Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 199-208

On Using Cutting Planes in Pseudo-Boolean Optimization

Vasco M. Manquinho vmm@sat.inesc-id.pt
IST/INESC-ID, Technical University of Lisbon, Portugal

João Marques-Silva jpms@ecs.soton.ac.uk
University of Southampton, UK

Abstract

Cutting planes are a well-known, widely used, and very effective technique for Integer Linear Programming (ILP). However, cutting plane techniques are seldom used in Pseudo-Boolean Optimization (PBO) algorithms. This paper addresses the utilization of Gomory mixed-integer and clique cuts, in Satisfiability-based algorithms for PBO, and shows how these cuts can be used for computing lower bounds and for learning new constraints. A side result of learning new constraints is that the utilization of cutting planes enables non-chronological backtracking. Besides cutting planes, the paper also shows that the utilization of search restarts in PBO can be effective in practice, allowing the computation of tighter lower bounds each time the search restarts. The more aggressive lower bounds result from the constraints learned due to the utilization of cutting planes. Experimental results show that the integration of cutting planes and search restarts in a SAT-based algorithm for PBO yields a competitive new solution for PBO.

Keywords: *Pseudo-Boolean Optimization, Cutting Planes, Search Restarts*

Submitted October 2005; revised December 2005; published March 2006

1. Introduction

Motivated by recent advances in algorithms for Boolean Satisfiability (SAT) [20, 21], algorithms for Pseudo-Boolean (PB) Solving (PBS) and Optimization (PBO) have also been the subject of significant improvements [3, 9, 12]. As a result, the most effective Boolean Satisfiability (SAT) techniques, including clause learning, lazy data structures and conflict-driven branching heuristics, have been extended to PBO. In addition to the significant amount of work on extending SAT techniques to PBS, there has also been work specific to PBO, which entails techniques specific for optimizing the cost function in PBO formulations [18, 19].

This paper proposes to apply the identification of Gomory mixed-integer cuts and clique cuts [7, 15, 16] in SAT-based PBO algorithms. The objective is to use these cutting plane techniques for computing more accurate lower bounds, and consequently obtaining additional pruning ability. Moreover, the paper shows how to exploit the computation of cutting planes for creating new constraints, which enable non-chronological backtracking from lower bounding information. This paper also shows that search restarts [14] (commonly used in state-of-the-art SAT solvers) can also be very effective for PBO. Since cutting plane thechniques are also used for generating new constraints, it is reasonable to assume that search restarts may yield tighter lower bounds each time the search is restart.

The paper is organized as follows. The following sections address, respectively, definitions, a survey of PBO algorithms, and a brief survey of cutting planes techniques. Afterwards, section 4 outlines the integration of cutting planes in SAT-based PBO algorithms, and afterwards we address the utilization of search restarts. Next, we address our submission to the pseudo-boolean evaluation and experimental results. Finally, the paper concludes in section 8.

2. Preliminaries

In a propositional formula, a literal l_j denotes either a variable x_j or its complement $\bar{x}_j$. A literal is said to be a positive literal if it denotes a variable x_j. Otherwise is said to be a negative literal. If a literal $l_j = x_j$ and x_j is assigned value 1 or $l_j = \bar{x}_j$ and x_j is assigned value 0, then the literal is said to be true. Otherwise, the literal is said to be false. An instance P of the Pseudo-Boolean Optimization (PBO) problem can be defined as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n} c_j \cdot x_j \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} l_j \geq b_i, \quad x_j \in \{0,1\}, a_{ij}, b_i \in \mathbb{N}_0^+, i \in \{1..m\}, \end{array} \tag{1}$$

