Using diversification, communication and parallelism to solve mixed-integer linear programs

R. Carvajal[a,*], S. Ahmed[a], G. Nemhauser[a], K. Furman[b], V. Goel[c], Y. Shao[c]

[a]*Industrial and Systems Engineering, Georgia Institute of Technology*
[b]*ExxonMobil Research and Engineering*
[c]*ExxonMobil Upstream Research Company*

Abstract

Performance variability of modern mixed-integer programming solvers and possible ways of exploiting this phenomenon present an interesting opportunity in the development of algorithms to solve mixed-integer linear programs (MILPs). We propose a framework using multiple branch-and-bound trees to solve MILPs while allowing them to share information in a parallel execution. We present computational results on instances from MIPLIB 2010 illustrating the benefits of this framework.

Keywords:
Integer programming, branch-and-bound, diversification, communication, parallelism, performance variability

1. Introduction

Mixed-Integer Linear Programming (MILP) problems are commonly solved using a linear programming based branch-and-cut scheme. The scheme proceeds by partitioning the space of solutions in the form of a search tree obtained by fixing variable bounds (branching), and uses lower bounds from linear programming (LP) relaxations, typically tightened by the addition of cutting planes, and upper bounds, from feasible solutions, to prune parts of the search space. Modern implementations of this generic scheme involve a multitude of options, e.g. how to select a partition to further subdivide (node

[*]Corresponding author
Email address: rocarvaj@gatech.edu (R. Carvajal)

selection), how to select a variable to branch on (variable selection), what kinds of cutting planes to add and how often to add them, what types of heuristics to use and how often to use them and so on. It is well known that different choices of these options can have very significant effect on the performance of the branch-and-cut scheme [2]. In addition to these algorithmic options, it has been noted in [3] that the performance of a specific implementation on a particular problem instance can vary very significantly with less understood factors, such as the computational environment, random seeds used in the inner workings of the implementation, and permutation of the rows and columns of the instance. To alleviate inconsistency of computational results due to performance variability issues, [9] suggest a *performance variability score* to present computational results on the MIPLIB 2010 instances. For more details regarding performance variability, the reader is referred to the recent survey article [11].

One way of exploiting performance variability in solving an instance is processing it with multiple different settings (called *configurations* throughout the rest of this paper) of a MILP solver and then selecting the best of these executions. In [12] commercial solvers like CPLEX and Gurobi are executed multiple times with different random seeds (5 at the time of this writing) to solve a set of MILP benchmark instances and the best result is reported. The results show improvements in time to optimality relative to the default settings of these solvers (around 40% for CPLEX and 30% for Gurobi). In [5], a *bet-and-run* approach is proposed, where multiple configurations of a MILP solver obtained by different kinds of randomization of the root LP are run simultaneously until a given number of nodes is explored. Then a bet is placed on the "best" configuration, which is then run to optimality. The authors report a reduction in the number of branch-and-bound nodes. A similar approach is reported in the context of implementing branch and bound in a parallel architecture [13, 10, 14]. A well known issue here is the "ramp-up" phase of the algorithm in which not many branch-and-bound nodes have been generated, making it hard to balance work among available multiple processors and potentially leaving many idle resources. In [13] and then in [10, 14] a technique (that is called *racing ramp-up* in [10]) is proposed, which involves running different configurations of a MILP solver in parallel independently across the available processors until a given criterion is met. Then the best generated branch-and-bound tree is used to continue the algorithm by sending its nodes to the different processors. All other generated trees are discarded. According to [10], this approach has not proven to be very effective. In [4], the effect of variability in collecting good cutting planes and solutions within the CPLEX MILP solver is investigated. The approach starts by running a *sampling* phase that executes the default root-node cut loop starting from different optimal bases of the initial LP (through the use of the random seed parameter available in CPLEX) multiple times, while collecting cutting planes and feasible solutions. After this, a final run is done where the instance is solved by using the collected pools of cuts and solutions. They report considerable reduction in the variability of the percentage root node LP gap closed on a set of "unstable" instances from MIPLIB 2010. They also note improvements in primal solutions at the root node and reductions in the time of the final run of the algorithm (without taking into account the sampling phase).

