# Effective Lower Bounding Techniques for Pseudo-Boolean Optimization

Vasco M. Manquinho, Joao Marques-Silva

▶ **To cite this version:**

Vasco M. Manquinho, Joao Marques-Silva. Effective Lower Bounding Techniques for Pseudo-Boolean Optimization. DATE'05, Mar 2005, Munich, Germany. pp.660-665. hal-00181183

**HAL Id: hal-00181183**
**https://hal.science/hal-00181183**

Submitted on 23 Oct 2007

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Effective Lower Bounding Techniques for Pseudo-Boolean Optimization

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

## Abstract

*Linear Pseudo-Boolean Optimization (PBO) is a widely used modeling framework in Electronic Design Automation (EDA). Due to significant advances in Boolean Satisfiability (SAT), new algorithms for PBO have emerged, which are effective on highly constrained instances. However, these algorithms fail to handle effectively the information provided by the cost function of PBO. This paper addresses the integration of lower bound estimation methods with SAT-related techniques in PBO solvers. Moreover, the paper shows that the utilization of lower bound estimates can dramatically improve the overall performance of PBO solvers for most existing benchmarks from EDA.*

## 1. Introduction

Recent advances in Boolean Satisfiability (SAT) have resulted in new practical algorithms for solving the Linear Pseudo-Boolean Optimization problem [2, 4, 6]. These algorithms perform a linear search on the possible values of the cost function, starting from the highest, 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 optimal value is given by the last computed solution. By incorporating important features from SAT solvers, like non-chronological backtracking in the search tree, conflict-based learning mechanisms and lazy data structures, these solvers have been able to solve with success several classes of highly constrained pseudo-boolean instances. However, these solvers also fail to handle effectively the information provided by the cost function, which can be crucial as the experimental results in this paper clearly demonstrate. In order to prune the search due to the value of the cost function we propose the use of methods to estimate a lower bound on the value of the cost function. Whenever the lower bound estimation is higher or equal to the best solution found so far, we are able to prune the search tree. Moreover, we also establish conditions for backtracking non-chronologically in the search tree when the search backtracks due to the lower bound estimate. This paper extends [9] by considering alternative lower bounding procedures.

The paper is organized as follows. The first part concentrates on describing how different lower bound estimation methods can be applied in pseudo-boolean optimization problem, focusing on linear-programming relaxation and Lagrangian relaxation. Section 4 proposes techniques for obtaining explanations that allow backtracking non-chronologically when the search is bound due to the lower bound estimate. In addition, the paper presents a number of additional techniques that have proved useful in practical PBO solving. Preliminary experimental results are analyzed in section 6 for most existing PBO benchmarks for EDA problems. 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 $\bar{x}_j$. 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 a Linear Pseudo-Boolean Optimization problem can be defined as follows,

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

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

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)
1530-1591/05 $ 20.00 IEEE
IEEE COMPUTER SOCIETY

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 that can be found in [5, 9, 15].

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 $1 - x_j$. In section 3 we will use this latter formulation.

## 3. Pseudo-Boolean Optimization Algorithms

In [3], P. Barth first proposed an approach based on Boolean Satisfiability (SAT) techniques for solving Pseudo-Boolean Optimization (PBO). This approach consists of performing a linear search on the possible values of the cost function, starting from the highest, 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 resulted in new successful algorithms [2, 4, 6] for several sets of PBO instances, namely the incorporation of non-chronological backtracking in the search tree, conflict-based learning mechanisms and lazy data structures have been applied with success. The SAT-based approach focuses primarily on finding solutions for the problem constraints. Therefore, for highly constrained problems these techniques are very effective. However, these algorithms find it difficult to deal with the information from the cost function.

Unlike the SAT-based approach, branch-and-bound algorithms [5, 8] have proved to be very effective when the instances to be solved are not highly constrained since they are able to prune the search tree earlier due to estimate of 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 set of variable assignments. For a given instance $P$ of a pseudo-boolean optimization problem, let $P.upper$ denote the upper bound on the value of the cost function. The search is pruned whenever the lower bound estimation is higher 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 [5, 8, 9, 15] for the binate covering problem follow this approach as well as several general integer programming solvers.

