Exploiting Cyberinfrastructure to Solve Real-Time Integer Programs

Shabbir Ahmed, George L. Nemhauser, Martin W.P. Savelsbergh
H. Milton Stewart Industrial and Systems Engineering
Georgia Institute of Technology
765 Ferst Drive, NW, Atlanta, Georgia 30332-0205, USA,

NSF Grant # DMI 0521953

Final Report

1. Approximating the Stability Region for Binary Mixed-Integer Programs

The *stability region* of a solution is the polyhedral set of objective coefficients for which the solution is optimal. It provides valuable information for sensitivity analysis and re-optimization. An exact description of it may require an exponential number of inequalities. We develop polyhedral inner and outer approximations of linear size.

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 $c^*x = \sum_{i \in N} c_i^* x_i$, $N = \{1,\dots,n\}$ and $h : \text{proj}_y(X) \to \mathbb{R}$. We do not give specific requirements on h and X, but we assume that an optimal solution (x^*, y^*) exists and can be computed. We are interested in the stability of (x^*, y^*) with respect to variations of the cost vector c^*. By possibly complementing variables, we assume without loss of generality that $x^* = 0$, so the optimal value of P is $z^* = h(y^*)$.

Definition 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, $C = \{c \in \mathbb{R}^n : c(x - x^*) \le h(y^*) - h(y), \forall\ (x,y) \in X\}$.*

A familiar example of a stability region comes from linear optimization. In a linear program, the stability region of a solution is the cone generated by the set of constraints that are binding at the solution.

Remark 1. *Let $(\hat{x}, \hat{y})$ be feasible for P, with objective value $\hat{z} = c^*\hat{x} + h(\hat{y}) \le z^*$. 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. *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 $C = \{c \in \mathbb{R}^n : cx \le h(y^*) - v(x), \forall\ x \in \{0,1\}^n\}$.*

Remark 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. *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 $C^- \subseteq C$.*

Let $(\hat{x}, \hat{y})$ be feasible for P. By Remark 1, the set $\{c \in \mathbb{R}^n : c\hat{x} \leq z^* - h(\hat{y})\}$ is an outer approximation of C. The set $\{c \in \mathbb{R}^n : c \leq c^*\}$ is a trivial inner approximation of C. The following are two simple but important consequences of Definition 2 that help us obtain small outer approximations and large inner approximations.

Proposition 1.

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, $\operatorname{conv}(C_1^- \cup C_2^-)$ 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. If P_j is infeasible for some $j \in N$, then every feasible solution to P has $x_j = 0$. Thus, c_j cannot in any way affect optimality, and we can ignore it. 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. An important question is whether solving the problems P_j is necessary to obtain the z_j values, or if a more efficient method exists. This is especially pertinent if P is NP-hard, because each P_j is then NP-hard as well.

The following observation follows directly from Remark 1 and Proposition 1.

Remark 3. *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 3 satisfies

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

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 γ_j.

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

Proposition 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 1. *Suppose we order the x variables so that $z^* \geq z_1 \geq z_2 \geq \cdots \geq z_n$ holds. Then*

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

is an inner approximation of C.

