On Applying Cutting Planes in DLL-Based Algorithms for Pseudo-Boolean Optimization

Vasco Manquinho and João Marques-Silva

Technical University of Lisbon,
IST/INESC-ID, Lisbon, Portugal
{vmm, jpms}@sat.inesc-id.pt

Abstract. The utilization of cutting planes is a key technique in Integer Linear Programming (ILP). However, cutting planes have seldom been applied in Pseudo-Boolean Optimization (PBO) algorithms derived from the Davis-Logemann-Loveland (DLL) procedure for Propositional Satisfiability (SAT). This paper proposes the utilization of cutting planes in a DLL-style PBO algorithm, which incorporates the most effective techniques for PBO. We propose the utilization of cutting planes both during preprocessing and during the search process. Moreover, we also establish conditions that enable clause learning and non-chronological backtracking in the presence of conflicts involving constraints generated by cutting plane techniques. The experimental results, obtained on a large number of classes of instances, indicate that the integration of cutting planes with backtrack search is an extremely effective technique for PBO.

1 Introduction

In this paper we address algorithms for Pseudo-Boolean Optimization (PBO) and focus on exploiting effectively the information provided by the cost function. Our objective is to use this information for pruning the search space. Hence, we propose to augment SAT-based PBO algorithms with bounding capability, associated with information obtained from the Pseudo-Boolean (PB) constraints and from the cost function[1]. Moreover, we propose to extend a SAT-based PBO algorithm with lower bounding, that uses linear programming relaxation for compute lower bounds [2], and integrate the identification of cutting planes in this algorithm. We also establish conditions for learning new constraints from conflicts associated with cutting planes. Furthermore, we show that these new constraints can be used for performing non-chronological backtracking. Experimental results, obtained on representative problem instances, illustrate the effectiveness of integrating cutting planes in SAT-based algorithms for PBO.

[1] An extended version of this paper is available in [1].

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 451–458, 2005.

2 Preliminaries

An instance P of the Pseudo-Boolean Optimization problem can be defined as follows:

$$\begin{array}{ll} \text{minimize} & \sum\limits_{j \in N} c_j \cdot x_j \\ \text{subject to} & \sum\limits_{j \in N} a_{ij} l_j \geq b_i, \\ & x_j \in \{0,1\}, a_{ij}, b_i \in \mathbf{N}_0^+, i \in M \\ & N = \{1, \ldots, n\}, M = \{1, \ldots, m\} \end{array} \tag{1}$$

where c_j is a non-negative integer cost associated with variable $x_j, j \in N$ and a_{ij} denote the coefficients of the literals l_j in the set of m linear constraints.

Every pseudo-boolean formulation can be rewritten such that all coefficients a_{ij} and right-hand side b_i be non-negative. In a given constraint, if all a_{ij} coefficients have the same value k, then it is called a cardinality constraint, since it only requires that $\lceil b_i/k \rceil$ literals be true. A pseudo-boolean constraint where any literal set to true is enough to satisfy the constraint, can be interpreted as a propositional clause. This occurs when the value of all a_{ij} coefficients are greater than or equal to b_i. If every constraint can be interpreted as a propositional clause then P is an instance of the *binate covering problem* (BCP). Covering formulations have been the subject of thorough research work [3].

Notice that a linear pseudo-boolean optimization problem can also be viewed as a special case of linear integer programming problem. The linear integer programming formulation for the constraints can be obtained if we replace literals $\bar{x}_j$ by $1 - x_j$. Throughout the paper we refer extensively to backtrack search algorithms. In addition, the PB inference techniques of [4] are assumed.

3 Pseudo-Boolean Optimization Algorithms

Given that PBO is a restriction of generic ILP, all algorithms proposed in the past for ILP can also be used for PBO. Among these, complete approaches include branch-and-bound with linear programming relaxations [5], cutting planes [6], branch-and-cut [7], and branch-and-bound [3]. Besides algorithms for generic ILP, algorithms specific to PBO have also been proposed. These include SAT-based algorithms [4, 8, 9], branch-and-bound algorithms [3] and SAT-based algorithms with lower bounding [2]. A survey of these algorithms is available, for example, in [1]. In the reminder of this section we address the utilization of linear programming relaxations.

Linear programming relaxations (LPR) have long been used as a lower bound estimation procedure in branch-and-bound algorithms for solving integer programming problems [5, 7, 10]. It is also often the case that the LPR bound is tighter than the one obtained through other lower bounding procedures [2, 11]. The general formulation of the LPR for a pseudo-boolean problem instance is obtained from (1) as follows:

$$\begin{array}{ll} \text{minimize} & z_{lpr} = cx \\ \text{subject to} & Ax \geq b,\ 0 \leq x \leq 1 \end{array} \tag{2}$$

where vector c defines the non-negative integer cost associated with every decision variable in vector x. Entries of matrix A define the constraint coefficients and vector b the right-hand side of every constraint. It is well-known that the solution of (2) is a lower bound on the solution of (1) [5].

