AFRL-SR-AR-TR-09-0081

# REPORT DOCUMENTATION PAGE



**PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.**

| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| 18-02-2009 | Final | 15/02/07-30/11/08 |

**4. TITLE AND SUBTITLE**
Combining Exact and Heuristic Approaches to Discrete Optimization

**5a. CONTRACT NUMBER**
FA9550-07-1-0177

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
George L. Nemhauser
Mathieu W. Savelsbergh

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Georgia Tech Research Corporation
505 10th St.
Atlanta, GA 30332-0001

**8. PERFORMING ORGANIZATION REPORT NUMBER**
240664R

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Air Force Office of Scientific Research/NL
875 N. Randolph St., Room 3112
Arlington, VA 22203
Dr. Donald Hearn

**10. SPONSOR/MONITOR'S ACRONYM(S)**
AFOSR

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution is unlimited

**13. SUPPLEMENTARY NOTES**
none

**14. ABSTRACT**

In the last decade the computational power of discrete optimization methodology has increased remarkably to the point where problems that could not be solved with days of computation can now be solved in minutes by commercial solvers. This success has stimulated the need for methodology to solve even much larger problems and the desire to solve problems in real-time. We have conducted research that has yielded computationally effective algorithms to provide high-quality solutions to very large-scale planning problems and high-quality solutions in (nearly) real-time to operational problems. Traditionally, this goal has been pursued with heuristic approaches. The underlying theme of our research has been different. We use exact optimization whenever possible. For example, to solve problems that are too large for exact optimization, we embed exact optimization in a heuristic search algorithm. To solve real-time instances that differ only by small perturbations of coefficients, we use exact optimization as a planning tool whereby we solve a core instance exactly and use results from its solution to make the solution of the real-time instances much faster than would be the case if they were solved from scratch.

**15. SUBJECT TERMS**
Discrete optimization, heuristics, exact methods

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | R. Scott Goodwin |
| U | U | U | UU | | 19b. TELEPHONE NUMBER *(Include area code)* 404-894-6920 |

# Combining Exact and Heuristic Approaches for Discrete Optimization

*George L.Nemhauser, Martin W.P. Savelsbergh*

Contract #: FA9550-07-1-0177
Final report

## Executive Summary

In the last decade the computational power of discrete optimization methodology has increased remarkably to the point where problems that could not be solved with days of computation eau now be solved in minutes by commercial solvers such as CPLEX and XPRESS. This success has stimulated the need for methodology to solve even much larger problems and the desire to solve problems in real-time. We have eondueted researeh that has yielded computationally effective algorithms to provide high-quality solutions to very large-scale planning problems and high-quality solutions in (nearly) real-time to operational problems.

Traditionally, this goal has been pursued with heuristie approaches. The underlying theme of our research has been different. We use exact optimization whenever possible. For example, to solve problems that are too large for exact optimization, we embed exact optimization in a heuristic search algorithm. To solve real-time instances that differ only by small perturbations of coeffieients, we use exact optimization as a planning tool whereby we solve a eore instanee exactly and use results from its solution to make the solution of the real-time instanees mueh faster than would be the ease if they were solved from scratch. Furthermore, we revisit the least studied and least understood aspeet of integer programming methodology, namely branching, and develop novel braneliing strategies based on learning eoncepts and restart ideas.

The faculty involved in this research are Professor George Nemhauser (PI) and Professor Martin Savelsbergh (eo-PI). The students involved are Michael Hewitt, Fatma Kilinc-Karzan, and Juan Pablo Vielma. Hewitt is expected to complete his PhD thesis entitled Combining Exaet and Heuristic Methods for Discrete Optimization this summer. Vielma will also complete this summer and his thesis is entitled Mixed Integer Programming Approaches for Nonlinear and Stochastic Programming. Papers submitted for publication so far are:

M. Hewitt, G.L. Nemhauser, and M.W.P. Savelsbergh. "Combining Exaet and Heuristic Approaehes for the Capaeitated Fixed Charge Network Flow Problem." Submitted to INFORMS J. on Computing (2008).

F. Kılınç-Karzan, A. Toreillo, S. Ahmed, G.L. Nemhauser, M.W.P. Savelsbergh. "Approximating the Stability Region for Binary Mixed-Integer Programs." Submitted to Operations Research Letters (2008).

F. Kılınç-Karzan, G.L. Nemhauser, M.W.P. Savelsbergh. "Information Based Branching Rules for Binary Mixed Integer Problems." Submitted to INFORMS J. on Computing (2009).

# 1 Embedding Exact Optimization in Heuristic Search

Despite the impressive gains that have occurred in mixed-integer programming (MIP) solvers in the last decade, there are still many significant practical problems that cannot be solved to optimality using the leading commercial solvers such as CPLEX and XPRESS. These problems are either too large or too complex for reasons such as difficulty in finding a feasible solution or a large gap between the values of the linear programming (LP) relaxations and MIP optimal solutions. In such situations, a pragmatic alternative is to search for provably good solutions.

One of our research thrusts has been to use neighborhood search to obtain nearly optimal solutions to huge and complex problems. The basic idea is very simple. A subproblem involving a suitably small subset of the variables is selected and all other variables are fixed. This subproblem is solved by a MIP solver. Hopefully a better solution to the whole problem is obtained. Then we repeat the process by choosing a different subproblem (see Algorithm 1).

**Algorithm 1** Neighborhood Search

```
while the search time has not exceeded a prespecified limit T do
  Choose a subset of variables V
  Solve the IP defined by variables in V
  if an improved solution is found then
    Update the global solution
  end if
end while
```

The key to making this approach work is problem dependent. We illustrate this approach on the Fixed Charge Network Flow (FCNF) problem, which is a classic discrete optimization problem in which a set of commodities has to be routed through a directed network. Each commodity has an origin, a destination, and a quantity. Each network arc has a capacity. There is a fixed cost associated with using an arc and a variable cost that depends on the quantity routed along the arc. The objective is to minimize the total cost. Two versions of the problem are considered: commodities have to be routed along a single path and commodities can be routed along multiple paths.

By selecting a suitably small subset of the commodities and assuming that the flow paths for all other commodities are fixed a tractable integer program can be defined and solved using an IP solver. Similarly, by selecting a suitably small subnetwork and assuming all flows outside the subnetwork are fixed a tractable integer program can be defined and solved using an IP solver. Hopefully, a better solution to the whole problem is obtained by these subproblem solves. The process can be repeated multiple times by choosing different subsets of commodities and different subnetworks.