Algorithm 1 Binary Solution Cover

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

  Set $(x^0, y^0) \leftarrow (x^*, y^*)$, $z_0 \leftarrow z^*$, $I \leftarrow N$, $I_\infty \leftarrow \emptyset$.
  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
```

Corollary 1. *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 1.

Algorithm 1 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 1 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 $I_k^+ \subseteq N$ indicates which x-variables have value one in (x^k, y^k). The set $I_k^- \subseteq I_k^+$ indicates which of these variables have value one for the first time.

An outer approximation is easily obtained applying Remark 1 to each solution (x^k, y^k). To determine an inner approximation, we must first establish a preliminary fact.

Proposition 3. *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 3 with Theorem 1 to construct our inner approximation.

Theorem 2. *Suppose Algorithm 1 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_∞ be obtained from the algorithm. Then*

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

is an inner approximation of C.

We can also apply Corollary 1 to get an inner approximation that is not restricted to values greater than or equal to 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 1 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 1 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 1. 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. And although volume computation and general integration by sophisticated random walk techniques is an active area of research, no practical volume computation algorithm has yet been developed.

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

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

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 1, this means the variable does not belong to the set I_∞.
- 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 1 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 λ^- and λ^+ 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 λ^- and λ^+.

As Table 1 indicates, the estimated volume ratio of C^- and C^+ varies significantly from one instance to the next. On one end, Algorithm 1 delivers a fairly tight approximation for problems `dcmulti` (92%), `enigma` (95%), and `lseu` (87%), to name a few. In fact, Algorithm 1 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 numbers 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 1 (OIO, for "original is optimal") reports these counts for all instances except `p0033`, which did not have any directions along which $\lambda^- < \lambda^+$. 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.

The last experiment also suggests an additional option to explore when (x^*, y^*) is not optimal for the new objective. Algorithm 1 generates a list of solutions that in some sense "surrounds" the original. Therefore, if (x^*, y^*) is not optimal for an objective, one of the solutions from the list may in fact be optimal instead. To test this hypothesis, we designed a final set of experiments. For the first 1000 random directions d with $\lambda^- < \lambda^+$, define $c_\pm$ as before and also define

$$c_+ = c^* + \frac{\lambda^- + 3\lambda^+}{2}\frac{d}{\|d\|} \in \mathbb{R}^n \setminus C^+.$$

Note that just as $c_\pm$ is guaranteed to be in $C^+ \setminus C^-$, c_+ is guaranteed to be in $\mathbb{R}^n \setminus C^+$, and the distance along d from c_+ to C^+ equals the distance from $c_\pm$ to C^-. We re-optimize the problem with both of these new objectives, and count the number of times the best solution from the list is optimal for each objective. In addition, when none of the solutions is optimal, we calculate the relative gap between the best solution in the list and the new optimal value. The final four columns of Table 1 summarize the results, reporting both the number of times the best solution is optimal (BIO) and the average relative difference in objective values (ROD) for non-optimal cases, defined as $\frac{|\text{optimal}-\text{best}|}{|\text{optimal}|}$.