4 Cutting Planes

Integer Linear Programming algorithms use linear programming relaxations (as formulated in (2)) for estimating lower bounds on the value of the cost function. However, linear programming relaxations have other applications, including the identification of cutting planes.

The work on cutting planes can be traced to Gomory [6]. Gomory introduced a cutting plane technique that derives new linear inequalities in order to exclude some non-integer solutions from (2). However, the new linear inequalities are valid for the original integer linear program and so can be safely added to the original problem. Moreover, solving (2) with the added inequalities may yield a tighter lower bound estimate.

Since Gomory's original work, a large number of cutting plane techniques have been proposed [5, 10, 12]. This section addresses Gomory fractional cuts and clique cuts, which are integrated in a SAT-based PBO solver.

In simplex-based solutions for solving the LPR from (2), the simplex method adds a set S of slack variables such that each constraint can be formulated as:

$$\sum_{j \in N} a_{ij} x_j - s_i = b_i \ \ s_i \geq 0 \tag{3}$$

This formulation is called the slack formulation and it is used to create the original simplex tableau [5].

If the solution x^* of the LPR is integral, then x^* provides the optimal solution to the original problem. Otherwise, choose a basic[2] variable x_j such that its value on the LPR solution is not integral. Since x_j is a basic variable, after the pivot operations performed by the simplex algorithm on (3), there is a row in the simplex tableau of the form,

$$x_j + \sum_{i \in P} \alpha_i x_i + \sum_{i \in Q} \beta_i s_i = x_j^* \tag{4}$$

where P and Q are the sets of indexes of non-basic variables (problem variables and slack variables, respectively). In [6], Gomory proves that the inequality,

$$\sum_{i \in P} f(\alpha_i) x_i + \sum_{i \in Q} f(\beta_i) s_i \geq f(x_j^*) \tag{5}$$

[2] See for example [5] for a definition of basic and non-basic variables.

where $f(y) = y - \lfloor y \rfloor$, is violated by the solution of the LPR, but satisfied by all non-negative integer solutions to (4). Hence, it is also satisfied by all solutions to the original problem as formulated in (1) and can be added to the LPR. Solving the LPR with the new restriction (known as a Gomory *fractional cut*) will yield a tighter lower bound estimate on the value of the PBO instance. Observe that several methods for strengthening the original Gomory fractional cuts have been proposed [13, 14, 15], but are beyond the scope of this work.

Notice that we can use (3) to replaces each slack variable in (5) so that the constraint only contain variables from the original PBO problem. Afterwards, if we apply the rounding operation on the non integer coefficients we obtain a new pseudo-boolean constraint valid for the original PBO instance as defined in (1), since the rounding operation will only weaken the constraint.

Like the Gomory fractional cuts, clique cuts [5, 10] also provide a method that adds new inequalities in order to cut non-integral solutions from the LPR, hence improving the tightness of lower bound estimates.

In general, we can build a conflict graph in order to represent all incompatible assignments for a pseudo-boolean formula. In the conflict graph, each node represents an assignment to a problem variable and each edge between two nodes represents an assignment incompatibility. For each clique C in the conflict graph we can add a new constraint of the form $\sum_{i \in C} l_i \leq 1$, where l_i is the literal at node i of clique C. One should note that we are interested in finding all maximum cliques in the conflict graph, but it is well-known that that the problem of finding a maximum clique in an undirected graph is NP-Hard [5]. As a result, a heuristic greedy procedure is often used.

5 Cutting Planes in SAT-Based PBO

In a modern SAT-based algorithm, a conflict analysis procedure is carried out whenever a conflict arises [16]. Therefore, if the generated cutting plane is involved in the conflict analysis process, it must be able to determine its logical dependencies in order to backtrack to a valid node of the search tree. This section proposes conditions for associating dependencies with computed cutting planes, thus enabling clause learning and non-chronological backtracking from constraints inferred with cutting plane techniques.

