the first $i - 1$ cut hyperplanes is greater than 0.999, we discarded the i^{th} cut.

  From the remaining cuts we picked the first 250, and added them to the LP formulation.

We call the above method *cut selection by distance*. Another method that is quite natural is called *cut selection by usage*; we describe this method next. Suppose again, that we would like to select the “best” 250 cuts to be added to the LP formulation. We tentatively add all of them, then track the course of the reoptimization by the dual simplex algorithm using the steepest edge pivot rule. Whenever a cut is pivoted on, we mark it. We let dual simplex run, until 250 cuts get marked; these will be the selected ones. We never unmark a cut, and whether a cut is pivoted on once, or more than once does not matter. Perhaps surprisingly, we found that out of more than 600 cuts, each of which is violated by the current solution, we could never choose more than about 350 with this method – at most this many are *ever* pivoted on in the course of the reoptimization!

Cut selection by distance and by usage performed quite similarly on the *seymour* problem; it would be interesting to see how they work on other hard IP's, especially, when more than one type of cut (e.g. knapsack, flow-cover, Gomory-cuts) is used.

Figure 2 depicts the progress of the cutting plane algorithm with two different settings versus branch-and-bound with default branching and strong branching. The cutting strategy that worked best was cutting off an LP solution in the interior of the optimal face, as opposed to the usual vertex cutting. On the horizontal axis one mark means 10 thousand enumerated nodes for branch-and-bound, and one round of cutting for lift-and-project cuts. The progress made by branch-and-bound and cutting planes is quite well comparable this way, even though the time taken by the four different algorithms between two tickmarks on the horizontal axis can be different of course. For example branch-and-bound with SB took about a week to enumerate 100 thousand nodes, while with default branching it took only 3 days. The most time-consuming was cutting with the "interior point cutting" option; this took about 2 weeks. Still, in the case of *seymour* the question is simply solving it, or not; hence the few days difference in the running times is irrelevant in this case. From Figure 2 it is clear that after 100 thousand nodes, or 10 rounds, both algorithms completely "ran out of steam"; even after several more months, or years they would not solve the problem.

Formulation 1 was fed to the CPLEX branch-and-bound solver, again with the well-tested SB setting. After about 100 thousand nodes, the lower bound was further pushed up by about 3 units, for the total of about 15 units; at that point it was clear, that this way we will never solve the problem. At the same time, it also became clear that producing a limited number of nodes in a *branch-and-cut tree*, each with at least 15-16 units of the gap closed, would do the trick; we would simply need to process those nodes by branch-and-bound afterwards. Hence we set up a run to generate the required nodes, in which a certain number of cutting rounds was followed by branching for a number of levels in the branch-and-bound tree, then the process repeated. Precisely, we

- Generated 10 rounds of cuts at the root node.
- Ran B&B for 4 levels.
- At each of the 2^4 = 16 nodes, we generated 2 more rounds of cuts.
- Ran B&B for 4 levels.
- At each of the 2^8 = 256 nodes, we generated 1 more round.

That is, at the end we had 2^8 = 256 nodes in the tree, and on the way from the root to any one of them 13 rounds of cuts were generated. We used SB for the variable selection; interior-point cutting, and selecting 250 cuts by usage for cut generation. We remark that all cuts generated within the tree were globally valid, i.e. they were used at the other nodes as well.

In the end, the gap closed at

- the best node was: 16.77
- the worst: 15.17
- the median: 16.29

We remark that the problem was preprocessed at the root node by deleting all dominated rows and columns as usual in setcovering problems. The reduced problem has 4323 rows, and 882 columns; the IP value of 423 in the original problem corresponds to the value of 238 in the preprocessed problem. Although in the parallel processing of the nodes we had to set the cutoff values to take into account the preprocessing, we translated these values back to correspond to the original instance.

The Condor system

Heterogeneous clusters of workstations are becoming an important source of computing resources. One approach to use these clusters of machines more effectively allows users to run their (computing intensive) jobs on idle machines that belong to somebody else. The Condor system [10, 8] that has been developed at University of Wisconsin-Madison is one scheme that manages such resources in a local intranet setting. It monitors the activity on all participating machines, placing idle machines in the Condor pool, that are allocated to service job requests from users. Users' programs are allowed to run on any machine in the pool, regardless of whether the user submitting the job has an account there or not. The system guarantees that heavily loaded machines will not be selected for an application.

Machines enter the pool when they become idle, and leave when they get busy, e.g. the machine owner returns. To protect ownership rights, whenever a machine's owner returns, Condor immediately interrupts any job running on that machine, migrating the job to another idle machine. In fact, the running job is initially suspended in case the executing machine becomes idle again within a short timeout period. If the executing machine remains busy, then the job is migrated to another idle workstation in the pool or returned to the job queue. For a job to be restarted after migration to another machine a checkpoint file is generated that allows the exact state of the process to be re-created. This design feature ensures the eventual completion of a job. In order to use the checkpoint feature, the job to be executed must just be relinked before being submitted to the Condor manager. An additional benefit of this relinking is that remote I/O can be performed on the submitting machine, therefore limiting the footprint of the job on the executing machine.

There are various priority orderings used by Condor for determining which jobs and machines are matched at any given instance. A job advertises its requirements via the simple mechanism of a "job description file". This file informs Condor of the location of the executable and the input and output files, along with the required architecture, operating system and memory needs of the job. A machine similarly advertises its properties and a matching scheme (implemented within the resource manager) pairs jobs to machines. Based on the priority orderings, running jobs may sometimes be preempted to allow higher priority jobs to run instead. Condor is freely available and has been used in a wide range of production environments for more than ten years.