With the exception of `bell3a` and `vpm1`, the solution list yields good solutions for the new objective vectors $c_\pm$ and c_+. For c_+, even when none of the solutions is optimal, on average the relative gap is below 2% and significantly below for many instances. Thus, the solutions generated by Algorithm 1 provide an excellent warm start for re-solves, and they can also be used as high-quality heuristic solutions when re-optimization is not possible.

								$c_\pm \in C^+ \setminus C^-$			$c_+ \in \mathbb{R}^n \setminus C^+$	
Problem	# vars	# bin	# active	$\lVert c^* \rVert$	λ^- (avg.)	λ^+ (avg.)	$\frac{\lambda^-(\text{avg.})}{\lambda^+(\text{avg.})}$	OIO (cnt.)	BIO (cnt.)	ROD (avg.)	BIO (cnt.)	ROD (avg.)
`bell3a`	133	39	31	1.553E+05	1.129E+05	1.430E+05	0.79	724	724	2.243E-02	0	11.784
`bell5`	104	30	20	1.609E+05	8947	25875	0.35	1000	1000	-	680	7.174E-04
`blend2`	353	231	215	45.98	0.08	0.18	0.43	1000	1000	-	1000	-
`dcmulti`	548	75	71	4220	16.27	17.75	0.92	990	990	8.082E-06	865	1.869E-05
`dsbmip`	1886	160	14	0	0.48	0.62	0.78	0	1000	-	1000	-
`egout`	141	55	28	115	3.21	7.76	0.41	1000	1000	-	205	5.296E-03
`enigma`	100	100	14	1	0.31	0.32	0.95	1000	1000	-	1000	-
`fiber`	1298	1254	1239	3.167E+06	762	809	0.94	1000	1000	-	990	1.631E-04
`fixnet6`	878	378	378	5606	1.98	2.28	0.87	1000	1000	-	1000	-
`gen`	870	144	100	1670	17.51	24.45	0.72	990	1000	-	821	2.544E-05
`gesa2_o`	1224	384	383	7.623E+06	363	916	0.40	1000	1000	-	1000	-
`gesa2`	1224	240	240	7.307E+05	1720	2985	0.58	656	656	2.920E-06	64	2.247E-05
`gesa3_o`	1152	336	336	7.616E+06	1934	70313	0.03	997	997	5.000E-10	0	1.455E-03
`gesa3`	1152	216	216	6.487E+05	1.104E+04	2.488E+05	0.04	1000	1000	-	0	1.862E-02
`gt2`	188	24	12	1.302E+04	1180	4906	0.24	1000	1000	-	1000	-
`khb05250`	1350	24	24	1.225E+07	3.317E+05	6.482E+05	0.51	899	899	2.444E-04	106	1.931E-03
`l152lav`	1989	1989	1988	8800	3.84	10.04	0.38	1000	1000	-	39	1.216E-04
`lseu`	89	89	85	1941	4.55	5.22	0.87	1000	1000	-	998	2.667E-04
`manna81`	3321	18	16	4	0.29	2.44	0.12	1000	1000	-	2	1.139E-04
`mas76`	151	150	150	1.200E-04	90.91	159.67	0.57	1000	1000	-	961	3.055E-04
`misc03`	160	159	100	0	101	354	0.29	1000	1000	-	422	1.696E-02
`misc06`	1808	112	112	0	0.96	1.48	0.65	1000	1000	-	552	8.474E-06
`mod008`	319	319	311	1316	2.83	7.52	0.38	1000	1000	-	841	1.799E-03
`mod010`	2655	2655	2433	9350	1.40	4.25	0.33	1000	1000	-	645	7.996E-06
`mod011`	10958	96	96	1.400E+07	1.211E+05	3.738E+05	0.32	1000	1000	-	121	2.033E-03
`modglob`	422	98	98	1.292E+05	1753	12852	0.14	1000	1000	-	90	1.079E-04
`p0033`	33	33	21	945	3.61	3.61	1.00	-	-	-	-	-
`p0201`	201	201	131	5712	18.02	54.18	0.33	1000	1000	-	1000	-
`p0282`	282	282	282	2.975E+05	8.92	20.11	0.44	1000	1000	-	435	3.767E-06
`p0548`	548	548	484	1.420E+04	5.30	15.27	0.35	978	1000	-	1000	-
`p2756`	2756	2756	2391	3.417E+04	0.35	5.19	0.07	903	1000	-	10	1.219E-04
`pk1`	86	55	55	0	0.30	0.43	0.69	1000	1000	-	1000	-
`pp08a`	240	64	64	2437	11.18	15.59	0.72	1000	1000	-	485	4.077E-04
`pp08aCUTS`	240	64	64	2437	11.18	15.59	0.72	1000	1000	-	486	3.969E-04
`qnet1_o`	1541	1288	1260	0	0	8.03	0.00	1000	1000	-	996	2.697E-05
`qnet1`	1541	1288	1260	0	7.79	8.03	0.97	1000	1000	-	910	2.760E-05
`rentacar`	9557	55	22	0	6.020E+04	6.098E+04	0.99	0	1000	-	437	1.314E-04
`rgn`	180	100	60	0	5.67	7.93	0.71	0	1000	-	1000	-
`set1ch`	712	240	235	1.094E+04	8.52	11.73	0.73	1000	1000	-	854	1.509E-05
`vpm1`	378	168	114	10.68	0.11	4.33	0.03	960	960	4.330E-03	1	0.236
`vpm2`	378	168	136	10.21	0.08	0.56	0.14	834	1000	-	838	3.625E-03

Table 1: Shooting experiment results for MIPLIB 3.0 instances.

2. Integer Programming based Local Search

The Fixed Charge Network Flow (FCNF) problem 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. Many real-life instances of FCNF (or of instances of models that contain FCNF as a substructure) are very large - so much so that sometimes even the linear programming relaxation of the natural arc-based integer programming formulation can be intractable. In such situations, we can use a path-based integer programming formulation, but that necessitates the use of column generation techniques. Furthermore, the linear programming relaxation of the arc-formulation and the path-formulation have the same optimal value which is known to be weak. Hence, it may not just be difficult to find feasible solutions, it may also be challenging to determine the quality of these solutions. As a result, even though much research has been devoted to FCNF, exact methods are only capable of handling small instances, far smaller than many realistic-sized instance. Fortunately, in today's dynamic business environment getting high-quality solutions in a short amount of time is usually more important than getting provably optimal solutions. Hence the focus of our research is to develop a solution approach that produces *provably* high-quality solutions *quickly*.