The most straightforward solution, for safely determining a set of dependencies for the Gomory fractional cuts generated during the search process, is to declare that these cuts depend on all decision assignments made from the root node to the current node. This solution associates with each cutting plane all decisions in the search tree, thus forcing chronological backtracking and ensuring completeness. In this case, we can determine a set of literals ω_{cut} that defines the set of dependencies for the generated cut. When one of literals in ω_{cut} is set to 1, the cut will no longer be active (i.e. the associated constraint will be satisfied). Therefore, the generated cut would depend on all decision assignments and, for all decision variables x_j assigned from the root node to the current node, we would have $\bar{x}_j \in \omega_{cut}$ if $x_j = 1$ or $x_j \in \omega_{cut}$ if $x_j = 0$.

In order for the generated cut to be safely added to the set of pseudo-boolean constraints, we must add all literals $l_j \in \omega_{cut}$ to the cut. The coefficient of each added literal l_j must be large enough to satisfy the constraint whenever $l_j = 1$. Although this approach guarantees the completeness of the algorithm, if a conflict occurs involving the generated cuts at a node N of the search tree, the search cannot backtrack to a node higher than N.

Another technique would be to associate dependencies with cuts following the ideas proposed in [2] for LPR. Since each cut is derived from the outcome of solving the LPR formulation, then we can associate with each cut the same dependencies we associate with lower bound conflicts. However, one should note that the tableau constraint (4), from which the Gomory fractional cut is inferred, depends on the pivot operations performed while solving the LPR. As a result, the tableau constraint (4) contains the slack variables assigned value 0 from the constraints from which it depends.

Let S be the set of constraints with slack variables assigned value 0 in the tableau constraint (4). If the literals assigned value 0 in these constraints were to have a different value, the tableau constraint might not be inferred in the LPR. Therefore, we can consider the assignments to those literals as the responsible for inferring the cut and we can define ω_{cut} as:

$$\omega_{cut} = \{l : l = 0 \wedge l \in \omega_i \wedge \omega_i \in S\} \tag{6}$$

Notice that the generated cut might not depend on all decision assignments. Hence, if a conflict occurs involving generated cuts at node N with its dependences determined as in (6), it is possible to backtrack to a node higher than N in the search tree, i.e. a non-chronological backtrack step. Moreover, the generated cuts can also be used in different parts of the search tree, in addition to the subtree with root at the node N.

6 Results

In order to empirically evaluate the techniques described in the paper, we ran our solver (*bsolo*) on representative sets of PBO instances [17, 18, 19]. Besides bsolo, we also ran PBS [8], Galena [4] and the commercial MILP solver CPLEX (version 7.5). The CPU times presented are from a AMD Athlon processor at 1.9 GHz and the time limit for each instance was set to one hour. If the time limit was reached, we provide an indication of which was the best upper bound value found when the search was stopped. Additional results and details on the experimental procedure can be found at [1].

The experimental results are shown in Table 1. For bsolo we present results for four different configurations: without using cutting planes, using only fractional Gomory cuts during the search, using clique cuts and Gomory fractional cuts only during preprocessing, and using all cuts both during preprocessing and during the search. For all bsolo configurations, the lower bound estimates were obtained with linear programming relaxations.

Table 1. Experimental Results

					bsolo			
	Benchmark	sol.	pbs	galena	cplex	no cuts	Gomory	pre proc.	all cuts
[17]	9symml	4517	ub6453	ub6986	1.63	328.97	41.39	716.56	225.62
	C432	4822	ub6577	ub8070	3.34	ub4822	343.33	1253.60	208.52
	cmb	1053	ub1490	ub1476	0.03	0.25	0.23	0.12	0.10
	my_adder	4561	ub6271	ub5548	2.29	14.94	6.54	10.15	5.07
[18]	9sym.b	5	1718.98	0.26	0.14	0.89	0.68	0.73	0.72
	alu4.b	–	ub121	ub53	ub50	ub50	ub50	ub51	ub50
	apex4.a	776	ub2282	ub845	3.92	ub776	810.22	ub776	2404.60
	clip.b	15	ub30	1.68	0.36	9.65	1.15	4.14	5.54
	e64.b	–	ub99	ub53	ub49	ub50	ub49	ub50	ub49
	jac3	15	ub48	0.71	0.09	3.11	2.93	9.67	6.21
	rot.b	115	ub745	ub142	71.56	ub117	ub117	ub118	ub118
	sao2.b	25	ub39	132.91	0.50	11.3	5.46	9.34	18.99
[19]	aim100-1_6-y1-2	100	0.01	0.01	373.42	0.04	0.04	0.19	0.19
	aim100-3_4-y1-4	100	0.01	–	108.79	2.73	2.76	0.30	0.31
	aim200-1_6-y1-3	200	0.01	–	–	0.11	0.11	0.49	0.51
	aim200-6_0-y1-2	200	0.01	0.03	ub200	140.18	119.81	0.81	0.84
	ii8a2	139	ub150	ub144	63.77	ub141	ub141	1931.50	ub139
	ii8b1	191	ub272	ub213	7.11	2285.50	ub232	860.37	ub193
	ii8c1	302	ub405	ub399	2719.59	ub416	ub413	ub320	ub321
	ii8d1	–	ub515	ub432	ub351	ub470	ub470	ub359	ub401
	jnh1	92	0.01	0.20	39.56	42.81	324.99	35.84	158.63
	jnh7	89	0.02	0.04	7.42	20.55	9.80	1.91	3.04
	ssa7552-159	1327	ub1327	ub1327	ub1327	ub1327	ub1327	ub1327	1212.50
	ssa7552-160	1359	ub1359	ub1359	ub1359	ub1359	ub1359	ub1359	1369.60