Both the selection of a subset of commodities and the selection of a subnetwork define a subset of network arcs. If this subset of network arcs contains all the arcs that carry flow in the current best solution, then we can seed the IP with that solution and ensure that we only find solutions that are at least as good as the current best. Therefore, all our schemes for defining a smaller IP start from the subset of arcs associated with the current best solution. Also, our schemes do not choose arcs directly but choose paths, thus ensuring that every arc chosen exists in some feasible solution. Examples of commodity subset selection ideas include:

- Select commodities whose paths in the current best solution use arcs for which the reduced cost in the most recent LP solution are far from 0. Thus, we try to re-route commodities away from arcs that are far away from satisfying complementary slackness.

- Suppose in the current best solution, there is a node with two commodities entering on a single arc, but leaving on two different arcs. When these arcs are not fully used, the two commodities are good candidates for re-optimization.
- Since we want to find sets of commodities that are likely to share arcs, we search for commodities whose paths in the current best solution are close together.
- Suppose the network is very large and there are two commodities and that have only one feasible path from their source to their sink. In that case, it is unlikely that these paths have a common arc hence re-optimizing commodities the two commodities together is not likely to lead to improvements. Conversely, if the two commodities have many paths, then they are likely to share arcs and thus are good candidates to optimize together.

Examples of subnetwork selection ideas include (they refer to to the path-based formulation we maintain to get dual bounds):

- Select paths that often appear in improving solutions, but are not in our current best solution.
- Select paths that appear in the optimal solution to the LP relaxation of the path-based formulation.
- Select the path produced for each commodity by the pricing problem associated with the path-based formulation.

Note that the approach outlined above differs in two respects from more traditional neighborhood search methods: (1) the neighborhood is explored using integer programming technology, and (2) the neighborhood selection is guided by appropriatcly chosen problem dependent metrics and changes as the search progresses. Note also that this approach can be used on extremely large instances because the algorithm never requires the full instance to be in memory.

The resulting solution approach is very effective. For instances with 500 nodes, with 2000, 2500 and 3000 arcs, and with 50, 100, 150, and 200 commodities, we compared the quality of the solution produced by our solution approach with the best solution found by CPLEX after 15 minutes of computation and after 12 hours of computation. On average, the solution we found in less than 15 minutes is 35% better than CPLEX' best solution after 15 minutes and 20% better than CPLEX' best solution after 12 hours. Furthermore, we find a better solution than CPLEX' best solution after 15 minutes within 1 minute, and CPLEX' best solution after 12 hours within 3 minutes. On these instances the approach produces dual bounds that are 25% stronger than the LP relaxation. We also compared the quality of the solutions produced by our solution approach with the quality of the solutions produced by a recent implementation of the tabu search algorithm of Ghamlouche et al. For nearly all instances in their test set, our solution is better than the solution of the tabu search algorithm and this solution is found much faster.

Table 1 presents details of one of our experiments. It shows the value of the best solution found by CPLEX in 15 minutes and 24 hours, the value of the solution found by our heuristic search, and the difference in quality between the solutions found by CPLEX and the one found by our heuristic search. An "X" in a column indicates that no feasible solution was found. (The value reported for our heuristic search is the value of the solution produced within 15 minutes.)

We observe that in *every* instance IP Search finds a better solution in 15 minutes than CPLEX does in 12 hours. We also see that the improvement over the solution found by CPLEX in 15 minutes is significant, often greater than 30%. Even the improvement over the solution found by CPLEX in 12 hours is impressive, often greater than 20%. Unfortunately, little can be said with confidence regarding the true optimality gap of the solutions produced by IP Search since the dual

Table 1: Primal-side Comparison with CPLEX

| Problem | CPLEX-15M | CPLEX-12H | IP Search | CPLEX-15M Gap | CPLEX-12H Gap |
|---|---|---|---|---|---|
| 500,2000,50,F,L | 5,301,081 | 5,301,081 | 3,910,120 | -35.57 | -35.57 |
| 500,2000,50,F,T | X | 7,927,065 | 5,249,040 | N/A | -51.02 |
| 500,2000,100,F,L | 8,944,724 | 8,299,799 | 6,764,310 | -32.23 | -22.70 |
| 500,2000,100,F,T | 10,199,000 | 8,306,181 | 7,718,750 | -32.13 | -7.61 |
| 500,2000,150,F,L | 10,996,000 | 10,080,000 | 8,618,060 | -27.59 | -16.96 |
| 500,2000,150,F,T | 12,115,000 | 10,770,000 | 9,448,890 | -28.22 | -13.98 |
| 500,2000,200,F,L | 13,808,000 | 12,824,000 | 10,333,200 | -33.63 | -24.10 |
| 500,2000,200,F,T | X | X | 12,425,600 | N/A | N/A |
| 500,2500,50,F,L | 4,611,275 | 4,611,275 | 3,841,350 | -20.04 | -20.04 |
| 500,2500,50,F,T | 5,779,926 | 5,084,529 | 4,666,740 | -23.85 | -8.95 |
| 500,2500,100,F,L | 9,351,042 | 9,251,042 | 6,875,420 | -36.01 | -34.55 |
| 500,2500,100,F,T | 9,724,997 | 7,995,284 | 7,235,520 | -34.41 | -10.50 |
| 500,2500,150,F,L | 13,660,000 | 12,497,000 | 9,730,100 | -40.39 | -28.44 |
| 500,2500,150,F,T | 11,385,000 | 10,683,000 | 7,934,360 | -43.49 | -34.64 |
| 500,2500,200,F,L | 15,539,000 | 13,468,000 | 11,261,300 | -37.99 | -19.60 |
| 500,2500,200,F,T | 18,906,000 | 14,948,000 | 12,825,300 | -47.41 | -16.55 |
| 500,3000,50,F,L | 5,098,318 | 5,098,318 | 3,596,980 | -41.74 | -41.74 |
| 500,3000,50,F,T | 5,615,096 | 4,866,768 | 4,504,260 | -24.66 | -8.05 |
| 500,3000,100,F,L | 8,721,798 | 8,721,798 | 6,577,980 | -32.59 | -32.59 |
| 500,3000,100,F,T | 10,119,000 | 8,330,109 | 7,517,970 | -34.60 | -10.80 |
| 500,3000,150,F,L | 12,628,000 | 12,623,000 | 9,214,960 | -37.04 | -36.98 |
| 500,3000,150,F,T | 12,615,000 | 10,147,000 | 9,186,840 | -37.32 | -10.45 |
| 500,3000,200,F,L | 15,039,000 | 13,441,000 | 10,853,400 | -38.56 | -23.84 |
| 500,3000,200,F,T | 17,883,000 | 13,674,000 | 11,578,000 | -54.46 | -18.10 |
| Average | | | | -35.18 | -22.95 |

bounds produced by CPLEX change very little over the course of the execution and are likely to be weak. In fact, for many of the loosely capacitated instances, CPLEX did not find a significantly better primal solution in 12 hours than it did in 15 minutes. This highlights the difficulty that an LP-based branch-and-bound algorithm can have in finding good primal solutions when the dual bounds are weak.

