Satisfiability-Based Algorithms for Pseudo-Boolean Optimization Using Gomory Cuts and Search Restarts

Vasco M. Manquinho and João Marques-Silva
IST/INESC-ID, Technical University of Lisbon, Portugal
{vmm,jpms}@sat.inesc-id.pt

Abstract

Cutting planes are a well-known, widely used, and very effective technique for Integer Linear Programming (ILP). In contrast, the utilization of cutting planes in Pseudo-Boolean Optimization (PBO) is recent and results still preliminary. This paper addresses the utilization of cutting planes, namely Gomory mixed-integer 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 proposes the utilization of search restarts in PBO. We show that search restarts can be effective in practice, allowing the computation of more aggressive lower bounds each time the search restarts. Experimental results show that the integration of cutting planes and search restarts in a SAT-based algorithm for PBO yields a very efficient and robust new solution for PBO.

1. Introduction

Recent algorithms for Pseudo-Boolean (PB) Solving (PBS) and Optimization (PBO) have been the subject of significant improvements [2, 6, 9]. The most effective Boolean Satisfiability (SAT) techniques, including clause learning, lazy data structures and conflict-driven branching heuristics, have been extended to PBO. From this work resulted a generation of PB solvers significantly more effective than previous ones [4]. 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 associated with PBO formulations [15, 17, 18].

This paper proposes to apply the identification of cutting planes to SAT-based algorithms for PBO. The objective is to use cutting planes 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. In contrast with the work of [15], which uses cutting planes solely for computing lower bounds, we propose the use of cutting planes for constraint learning, and for backtracking non-chronologically from conflicts involving those constraints. Besides the utilization of cutting planes in SAT-based PBO algorithms, this paper also proposes the utilization of search restarts [10]. Since cutting planes are used for generating new constraints, it is reasonable to assume that search restarts may yield tighter lower bounds each time the search restarts. In practice, we observed that this can indeed be the case.

The paper is organized as follows. The following two sections address, respectively, definitions and a survey of PBO algorithms. Afterwards, Section 4 presents Gomory cutting planes and outlines its integration in SAT-based PBO algorithms. Section 5 addresses the utilization of search restarts and experimental results on representative problem instances are discussed in Section 6. Finally, the paper concludes in Section 7.

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. If a literal denotes $\bar{x}_j$ 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:

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

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). 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 [8, 16, 17].

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. In addition, the PB inference techniques of [6, 9] 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 [22], cutting planes [11] and branch-and-cut [20]. Besides algorithms for generic ILP, algorithms specific to PBO have also been proposed. These include SAT-based algorithms [4], branch-and-bound algorithms [8] and SAT-based algorithms with lower bounding [17, 18].

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 [8], and address SAT-based algorithms for PBO. Moreover, we describe the utilization of linear programming relaxations, an extremely effective technique in PBO algorithms that utilize lower bounding.

3.1. SAT-Based Algorithms

The first SAT-based approach for PBO was proposed by P. Barth [4]. This algorithm is based on the DLL procedure augmented with conditions on the value of the cost function. 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 [2, 6, 9] for several classes of PB instances. The most relevant techniques include non-chronological backtracking in the search tree, conflict-based constraint learning mechanisms and lazy data structures. Observe that most SAT-based approaches focus primarily on finding solutions to the PB constraints. As a result, for highly constrained problem instances, these techniques can be very effective. However, these algorithms are less effective at handling the information provided by the cost function [18].

3.2. Branch-and-Bound Algorithms

Unlike SAT-based algorithms, branch-and-bound algorithms [8, 16] have proved to be very effective for instances that are not highly constrained. 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 pseudo-boolean optimization, 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 [8, 16, 17] for the binate covering problem follow this approach.

For several classes of 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 [8] or linear-programming relaxations [16], among others.

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

Although the approximation of a maximum independent set of constraints (MIS) is the most widely used lower bound procedure for the binate covering problem [8], linear programming relaxation (LPR) has also been used with suc-

cess [16, 18]. It is also often the case that the LPR bound is tighter than the one obtained with the MIS approach.

The general formulation of the LPR for a pseudo-boolean problem instance is obtained from (1) as follows:

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

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

Linear Programming Relaxations are extensively used in Integer Linear Programming (ILP) algorithms for estimating lower bounds on the value of the cost function and also for the identification of cutting planes.