Our solution approach relies heavily on linear and integer programming to take advantage of the power of commercially available linear and integer programming solvers. Algorithmically, we combine mathematical programming techniques with heuristic search techniques. More specifically, we develop

- a primal local search algorithm using neighborhoods that involve solving carefully chosen integer programs, and
- a scheme to generate dual bounds that involves strengthening the linear programming relaxation via cuts discovered while solving these integer programs.

Our approach tightly integrates the use of the arc-based formulation of FCNF and the path-based formulation of FCNF. It also incorporates randomization to diversify the search and learning to intensify the search.

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 Ghamlouch 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.

The key characteristics of our solution approach for FCNF, which differentiate it from existing heuristic approaches, are:

- it uses exact methods to find improving solutions,

- it generates both a primal solution and a dual bound at each iteration, and
- it uses both the arc and path formulations of FCNF to guide the search.

The following paper concerning this material has been accepted for publication in INFORMS J. on Computing.

Mike Hewitt, George L. Nemhauser, and Martin W.P. Savelsbergh. "Combining Exact and Heuristic Approaches for the Capacitated Fixed Charge Network Flow Problem."

3. Information-Based Branching Schemes for 0-1 Integer Problems

Branching variable selection can greatly affect the effectiveness and efficiency of a branch-and-bound algorithm. Traditional approaches to branching variable selection rely on estimating the effect of the candidate variables on the objective function. We propose an approach which is empowered by exploiting the information contained in a family of fathomed subproblems, collected beforehand from an incomplete branch-and-bound tree. In particular, we use this information to define new branching rules that reduce the risk of incurring inappropriate branchings.

The information-based variable selection methods we propose are quite different from the existing approaches as they rely on collecting information upfront (as opposed to during B&B) and they estimate the impact of a candidate branching variable on the fathoming decision of child nodes (as opposed to the objective function value of the LP relaxation of child nodes). The methods recognize that the branching decisions made at the top of the B&B tree are the most important ones and explore two ideas:

i) A *restart* strategy. To make more informed decisions at the top of the B&B tree, we perform an incomplete B&B search and collect information on fathomed nodes and exploit this information in the actual B&B search.

ii) Fathoming-based branching variable selection. To generate small B&B trees, we choose branching variables that are likely to lead to fathoming of child nodes.

To strengthen the information obtained from the incomplete B&B search, we develop an integer program that derives a minimum cardinality clause for each fathomed node. We also provide various branching alternatives that use this information in the restart framework.

We consider the 0-1 integer program

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

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 $x_i \in \{0,1\}$ by $0 \leq x_i \leq 1$ for $i = 1, \ldots, n$.

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 definitions and notation.

Definition 3. *Let v^* be an upper bound on the objective value of the optimum solution to (P). A partial assignment $C = (C^0, C^1)$ is called a <u>clause</u> if*

$$\{(x, y) \in [0,1]^n \times \mathbb{R}^k_+ : \ c^{\mathrm{T}}x + d^{\mathrm{T}}y \le v^*, \ Ax + By \ge b, \ x_j = 0 \ \forall j \in C^0, \ x_j = 1 \ \forall j \in C^1\}$$