For further details on the approach and additional computational results see:

M. Hewitt, G.L. Nemhauser, and M.W.P. Savelsbergh. "Combining Exact and Heuristic Approaches for the Capacitated Fixed Charge Network Flow Problem." Submitted to INFORMS J. on Computing (2008).

## 2 Approximating the Stability Region for Binary Mixed-Integer Programs

Suppose we have a difficult discrete optimization problem with some binary variables and a linear objective function for these variables. In addition to finding an optimal solution, we would like to know how much the objective vector can change while maintaining the optimality of the solution. This stability information is useful, for example, when solving problems with perturbed objective functions rapidly.

We consider the optimization problem

$$\begin{aligned} z^* = \max\ & c^* x + h(y) \\ \text{s.t. } & (x, y) \in X \\ & x \in \{0,1\}^n, \end{aligned} \qquad (P(c^*))$$

where $c^*x = \sum_{i \in N} c_i^* x_i$, $N = \{1, \ldots, n\}$ and $(x^*, y^*)$ is an optimal solution. We are interested in the stability of $(x^*, y^*)$ with respect to variations of the cost vector $c^*$.

**Definition 2.1.** The *stability region* of $(x^*, y^*)$ is the region $C \subseteq \mathbb{R}^n$ s.t. $c \in C$ if and only if $(x^*, y^*)$ is optimal for $P(c)$. That is, $C = \{c \in \mathbb{R}^n : c(x - x^*) \leq h(y^*) - h(y), \forall\ (x, y) \in X\}$.

By possibly complementing variables, we assume without loss of generality that $x^* = 0$, so the optimal value of P is $z^* = h(y^*)$.

**Remark 2.1.** *Let $(\hat{x}, \hat{y})$ be feasible for P, with objective value $\hat{z} = c^*\hat{x} + h(\hat{y}) \leq z^*$. Suppose $\hat{c} \in \mathbb{R}^n$ satisfies $\hat{c}\hat{x} > z^* - \hat{z} + c^*\hat{x}$. Then $\hat{c} \notin C$.*

**Remark 2.2.** *Let $\hat{x} \in \{0,1\}^n$, and define $v(\hat{x}) = \max_{(\hat{x}, y) \in X} h(y)$, with $v(\hat{x}) = -\infty$ if no $y$ satisfies $(\hat{x}, y) \in X$. Then $C = \{c \in \mathbb{R}^n : cx \leq h(y^*) - v(x), \forall\ x \in \{0,1\}^n\}$.*

Remark 2.2 implies that $C$ is a polyhedron possibly defined by an exponential number of inequalities. We choose to approximate $C$ using polyhedra defined by polynomially many inequalities. The next two definitions further explain this approximation.

**Definition 2.2.** Let $C$ be the stability region of $(x^*, y^*)$. An *outer approximation* of $C$ is a region $C^+ \subseteq \mathbb{R}^n$ satisfying $C^+ \supseteq C$. An *inner approximation* of $C$ is a region $C^- \subseteq \mathbb{R}^n$ satisfying $C^- \subseteq C$.

The following are two simple but important consequences of Definition 2.2 that help us obtain small outer approximations and large inner approximations.

**Proposition 2.3.**

*i) If $C_1^+, C_2^+$ are outer approximations, $C_1^+ \cap C_2^+$ is an outer approximation.*

*ii) If $C_1^-, C_2^-$ are inner approximations, $\text{conv}(C_1^- \cup C_2^-)$ is an inner approximation.*

Next, we show how to obtain inner and outer approximations of $C$ by solving $n$ problems $\{P_j : j \in N\}$, where each $P_j$ is given by

$$\begin{aligned} z_j = \max\ & c^*x + h(y) \\ \text{s.t. } & (x, y) \in X \\ & x \in \{0,1\}^n \\ & x_j = 1. \end{aligned} \tag{$P_j$}$$

Throughout, we assume without loss of generality that all problems $P_j$ are feasible. Accordingly, we assume that we have solved the problems $P_j$ for every $j \in N$, and determined optimal solutions $(x^j, y^j)$ with corresponding objective values $z_j$.

The following observation follows directly from Remark 2.1 and Proposition 2.3.

**Proposition 2.1.** *The set $C^+ = \left\{c \in \mathbb{R}^n : cx^j \leq z^* - z_j + c^*x^j, \forall\ j \in N\right\}$ is an outer approximation of $C$.*

Let $\gamma_j = z^* - z_j + c_j^*$. Observe that the outer approximation $C^+$ of Proposition 2.1 satisfies

$$\{c \in C^+ : c \geq c^*\} \subseteq \{c \in \mathbb{R}^n : c_j^* \leq c_j \leq \gamma_j, \forall\ j \in N\}. \tag{1}$$

In other words, in the direction $c \geq c^*$, the stability region $C$ of $(x^*, y^*)$ is contained in a box defined by the values $\gamma_j$.

To determine an inner approximation, we make use of the next result.

**Algorithm 2** Binary Solution Cover

**Require:** An optimization problem P with optimal solution $(x^*, y^*)$ satisfying $x^* = 0$.

Set $(x^0, y^0) \leftarrow (x^*, y^*)$, $z_0 \leftarrow z^*$, $I \leftarrow N$.
Add cut $D \equiv (\sum_{i \in I} x_i \geq 1)$ to P.
**for** $k = 1, \ldots, n$ **do**
  Resolve the modified P; get new optimal solution $(x^k, y^k)$ and objective value $c^* x^k + h(y^k) = z_k$.
  **if** P is infeasible **then**
    Set $I_\infty \leftarrow I$.
    Return $k$ and exit.
  **end if**
  Set $I_k^+ \leftarrow \{i \in N : x_i^k = 1\}$, $I_k^- \leftarrow I \cap I_k^+$.
  Set $I \leftarrow I \setminus I_k^-$; modify cut $D$ accordingly.
**end for**

**Proposition 2.2.** *Let* $\tilde{c}^j = (c_1^*, \ldots, c_{j-1}^*, \gamma_j, c_{j+1}^*, \ldots, c_n^*)$. *Then* $\tilde{c}^j \in C, \forall\ j \in N$.

**Theorem 2.3.** *Suppose we order the* $x$ *variables so that* $z^* \geq z_1 \geq z_2 \geq \cdots \geq z_n$ *holds. Then*

$$C^- = \left\{ c \geq c^* : \sum_{i=1}^{j} c_i \leq \gamma_j + \sum_{i=1}^{j-1} c_i^*, \forall\ j \in N \right\} \tag{2}$$

*is an inner approximation of* $C$.

