Rapid Learning for Binary Programs

Timo Berthold*[1], Thibaut Feydy[2], and Peter J. Stuckey[2]

[1] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
berthold@zib.de
[2] National ICT Australia** and the University of Melbourne, Victoria, Australia
{tfeydy,pjs}@csse.unimelb.edu.au

Abstract. Learning during search allows solvers for discrete optimization problems to remember parts of the search that they have already performed and avoid revisiting redundant parts. Learning approaches pioneered by the SAT and CP communities have been successfully incorporated into the SCIP constraint integer programming platform.
In this paper we show that performing a heuristic constraint programming search during root node processing of a binary program can rapidly learn useful nogoods, bound changes, primal solutions, and branching statistics that improve the remaining IP search.

1 Introduction

Constraint programming (CP) and integer programming (IP) are two complementary ways of tackling discrete optimization problems. Hybrid combinations of the two approaches have been used for more than a decade. Recently both technologies have incorporated new *nogood learning* capabilities that derive additional valid constraints from the analysis of infeasible subproblems extending methods developed by the SAT community.

The idea of *nogood learning*, deriving additional valid *conflict constraints* from the analysis of infeasible subproblems, has had a long history in the CP community (see e.g. [1], chapter 6) although until recently it has had limited applicability. More recently adding carefully engineered nogood learning to SAT solving [2] has lead to a massive increase in the size of problems SAT solvers can deal with. The most successful SAT learning approaches use so called *first unique implication point (1UIP)* learning which in some sense capture the nogood closest to the failure that can infer new information.

Constraint programming systems have adapted the SAT style of nogood learning [3, 4], using 1UIP learning and efficient SAT representation for nogoods, leading to massive improvements for certain highly combinatorial problems.

Nogood learning has been largely ignored in the IP community until very recently (although see [5]). Achterberg [6] describes a fast heuristic to derive

* This research was partially funded by the DFG Research Center MATHEON in Berlin
** NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

small conflict constraints by constructing a dual ray with minimal nonzero elements. He shows that nogood learning for general mixed integer problems can result in an average speedup of 10%. Kılınc Karzan et. al. [7] suggest restarting the IP solver and using a branching rule that selects variables which appear in small conflict constraints for the second run. Achterberg and Berthold [8] propose a hybrid branching scheme for IP that incorporates conflict-based SAT and impact-based CP style search heuristics as dynamic tie-breakers.

2 Rapid Learning

The power of nogood learning arises because often search algorithms implicitly repeat the same search in a slightly different context in another part of the search tree. Nogoods are able to recognize such situations and avoid redundant work. As a consequence, the more search is performed by a solver and the earlier nogoods are detected the greater the chance for nogood learning to be beneficial.

Although the nogood learning methods of SAT, CP, and IP approaches are effectively the same, one should note that because of differences in the amount of work per node each solver undertakes there are different design tradeoffs in each implementation. An IP solver will typically spend much more time processing each node than either a SAT or CP solver. For that reason SAT and CP systems with nogoods use 1UIP learning and frequent restarts to tackle problems while this is not the case for IP. IP systems with nogoods typically only restart at the root, and use learning methods which potentially generate several nogoods for each infeasibility (see [6]).

The idea of Rapid Learning is based on the fact that a CP solver can typically perform a partial search on a few hundred or thousand nodes in a fraction of the time that an IP solver needs for processing the root node of the search tree. Rapid Learning applies a fast CP branch-and-bound search for a few hundred or thousand nodes, before we start the IP search, but after IP presolving and cutting plane separation.

Each piece of information collected in this rapid CP search can be used to guide the IP search or even deduce further reductions during root node processing. Since the CP solver is solving the same problem as the IP solver

- each generated conflict constraint is valid for the IP search,
- each global bound change can be applied at the IP root node,
- each feasible solution can be added to the IP solver's solution pool,
- the branching statistics can initialize a hybrid IP branching rule [8], and
- if the CP solver completely solves the problem, the IP solver can abort.

All five types of information may potentially help the IP solver. Rapid Learning performs a limited CP search at the root node, after most of the IP presolving is done to collect potential new information for the IP solver.