where c_j is a non-negative integer cost associated with variable x_j, 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). When all literals of the propositional clauses are positive then P is an instance of the *unate covering problem* (UCP). Covering formulations have been the subject of thorough research work [10, 17, 18, 24]. Observe 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. Most if not all backtrack search SAT algorithms apply the *unit clause rule* [11]. If a clause is unit, then the sole free literal must be assigned value 1 for the formula to be satisfiable. In this case, the value of the literal and of the associated variable are said to be *implied*. The iterated application of the unit clause rule is often referred to as unit propagation. In the following sections we often need to associate *dependencies* (or an *explanation*) with each implied variable assignment. Dependencies represent sufficient conditions for variable assignments to be implied. For example, let $x = v_x$ be a truth assignment implied by applying the unit clause rule to a unit clause clause ω. Then the explanation for this assignment is the set of assignments associated with the remaining literals of ω, which are assigned value 0. In addition, pseudo-Boolean inference techniques of [9, 12] are assumed.

3. Pseudo-Boolean Optimization Algorithms

This section addresses algorithms for PBO that are relevant to the work described in the paper. We briefly overview branch-and-bound algorithms for the Binate Covering Problem (BCP), representing a well-known restriction of PBO [10], where the notion of lower bounding has been extensively used, and address SAT-based algorithms for PBO. Moroever, we describe the utilization of linear programming relaxations, a key technique in branch-and-bound algorithms for ILP, but also extremely effective in PBO algorithms that utilize lower bounding.

3.1 SAT-Based Algorithms

The SAT-based approach for PBO is based on the Davis-Logemann-Loveland [11] procedure augmented with conditions on the value of the cost function [6]. The algorithm performs a linear search on the possible values of the cost function, starting from the highest value, and at each step requiring the next computed solution to have a cost lower than the previous one. If the resulting instance is not satisfiable, then the solution is given by the last recorded solution. The generalization of recent advances in SAT to PB constraints resulted in new effective algorithms [3, 9, 12] for several classes of PB instances. The most relevant techniques include non-chronological backtracking in the search tree, conflict-based learning mechanisms and lazy data structures.

3.2 Branch-and-Bound Algorithms

Unlike SAT-based algorithms, branch-and-bound algorithms [10, 17] have proved to be very effective for instances where it is not hard to find a variable assignment that satisfies all constraints. In general, these algorithms are able to prune the search tree earlier by using estimates on the value of the cost function. In branch-and-bound algorithms *upper bounds* on the value of the cost function are identified for each solution to the constraints, and *lower bounds* on the value of the cost function are estimated considering the current variable assignments. For a given instance P of PBO, let $P.upper$ denote the upper bound on the value of the cost function. The search is pruned whenever the lower bound estimation is larger than or equal to $P.upper$. In this case it is guaranteed that a better solution cannot be found with the current variable assignments and therefore the search can be pruned. The algorithms described in [10, 17, 18, 24] for the binate covering problem follow this approach.

For several classes of instances, specially for less constrained instances, the tightness of the lower bounding procedure is crucial for the algorithm's efficiency, because with higher estimates of the lower bound, the search can be pruned earlier. Several procedures can be used for lower bound estimation, namely the approximation of a maximum independent set of constraints (MIS) [10, 18], linear-programming relaxations [17] or Lagrangian relaxations [2].

In the remainder of the paper, we refer to *lower bound conflicts* to denote the situations when the search process backtracks because the lower bound estimate is greater than or equal to a previously computed upper bound on the value of the cost function.

3.3 Linear Programming Relaxations

Linear programming relaxation (LPR) has been used with success [17, 19] in solving PBO since it often provides tighter bounds than other methods. Moreover, LPR have long been used as a lower bound estimation procedure in branch-and-bound algorithms for solving ILP problems [7, 22]. The general formulation of the LPR for a pseudo-boolean problem instance is obtained from (1) as follows:

$$\begin{array}{lll} \text{minimize} & z_{lpr} = \sum_{j=1}^{n} c_j \cdot x_j & \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & 0 \leq x_j \leq 1, a_{ij}, b_i \in \mathbb{Z} \end{array} \tag{2}$$

The solution of (1) is referred to as z^*_{pbo}, whereas the solution of (2) is referred to as z^*_{lpr}. It is well-known that the solution z^*_{lpr} of (2) is a lower bound on the solution z^*_{pbo} of (1) [22]. Basically, any solution of (1) is also a feasible solution of (2), but the converse is not true. Moreover, for a given solution of (2) where $x \in \{0,1\}^n$, we necessarily have $z^*_{pbo} = z^*_{lpr}$. Hence, the result follows.