**Corollary 2.4.** *The set* $\{c + d : c \in C^-, d \leq 0\}$ *is an inner approximation of* $C$.

These last two results motivate a natural algorithm for determining an inner approximation. Solve each of the problems $P_j$ in turn, sort them by objective value, and compute the inner approximation as indicated in Theorem 2.3. This procedure can be modified slightly to potentially reduce the number of solves.

Algorithm 2 begins with a problem of the form P and an optimal solution $(x^*, y^*)$ with $x^* = 0$. It then adds a cut $\sum_{i \in N} x_i \geq 1$, forcing at least one of the $x$ variables to one. After resolving, it determines which of the $x$ variables switched, removes their coefficients from the cut, and repeats. The end result is a list of solutions, ordered by objective value, which covers all possible $x$ variables. (Variables not covered by any solution on the list are those fixed to zero in any feasible solution.) For future reference, we formally define the relevant information gathered during the execution of Algorithm 2 as follows. The solution in the $k$-th iteration is denoted by $(x^k, y^k)$ and has objective function value $z_k$. The set $I_k^+$ indicates which binary variables have value one in $(x^k, y^k)$. The set $I_k^-$ indicates which of these variables have value one for the first time.

An outer approximation is easily obtained applying Remark 2.1 to each solution $(x^k, y^k)$. To determine an inner approximation, we must first establish a preliminary fact.

**Proposition 2.1.** *Let* $i \in I_k^-$, *for some* $k$. *Then* $(x^k, y^k)$ *is optimal for* $P_i$.

Note that $(x^k, y^k)$ is not necessarily optimal for $P_j$, for $j \in I_k^+ \setminus I_k^-$. We now combine Proposition 2.1 with Theorem 2.3 to construct our inner approximation.

**Theorem 2.2.** *Suppose Algorithm 2 is run on problem P, terminating after* $\ell \leq n$ *steps. Let* $(x^k, y^k), z_k, I_k^+, I_k^-, \forall\ k = 1, \ldots, \ell$ *and* $I_\infty$ *be obtained from the algorithm. Then*

$$C^- = \left\{ c \geq c^* : \sum_{i \in I_1^- \cup \cdots \cup I_k^-} c_i \leq z^* - z_k + \sum_{i \in I_1^- \cup \cdots \cup I_k^-} c_i^*, \forall\ k = 1, \ldots, \ell \right\} \tag{3}$$

*is an inner approximation of $C$.*

It is important to note that the stability region $C$ depends only on $(x^*, y^*)$ and $h(y)$, and not on $c^*$. However, the inner approximation we calculate using Algorithm 2 does depend on $c^*$, because $c^*$ appears in the right-hand side of each inequality and also because different starting costs may determine different solutions in the algorithm when we add the inequality $\sum_{i \in I} x_i \geq 1$. So any $c^* \in C$ can be used in Algorithm 2 to obtain a possibly different inner approximation.

We next address the quality of approximation of the inner and outer regions $C^-, C^+$ generated by Algorithm 2. For any $c \in C^+ \setminus C^-$, we are unable to determine the optimality of $(x^*, y^*)$ for $P(c)$ without re-optimizing. Ideally, we would like to estimate the volume of this uncertainty region $C^+ \setminus C^-$, and perhaps compare it to the volume of $C^-$ and $C^+$. However, even for problems with modest dimension we cannot efficiently estimate this volume.

In light of these computational difficulties, we have developed a "shooting" experiment to give some idea of the relative sizes of $C^-$, $C^+$ and $C^+ \setminus C^-$. Starting from the original objective vector $c^*$, we generate a random direction $d$ by uniformly sampling each component from $[0, 1]$. We then compute the quantities

$$\lambda^- = \max_{\lambda} \left\{ c^* + \lambda \frac{d}{\|d\|} \in C^- \right\} \qquad \lambda^+ = \max_{\lambda} \left\{ c^* + \lambda \frac{d}{\|d\|} \in C^+ \right\}, \tag{4}$$

and use the values to compare the relative sizes of $C^-$ and $C^+$ in the direction $d$.

We performed the shooting experiment on the problem instances contained in MIPLIB 3.0. For a given instance, we considered as $x$ variables all binary variables that satisfied the following conditions:

- The variable is not fixed to its optimal value in all feasible solutions. In terms of Algorithm 2, this means the variable does not belong to the set $I_\infty$.
- There is no alternate optimal solution with the variable set to its complement. That is, we have $z^* > z_j$.

We refer to such binary variables as "active." If a particular MIPLIB instance did not have any active binary variables, we discarded it. We also skipped problems with more than 5,000 binary variables and problems which did not solve to optimality within one hour of CPU time. All computational experiments were carried out on a system with two 2.4 GHz Xeon processors and 2 GB RAM, and using CPLEX 11.1 (with default settings) as the optimization engine. For each instance, we generated 100,000 direction vectors $d$.

Table 2 contains average results for our experiments. For each instance, we report the total number of decision variables (# vars), the total number of binary variables (# bin), the number of active binary variables (# active), the Euclidean norm of the cost vector corresponding to the active variables ($\|c^*\|$), the average $\lambda^-$ and $\lambda^+$ values, and their ratio. For example, for problem `lseu`, 85 of 89 decision variables are active. In the directions tested, $C^-$ occupies 87% of the volume of $C^+$, but both regions allow only for a relatively small change in $c^*$, since $\|c^*\|$ is significantly larger than $\lambda^-$ and $\lambda^+$.

As Table 2 indicates, the estimated volume ratio of $C^-$ and $C^+$ varies significantly from one instance to the next. On one end, Algorithm 2 delivers a fairly tight approximation for problems `dcmulti` (92%), `enigma` (95%), and `lseu` (87%), to name a few. In fact, Algorithm 2 even delivers the exact stability region for `p0033`. On the other, the volume ratio is very small on instances such as `p2756` (7%), `vpm1` (3%) and `vpm2` (14%). This discrepancy is certainly in part due to differences in the number of active binary variables (14 in `enigma` and 2,391 in `p2756`,) since volume differences

tend to grow as dimension grows. However, part of this discrepancy is also instance dependent, as some problems with similar number of active variables have very different ratios.

Overall, the experimental results indicate that the volume of $C^+ \setminus C^-$ is significant for some instances.