The basic idea of Rapid Learning is related to the concept of *Large Neighborhood Search* heuristics in IP. But rather than doing a partial search on a sub-problem using the same (IP search) algorithm, we perform a partial search

on the same problem using a much faster algorithm. Rapid Learning also differs from typical IP heuristics in the sense that it can improve both primal and dual bounds at the same time.

3 Computational Results

Our computational study is based on the branch-cut-and-price framework SCIP (Solving Constraint Integer Programs). This system incorporates the idea of *Constraint Integer Programming* [9, 10] and implements several state-of-the-art techniques for IP solving, combined with solving techniques from CP and SAT, including nogood learning. The Rapid Learning heuristic presented in this article was implemented as a separator plugin.

For our experiments, we used SCIP 1.2.0.5 with Cplex 12.10 as underlying LP solver, running on a Intel® Core™2 Extreme CPU X9650 with 6 MB cache and 8 GB RAM. We used default settings and a time limit of one hour for the main SCIP instance which performs the IP search.

For solving the CP problem, we used a secondary SCIP instance with "emphasis cpsolver" (which among other things turns off LP solving) and "presolving fast" settings (which turns off probing and pairwise comparison of constraints) and the parameter "conflict/maxvarsfac" set to 0.05 (which only creates nogoods using at most 5% of the variables of the problem). As node limit we used $\max(500, \min(\mathit{niter}, 5000))$, with *niter* being the number of simplex iterations used for solving the root LP in the main instance. We further aborted the CP search as soon as 1000 conflicts were created, or no useful information was gained after 20% of the node limit.

As test set we chose all 41 *Binary programs (BPs)* of the Miplib 3.0 [11], the Miplib2003 [12] and the IP collection of Hans Mittelmann [13] which have less then 10 000 variables and constraints after SCIP presolving. BPs are an important subclass of IPs and finite domain CPs. where all variables take values 0 or 1. Note, that for a BP, all conflict constraints are Boolean clauses, hence linear constraints.

Table 1 compares the performance of SCIP with and without Rapid Learning applied at the root node (columns "SCIP" and "SCIP-RL"). Columns "RL" provide detailed information on the performance of Rapid Learning. "Ngds" and "Bds" present the number of applied nogoods and global bound changes, respectively, whereas "S" indicates, whether a new incumbent solution was found. For instances which could not be solved within the time limit, we present the lower and upper bounds at termination.

Note first that Rapid Learning is indeed rapid, it rarely consumes more than a small fraction of the overall time (except for `mitre`). We observe that for many instances the application of Rapid Learning does not make a difference. However, there are some, especially the `acc` problems, for which the performance improves dramatically. There are also a few instances, such as `qap10`, for which Rapid Learning deteriorates the performance. The solution time decreases by 12% in geometric mean, the number of branch-and-bound nodes by 13%. For the four unsolved instances, we see that Rapid Learning leads to a better primal

bound in three cases. The dual bound is worse for the instance `protfold`. For the instances `acc-2` and `nug08`, Rapid Learning completely solved the problem.

Additional experiments indicate that the biggest impact of Rapid Learning comes from nogoods and learning new bounds, but all the other sources of information are also beneficial to the IP search on average.

4 Conclusion and Outlook

Rapid Learning takes advantage of fast CP search to perform a rapid heuristic learning of nogoods, global bound changes, branching statistics and primal solutions before the IP search begins. Our computational results demonstrate that this information can improve the performance of a state-of-the-art non-commercial IP solver on BPs substantially.

We plan to investigate Rapid Learning for general IP problems, where we need to use bound disjunction constraints [6] to represent nogoods. We also plan to investigate the application of rapid learning at other nodes than the root, and combinations of CP and IP search that continually communicate nogoods, using a hybrid of SCIP and a native CP system.