Since the CPLEX suite of optimization procedures comes in library form, it is very easy to carry out the relinking of a simple driver program to run the *seymour* problem. On June 23, 1999, we submitted two separate CPLEX 6.0 jobs in an attempt to solve Formulation 1 described above. Both were set to run in depth first search mode to ensure the size of the stored branch and bound tree did not exceed the memory of the machines on which it ran. A cutoff value was set that excludes the presumed optimal solution by 1. In one job, the remaining parameters to CPLEX were set to default values, while in the other job, strong branching was carried out. At the time of writing this article, both jobs are still running. Condor has provided both jobs with over 600 days of CPU time in the ensuing two years.

One of the jobs has explored over 13.4 million nodes of the tree, while the other has processed close to 2.5 million nodes. As expected, neither has found a solution that exceeds the cutoff value, and furthermore, the lower bound has remained essentially stagnant for the large part of this time. Clearly, just applying brute force execution time to this problem is not going to solve it. However, it is interesting to note the reliability of both the Condor and CPLEX systems to be able to continue executing on a variety of different machines during this two year period.

One issue about the above computation is that each execution is limited to one processor. While it would be possible to use the parallel version of CPLEX to increase resources applied to the solution, it is not at all clear whether the parallel code would be able to run in the Condor environment.

Processing the nodes on Condor

Instead, we submitted the 256 IP subproblems described above, as 256 separate tasks. While the efficiencies generated by intercommunication between these tasks would be lost, the extra processing available at the root nodes of all the tasks that was described above was thought to more than compensate for this loss. Furthermore, any collection of resources could be used to solve these 256 instances, involving state-of-the-art commercial packages.

MPS input files for all 256 subproblems can be found at [7]. Each problem is listed with the ID of the corresponding node in the branch-and-bound tree. The file you get by clicking on the link will be called "node.mps". The nodes are sorted by lower bound, which is computed as the value of the LP relaxation.

These problems were processed using CPLEX 6.6 and XPRESS 11.50. 219 of the problems were solved using CPLEX via Condor at Wisconsin. The remainder were processed using XPRESS 11.50 and CPLEX 6.6 at Columbia.

In general, we used 423.01 as the upper cutoff for the solvers, since at the outset of this work we were somewhat skeptical regarding the existence of a solution of value 423; we did have one with value 424 though. On July 4, 2000, we did find a solution with 423, after this the remaining subproblems were set up with an upper cutoff of 422.01. In the end, we were able to generate several solutions of value 423. The electronic citation [7] gives the binary variables that take on value 1 in two distinct optimal solutions.

For the 219 jobs that were run under Condor, the total CPU time used to process them all was 443.6 days, with 41.7 days idle time for jobs waiting in the Condor queue. During this time 10,244,500 nodes were explored using 3,261,696,402 pivots running on a total of 883 different machines. The longest single node took 36 days to complete, and the shortest completed in just under 53 minutes. At Columbia, a further 48.9 CPU days were used to explore 934,868 nodes. The actual elapsed time between starting the process and ending the process was 37 days, starting in June 2000 and ending on July 26, 2000.

Acknowledgement

Thanks are due to Oktay Günlük for helpful comments on a draft of this paper.

References

[1] E. Balas, Disjunctive programming, *Annals of Discrete Mathematics 5* (1979) 3-51.

[2] E. Balas, S. Ceria and G. Cornuéjols, A lift-and-project cutting plane algorithm for mixed 0-1 programs, *Mathematical Programming 58* (1993) 295-324.

[3] E. Balas, S. Ceria and G. Cornuéjols, Mixed 0-1 Programming by lift-ad-project in a branch-and-cut framework, *Management Science 42* (1996) 1229-1246.

[4] R. E. Bixby, E. A. Boyd, and R. R. Indovina, MIPLIB: A Test Set of Mixed Integer Programming Problems, *SIAM News* 25 (1992).

[5] R. E. Bixby, S. Ceria, C. M. McZeal, M. W. P. Savelsbergh, An Updated Mixed Integer Programming Library: MIPLIB 3.0, *Optima 54* (1998), 12-15.

[6] S. Ceria and G. Pataki, Solving Integer and Disjunctive Programs by Lift-and-Project, *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization* (1998) 271-283.

[7] http://firulete.ieor.columbia.edu/~schmieta/seymour.html, List of 256 nodal subproblems.

[8] http://www.cs.wisc.edu/condor, The Condor Project, High Throughput Computing.

[9] http://www.math.gatech.edu/~thomas/FC/fourcolor.html, The four color theorem.

[10] M. J. Litzkow, M. Livny, and M. W. Mutka, Condor: A hunter of idle workstations, In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104-111, June 1988.

[11] L. Lovász and A. Schrijver, Cones of matrices and set-functions and 0-1 optimization, S*IAM Journal on Optimization 1* (1991), 166-190.

[12] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas, The four colour theorem, *J. Combin. Theory* Ser. B. 70 (1997), 2-44.

[13] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas, A new proof of the four colour theorem, *Electron. Res. Announc. Amer. Math. Soc. 2* (1996), 17-25 (electronic).

[14] H. Sherali and W. Adams, A hierarchy of relaxations between the continuous and convex hull representation for zero-one programming problems, *SIAM Journal on Discrete Mathematics 3* (1990) 411-430.