Therefore, we next address the cause of this discrepancy: Are we under-approximating $C^-$ or over-approximating $C^+$? To answer this question, we performed the following additional experiment. For each of the first 1000 random directions $d$ that yield $\lambda^- < \lambda^+$, we construct a new cost vector

$$c_\pm = c^* + \frac{\lambda^- + \lambda^+}{2} \frac{d}{\|d\|} \in C^+ \setminus C^-,$$

and re-optimize the problem with this new objective. We then count the number of times the original optimal solution $(x^*, y^*)$ remains optimal for the new cost vector $c_\pm$. The next column in Table 2 (OIO, for "original is optimal") reports these counts for all instances except `p0033`. With the exception of `dsbmip`, `rentacar` and `rgn`, the original solution is optimal most of the time, with most instances recording counts above 800 and many getting 1000. These results indicate that the uncertainty region $C^+ \setminus C^-$ is caused primarily by an excessive under-approximation of $C^-$. They also suggest that $(x^*, y^*)$ is likely to remain optimal inside $C^+$, even when this fact cannot be guaranteed.

| Problem | # vars | # bin | # active | $\|c^*\|$ | $\lambda^-$ (avg.) | $\lambda^+$ (avg.) | $\frac{\lambda^-(\text{avg.})}{\lambda^+(\text{avg.})}$ | OIO (cnt.) |
|---|---|---|---|---|---|---|---|---|
| bell3a | 133 | 39 | 31 | 1.553E+05 | 1.129E+05 | 1.430E+05 | 0.79 | 724 |
| bell5 | 104 | 30 | 20 | 1.609E+05 | 8947 | 25875 | 0.35 | 1000 |
| blend2 | 353 | 231 | 215 | 45.98 | 0.08 | 0.18 | 0.43 | 1000 |
| dcmulti | 548 | 75 | 71 | 4220 | 16.27 | 17.75 | 0.92 | 990 |
| dsbmip | 1886 | 160 | 14 | 0 | 0.48 | 0.62 | 0.78 | 0 |
| egout | 141 | 55 | 28 | 115 | 3.21 | 7.76 | 0.41 | 1000 |
| enigma | 100 | 100 | 14 | 1 | 0.31 | 0.32 | 0.95 | 1000 |
| fiber | 1298 | 1254 | 1239 | 3.167E+06 | 762 | 809 | 0.94 | 1000 |
| fixnet6 | 878 | 378 | 378 | 5606 | 1.98 | 2.28 | 0.87 | 1000 |
| gen | 870 | 144 | 100 | 1670 | 17.51 | 24.45 | 0.72 | 990 |
| gesa2_o | 1224 | 384 | 383 | 7.623E+06 | 363 | 916 | 0.40 | 1000 |
| gesa2 | 1224 | 240 | 240 | 7.307E+05 | 1720 | 2985 | 0.58 | 656 |
| gesa3_o | 1152 | 336 | 336 | 7.616E+06 | 1934 | 70313 | 0.03 | 997 |
| gesa3 | 1152 | 216 | 216 | 6.487E+05 | 1.104E+04 | 2.488E+05 | 0.04 | 1000 |
| gt2 | 188 | 24 | 12 | 1.302E+04 | 1180 | 4906 | 0.24 | 1000 |
| khb05250 | 1350 | 24 | 24 | 1.225E+07 | 3.317E+05 | 6.482E+05 | 0.51 | 899 |
| l152lav | 1989 | 1989 | 1988 | 8800 | 3.84 | 10.04 | 0.38 | 1000 |
| lseu | 89 | 89 | 85 | 1941 | 4.55 | 5.22 | 0.87 | 1000 |
| manna81 | 3321 | 18 | 16 | 4 | 0.29 | 2.44 | 0.12 | 1000 |
| mas76 | 151 | 150 | 150 | 1.200E-04 | 90.91 | 159.67 | 0.57 | 1000 |
| misc03 | 160 | 159 | 100 | 0 | 101 | 354 | 0.29 | 1000 |
| misc06 | 1808 | 112 | 112 | 0 | 0.96 | 1.48 | 0.65 | 1000 |
| mod008 | 319 | 319 | 311 | 1316 | 2.83 | 7.52 | 0.38 | 1000 |
| mod010 | 2655 | 2655 | 2433 | 9350 | 1.40 | 4.25 | 0.33 | 1000 |
| mod011 | 10958 | 96 | 96 | 1.400E+07 | 1.211E+05 | 3.738E+05 | 0.32 | 1000 |
| modglob | 422 | 98 | 98 | 1.292E+05 | 1753 | 12852 | 0.14 | 1000 |
| p0033 | 33 | 33 | 21 | 945 | 3.61 | 3.61 | 1.00 | - |
| p0201 | 201 | 201 | 131 | 5712 | 18.02 | 54.18 | 0.33 | 1000 |
| p0282 | 282 | 282 | 282 | 2.975E+05 | 8.92 | 20.11 | 0.44 | 1000 |
| p0548 | 548 | 548 | 484 | 1.420E+04 | 5.30 | 15.27 | 0.35 | 978 |
| p2756 | 2756 | 2756 | 2391 | 3.417E+04 | 0.35 | 5.19 | 0.07 | 903 |
| pk1 | 86 | 55 | 55 | 0 | 0.30 | 0.43 | 0.69 | 1000 |
| pp08a | 240 | 64 | 64 | 2437 | 11.18 | 15.59 | 0.72 | 1000 |
| pp08aCUTS | 240 | 64 | 64 | 2437 | 11.18 | 15.59 | 0.72 | 1000 |
| qnet1_o | 1541 | 1288 | 1260 | 0 | 0 | 8.03 | 0.00 | 1000 |
| qnet1 | 1541 | 1288 | 1260 | 0 | 7.79 | 8.03 | 0.97 | 1000 |
| rentacar | 9557 | 55 | 22 | 0 | 6.020E+04 | 6.098E+04 | 0.99 | 0 |
| rgn | 180 | 100 | 60 | 0 | 5.67 | 7.93 | 0.71 | 0 |
| set1ch | 712 | 240 | 235 | 1.094E+04 | 8.52 | 11.73 | 0.73 | 1000 |
| vpm1 | 378 | 168 | 114 | 10.68 | 0.11 | 4.33 | 0.03 | 960 |
| vpm2 | 378 | 168 | 136 | 10.21 | 0.08 | 0.56 | 0.14 | 834 |

Table 2: Shooting experiment results for MIPLIB 3.0 instances.

For further details on the approach and computational experience with the algorithms see:

F. Kılınç-Karzan, A. Toreillo, S. Ahmed, G.L. Nemhauser, M.W.P. Savelsbergh. "Approximating the Stability Region for Binary Mixed-Integer Programs." Submitted to Operations Research Letters (2008).