is infeasible.

Let N_0 denote the root node, N_i denote a node in the tree and $C_i = (C_i^0, C_i^1)$ 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 can assume that $C_0 = \emptyset$. Therefore for any given fathomed node N_i in a B&B tree, the corresponding pair of index sets (C_i^0, C_i^1), gives a clause. Let f_j^l denote *fixing* the binary variable x_j to the value $l \in \{0, 1\}$, i.e. setting $x_j = l$.

Definition 4. *A clause C is called <u>minimal</u> if for all $f_j^l \in C$, the partial assignment $C \setminus \{f_j^l\}$ is not a clause.*

If C is a minimal clause, no node N_i in the tree with $C_i \subset C$ can 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 B&B tree.

Given a clause $C = (C^0, C^1)$, the inequality

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

eliminates all solutions that do not comply with it. Clearly (1) might cut off some feasible solutions. However, the region cut off by (1) is guaranteed not to contain any feasible solution with an objective value better than the optimal objective value. Furthermore (1) is a generalized cover inequality and hence, by obtaining minimal clauses, we actually derive a generalized cover inequality for (P).

Since shorter clauses yield fathoming more quickly, we are particularly interested in finding a minimal clause of *minimum cardinality.* It is quite likely that the clauses obtained from an incomplete tree are not minimal. We therefore strengthen the information on fathomed nodes by identifying the branching decisions that are essential to the fathoming of the node. We do this by solving a 0-1 integer program that obtains a minimum cardinality clause from a given basic clause.

Without loss of generality, we assume that the upper and lower bounds of the binary variables are included in the original formulation as constraints, i.e., $Ax + By \ge b$ includes $0 \le x_i \le 1 \ \forall i \in \{1, \ldots, n\}$. Let v^* be an upper bound on 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 based on the integrality of the LP solution is quite infrequent and given v^*, we can simply fathom all nodes with objective value greater than or equal to v^* by bound, which includes fathoming of all integer solutions that are no better than the current best bound as well.

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

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

Define the following variables z_i^0 (z_i^1) for $i \in C^0$ ($i \in C^1$):

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

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 MIQP

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

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

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

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 quadratic formulation where we minimize the sum of z_i^0 and z_i^1:

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

In order to bound the dual variables γ and λ from above by 1, we introduce another variable α, we relabel $\alpha\lambda$ and $\alpha\gamma$ as λ and γ respectively and 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. This yields the formulation:

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

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 z_i corresponding to the last branched variable to 1 and also we replace $\alpha > 0$ with $\alpha \geq 10^{-5}$. In fact, this last linearization step is equivalent to a big-M formulation where the big-M value can be chosen as $\frac{1}{\alpha} = 10^5$.

Once the clause information is collected, there is still the question of how to use this information to aid the search in the restart phase. Let $\mathcal{C} = \{C_1, ..., C_K\}$ be a set of clauses. 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 $C = (C^0, C^1)$ 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 is possible to obtain a descendent node (not necessarily a child) 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 descendent 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 $C = (C^0 \setminus \tilde{C}^0, C^1 \setminus \tilde{C}^1)$ 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 $\omega(C_i)$:

0. $\omega(C_i) = 1$ for all clauses C_i (i.e. all clauses are of equal importance),