For several instances, specially for low 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) [5, 9], linear-programming relaxations [8] or Lagrangian relaxations [12].

### 3.1. 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 (a particular case of PBO) [5, 15], linear programming relaxation (LPR) has also been used with success [8] . It is also often the case that the linear programming relaxation bound is higher than the one obtained with the MIS approach. Moreover, linear programming relaxations have long been used as a lower bound estimation procedure in branch-and-bound algorithms for solving integer programming problems [11].

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

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

where vector $c$ defines the non-negative integer cost associated with every decision variable in vector $x$. Entries of matrix $A$ defines the constraint coefficients and vector $b$ the right-hand side of every constraint. The solution of (1) is referred to as $z^*_{cp}$, 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 $z^*_{cp}$ of (1) [11]. 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 $z^*_{cp} = z^*_{lpr}$. Hence, the result follows. Furthermore, different linear programming algorithms can be used for solving (2), some with guaranteed worst-case polynomial run time [11].

### 3.2. Lagrangian Relaxations

Lagrangian relaxation (LGR) is a widely used method for computing bounds on the optimal value of the cost function from network optimization to nonlinear programming [12, 13]. It also known that for some instances, the bound provided by the Lagrangian relaxation method is tighter than the one obtained by the linear programming relaxation [12]. Therefore, Lagrangian relaxation can be used to provide a quick and tight lower bound on the value of the cost function for pseudo-boolean optimization problems.

While in linear programming relaxations we are able to find a lower bound estimate by solving the problem constraints and relaxing the possible variable values, in Lagrangian relaxations we relax the problem constraints and


IEEE COMPUTER SOCIETY

incorporate them in the objective function with associated Lagrangian multipliers.

Given a generic linear optimization problem formulated as:

$$\begin{array}{ll} \text{minimize} & z^* = cx \\ \text{subject to} & Ax = b \\ & x \in X \end{array} \quad (3)$$

we can define the *Lagrangian function* $L(\mu)$ as:

$$L(\mu) = min\{cx + \mu(Ax - b) : x \in X\} \quad (4)$$

where vector $\mu$ defines the *Lagrangian multiplier* associated with each constraint. The Lagrangian Bounding Principle [12] states that for any vector $\mu$ of the Lagrangian multipliers, the value of $L(\mu)$ is a lower bound on the optimal solution of the original optimization problem.

In (3) all constraints are formulated as equalities, while in the pseudo-boolean optimization problem (1) we have inequality constraints. Therefore, in that case the Lagrangian relaxation problem is formulated as:

$$L^* = max\{L(\mu) : \mu \geq 0\} \quad (5)$$

where $L^*$ is the optimum value of the Lagrangian relaxation. The most tight lower bound estimate we can obtain using this method is given by $\lceil L^* \rceil$.

Before trying to solve the Lagrangian relaxation problem in order to obtain $L^*$, we must determine the value of $L(\mu)$ for a given value of $\mu$. Notice that by expanding (4) we have:

$$\begin{array}{l} L(\mu) = min\{\sum\limits_{j \in N} c_j x_j + \sum\limits_{i \in M} \mu_i((\sum\limits_{j \in N} a_{ij} x_j) - b_i)\} \\ L(\mu) = min\{\sum\limits_{j \in N} \alpha_j x_j - \sum\limits_{i \in M} \mu_i b_i\} \text{ where} \\ \alpha_j = (c_j + \sum\limits_{i \in M} \mu_i a_{ij}) \end{array} \quad (6)$$

In order to obtain the value of $L(\mu)$, for a given $\mu$, we must determine the value of the decision variables $x_j$ to be able to minimize the expression. Therefore, we must have $x_j = 0$ whenever $\alpha_j \geq 0$ and $x_j = 1$ when $\alpha_j < 0$.

Literature from nonlinear programming [13] and network optimization [12] provide methods to solve (5), namely gradient methods to approximate the value of $L^*$. For the results presented in this paper the approach outline in [12] is used.

## 4. Bound-based Conflicts

Conditions for backtracking non-chronologically due to the lower bound estimate on the value of the cost function first proposed in [9] assume that the lower bounding procedure is the MIS approximation. Next, we review the main ideas about pruning the search tree based on the estimated value of the cost function and describe the conditions to apply when using linear-programming relaxation or Lagrangian relaxation as a lower bound estimation procedure.