## 3 Information-based Branching for Integer Programming

Branch-and-bound (*B&B*) empowered with advanced features such as presolve, cuts, heuristics and strong branching is the preferred algorithm for solving Mixed-Integer Linear Programming (MIP) problems. We focus on branching. A subproblem corresponding to a node in a *B&B*-tree is *fathomed* when we are able to certify that it has been fully explored, that is, all feasible solutions to it have been explicitly or implicitly visited; otherwise, it's referred to as an *open* subproblem. In a *B&B*-scheme, open subproblems are generated in a rigid way, depending on the branching rule. In fact, every open subproblem is restricted to be identical to a previously examined subproblem except for its last chronologically branched variable. Therefore, inappropriate branchings performed at the beginning can jeopardize the effectiveness of the branch-and-bound method.

Changing the policy for branching variable selection can have a dramatic effect on the overall time needed to solve a problem. Most existing branching variable selection methods either estimate the impact of the candidate variables on the objective function of the LP relaxation or provide bounds on the degradation. The candidate variable having the greatest estimated impact is then chosen for branching. The motivation is to select the branching variable that maximizes the degradation of the objective function value at the optimal solution of the child node LP relaxation, which gives a tighter bound on the unsolved nodes.

The "information-based variable selection methods" we have developed use a variety of means to estimate the impact of each candidate variable on fathoming based on previously collected information. These new methods recognize that the branching decisions made at the top of the tree are the most important ones. In order to make more informed decisions at the top, first a traditional *B&B*is started and after a certain amount of information is collected from the fathomed nodes, the process is halted. A restart of the *B&B*empowered by exploiting the information contained in a set of previously fathomed subproblems is performed. In contrast to the current branching variable selection methods based on the degradation estimate in the objective function value, we favor the variables that have the most impact on the fathoming of the child nodes. The general idea is to arrive at child nodes which are closer to being fathomed, in the hope that one or both of the child nodes will never be expanded. Note that using the information derived directly from the previous search nodes, as in backjumping or learning *B&B*, may be ineffective, as they have been "spoiled" by inappropriate branchings earlier in the process. Instead, in our approach, we strengthen the information on fathomed subproblems by eliminating the unnecessary branching decisions which had no effect on their fathoming. In addition to new branching methods, we use this information in propagation and in the generation of valid inequalities.

We consider the MIP problem

$$\min \left\{ c^{\mathrm{T}} x + d^{\mathrm{T}} y \;\mid\; Ax + By \geqslant b,\; x \in \{0,1\}^n,\; y \in \mathbb{R}^k_+ \right\} \tag{P}$$

where $c \in \mathbb{R}^n$, $d \in \mathbb{R}^k$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times k}$. Its LP relaxation is obtained by replacing $x \in \{0,1\}^n$ by $0 \leq x \leq 1$.

Recall that in a *B&B*-tree, or for short a tree, a node can be fathomed in three ways: (i) the node LP-relaxation is infeasible, (ii) the optimal solution to the node LP-relaxation is integer, and (iii) the optimum value of the node LP-relaxation is no better than the objective value of the

currently best known integer feasible solution. To formalize the notion of fathoming, we use the following notation.

We say $f_j^l$ is the *fixing* of the variable $j$ to the value $l \in \{0,1\}$. Let $N_0$ denote the root node, $N_i$ denote a node in the tree and $C_i = C_i^0 \cup C_i^1$ be the set of binary variables fixed at node $N_i$, where $C_i^0$ ($C_i^1$) denotes the indices of binary variables fixed to 0 (1). Without loss of generality, we assume that $C_0 = \emptyset$.

**Definition 1.** *If $N_i$ is a fathomed node of a binary tree, then $C_i$ is called a clause corresponding to node $N_i$. When $|C_i| = n$, the clause is called trivial.*

**Definition 2.** *Let $C$ be a clause. If $C \setminus f_j^l$ is not a clause for any $f_j^l \in C$, then $C$ is called a minimal clause.*

If $C$ is a minimal clause, any node $N_i$ in the tree with $C_i \subset C$ cannot be fathomed by any of the fathoming rules. Moreover any node $N_i$ with $C_i \supseteq C$ can be fathomed together with the subtree rooted at $N_i$. Note that there exists a minimal clause (not necessarily unique) associated with each fathomed node in any binary tree.

Given a clause $C = C^0 \cup C^1$, the inequality

$$\sum_{i \in C^0} x_i + \sum_{i \in C^1} (1 - x_i) \geq 1 \tag{5}$$

eliminates all solutions that do not comply with it. Clearly (5) might cut off some feasible solutions. However, the region cut off by (5) is guaranteed not to contain any feasible solution with an objective value better than the optimal objective value. Note that (5) is a generalized cover inequality and hence by obtaining minimal clauses, we actually derive a minimal generalized cover inequality for (P).

We wish to efficiently identify the most useful clauses and use them effectively in *B&B*by exploring the restart framework. Our approach is mainly based on deriving information in the form of clauses from the fathomed nodes of a partial tree. We employ this information in guiding the search through designing advanced preprocessing, propagation and branching schemes as well as in generating valid inequalities of the form (5). It is quite likely that the clauses obtained from a partial tree are not minimal. So we strengthen the information on fathomed nodes by eliminating the unnecessary fixings which had no effect on fathoming. We do this by solving a MIP model that obtains a minimal clause from a given clause.

Next, we present a model that can be used to generate a minimal clause of minimum cardinality. Without loss of generality, we assume that the upper and lower bounds of the binary variables are also included in the original formulation as constraints. Let $v^*$ be the objective value of the optimum solution to (P). We can fathom any node of the tree which is either infeasible or has an objective value greater than $v^*$. Fathoming by integrality is quite infrequent in practice and given $v^*$, we can simply fathom all nodes with objective value greater than or equal to $v^*$, which includes fathoming of all integer solutions as well.

Consider a leaf node of the tree that is fathomed by bound and denote the corresponding set of variable fixings by $C = C^0 \cup C^1$. The LP relaxation of this leaf node is

$$\begin{aligned}
&\min c^T x + d^T y \\
&\text{s.t. } Ax + By \geq b \\
&\qquad x_i \geq 1 \qquad \forall i \in C^1 \\
&\qquad -x_i \geq 0 \qquad \forall i \in C^0 \\
&\qquad x \in \mathbb{R}^n_+, \quad y \in \mathbb{R}^k_+
\end{aligned}$$

Define the following variables:

$$z_i^0 = \begin{cases} 1, & \text{if inequality } -x_i \geq 0 \text{ is added to the LP relaxation;} \\ 0, & \text{otherwise,} \end{cases}$$

and

$$z_i^1 = \begin{cases} 1, & \text{if inequality } x_i \geq 1 \text{ is added to the LP relaxation;} \\ 0, & \text{otherwise.} \end{cases}$$

Using these new variables, the following MIP

$$\begin{aligned} \min\ & c^T x + d^T y \\ \text{s.t. } & Ax + By \geq b \\ & z_i^1 x_i \geq z_i^1 \qquad \forall i \in C^1 \\ & -z_i^0 x_i \geq 0 \qquad \forall i \in C^0 \\ & x \in \mathbb{R}^n_+, \quad y \in \mathbb{R}^k_+ \\ & z_i^0, z_i^1 \in \{0,1\}. \end{aligned}$$

is equivalent to the node LP when all of the binaries in it are set to 1.