References

1. Dechter, R.: Constraint Processing. Morgan Kaufman (2003)
2. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of DAC'01. (2001) 530–535
3. Katsirelos, G., Bacchus, F.: Generalised nogoods in CSPs. In: Proceedings of AAAI-2005. (2005) 390–396
4. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3) (2009) 357–391
5. Davey, B., Boland, N., Stuckey, P.: Efficient intelligent backtracking using linear programming. INFORMS Journal of Computing **14**(4) (2002) 373–386
6. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization **4**(1) (2007) 4–20 Special issue: Mixed Integer Programming.
7. Kılınç Karzan, F., Nemhauser, G.L., Savelsbergh, M.W.P.: Information-based branching schemes for binary linear mixed-integer programs. Math. Progr. C **1**(4) (2009) 249–293
8. Achterberg, T., Berthold, T.: Hybrid branching. In van Hoeve, W.J., Hooker, J.N., eds.: Proc. of CPAIOR 2009. Volume 5547 of LNCS., Springer (May 2009) 309–311
9. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007)
10. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: A new approach to integrate CP and MIP. In Perron, L., Trick, M.A., eds.: Proc. of CPAIOR 2008. Volume 5015 of LNCS. (2008) 6–20
11. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: MIPLIB 3.0. Optima (58) (1998) 12–15
12. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters **34**(4) (2006) 1–12
13. Mittelmann, H.: Decision tree for optimization software: Benchmarks for optimization software http://plato.asu.edu/bench.html.

Table 1. Impact of Rapid Learning on the performance of SCIP

	SCIP		SCIP-RL		Rapid Learning				
Name	Nodes	Time	Nodes	Time	Nodes	Time	Ngds	Bds	S
10teams	197	7.2	197	7.3	716	0.1	0	0	
acc-0	1	0.9	1	0.9	0	0.0	0	0	
acc-1	112	32.6	113	34.4	3600	0.4	1332	0	
acc-2	54	58.8	1	4.4	2045	0.4	427	0	✓
acc-3	462	392.5	64	76.0	2238	0.7	765	0	
acc-4	399	420.2	364	115.4	2284	0.7	722	0	
acc-5	1477	354.1	353	126.6	2054	0.5	756	0	
acc-6	251	71.0	899	138.2	2206	0.5	591	0	
air04	159	45.4	159	45.6	1000	0.2	0	0	
air05	191	22.6	191	22.8	369	0.1	0	0	
cap6000	2755	2.6	2755	2.7	100	0.0	0	17	
disctom	1	2.2	1	2.2	0	0.0	0	0	
eilD76	3	17.2	3	17.2	100	0.0	0	0	
enigma	733	0.5	1422	0.5	500	0.0	9	0	
fiber	51	1.1	53	1.1	100	0.0	0	0	
harp2	352292	209.2	306066	191.3	1135	0.4	7	0	✓
l152lav	56	2.1	56	2.2	423	0.1	0	0	
lseu	366	0.5	450	0.5	500	0.0	146	0	✓
markshare4_0	1823558	111.7	2140552	234.4	500	0.0	305	0	✓
misc03	176	0.8	284	0.8	500	0.0	138	0	
misc07	31972	21.4	34416	22.4	100	0.0	0	0	
mitre	6	7.5	6	10.0	4177	2.5	284	1610	
mod008	366	0.8	366	0.8	100	0.0	0	0	✓
mod010	5	0.8	5	1.0	854	0.2	357	52	
neos1	1	3.1	1	3.2	727	0.1	325	0	✓
neos21	2020	18.7	1538	17.5	141	0.0	0	0	✓
nug08	1	56.2	1	10.2	1011	0.2	460	1392	
p0033	3	0.5	3	0.5	500	0.0	287	4	✓
p0201	76	0.7	76	0.7	100	0.0	0	0	
p0282	24	0.5	24	0.5	100	0.0	0	0	✓
p0548	53	0.5	38	0.5	100	0.0	0	10	
p2756	213	1.7	111	1.6	100	0.0	0	80	
prod1	23015	17.1	25725	20.0	500	0.1	0	0	✓
prod2	68682	80.3	68635	79.2	500	0.1	17	0	✓
qap10	5	146.8	12	542.0	2107	0.5	1666	0	
stein27	4041	0.8	4035	1.1	500	0.0	328	0	✓
stein45	50597	18.0	51247	18.1	500	0.0	0	0	✓
markshare1	[0.0,7.0]		[0.0,5.0]		500	0.0	199	0	✓
markshare2	[0.0,14.0]		[0.0,11.0]		500	0.0	174	0	✓
protfold	[-36.9135,-21.0]		[-37.0898,-22.0]		3078	1.6	510	0	
seymour	[414.318,425.0]		[414.313,426.0]		653	0.0	0	0	✓
geom. mean	212	8.3	185	7.3					
arithm. mean	63 902	57.5	71 357	47.4					