Among the SAT-based PBO algorithms, bsolo is by far the most effective algorithm, for the instances in this paper and for the instances in [2]. In fact, bsolo without the utilization of cutting planes is already significantly more effective than the other SAT-based algorithms. The utilization of cutting planes further improves the results of bsolo, making it more robust than the commercial ILP solver cplex. Observe that the worst results for cplex occur for instances of the minimum-size prime implicant problem. These instances are derived from CNF formulas, where SAT-based techniques are particularly relevant.

7 Conclusions

The paper describes the integration of cutting plane techniques in SAT-based algorithms for Pseudo-Boolean Optimization and outlines conditions for performing constraint learning and non-chronological backtracking based on previously inferred cutting planes. These conditions provide novel mechanisms for extending the most effective SAT techniques to PBO. Experimental results clearly indicate

that the utilization of cutting plane techniques can be extremely effective in PBO. Moreover, the results are also clear in demonstrating that lower bounding techniques are essential for hard instances of PBO. The experimental results for the most well-known PBO solvers that do not integrate lower bounding techniques, clearly demonstrate that lower bounding is essential for representative instances of pseudo-boolean optimization.

References

1. Manquinho, V., Marques-Silva, J.: On applying cutting planes in dll-based algorithms for pseudo-boolean optimization. Technical Report RT/003/05-CDIL, INESC-ID (2005)
2. Manquinho, V., Marques-Silva, J.P.: Effective lower bounding techniques for pseudo-boolean optimization. In: Design, Automation and Test in Europe Conference. (2005)
3. Coudert, O.: On Solving Covering Problems. In: Design Automation Conference. (1996) 197–202
4. Chai, D., Kuehlmann, A.: A Fast Pseudo-Boolean Constraint Solver. In: Design Automation Conference. (2003) 830–835
5. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. John Wiley & Sons (1988)
6. Gomory, R.: Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society **64** (1958) 275–278
7. Mitchell, J.: Branch-and-cut algorithms for combinatorial optimization problems. In: Handbook of Applied Optimization. Oxford University Press (2002) 65–77
8. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: Generic ILP versus specialized 0-1 ILP: An update. In: International Conference on Computer Aided Design. (2002) 450–457
9. Barth, P.: A Davis-Putnam Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical Report MPI-I-95-2-003, Max Plank Institute for Computer Science (1995)
10. Bixby, R.E.: Progress in linear programming. ORSA Journal on computing **6** (1994) 15–22
11. Liao, S., Devadas, S.: Solving Covering Problems Using LPR-Based Lower Bounds. In: Design Automation Conference. (1997) 117–120
12. Chvátal, V.: Edmonds polytopes and a hierarchy of combinatorial problems. Discrete Mathematics **4** (1973) 305–337
13. Balas, E., Ceria, S., Cornuéjols, G., Natraj, N.: Gomory cuts revisited. Operations Research Letters **19** (1996) 1–9
14. Ceria, S., Cornuéjols, G., Dawande, M.: Combining and strengthening Gomory cuts. In Springer-Verlag, ed.: Lecture Notes in Computer Science. Volume 920. E. Balas and J. Clausen (eds.) (1995)
15. Gomory, R.: An algorithm for integer solutions to linear programs. In Graves, R., (eds.), P.W., eds.: Recent Advances in Mathematical Programming. McGraw-Hill (1963) 269–302
16. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A new search algorithm for satisfiability. In: International Conference on Computer-Aided Design. (1996) 220–227

17. Zhu, Z.: Synthesis for mixed ptl/cmos circuit. (http://www-unix.ecs.umass.edu/-~zzhu/)
18. Yang, S.: Logic Synthesis and Optimization Benchmarks User Guide. Microelectronics Center of North Carolina (1991)
19. Pizzuti, C.: Computing Prime Implicants by Integer Programming. In: IEEE International Conference on Tools with Artificial Intelligence. (1996) 332–336