In this paper, we study a possible way of exploiting performance variability by considering a diverse set of configurations of a MILP solver and executing these in parallel while allowing them to share information among each other. We test different types of information to be shared and compare the performance with that of the base solver with default settings. Our experimental results confirm previous evidence [5] that by simply selecting the best of multiple runs of a MILP solver with different configurations can yield significant performance improvements. We also show that the addition of communication yields substancial benefits in terms of reaching good feasible solutions or good upper bounds quickly. Although previous work (like [4]and [5]) has explored ideas that can be potentially used in a parallel setting, in most cases a serial implementation is used. However our approach allows for communication on-the-fly and parallelism is a core part of our implementation.

2. The diversification-communication framework

Our scheme consists of multiple configurations of a MILP solver running in parallel on the same instances. We call the set of different configurations a *diversification*. The configurations can share communication in different modes:

- **No communication.** Configurations run independently until one of them proves optimality or all of them run out of time.
- **Light communication.** Configurations run independently, but the best lower and upper bound obtained among all configurations are

recorded. Thus optimality can be proved earlier by having one configuration providing the optimal solution and another the best dual bound.

- **Communication.** Configurations share some information with the other configurations, which then use the information in their own searches.

In the communication mode, the configurations can share one or more of the following types of information:

- **Feasible solutions.** Configurations send feasible solutions they find to the other configurations. Each configuration that receives a solution adds it as a heuristic solution to its search.

- **LP bounds.** Configurations send their best LP bound to other configurations. Each configuration uses the value to add an objective value cut of the form $c^T x \geq \bar{z}$, where $\bar{z}$ is a global lower bound for the problem.

- **Cuts at the root node.** The pool of cutting planes generated during the root node cut loop in each configuration is shared with the other configurations. When a configuration receives cutting planes from other configurations, it adds these as new rows to the problem. More details on the way this is implemented are given in Section 3.

- **Feasible solution as forbidden paths.** In an instance with binary variables, configurations share feasible solutions. When a configuration receives a solution, it uses the value of its binary variables to forbid it from the feasible region. Given $B \subseteq I$ the set of indices of binary variables in the problem, a feasible solution x^* and all the paths in the branch-and-bound tree that coincide with it in its binary variables can be forbidden by adding the cut:

$$\sum_{\substack{j \in B, \\ x_j^* = 0}} x_j + \sum_{\substack{j \in B, \\ x_j^* = 1}} (1 - x_j) \geq 1.$$

  Additionally, in order to use the upper bound information provided by the solution, an objective-value cut of the form $c^T x \leq c^T x^*$ is also added.

- **Fathoming paths.** Again, in the context of problems with binary variables, the configurations keep track of paths in the branch-and-bound tree that end in fathomed nodes. These paths are then shared with the other configurations which cut them off from their feasible regions with cuts of the form:

  $$\sum_{j \in B \cap P_0} x_j + \sum_{j \in B \cap P_1} (1 - x_j) \geq 1.$$

  Here, a path P consists of two sets of indices of variables that are fixed to 0 and 1, P_0 and P_1 respectively.

3. Parallel implementation and experimental framework

We implemented the diversification-communication framework described in the previous section by using a master-worker scheme, in which the master process is only in charge of managing communication among the workers, which are in turn running the different configurations of the MILP solver in parallel. Our code is written in C++ using OpenMPI [6] to implement the communication and CPLEX 12.4 [7] as the base MILP solver. All the communication with the worker processes is done through the use of CPLEX callback functions implemented using its C callable library.

The sharing of cuts at the root node is implemented using a two-phase approach. First, a set of configurations with different emphases in generating cutting planes execute CPLEX's root node cut loop until it reaches its end or a certain time limit. Second, the configurations share their corresponding cut pools and best solution found so far (if any). Then, a different diversification is used, which we discuss below, and the optimization is restarted using the cuts received as new rows. Because we need to reference the original variables when collecting cuts, we apply presolve to every instance in our test set and therefore the presolve feature in CPLEX is turned off.

Our experiments are performed over the *Benchmark* class of problems from MIPLIB 2010 [9]. We only use the feasible instances from this class due to the requirements of our algorithm and we also omit instance `dfn-gwin-UUM` because it does not have any binary variables. This leaves 84 instances in total. All runs are given a time limit of 3 hours of wall-clock time. When sharing cuts at the root node, a 1 hour time limit is given to the root node cut loop and 3 hours for the re-optimization phase.