4. Cutting Planes

Work on cutting planes can be traced back to Gomory [15]. 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 [7, 22]. This section addresses Gomory mixed-integer and clique cuts, as well as their integration in a SAT-based PBO solver. Moreover, we also establish conditions in order to backtrack non-chronologically in the search tree when a conflict arises involving learned cutting planes.

4.1 Gomory Mixed-Integer Cuts

Section 3.3 describes the utilization of linear programming relaxation (LPR) for estimating lower bounds in Pseudo-Boolean Optimization (PBO). In simplex-based solutions for solving the LPR from (2), the simplex method adds a set S of slack variables (one for each constraint) such that,

$$\sum_{j=1}^{n} a_{ij} x_j - s_i = b_i \qquad s_i \geq 0, a_{ij}, b_i \in \mathbb{Z} \tag{3}$$

This formulation is called the slack formulation and it is used to create the original simplex tableau [22].

If the solution x^* of the LPR is integral, then x^* provides the optimal solution to the original problem. Otherwise, choose a basic[1] variable x_j such that its value on the LPR

1. See for example [22] for a definition of basic and non-basic variables.

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 [15], Gomory proves that the inequality,

$$\sum_{i \in P} f(\alpha_i) x_i + \sum_{i \in Q} f(\beta_i) s_i \geq f(x_j^*) \qquad f(y) = y - \lfloor y \rfloor, y \in \Re \tag{5}$$

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 will yield a tighter lower bound estimate on the value of the PBO instance. Several methods for strengthening the original Gomory cuts have been proposed [5, 8, 16]. In [16], Gomory proves that the cut

$$\begin{array}{l} \sum_{i \in P} g(\alpha_i) x_i + \sum_{i \in Q} g(\beta_i) s_i \geq 1 \\ \text{where } g(y) = \begin{cases} \frac{f(y)}{f(x_j^*)} & : \ f(y) \leq f(x_j^*) \\ \frac{1-f(y)}{1-f(x_j^*)} & : \ f(y) > f(x_j^*) \end{cases} \end{array} \tag{6}$$

is stronger than (5) and satisfied by all solutions of (3).

Observe that from (3) each slack variable depends only from the original problem variables and can be replaced in (6) by $s_i = \sum_{j=1}^{n} a_{ij} x_j - b_i$. 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.

One should note that in a modern SAT-based algorithm, a conflict analysis procedure is carried out whenever a conflict arises [20, 21]. 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. In the section 4.3 we propose conditions for associating dependencies with computed cutting planes, thus enabling constraint learning and non-chronological backtracking from constraints inferred with cutting plane techniques.

4.2 Clique Cuts

Like Gomory cuts, Clique cuts [7, 22] 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. To show the use of Clique cuts, suppose that among other constraints in our pseudo-boolean formula we have,

$$x_1 + \bar{x}_2 \leq 1 \quad x_1 + x_3 \leq 1 \quad \bar{x}_2 + x_3 \leq 1 \tag{7}$$

From these constraints, we can infer that the set of assignments $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$ are not compatible in the pseudo-boolean formula and so we can safely add a new clique constraint,

$$x_1 + \bar{x}_2 + x_3 \leq 1 \tag{8}$$

Note that when solving the LPR for the formula with the new inferred constraint (8), several possible solutions for the LPR are cut. For example, the assignments $x_1 = x_2 = x_3 = 0.5$ satisfy the original constraints (7), but do not satisfy (8). In general, we can build a conflict graph in order to represent all incompatible assignments for a pseudo-boolean formula (see details in [4]). 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,

$$\sum_{i \in C} l_i \leq 1 \tag{9}$$

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 [22]. As a result, a heuristic greedy procedure is often used.