### 4.1. Backtracking on Bound-based Conflicts

A bound conflict in an instance of the pseudo-boolean optimization problem (PBO) *P* arises when the lower bound is equal to or higher than the upper bound. This condition can be written as

$$P.path + P.lower \geq P.upper \quad (7)$$

where *P.path* is the cost of the assignments already made, *P.lower* is a lower bound estimate on the cost of satisfying the constraints not yet satisfied (as given for example using Lagrangian relaxation), and *P.upper* is the best solution found so far.

In this situation, our approach is to identify a set of assignments responsible for the bound conflict and build a new propositional clause $\omega_{bc}$ such that it prevents those assignments of being repeated during the search process. When $\omega_{bc}$ is added, a conflict analysis procedure must be carried out to determine to which level of the search tree to backtrack to.

A straightforward approach to build $\omega_{bc}$ would be to consider the decision variable assignments from all levels of the search tree, but in that case the resulting backtrack would necessarily be chronological. In [9] it was already shown that the assignments responsible for the bound conflict might not be associated with all levels of the search tree.

From (7), we can readily conclude that *P.path* and *P.lower* are the unique components involved in each bound conflict. Therefore, we will analyze both the *P.path* and *P.lower* components in order to establish the assignments responsible for a given bound conflict. Our goal is to define two sets of literals $\omega_{pp}$ and $\omega_{pl}$ containing the explanation for *P.path* and *P.lower*, respectively. Our bound conflict clause $\omega_{bc}$ is defined by the set union of the literals in $\omega_{pp}$ and $\omega_{pl}$.

We start by studying *P.path*. Clearly, the variable assignments that cause the value of *P.path* to grow are solely those assignments with a value of 1. Hence, we can define $\omega_{pp}$ such that each variable in $\omega_{pp}$ has positive cost and is assigned value 1:

$$\omega_{pp} = \{l = \bar{x}_j : Cost(x_j) > 0 \wedge x_j = 1\} \quad (8)$$

which basically states that in order to decrease the value of the cost function (i.e.*P.path*) at least one variable that is assigned value 1 has instead to be assigned value 0.

We now consider *P.lower*. Since different lower bound estimation procedures can be used, we will describe in the remainder of this section how to identify an explanation for the bound conflict when using either linear-programming relaxation or Lagrangian relaxation.


IEEE COMPUTER SOCIETY

### 4.2. Lower Bound Conflicts from Linear-Programming Relaxation

When using linear-programming relaxations as a lower bound estimation procedure, the value of $P.lower$ is obtained according to the formulation described in section 3.1. In order to determine the set of assignments we can deem responsible for $P.lower$, we must define $S$ as the set of constraints with *slack*[1] variables assigned value 0 in the linear program solution. These are the constraints which actually limit the value of $P.lower$. If the literals that assume value 0 in these constraints were to have a different value, some constraints might be satisfied and the value of $P.lower$ would be lower. Therefore, we can consider the assignments to those literals as the responsible for $P.lower$ and define $\omega_{pl}$ as:

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

Clearly, $\omega_{pl}$ does not necessarily depend on all decision levels in the search tree. Hence, non-chronological backtracking can result from the conflict analysis procedure.

### 4.3. Lower Bound Conflicts from Lagrangian Relaxation

In order to determine $\omega_{pl}$ using the Lagrangian relaxation lower bound estimation procedure as described in section 3.2, we can follow a similar approach to the one described for linear-programming relaxation. Let $S$ be the set of constraints used in obtaining the value of $P.lower$ whose Lagrangian multiplier is different from 0. We can clearly notice from (6) that the constraints with Lagrangian multiplier equal to 0 are irrelevant for computing $P.lower$. In this case, $\omega_{pl}$ can be determined as formulated in (9).

Another approach to determine $\omega_{pl}$ is to consider the value of $\alpha_j$ for each assigned variable from $S$. If a given variable $x_j$ is assigned value 0 and $\alpha_j > 0$, then by changing its value to 1 we would increase the value of $P.lower$. Or if variable $x_j$ is assigned value 1 and $\alpha_j < 0$, if we were to change the value of $x_j$, $P.lower$ would raise. Hence, these assignments cannot be deemed responsible for the value of $P.lower$ and should not be considered in $\omega_{pl}$.