One of our goals is to compare the performance of the proposed diversification-communication framework with default CPLEX using the same number of threads as processors that run in parallel. We limit our experiments to using 8 processes because of machine availability. Since our implementation of diversification-communication uses user callback functions in CPLEX, dummy callbacks were used when CPLEX was executed with default settings.

In order to test the effect of communication, we fixed a diversification consisting of 8 configurations, listed below, each using one thread, obtained by changing one parameter of CPLEX 12.4 at a time (The corresponding CPLEX parameter considered is indicated in parenthesis.)

1. CPLEX 12.4 with default settings.
2. Emphasis on feasibility (`CPX_MIPEMPHASIS_FEASIBILITY`).
3. Emphasis on optimality (`CPX_MIPEMPHASIS_OPTIMALITY`).
4. Increase of heuristic frequency to every 10 nodes (`CPX_PARAM_HEURFREQ`).
5. Use best estimate for node selection (`CPX_NODESEL_BESTEST`).
6. Use of pseudo costs for variable selection (`CPX_VARSEL_PSEUDO`).
7. Use of pseudo reduced costs for variable selection (`CPX_VARSEL_PSEUDOREDUCED`).
8. Generate Gomory fractional cuts aggressively (`CPX_PARAM_FRACCUTS`).

These configurations consider various aspects of the branch-and-bound algorithm and seem to have a significant effect on performance, based on our experience with CPLEX 12.4.

We consider several metrics to measure the performance of the proposed framework. Note that the optimal values are known for the instances used here.

- *Number of instances solved to optimality.*
- *Time to optimality* reported by the algorithm.
- *Times to reach 1%, 5% and 10% gap*, as reported by the algorithm. At a given time, let z_{best} and z_{bound} be the values of the best feasible feasible solution and best lower bound found so far. The current gap is computed as $(z_{best} - z_{bound})/(1.0^{-10} + |z_{best}|)$.
- *Times to reach 1%, 5% and 10% primal gap.* Let z_{opt} be the optimal value. At a given time, the current *primal gap* is computed as $(z_{best} - z_{opt})/(1.0^{-10} + |z_{best}|)$.

- *Times to reach 1%, 5% and 10% dual gap.* At a given time, the current *dual gap* is computed as $(z_{opt} - z_{bound})/(1.0^{-10} + |z_{opt}|)$.

Every time an algorithm fails to prove optimality or to reach one of the different gap values mentioned above, the corresponding time is counted as the time limit.

4. Results

In Table 1 we present a performance comparison of the various modes of our framework against default CPLEX 12.4 using 8 threads (labeled CPX8T in the table). We use the three modes of communication given in Section 2, the no-communication and light-communication modes (NoComm and Light-Comm, in the table) and within the communication mode:

- Feasible solutions (SolsComm in table).
- Root node cuts (CutsComm in table).
- Root node cuts and feasible solutions in the re-optimization (CutsSolsComm in table).
- Feasible solutions as forbidden paths (SolsForbComm in table).
- Feasible solutions as forbidden paths and paths that lead to fathomed nodes (SPForbComm in table).

We do not include results on sharing lower bounds since the performance obtained was significantly worse than in the rest of the modes. This can be explained by the fact that the cuts used to add the bounds are parallel to the objective function and can lead to significant dual degeneracy [8].

The set of instances is separated into three difficulty classes. These classes are determined by ranking the instances according to the maximum time to optimality among all the modes as well as default CPLEX. The *easy*, *medium* and *hard* classes consist of instances solved within 0 to 1000 seconds, 1001 to 10000 seconds, and more than 10000 seconds, respectively. The number of instances in each class is indicated in Table 1, in parenthesis next to the name of the class.