4.3 Dependencies from Gomory Mixed-Integer Cuts

In order to integrate Gomory mixed-integer cuts in a SAT-based approach we must associate with each cutting plane a set of literals ω_{cut} that define the cutting plane dependencies[2]. When one literal in ω_{cut} is set to 1, the cut will no longer be active (i.e. the associated constraint will be satisfied). 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 (i.e. the value of the right-hand-side of the cutting plane) in order to satisfy the constraint whenever $l_j = 1$.

One should note that the tableau constraint (4), from which the Gomory mixed-integer 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:

$$\omega_{cut} = \{l : l = 0 \wedge l \in \omega_i \wedge \omega_i \in S\} \tag{10}$$

Note 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 (10), 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.

4.4 Dependencies from Clique Cuts

It was shown in section 4.2 that a clique cut is generated from a clique in a conflict graph representing incompatible assignments to problem variables. The set of dependencies of the

2. In [5], an approach is proposed to generate global Gomory mixed-integer cuts that are valid in all nodes of the search tree. However, these do not readily apply to SAT-based PBO.

clique cut is the set of dependencies of all edges in the clique. Therefore, for a given clique cut generated from a clique C in the conflict graph, the set of dependences ω_{cut} can be defined as $\omega_{cut} = \bigcup_{e \in C} \omega_e$, i.e. the set union of all dependencies of each edge in C where ω_e is the set of dependencies of the edge $e \in C$.

As described in [4], an edge is added to the conflict graph through a procedure based on probing assignments. For example, if in the probing of assignment $x_j = 1$ we deduce that $x_i = 1$ is a necessary assignment, then $x_j = 1$ and $x_i = 0$ are incompatible assignments and an edge between nodes x_j and $\bar{x}_i$ can be added to the conflict graph. The dependencies of an edge between these two nodes can be defined as the literals assigned value 0 (at previous levels of the search tree) in the constraints involved in the deduction procedure used to infer that $x_i = 1$ is a necessary assignment if $x_j = 1$. If any of those literals were to have a different value, the probing procedure might not have been able to deduce that $x_j = 1$ implies $x_i = 1$.

5. Search Restarts

Search restarts have been proposed by Gomes et al. [14] and have been successfully applied to SAT [21]. However, despite its success in SAT, search restarts have seldom been used in PBO. Our motivation to use search restarts is that our algorithm not only learns new propositional clauses when conflicts arise, but also learns new constraints by using cutting plane techniques. Hence, at each search restart, the new lower bound at the root node can be higher than in the previous restart. Moreover, since the decision assignment procedure is based on the information provided by the LPR solution, by restarting the search, it is possible that the new decision assignments might drive the search into other areas of the search space where new learned constraints are more effective at pruning the search.

There are several methods to guarantee completeness of backtrack search algorithms with search restarts. One of the approaches is to keep a set of learned constraints (possibly all learned constraints) in order to avoid exploring areas of the search space already explored. However, in our algorithm, we simply increase the cutoff point after each search restart [21]. For each run of the algorithm there is a conflict counter that counts the number of conflicts in that run. In the first run, the algorithm restarts when the counter equals a given number k that defines the initial cutoff. Each time a new restart occurs, the cutoff limit is increased by k. If during a given run of the algorithm, a new solution is found that improves the upper bound value, we reset the conflict counter of that run since the algorithm is being able to improve on its previous solution.

6. Pseudo-Boolean Evaluation

Our solver *bsolo* integrates several features as different lower bounding procedures, cutting plane and problem simplification techniques, search restarts, among others. However, since in the final submission for the Pseudo-Boolean Evaluation, we could only submit one version of the solver, we tried to incorporate several techniques into a hybrid version of *bsolo* to be more robust and provide solutions to a more diversified number of problem instances. For a given instance, *bsolo* starts by using linear search on the value of the cost function and search restarts, increasing the cut off at each restart (as described in section 5). However, after a given number of restarts without improving on the best solution found, we run MIS and LPR lower bound estimation procedures. If the lower bound value provided by MIS is better than LPR or worst by 5% or less, we chose to use MIS. Otherwise, we use LPR with cutting planes. The reason is that if the value of the MIS bound is at least close to the value provided by LPR, then we should use MIS since the overhead of computing the LPR is much larger. Overall, the results for the final version of the solver were largely better than the previous versions submitted for the first stage. Our solver was able to prove optimality in 196 instances and find approximate solutions for another 353 instances. Moreover, we were also able to prove unsatisfiability for 136 instances.