The work on cutting planes can be traced to Gomory [11]. 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, 7, 22]. This section addresses Gomory mixed-integer cuts and its 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

$$\begin{array}{ll} \text{minimize} & z_{lpr} = \sum_{j=1}^{n} c_j \cdot x_j \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j - s_i = b_i \\ & 0 \leq x_j \leq 1, s_i \geq 0, a_{ij}, b_i \in \mathbb{Z} \end{array} \tag{3}$$

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 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 [11], Gomory proves that the inequality,

$$\begin{array}{l} \sum_{i \in P} f(\alpha_i) x_i + \sum_{i \in Q} f(\beta_i) s_i \geq f(x^*_j) \\ \text{where } f(y) = y - \lfloor y \rfloor, y \in \mathbb{R} \end{array} \tag{5}$$

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 [3, 12, 14]. In [12], Gomory proves that the cut

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

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

Notice that from (3) each slack variable depends only from the original problem variables and can be replaced in (6) by,

$$s_i = \sum_{j=1}^{n} a_{ij} x_j - b_i \tag{7}$$

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 (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 [19, 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 next section 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.

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

4.2. Learning from Gomory Mixed-Integer Cuts

The most straightforward solution, for safely determining a set of dependencies for the Gomory mixed-integer 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. The main idea is that if one of the decision assignments were to be different, then the generated cut could not be applied. In this case, we can determine a set of literals ω_{cut} that defines the set of dependencies for the generated cut. 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). 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,

$$\begin{array}{l} \bar{x}_j \in \omega_{cut} \text{ if } x_j = 1 \\ x_j \in \omega_{cut} \text{ if } x_j = 0 \end{array} \tag{8}$$

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 $l_j = 1$.

With this approach, we can safely add any new cuts to our set of pseudo-boolean constraints, guaranteeing the completeness of the algorithm. However, since the cuts depend on all decision assignments made in the search tree from the root node to the current node N where the cut was generated, the constraint will not be used other than at the subtree with root at the node N. Moreover, if a conflict occurs involving the generated cuts at node N, the search cannot backtrack to a node higher than N in the search tree.

The second technique for associating dependencies with cuts follows the ideas proposed in [18] 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. Given the solution to the LPR formulation, the dependencies are identified as all 0-valued literals in all clauses for which the value of the slack variable is 0 [18]. Albeit this technique is more accurate than the first one, it is possible to achieve increased accuracy, by analyzing the process associated with the identification of Gomory mixed-integer cuts.

For describing the third technique, 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:

$$\omega_{cut} = \{l : l = 0 \wedge l \in \omega_i \wedge \omega_i \in S\} \tag{9}$$

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 (9), 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.

5. Search Restarts

Search restarts have been proposed by Gomes [10] 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, each time the search restarts, the 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 also possible that the new decision assignments might drive the search towards new areas of the search space where the 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. Experimental Results

In order to empirically evaluate the techniques described in the paper, we incorporated these techniques in *bsolo* and

ran it on PBO instances from logic synthesis [24] and satisfiable instances of the DIMACS benchmark set [13], using the model described in [23].

The *bsolo* solver also incorporates SAT-based techniques, namely unit propagation, non-chronological backtracking in the search tree and conflict-based learning mechanisms [17, 18]. The results presented in the paper were obtained by configuring *bsolo* to use the constraint strengthening technique described in [9]. Besides *bsolo*, we also ran *PBS* [2], *Galena* [6] and the commercial MILP solver *CPLEX* (version 7.5) [1].[2]

The experimental results are shown in Table 1. After the column with the instance name, there is the indication of the optimum value of the cost function. Notice that there are some instances for which no solver was able to prove optimality [3]. For *bsolo* we present results for different configurations: without using cutting planes, using strengthened Gomory mixed-integer cuts during the search and using both Gomory cuts and restarts with different initial cutoffs (100, 200 and 500). For all *bsolo* configurations, the lower bound estimates were obtained using LPR.

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 show that pure SAT-based algorithms (*PBS* and *Galena*) are not suitable to deal with these logic synthesis instances due to their lack of lower bound estimation procedures. These algorithms can easily find a solution to the constraints, but are unable to identify the optimum value of the cost function. 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.

SAT-based algorithms perform better than *CPLEX* for instances of the minimum-size prime implicant problem, since finding a solution to the problem constraints is harder. Nevertheless, *bsolo* is the most robust algorithm. Not only is *bsolo* able to solve all *aim* instances, essentially exploiting its SAT-based techniques, but is also able to solve the *ssa* instances when using Gomory mixed-integer cuts. Results also show that the use of search restarts allow *bsolo* to perform better for the *ii8* instances, being able to find better solutions than *CPLEX* for several instances.

[2]The *Eclipse* tool [15] is not yet available for benchmarking purposes.

[3]Entries with – denote instances for which a given solver was unable to provide either an upper bound or the optimum value.