Class	Algorithm	Solved	Time	Time1p	Time 5p	Time10p	TimeRP1p	TimeRP5p	TimeRP10p	TimeRD1p	TimeRD5p	TimeRD10p
Easy (26)	CPX8T	26										
	NoComm	26	**52.56%**	59.29%	63.98%	61.24%	63.67%	68.39%	69.05%	54.97%	60.75%	62.89%
	LightComm	26	47.93%	56.98%	62.91%	63.93%	64.05%	70.72%	**71.23%**	53.38%	59.48%	60.46%
	SolsComm	26	52.17%	**60.84%**	**64.90%**	**68.73%**	63.66%	**71.82%**	69.05%	**55.87%**	**62.42%**	63.59%
	SolsForbComm	26	50.49%	60.62%	64.59%	64.97%	**65.59%**	69.22%	69.61%	45.19%	50.26%	52.48%
	SPForbComm	26	50.22%	58.58%	63.68%	65.96%	63.01%	71.10%	70.09%	43.96%	49.43%	52.87%
	CutsComm	26	36.35%	40.84%	44.74%	48.09%	44.65%	49.39%	48.62%	42.92%	47.64%	**63.60%**
	CutsSolsComm	26	28.49%	40.42%	48.85%	45.08%	43.23%	49.30%	50.41%	40.70%	47.53%	58.63%
Medium (28)	CPX8T	28										
	NoComm	28	-20.02%	-3.21%	4.51%	9.23%	47.12%	35.77%	29.02%	**16.03%**	**29.50%**	**27.91%**
	LightComm	28	-39.54%	-8.16%	16.00%	22.80%	45.46%	14.77%	7.57%	9.68%	27.48%	26.41%
	SolsComm	28	-22.49%	1.17%	**19.15%**	**27.86%**	**49.16%**	36.23%	31.06%	8.12%	22.54%	24.28%
	SolsForbComm	28	-8.31%	5.27%	13.43%	18.69%	34.57%	**42.85%**	**34.01%**	8.09%	17.59%	20.81%
	SPForbComm	27	-7.82%	**5.47%**	18.82%	17.85%	41.16%	7.84%	8.44%	13.09%	25.00%	25.37%
	CutsComm	28	-79.33%	-49.04%	-56.10%	-31.24%	21.10%	-6.12%	-23.30%	-69.55%	-69.95%	-43.53%
	CutsSolsComm	28	-135.00%	-105.60%	-114.69%	-100.75%	-27.65%	-53.20%	-55.66%	-59.07%	-48.63%	-27.17%
Hard (29)	CPX8T	20										
	NoComm	21	0.96%	27.21%	61.31%	50.72%	60.40%	58.76%	62.78%	21.77%	48.79%	47.80%
	LightComm	20	-6.19%	32.77%	62.87%	57.94%	61.24%	60.22%	65.45%	16.90%	47.35%	46.06%
	SolsComm	21	2.92%	36.54%	57.55%	59.76%	**61.26%**	60.35%	65.85%	30.83%	**50.82%**	**50.62%**
	SolsForbComm	19	9.46%	36.32%	61.09%	54.95%	40.93%	55.28%	64.85%	6.22%	35.64%	26.58%
	SPForbComm	21	**11.97%**	**41.67%**	**65.93%**	**60.16%**	46.61%	**65.84%**	**68.43%**	8.40%	38.17%	48.31%
	CutsComm	18	-40.09%	26.65%	30.27%	-2.61%	14.85%	5.97%	9.29%	**45.51%**	44.21%	38.58%
	CutsSolsComm	15	-43.33%	23.50%	35.95%	18.74%	21.53%	31.98%	37.15%	37.00%	39.24%	35.76%

Table 1: Column "Solved" reports the number of instances solved to optimality within the time limit, column "Time" reports improvement in the time to prove optimality, columns "Time$\{1, 5, 10\}$p" report improvement in time to reach 1%, 5% and 10% gap, respectively, and columns "Time$\{1, 5, 10\}\{\text{RP}, \text{RD}\}$p" report improvement in time to reach 1%, 5% and 10% primal/dual gap, respectively. The maximum improvements are in bold font. Note that positive numbers indicate better performance than default CPLEX using 8 threads.

For each mode of our framework, Table 1 shows the percentage improvement in geometric mean (across all instances in a class) of the performance measures presented in Section 3 relative to CPLEX 12.4 with default settings using 8 threads.

A first observation from the results in Table 1 is that by just using the 8 diverse configurations each using one thread in no-communication mode, we already outperform default CPLEX using eight threads. The only exceptions are for the medium class for time to optimality and time to 1%.

Surprisingly, the effect of the light-communication mode is not very significant and can even be detrimental, probably due to the communication overhead. This indicates that it might typically be the case that one configuration proves optimality by itself, instead of one configuration finding the optimal solution and another finding the best lower bound to prove optimality.