7. Experimental Results

In order to empirically evaluate the techniques described in the paper, we ran *bsolo* on PBO instances from logic synthesis [25]. The *bsolo* solver also incorporates SAT-based techniques, namely unit propagation, non-chronological backtracking in the search tree and conflict-based learning mechanisms [18, 19]. The results presented in the paper were obtained by configuring *bsolo* to use the constraint strengthening technique described in [12] and the simplification techniques described in [24]. Besides *bsolo*, we also ran *Pueblo* [23], *minisat+* [13] and the commercial MILP solver *CPLEX* (version 7.5) [1]. The experimental results are shown in Table1 1. After the column with the instance name, there is the indication of the optimum value of the cost function. Observe that there are some instances for which no solver was able to prove optimality. For *bsolo* we present results for different configurations: using cutting planes described in the paper and using cuts and restarts with different initial cutoffs (100 and 200). The last line in the table provides the total number of instances for which the optimum value was found. The CPU times presented are from a AMD Athlon processor at 1.9 GHz with 1 GB of physical memory. The time limit for each instance was set to one hour. If the time limit is reached, we provide an indication of which was the best upper bound value found when the search was stopped.

Experimental results in Table 1 show that methods based in linear search on the value of the cost function are not suitable to deal with these instances due to their lack of lower bound estimation procedures. On the other hand, *CPLEX* and *bsolo* are able to prove optimality for most of these instances. Moreover, by using search restarts, *bsolo* is able to prove optimality for instance *alu4.b* and find a better solution than *CPLEX* for *e64.b* and *test4.pi*. One should note that *CPLEX* is faster than *bsolo* for some instances since *CPLEX* is a commercial tool, with highly optimized code. Code tuning in *bsolo* is expected to allow significant improvements. Overall the results provide evidence that *bsolo* becomes quite robust with the integrated utilization of cutting planes and search restarts. Indeed, *bsolo* is able to solve instances no other solver can solve, and for other instances *bsolo* is the solver that achieves the lowest upper bound.

8. Conclusions

This paper describes the integration of Gomory mixed-integer cuts and clique cuts in SAT-based algorithms for Pseudo-Boolean Optimization. Moreover, the paper outlines conditions for performing constraint learning and non-chronological backtracking based on previously

					bsolo		
Benchmark	sol.	*Pueblo*	*minisat+*	*CPLEX*	Cuts	rst 100	rst 200
5xp1.b	12	237.53	3244.01	4.43	**3.01**	**3.02**	**3.01**
9sym.b	5	16.75	0.24	**0.14**	0.78	0.79	0.78
alu4.b	50	ub 53	ub 52	ub 50	ub 50	**408.10**	2472.00
apex4.a	776	ub 798	ub 848	**3.92**	55.04	55.02	55.65
bench1.pi	121	ub 209	ub 173	**2.89**	36.09	36.06	36.44
clip.b	15	45.14	6.65	**0.36**	0.88	0.84	0.84
count.b	24	88.42	1835.46	**0.45**	1.07	1.08	1.07
e64.b	–	ub 59	ub 52	ub 49	ub 49	**ub 48**	**ub 48**
ex5.pi	65	ub 99	ub 85	40.59	**33.62**	**33.59**	**33.63**
exam.pi	63	ub 136	ub 90	**4.36**	409.63	138.94	244.54
f51m.b	18	53.19	1336.66	**0.89**	3.38	4.30	3.36
jac3	15	ub 20	ub 15	**0.09**	2.46	2.44	2.44
max1024.pi	259	ub 304	ub 279	**11.41**	ub 260	286.25	1425.00
prom2.pi	287	ub 445	ub 438	**3.34**	117.74	118.41	120.13
rot.b	115	ub 147	ub 123	**71.56**	ub 117	323.99	507.64
sao2.b	25	1605.05	ub 26	**0.50**	3.00	3.00	3.12
test4.pi	–	ub 179	ub 180	ub 103	ub 104	**ub 97**	ub 98
# Opt. Found		6	5	14	12	**15**	**15**