We want to find a minimum number of bound inequalities such that their inclusion in the original linear programming relaxation of (P) will still lead to a fathoming, i.e., either the LP relaxation is infeasible or the objective value exceeds the lower bound value, $v^*$. Since the original root LP relaxation is assumed to be feasible and bounded from below, its dual is also feasible. Thus we know that in the case of infeasibility of the node LP, the dual of the node LP is unbounded since when we add new bound inequalities to the primal, we are just adding new columns to the dual of the node LP. Therefore when a node is fathomed by infeasibility, we will be able to find a dual solution with an objective value strictly greater than $v^*$. So now we consider the dual of the above formulation by treating the variables $z_i^0$ and $z_i^1$ as parameters and we obtain

$$\begin{aligned} \max\ & \lambda^T b + \sum_{i \in C^1} \gamma_i^1 z_i^1 \\ \text{s.t. } & \lambda^T A_i + \gamma_i^1 z_i^1 \leq c_i \qquad \forall i \in C^1 \\ & \lambda^T A_i - \gamma_i^0 z_i^0 \leq c_i \qquad \forall i \in C^0 \\ & \lambda^T A_i \leq c_i \qquad \forall i \in \{1, \ldots, n\} \setminus (C^0 \cup C^1) \\ & \lambda^T B_j \leq d_j \qquad \forall j \in \{1, \ldots, k\} \\ & \lambda_l \geq 0 \qquad \forall l \in \{1, \ldots, m\} \\ & \gamma_i^0, \gamma_i^1 \geq 0. \end{aligned}$$

Since we are only interested in the existence of dual solutions with objective value greater than or equal to $v^*$, we can equivalently turn the objective into a constraint. By considering $z_i^0$ and $z_i^1$ as binary variables with the condition that at most one of them can be set to 1 for each $i$ (since $C^0 \cap C^1 = \emptyset$, we don't need to include these constraints explicitly), we obtain the following

formulation where we minimize the number number of $z_i^0$ and $z_i^1$:

$$\begin{aligned}
\min\ & \sum_{i\in C^0} z_i^0 + \sum_{i\in C^1} z_i^1 \\
\text{s.t. } & \lambda^T b + \sum_{i\in C^1} \gamma_i^1 z_i^1 \geq v^* \\
& \lambda^T A_i + \gamma_i^1 z_i^1 \leq c_i \qquad \forall i \in C^1 \\
& \lambda^T A_i - \gamma_i^0 z_i^0 \leq c_i \qquad \forall i \in C^0 \\
& \lambda^T A_i \leq c_i \qquad \forall i \in \{1,\ldots,n\} \setminus (C^0 \cup C^1) \\
& \lambda^T B_j \leq d_j \qquad \forall j \in \{1,\ldots,k\} \\
& \lambda_l \geq 0 \qquad \forall l \in \{1,\ldots,m\} \\
& z_i^0, z_i^1 \in \{0,1\}, \quad \gamma_i^0, \gamma_i^1 \geq 0.
\end{aligned}$$

In order to bound the dual variables $\gamma$ and $\lambda$ from above by 1, we introduce another variable $\alpha$ and also to linearize the model, we introduce $u_i^0$ and $u_i^1$ for the nonlinear terms $\gamma_i^0 z_i^0$ and $\gamma_i^1 z_i^1$ respectively:

$$\begin{aligned}
\min\ & \sum_{i\in C^0} z_i^0 + \sum_{i\in C^1} z_i^1 \\
\text{s.t. } & \lambda^T b + \sum_{i\in C^1} u_i^1 - \alpha v^* \geq 0 \\
& \lambda^T A_i + u_i^1 - \alpha c_i \leq 0 \qquad \forall i \in C^1 \\
& \lambda^T A_i - u_i^0 - \alpha c_i \leq 0 \qquad \forall i \in C^0 \\
& \lambda^T A_i - \alpha c_i \leq 0 \qquad \forall i \in \{1,\ldots,n\} \setminus (C^0 \cup C^1) \\
& \lambda^T B_j - \alpha d_j \leq 0 \qquad \forall j \in \{1,\ldots,k\} \\
& u_i^0 \leq \gamma_i^0, \quad u_i^0 \leq z_i^0, \quad u_i^0 - \gamma_i^0 - z_i^0 \geq -1 \qquad \forall i \in C^0 \\
& u_i^1 \leq \gamma_i^1, \quad u_i^1 \leq z_i^1, \quad u_i^1 - \gamma_i^1 - z_i^1 \geq -1 \qquad \forall i \in C^1 \\
& 0 \leq \lambda_l \leq 1 \qquad \forall l \in \{1,\ldots,m\} \\
& z_i^0, z_i^1 \in \{0,1\}, \quad 0 \leq u_i^0, u_i^1, \gamma_i^0, \gamma_i^1 \leq 1 \\
& 0 < \alpha.
\end{aligned}$$

Note that the fixing of the last variable on the path from root node to a leaf node is the main cause of fathoming done at the leaf node. Hence in any feasible solution to the above formulation, we will always have the corresponding binary variable ($z_i^0$ or $z_i^1$) fixed to 1. In our computations, we take advantage of this fact by actually fixing the value of the last branched variable in this MIP and we replace $\alpha > 0$ with $\alpha \geq 10^{-6}$.

Let $\mathcal{C} = \{C_1, ..., C_K\}$ be a set of clauses in a partial tree. Consider a node of the tree $\tilde{N}$ other than the root node, and let $\tilde{C}^0(\tilde{C}^1)$ denote the set of binary variables fixed to 0 (1). We say a clause $C = C^0 \cup C^1$ is *active* at $\tilde{N}$ if $C^0 \cap \tilde{C}^1 = \emptyset$ and $C^1 \cap \tilde{C}^0 = \emptyset$, i.e. if it's possible to obtain a child node (not necessarily immediate) from the current node that can be fathomed by the clause $C$. Whenever a clause becomes inactive for a particular node, it will remain inactive for all the child nodes of that node. Since some variables are already fixed at $\tilde{N}$, the active clauses can be updated using this information, i.e., the clause $C = (C^0 \setminus \tilde{C}^0) \cup (C^1 \setminus \tilde{C}^1)$ indicates the possible extension of the current node to a child node which can be fathomed by clause $C$. In the rest of the text whenever we refer to an active clause at a node, we actually refer to the updated version of the clause.