The addition of communication has a positive impact on performance relative to the no-communication setting. Communication of feasible solutions and fathoming paths (algorithms SolsComm, SolsForbComm and SPForbComm) show consistent good performance across classes, and are always located among the top modes.

Communication of cuts appears to be the worst performer, except for a few exceptions on the dual gaps for hard instances. These results may be a little unfair since the execution times considered here include the time spent in the first phase in which the cuts are collected. However, considering the performance only during the re-optimization phase (results that are not presented in this article) this communication mode gives better results, especially in the times to reach good dual gaps. This is consistent with the experience reported in [4].

One possible reason that can explain the weak performance of communication modes in terms of time to optimality and time to good gaps in the medium instances is that when a configuration on a particular instance reaches a point where upper and lower bounds changes are small, sharing information may not be that helpful anymore.

Finally, in terms of the number of instances solved to optimality, some of the modes are able to solve one more instance than CPLEX using 8 threads in the hard class. Again, the modes that share cuts failed to prove optimality in more instances than the others in the same class.

5. Conclusions

We have studied a potential way to take advantage of performance variability in MILP solvers, by sharing of information among diverse configurations during parallel execution. We have tested various types of information, such as feasible solutions, cutting planes and branching information in terms of paths that lead to fathomed nodes. Experimental results on instances from MIPLIB 2010 indicate that we outperform CPLEX using 8 threads by simply considering 8 different settings of CPLEX using a single thread that run independently. This confirms previous experiences cited in the literature.

The impact on performance of allowing communication among different configurations of the solver is positive, but usually mild relative to the setting without communication. In our experience, it seems that sharing of feasible solutions and some branching information are the most promising approaches. On the other hand the communication of cutting planes seems not to be helpful, but this may be because of the preliminary phase of cut collection.

In summary, our work indicates that there is potential in the use of communication among different configurations of a MILP solver. Future efforts should consider ways of obtaining diverse branching information from the participating configurations and using it to ensure that a more efficient search is done by avoiding bad branching decisions and promoting those that quickly fathom big parts of the tree.

Acknowledgements

This research has been supported in part by ExxonMobil Upstream Research Company, AFOSR and the School of Bussiness at Universidad Adolfo Ibáñez.

References

[1] T. Achterberg, Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin, 2009.

[2] R. Bixby, E. Rothberg, Progress in computational mixed integer programming–A look back from the other side of the tipping point, *Annals of Operations Research* 149, pp. 37–41, 2007.

[3] E. Danna, Performance variability in mixed integer programming, Presentation at the Workshop on Mixed Integer Programming, 2008.

[4] M. Fischetti, A. Lodi, M. Monaci, D. Salvagnin, A. Tramontani, Tree search stabilization by random sampling. Technical report OR/13/5, DEI, University of Bologna, 2013.

[5] M. Fischetti, M. Monaci, Exploiting erraticism in search. Submitted to *Operations Research*, 2012.

[6] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pp. 97–104, 2004.

[7] IBM, IBM ILOG CPLEX Optimizer, 2010.

[8] IBM, IBM ILOG CPLEX Optimizer forum: Adding objective value cuts, `https://www.ibm.com/developerworks/forums/forum.jspa?forumID=2059&start=0`, 2013.

[9] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, K. Wolter, MIPLIB 2010. *Mathematical Programming Computation* 3, pp. 103–163, 2011.

[10] T. Koch, T. Ralphs, Y. Shinano, Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research* 76, pp. 67–93, 2012.

[11] A. Lodi, A. Tramontani, Performance variability in MIP, Preprint, 2013.

[12] H. Mittelmann, Mixed integer linear programming benchmark, `http://plato.asu.edu/ftp/milpc.html`, 2013.

[13] V. Nwana, K. Darby-Dowman, G. Mitra, A twostage parallel branch and bound algorithm for mixed integer programs, *IMA Journal of Management Mathematics* 15, pp. 227–242, 2004.

[14] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, ParaSCIP: A parallel extension of SCIP. In C. Bischof, H.-G. Hegering, W. E.

Nagel, G. Wittum (Eds.), *Competence in High Performance Computing 2010*, Springer Berlin Heidelberg, pp. 135–148, 2012.