7. Conclusions

This paper describes the integration of Gomory mixed-integer cuts in SAT-based algorithms for PBO. In addition, the paper 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, including the ability for backtracking non-chronologically from lower bound conflicts. In clear contrast with the work of [15], our approach uses cutting planes for learning new constraints, and consequently for performing non-chronological backtracking from lower bound conflicts.

Besides the integration of cutting planes in SAT-based algorithms, 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 use of search restarts.

The experimental results show that the utilization of cutting planes can be extremely effective. In fact, *bsolo* using Gomory cuts and search restarts is able to solve problem instances that *no* other PBO solver is able to solve (including the commercial ILP solver *CPLEX*). Moreover, the results give further evidence that the use of lower bounding techniques is essential for developing effective PBO algorithms.

References

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

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

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

[4] 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.

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

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

[7] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

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

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

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

Table 1. Results for logic synthesis and minimum-size prime implicant

						bsolo		***bsolo* with restarts**		
	Benchmark	sol.	*PBS*	*Galena*	*CPLEX*	No Cuts	Gomory	RS 100	RS 200	RS 500
[24]	5xp1.b	12	ub33	341.37	4.43	15.57	16.68	64.75	17.09	17.41
	alu4.b	50	ub121	ub53	ub50	ub50	ub50	593.98	404.00	ub50
	apex4.a	776	ub2282	ub845	3.92	ub776	60.33	61.27	60.17	60.22
	clip.b	15	ub30	1.68	0.36	9.65	2.15	2.17	2.14	2.17
	e64.b	–	ub99	ub53	ub49	ub50	ub50	ub48	ub48	ub48
	ex5.pi	65	ub594	ub99	40.59	224.48	253.30	408.05	251.71	258.56
	f51m.b	18	ub40	1.94	0.89	8.68	4.28	5.71	3.22	4.18
	max1024.pi	259	ub454	ub324	11.41	210.15	200.56	148.09	196.62	195.47
	rot.b	115	ub745	ub142	71.56	ub117	ub117	156.48	911.96	150.94
	sao2.b	25	ub39	132.91	0.50	11.3	5.70	5.73	5.76	6.02
	test4.pi	–	ub1288	ub2804	ub103	ub104	ub104	ub99	ub98	ub98
[23]	aim-100-1_6-y1-2	100	0.01	0.01	373.42	0.04	0.32	0.32	0.33	0.33
	aim-100-3_4-y1-4	100	0.01	–	108.79	2.73	7.04	7.52	6.93	7.03
	aim-200-1_6-y1-3	200	0.01	–	–	0.11	5.11	5.23	5.10	5.10
	aim-200-6_0-y1-2	200	0.01	0.03	ub200	140.18	0.69	2.17	1.19	0.69
	aim-50-2_0-y1-2	50	0.01	–	4.11	0.04	0.19	0.19	0.18	0.19
	aim-50-3_4-y1-3	50	0.01	–	2.05	0.32	0.54	0.55	0.54	0.60
	ii8a1	54	1.78	0.18	0.30	1.15	2.12	2.17	2.12	2.26
	ii8b1	191	ub272	ub213	7.11	2285.50	3103.40	1358.40	1590.60	1985.80
	ii8c1	302	ub405	ub399	2719.59	ub416	ub357	ub302	ub302	ub302
	ii8d1	–	ub515	ub432	ub351	ub470	ub381	ub346	ub359	ub359
	ii8e1	–	ub438	ub448	ub357	ub400	ub378	ub343	ub361	ub350
	jnh12	94	0.02	0.01	7.59	0.31	0.23	0.08	0.22	0.22
	jnh7	89	0.02	0.04	7.42	20.55	6.82	6.98	7.08	6.82
	ssa7552-038	1448	ub1448	ub1448	ub1448	ub1452	3090.30	2405.30	3072.00	2691.00
	ssa7552-160	1359	ub1359	ub1359	ub1359	ub1359	3417.00	1930.00	2002.20	ub1359

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

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

[13] D. S. Johnson and M. A. Trick. Second DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1994.

[14] A. Letchford and A. Lodi. Strengthening chvatal-gomory cuts and gomory fractional cuts. *Operations Research Letters*, 30(2):74–82, 2002.

[15] X.-Y. Li, M. Stallmann, and F. Brglez. Effective Bounding Techniques for Solving Unate and Binate Covering Problems. In *Proc. Design Automation Conference*, June 2005.

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

[17] 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.

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

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

[20] J. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. In *Handbook of Applied Optimization*, pages 65–77. Oxford University Press, 2002.

[21] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Proc. 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] C. Pizzuti. Computing Prime Implicants by Integer Programming. In *Proc. International Conference on Tools with Artificial Intelligence*, pages 332–336, November 1996.

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