1. $\omega(C_i) = \frac{1}{|C_i|}$ where $|C_i|$ is the number of variables included in clause C_i (i.e. short clauses are preferred),
2. $\omega(C_i) = 2^{-|C_i|}$ (i.e. exponentially higher preference given to the shorter clauses).
3. $\omega(C_i) = \frac{1}{\max\{\sum_{j \in C_i^0} \tilde{x}_j + \sum_{j \in C_i^1}(1-\tilde{x}_j) - 1, 10^{-10}\}}$ where we use the closeness to violation of the clause inequality (1),

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

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

Let β_j denote the overall effect of branching on variable x_j. To estimate β_j, we combine β_j^0 and β_j^1 using rules that have been derived to combine pseudocosts. We suggest and test the following alternatives:

0. $\beta_j = \min\{\tilde{x}_j, 1 - \tilde{x}_j\} * (\beta_j^0 + \beta_j^1)$,
1. $\beta_j = \beta_j^0 + \beta_j^1$,
2. $\beta_j = \max\{\beta_j^0, \beta_j^1\} + 10 * \min\{\beta_j^0, \beta_j^1\}$,
3. $\beta_j = \max\{\beta_j^0, 10^{-6}\} * \max\{\beta_j^1, 10^{-6}\}$.

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

We denote a specific rule for determining a branching variable as $a - b - c$ where $a \in \{1, 2, 3, 4\}$, $b \in \{1, 2\}$ and $c \in \{1, 2, 3, 4\}$.

In Table 2, we report a summary of the computational results for our test set of instances. In the first part of Table 2, we provide information about the *Collection phase*, namely the number of nodes in the incomplete tree and the time (in seconds) required to construct the incomplete tree, and the number of clauses collected and their average size, i.e., the average number of binary variables involved in the clauses. In the second part, we provide information about the *Improvement phase*, namely the minimum, the maximum, and the total number of nodes as well as the corresponding solution times required for solving the minimum clause identification problems, the number of clauses improved together with the average size of the improved set of clauses. Finally, we provide information about the *Restart phase*, namely the node count and solution time statistics for *B-Cuts&Prop&Branch* and *I-Cuts&Prop&Branch*, and, for comparison, the node count and solution time of *CPLEX* with and without dynamic search enabled, *CPLEX-DS* and *CPLEX-BB*, respectively. In *B-Cuts&Prop&Branch* we use basic clause information, i.e., information obtained directly from the incomplete tree, whereas in *I-Cuts&Prop&Branch* we use improved clauses, i.e., information resulting from solving a minimum clause identification problem for each basic clause obtained from the incomplete tree. In both cases, clause inequalities are added to the cut pool, and at the nodes, whenever we have clause information on a candidate branching variable, we perform branching based on the 3-1-1 combination as well as propagation.

When we compare the performance of *B-Cuts&Prop&Branch* with the performance of CPLEX B&B , we see that the node counts and solution time improve on average (even when the statistics are added from the collection phase and restart phase and compared with just the restart phase of CPLEX B&B or dynamic search.). The node count improvements are substantial for some instances: for `neos-1480121` the node count goes from 879,665 to 579,231, and for `neos-1228986` from 65,558 to 38,640. Although the computation times improve too on average, the changes are not as significant due to the overhead of our branching strategy at each node. Note that this overhead can be reduced with advanced data structures and the use of internal CPLEX structures. When we compare the performance of *I-Cuts&Prop&Branch* with the performance of *B-Cuts&Prop&Branch*, we see that for the majority of instances the B&B trees get even smaller, although the arithmetic mean does not show this, it is clear from the geometric mean. This improved behavior obtained with improved clause information is much more significant for the instances from the *hard* set. Note that although our methods are built on top of CPLEX B&B , the overall performance of our approach is comparable with CPLEX dynamic search. By simply adding up the overall average number of nodes from the collection, improvement and restart phases for *I-Cuts&Prop&Branch* we obtain 77848.97 whereas average CPLEX B&B node count is 104274.63 and CPLEX DS node count is 81421.88. The same procedure for overall solution times leads to 1090 seconds for *I-Cuts&Prop&Branch*, whereas CPLEX B&B is 1116.15 seconds and CPLEX DS is 1026.66 seconds. We want to emphasize that the improvements over CPLEX are clearer when we restrict ourselves to *hard* instances.