Whenever an active clause $C$ has only one variable, i.e., $|C| = 1$, we can immediately fix the value of that variable. Suppose $C = C^0 = \{j\}$, then we can set $x_j = 1$ and create only a single child node, as the other branch will automatically be fathomed by the clause. We refer to this as *propagation*.

Given a node and a set of active clauses at that node, there are several ways to use this information in determining a branching variable. Let $\tilde{x}$ be the vector of current LP relaxation values of variables at the node. We first weight each active clause, denoted by $\omega(C_i)$, to estimate its importance in fathoming. We have tested four different alternatives to estimate $\omega(C_i)$:

*i)* $\omega(C_i) = 1$ for all clauses $C_i$ (i.e. all clauses are of equal importance),

*ii)* $\omega(C_i) = \frac{1}{|C_i|}$ where $|C_i|$ is the number of variables included in clause $C_i$ (i.e. short clauses are preferred),

*iii)* $\omega(C_i) = \frac{1}{\sum_{j \in C_i^0} \tilde{x}_j + \sum_{j \in C_i^1} (1 - \tilde{x}_j) - 1}$ where we look at the possible closeness to violation of the clause inequality (5),

*iv)* $\omega(C_i) = 2^{-|C_i|}$ (exponentially higher preference given to the shorter clauses).

Then using the weights of the clauses, we estimate the effectiveness of fixing the binary variable $x_j$ to 0 (1), denoted by $\beta_j^0$ ($\beta_j^1$). We have tested two alternatives to estimate the overall effect of creating a branch with $x_j = 0(1)$:

*i)* $\beta_j^0 = \sum_{i: j \in C_i^0} \omega(C_i)$ and $\beta_j^1 = \sum_{i: j \in C_i^1} \omega(C_i)$,

*ii)* $\beta_j^0 = \max\{\omega(C_i) : j \in C_i^0\}$ and $\beta_j^1 = \max\{\omega(C_i) : j \in C_i^1\}$.

Let $\beta_j$ denote the overall effect of a branching on variable $x_j$. To estimate $\beta_j$, we need to combine $\beta_j^0$ and $\beta_j^1$. Inspired by currently used branching rules, we suggest and test the following alternatives:

*i)* $\beta_j = \min\{\tilde{x}_j, 1 - \tilde{x}_j\} * (\beta_j^0 + \beta_j^1)$,

*ii)* $\beta_j = \beta_j^0 + \beta_j^1$,

*iii)* $\beta_j = \max\{\beta_j^0, \beta_j^1\} + 10 * \min\{\beta_j^0, \beta_j^1\}$.

Note that the first alternative considers the fractionality of the variable, whereas the second one simply adds the individual effects and the third one is similar to the weights used in strong branching.

In Table 3, we report computational results for a set of difficult instances from MIPLIB. We provide information about the collection phase (solve time, number of clauses collected and the average size), the improvement phase (minimum, average, and maximum solve time, number of clauses improved, average size after improvement), and the solve phase (number of nodes and solution time *CPLEX*, information-based search with basic clauses *Basic*, and information-based search with improved clauses *Improved*. If we compare the node counts for CPLEX and *Improved*, we see that in all of the instances the node counts decrease and the improvements in harder problems are more significant (for `mas74` reducing from 1656970 nodes to 1167900 nodes, for `qiu` from 15862 to 5139, and for `rout` from 45051 to 10933). These improvements in node counts also generally lead to improvements in solution times.

| Problem | | Collection phase | # | size | Improvement Phase (min) | (max) | (total) | # | size | *CPLEX* | *Basic* | *Improved* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| gesa2_o | # nodes | 5395 | 9 | 5.22 | 0 | 0 | 0 | 1 | 4.89 | 6630 | 5484 | 5733 |
| | time | 96.04 | | | 0.91 | 1.06 | 8.57 | | | 101.54 | 96.36 | 100.9 |
| mas74 | # nodes | 1254 | 200 | 26.29 | 0 | 299 | 4784 | 134 | 19.25 | 1656970 | 1288974 | 1167900 |
| | time | 5 | | | 0.03 | 0.43 | 21.76 | | | 2898.87 | 2547 | 2510.7 |
| mas76 | # nodes | 1036 | 200 | 22.97 | 0 | 327 | 2236 | 128 | 13.95 | 150624 | 116565 | 136133 |
| | time | 2.69 | | | 0.01 | 0.28 | 10.94 | | | 171.52 | 158.69 | 191.89 |
| misc07 | # nodes | 578 | 200 | 23.65 | 0 | 150 | 1018 | 142 | 10.59 | 9237 | 7693 | 7204 |
| | time | 20.07 | | | 0.07 | 0.95 | 46.15 | | | 224.25 | 211.79 | 191.39 |
| mod011 | # nodes | 581 | 200 | 17.8 | 0 | 0 | 0 | 0 | 17.8 | 1286 | 1112 | 1112 |
| | time | 1091.19 | - | | 54.96 | 60.83 | 11987.05 | | | 1817.07 | 1720.73 | 1720.73 |
| pk1 | # nodes | 1359 | 200 | 29.72 | 0 | 491 | 4986 | 116 | 19.25 | 99589 | 74349 | 79829 |
| | time | 6.57 | | | 0.01 | 0.47 | 17.76 | | | 254.99 | 235.71 | 252.47 |
| qiu | # nodes | 1101 | 200 | 15.63 | 0 | 2073 | 15230 | 108 | 12.69 | 15862 | 10227 | 5139 |
| | time | 218.74 | | | 0.66 | 51.65 | 812.82 | | | 1956.93 | 1444.39 | 762.43 |
| rout | # nodes | 17213 | 26 | 7.04 | 0 | 1 | 6 | 9 | 5.27 | 45051 | 29861 | 10933 |
| | time | 268.89 | | | 0.05 | 0.29 | 2.53 | | | 567.64 | 464.96 | 167.86 |
| stein45 | # nodes | 1571 | 200 | 15.54 | 0 | 8 | 11 | 0 | 15.54 | 21429 | 18969 | 18969 |
| | time | 14.29 | | | 0 | 0.04 | 3.65 | | | 97.57 | 94.37 | 94.37 |

Table 3: Results for difficult instances

For further details on the approach and additional computational results, see:

F. Kılınç-Karzan, G.L. Nemhauser, M.W.P. Savelsbergh. "Information Based Branching Rules for Binary Mixed Integer Problems." Submitted to INFORMS J. on Computing (2009).