## 5. Additional Techniques

In this section we describe additional techniques that have proved useful in implementing a branch-and-bound based PBO tool.

When using LPR for computing lower bounds, one can use the information provided by the LP solution for branching purposes. Essentially, branching is restricted to variables for which the LP solution is *not* integer. Of these variables, the one closest to 0.5 is selected. In the case more than one variable has been assigned value 0.5, then the VSIDS heuristic of Chaff is applied [10].

Another additional technique is the generation of a new constraint when a new better solution is found. When this occurs, $P.upper$ is updated, and one can generate a new constraint:

$$\sum_{j \in N} c_j x_j \leq P.upper - 1 \tag{10}$$

which requires that a new solution must have cost lower that the lowest cost already identified for a solution. This type of constraint is referred to as a knapsack constraint [11]. Suppose also the existence of a cardinality constraint of the form:

$$\sum_{j \in K} x_j \geq U, \quad K \subseteq N \tag{11}$$

In this case we can infer a new constraint, that can be added to the set of constraints. Let $V$ be the sum of the $U$ smallest coefficients $c_j$ of variables denoted by set $K$. Thus we can conclude:

$$\sum_{j \in K} c_j x_j \geq V \tag{12}$$

As a result, the following constraint can be inferred:

$$\sum_{j \in N-K} c_j x_j \leq P.upper - 1 - V \tag{13}$$

## 6. Experimental Results

In this section we present empirical results for the techniques described in the paper using our pseudo-boolean optimizer (*bsolo*) which incorporates the branch-and-bound techniques described in the paper and SAT-based techniques, namely boolean constraint propagation, non-chronological backtracking in the search tree and conflict-based learning mechanisms.

The CPU times presented are for a AMD Athlon processor at 1.9GHz with 1GB of physical memory. 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 (**ub**) value found when the search was stopped.

In order to empirically evaluate the lower bound procedures described in the paper, we ran our solver on most PBO instances available, most of which from EDA applications [2, 18, 17, 16] [2]. Besides bsolo, we also ran PBS [2], Galena [4] and the commercial MILP solver CPLEX (version 7.5) [1]. For bsolo, we present results when using different lower bound methods (MIS, LGR and LPR) and also when no lower bound estimation procedure is used (plain).

The bsolo solver was configured to use the constraint strengthening technique described in [6] and widely used

1 See [13] for a definition of slack and artificial variables.

2 Observe that the ones in [18] represent synthesis problems for mixed PTL/CMOS circuits, and the ones in [16] represent PB satisfaction problems.


IEEE COMPUTER SOCIETY

in mixed integer programming [14]. The probing used in the constraint strengthening is also used to detect necessary assignments during preprocessing. We also used simplification techniques described in [7, 15] in the synthesis benchmark set.

The results are presented in Table 1. As can be readily concluded, different solvers perform better for different sets of instances. In any case, there are a few clear trends. bsolo (with LPR) is by far the most regular solver, and overall the best performing solver, being able to solve 35 instances out of a total of 40 instances. The other publicly available PBO solvers, PBS and Galena, are solely able to solve 17 and 26 instances, respectively. It should be noted that the general purpose MILP solver CPLEX is extremely competitive, being clearly better that both PBS and Galena over all instances. It is also clear that CPLEX is better suited for optimizing a cost function than for solving a set of pseudo-boolean constraints, as the results for the last benchmarks illustrate.

Despite being a more regular and overall better performing algorithm, bsolo performs worse than both PBS and Galena for the last class of instances [16]. This is essentially due to a significantly larger time per decision required by bsolo, motivated by a less optimized implementation of pseudo-boolean constraint propagation. Finally, bsolo with LPR is significantly more efficient than bsolo with LGR. This is motivated by the slow convergence observed for the Lagrangian relaxation on most instances.