The following paper concerning this material has been submitted for publication in Mathematical Programming Computation.

Fatma Kilinç-Karzan, George L. Nemhauser, and Martin W.P. Savelsbergh. “Information-Based Branching Schemes for Binary Linear Mixed Integer Problems.”

Table 2: Value of the Restart Strategy with Fathoming-based Branching

		Collection phase			Improvement phase					Restart phase			
												Cuts&Prop&Branch	
Problem			#	size	min	max	total	# (%)	size	*CPLEX-BB*	*CPLEX-DS*	*B*	*I*
`Average(All)`	# nodes	795.75	200.00	23.08	0.00	84.27	653.63	70.57 (0.4)	16.61	104274.63	81421.88	74498.55	76399.59
	time	88.00			0.68	2.62	249.35			1116.15	1026.66	861.41	752.65
`Geo.Mean(All)`	# nodes	691.17	200.00	21.07	0.00	13.73	56.79	24.56 (0.1)	15.14	15729.74	11277.74	11688.61	10095.23
	time	41.88			0.45	1.86	86.90			677.32	524.34	494.22	417.23
`Average(Hard)`	# nodes	948.31	200.00	26.33	0.00	121.38	962.15	101.15 (0.5)	16.58	376527.00	287286.23	265678.77	275569.92
	time	21.28			0.15	1.32	73.60			2166.49	1851.16	1741.03	1574.68
`Geo.Mean(Hard)`	# nodes	866.68	200.00	24.80	0.00	35.36	183.95	77.72 (0.4)	15.37	187595.86	125247.38	139124.40	119371.08
	time	12.77			0.12	0.88	27.15			1486.44	1083.16	1185.47	965.61
`Average(Medium)`	# nodes	789.04	200.00	22.52	0.00	77.26	648.22	73.87 (0.4)	15.55	16689.91	16107.96	13540.04	12210.61
	time	71.27			0.62	2.80	228.91			866.06	776.76	608.49	500.50
`Geo.Mean(Medium)`	# nodes	710.30	200.00	20.31	0.00	12.04	56.93	21.32 (0.1)	14.13	13503.85	7686.48	9465.24	7512.41
	time	46.31			0.44	2.06	96.37			654.77	383.31	428.29	342.26
`Average(Easy)`	# nodes	673.80	200.00	21.12	0.00	62.87	394.53	39.00 (0.2)	18.27	2619.13	3154.13	2278.73	2208.40
	time	171.49			1.26	3.49	433.02			589.32	695.29	486.90	426.86
`Geo.Mean(Easy)`	# nodes	544.74	200.00	19.34	0.00	7.12	20.01	10.86 (0.1)	16.62	2318.53	2519.08	1887.41	1866.28
	time	97.64			0.85	2.72	200.60			360.75	451.80	288.12	272.93

Table 3: Effect of providing information in different forms: Cuts, Propagation, Branching

Problem		*CPLEX*		*Cuts*		*Prop*		*Prop&Branch*		*Cuts&Prop*		*Cuts&Prop&Branch*	
		B&B	DS	B	I	B	I	B	I	B	I	B	I
`Average(All)`	# nodes	104274.63	81421.88	98267.53	78190.55	99286.00	71424.20	78554.31	76915.43	96766.12	69657.12	74498.55	76399.59
	time	1116.15	1026.66	1059.78	926.24	1158.10	949.68	877.78	740.06	1059.84	920.88	861.41	752.65
`Geo.Mean(All)`	# nodes	15729.74	11277.74	13826.94	12352.97	15774.60	13243.56	11782.93	10050.71	13971.07	12276.95	11688.61	10095.23
	time	677.32	524.34	607.78	554.80	694.35	588.90	488.32	402.83	626.66	560.56	494.22	417.23
`Average(Hard)`	# nodes	376527.00	287286.23	356392.77	278861.00	356427.00	249255.85	281874.15	277031.23	350429.31	244892.08	265678.77	275569.92
	time	2166.49	1851.16	2226.76	1842.92	2241.33	1756.61	1834.83	1548.47	2177.95	1721.81	1741.03	1574.68
`Geo.Mean(Hard)`	# nodes	187595.86	125247.38	188082.89	157273.92	190045.00	140600.55	149230.09	121213.75	180646.81	136568.11	139124.40	119371.08
	time	1486.44	1083.16	1510.87	1295.21	1523.38	1162.59	1229.72	949.59	1495.53	1153.59	1185.47	965.61
`Average(Medium)`	# nodes	16689.91	16107.96	15046.39	14350.52	16563.91	15801.17	13402.39	12580.26	15070.87	14628.48	13540.04	12210.61
	time	866.06	776.76	767.86	682.91	869.59	753.43	594.37	498.77	797.92	729.83	608.49	500.50
`Geo.Mean(Medium)`	# nodes	13503.85	7686.48	11602.47	10202.55	13212.04	11311.45	9169.67	7460.13	12146.77	10986.99	9465.24	7512.41
	time	654.77	383.31	574.86	513.13	651.18	564.54	408.43	332.11	620.87	560.29	428.29	342.26
`Average(Easy)`	# nodes	2619.13	3154.13	2164.73	2164.20	3271.00	2592.07	2243.40	2129.00	2190.73	2164.07	2278.73	2208.40
	time	589.32	695.29	495.99	504.91	661.69	551.25	482.92	409.43	492.43	519.68	486.90	426.86
`Geo.Mean(Easy)`	# nodes	2318.53	2519.08	1883.09	1825.50	2393.81	2176.12	1916.31	1833.71	1882.95	1803.32	1887.41	1866.28
	time	360.75	451.80	300.38	299.67	387.54	348.29	288.15	257.34	298.82	299.89	288.12	272.93