Table 1. Results for logic synthesis instances

inferred cutting planes. These conditions provide novel mechanisms for extending the most effective SAT techniques to PBO, including the ability for backtracking non-chronologically from lower bound conflicts. The paper also proposes the utilization of search restarts, and shows that the constraints learned from identified cutting planes can be useful for accurating the computed lower bounds. Hence, the constraints inferred from identified cutting planes motivate the utilization of search restarts.

Experimental results, obtained on representative binate and unate covering instances (a particular case of PBO), demonstrate that the utilization of cutting planes can be extremely effective. Its integration into a SAT-based framework results in a competitive PBO solver. Besides solving instances that other solvers are unable to solve, for instances which no existing solver is able to solve *bsolo* can obtain tighter approximation values to the optimum.

References

[1] CPLEX MILP solver. http://www.ilog.com/products/cplex/.

[2] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Pearson Education, 1993.

[3] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In *International Conf. on Computer Aided Design*, pages 450–457, November 2002.

[4] A. Atamturk, G. Nemhauser, and M. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121:40–55, 2000.

[5] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[6] P. Barth. A Davis-Putnam Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Technical Report MPI-I-95-2-003, Max Plank Institute for Computer Science, 1995.

[7] R. E. Bixby. Progress in linear programming. *ORSA Journal on computing*, 6(1):15–22, 1994.

[8] S. Ceria, G. Cornuéjols, and M. Dawande. Combining and strengthening Gomory cuts. In Springer-Verlag, editor, *Lecture Notes in Computer Science*, volume 920. E. Balas and J. Clausen (eds.), 1995.

[9] D. Chai and A. Kuehlmann. A Fast Pseudo-Boolean Constraint Solver. In *Design Automation Conference*, pages 830–835, 2003.

[10] O. Coudert. On Solving Covering Problems. In *Design Automation Conference*, pages 197–202, June 1996.

[11] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, July 1962.

[12] H. Dixon and M. Ginsberg. Inference Methods for a Pseudo-Boolean Satisfiability Solver. In *National Conference on Artificial Intelligence*, pages 635–640, 2002.

[13] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–25, 2006. Special Issue on SAT 2005 competition and evaluations.

[14] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *National Conference on Artificial Intelligence*, pages 431–437, July 1998.

[15] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[16] R. Gomory. An algorithm for the mixed-integer problem. Technical Report RM-2597, Rand Corporation, 1960.

[17] S. Liao and S. Devadas. Solving Covering Problems Using LPR-Based Lower Bounds. In *Design Automation Conference*, pages 117–120, June 1997.

[18] V. Manquinho and J. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithms for the binate covering problem. *IEEE Transactions on Computer-Aided Design*, 21(5):505–516, May 2002.

[19] V. Manquinho and J. P. Marques-Silva. Effective lower bounding techniques for pseudo-boolean optimization. In *Design, Automation and Test in Europe Conference*, March 2005.

[20] J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *International Conf. on Computer-Aided Design*, pages 220–227, November 1996.

[21] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, June 2001.

[22] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[23] H. Sheini and K. Sakallah. Pueblo: A hybrid pseudo-boolean sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:157–181, 2006. Special Issue on SAT 2005 competition and evaluations.

[24] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Explicit and Implicit Algorithms for Binate Covering Problems. *IEEE Transactions on Computer Aided Design*, vol. 16(7):677–691, July 1997.

[25] S. Yang. Logic Synthesis and Optimization Benchmarks User Guide. Microelectronics Center of North Carolina, January 1991.