## 7. Conclusions

The paper describes the integration of lower bound estimation procedures with SAT-based techniques in linear pseudo-boolean optimization, emphasizing two lower bounding methods: linear-programming relaxation and Lagrangian relaxation. The paper also outlines a procedure for enabling non-chronological backtracking in the search tree when bound conflicts occur. In addition, the paper proposes a number of additional techniques, including an informed branching heuristic, and a technique for inferring constraints from knapsack and cardinality constraints.

Preliminary experimental results, obtained on most of existing PBO benchmarks for EDA problems, are clear: bsolo is the most effective PBO solver, far more effective than either PBS [2] or Galena [4], and is the only dedicated PBO solver that is competitive with (and overall significantly more efficient than) the commercial MILP solver CPLEX [1].

The very promising experimental results clearly demonstrate that the integration of lower bounding techniques is a crucial aspect in the development of PBO solvers, allowing bsolo to be dramatically more efficient than the best publicly available PBO solvers, PBS and Galena, which do not utilize lower bounding techniques. As the experimental results also suggest, the time per decision in bsolo is larger than in either PBS and Galena. This motivates fine-tuning the implementation and data structures of the bsolo prototype.

## 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 *ACM/IEEE International Conference on Computer Aided Design*, pages 450–457, November 2002.

[3] 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.

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

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

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

[7] J. Hooker. Logic-Based Methods for Optimization. In . Jon Wiley & Sons, 1996.

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

[9] 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.

[10] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. ACM/IEEE Design Automation Conference*, pages 530–535, June 2001.

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

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

[13] A. S. S. Nash. Linear and Nonlinear Programming. In . McGraw-Hill, 1996.

[14] M. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[15] 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.

[16] J. Walser. 0-1 integer programming benchmarks. http://www.ps.uni-sb.de/∼walser/benchmarks/-benchmarks.html.

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

[18] Z. Zhu. Synthesis for mixed ptl/cmos circuit. http://www-unix.ecs.umass.edu/∼zzhu/.





IEEE COMPUTER SOCIETY

| Ref. | Benchmark | Sol. | pbs | galena | cplex | bsolo plain | bsolo MIS | bsolo LGR | bsolo LPR |
|---|---|---|---|---|---|---|---|---|---|
| [2] | grout-4-3-1 | 62 | ub 64 | 2.85 | 0.21 | ub 64 | ub 64 | ub 64 | 0.28 |
| | grout-4-3-2 | 64 | ub 66 | 19.42 | 0.07 | ub 66 | ub 64 | ub 64 | 3.15 |
| | grout-4-3-3 | 62 | ub 66 | 4.11 | 0.07 | ub 64 | ub 64 | ub 66 | 1.34 |
| | grout-4-3-4 | 60 | ub 62 | 6.51 | 0.04 | ub 60 | ub 60 | ub 60 | 0.49 |
| | grout-4-3-5 | 60 | ub 64 | 6.02 | 0.04 | ub 62 | ub 62 | 2316.90 | 5.90 |
| | grout-4-3-6 | 66 | 617.61 | 8.26 | 0.05 | ub 66 | ub 66 | 832.94 | 3.51 |
| | grout-4-3-7 | 64 | 1334.94 | 0.86 | 0.08 | ub 66 | ub 66 | 2777.80 | 2.00 |
| | grout-4-3-8 | 36 | ub 44 | 17.90 | 0.06 | ub 40 | 441.06 | 37.23 | 0.51 |
| | grout-4-3-9 | 68 | 1227.29 | 2.24 | 0.06 | 654.70 | 2106.5 | 204.32 | 1.73 |
| | grout-4-3-10 | 70 | 36.16 | 0.36 | 0.21 | 174.02 | 185.81 | 5.42 | 0.43 |
| [18] | 9symml | 4517 | ub 6453 | ub 6986 | 1.63 | ub 20818 | ub 16792 | 347.13 | 328.97 |
| | C17 | 260 | 0.00 | 0.01 | 0.00 | 0.00 | 0.00 | 0.00 | 0.02 |
| | C432 | 4822 | ub 6577 | ub 8070 | 3.34 | ub 25207 | ub 23014 | 279.25 | ub 4822 |
| | b1 | 128 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 |
| | c8 | 1194 | ub 1542 | ub 1528 | 0.19 | ub 4350 | ub 3819 | 0.19 | 0.23 |
| | cc | 1567 | ub 1692 | ub 1786 | 0.00 | ub 3305 | ub 2738 | 0.09 | 0.08 |
| | cm42a | 694 | ub 754 | ub 696 | 0.05 | ub 1072 | ub 789 | 0.06 | 0.05 |
| | cmb | 1053 | ub 1490 | ub 1476 | 0.03 | ub 7113 | ub 5520 | 0.43 | 0.24 |
| | mux | 872 | ub 1321 | ub 1333 | 0.01 | ub 3430 | ub 2060 | 0.00 | 0.06 |
| | my_adder | 4561 | ub 6271 | ub 5548 | 2.29 | ub 19798 | ub 12012 | 0.77 | 14.24 |
| [17] | 5xp1.b | 12 | ub 33 | 341.37 | 4.43 | ub 14 | 78.17 | ub 12 | 15.57 |
| | 9sym.b | 5 | 1718.98 | 0.26 | 0.14 | ub 5 | 2.34 | ub 6 | 0.89 |
| | alu4.b | – | ub 121 | ub 53 | ub 50 | ub 52 | ub 52 | ub 50 | ub 50 |
| | apex4.a | 776 | ub 2282 | ub 845 | 3.92 | ub 785 | ub 783 | ub 781 | ub 776 |
| | bench1.pi | 121 | ub 1366 | ub 784 | 2.89 | ub 134 | ub 131 | ub 123 | 4.13 |
| | clip.b | 15 | ub 30 | 1.68 | 0.36 | ub 16 | 49.11 | ub 15 | 9.64 |
| | count.b | 24 | ub 48 | ub 25 | 0.45 | ub 25 | ub 25 | ub 25 | 0.81 |
| | e64.b | – | ub 99 | ub 53 | ub 49 | ub 54 | ub 53 | ub 48 | ub 50 |
| | f51m.b | 18 | ub 40 | 1.94 | 0.89 | ub 19 | 6.62 | ub 18 | 8.58 |
| | rot.b | 115 | ub 745 | ub 142 | 71.56 | ub 123 | ub 122 | ub 117 | ub 117 |
| [16] | acc-tight:0 | SAT | 1.00 | 0.05 | 1.55 | 4.78 [a] | 4.78 [a] | 4.78 [a] | 4.78 [a] |
| | acc-tight:1 | SAT | 1.28 | 0.06 | 52.17 | 2.20 [a] | 2.20 [a] | 2.20 [a] | 2.20 [a] |
| | acc-tight:2 | SAT | 0.78 | 0.09 | 141.28 | 2.30 [a] | 2.30 [a] | 2.30 [a] | 2.30 [a] |
| | acc-tight:3 | SAT | 2.14 | 0.29 | time | 43.61 [a] | 43.61 [a] | 43.61 [a] | 43.61 [a] |
| | acc-tight:4 | SAT | 14.23 | 2.13 | time | 345.26 [a] | 345.26 [a] | 345.26 [a] | 345.26 [a] |
| | acc-tight:5 | SAT | 20.06 | 1.35 | 1897.48 | 57.82 [a] | 57.82 [a] | 57.82 [a] | 57.82 [a] |
| | acc-tight:6 | SAT | 26.98 | 0.61 | time | 41.90 [a] | 41.90 [a] | 41.90 [a] | 41.90 [a] |
| | acc-tight:7 | SAT | 12.84 | 1.28 | time | 7.85 [a] | 7.85 [a] | 7.85 [a] | 7.85 [a] |
| | acc-tight:8 | SAT | 9.51 | 2.77 | time | 58.80 [a] | 58.80 [a] | 58.80 [a] | 58.80 [a] |
| | acc-tight:9 | SAT | 10.44 | 0.20 | time | 36.51 [a] | 36.51 [a] | 36.51 [a] | 36.51 [a] |
| | #Solved | 40 | 17 | 26 | 32 | 14 | 19 | 26 | 35 |

*a* For the instances from [16] the CPU times for the different versions of bsolo are the same. No lower bound estimation is used, since there is no cost function.

**Table 1. Experimental results**


IEEE COMPUTER SOCIETY