Solving Optimization Problems with Boolean Combinations of Pseudo-boolean Constraints (a preliminary report)

Lengning Liu and Mirosław Truszczyński

Department of Computer Science, University of Kentucky,
Lexington, KY 40506-0046, USA

Abstract. We study the optimization problems where the constraints are boolean combinations of pseudo-boolean constraints and the objective function is a linear function with integer coefficients and 0-1 variables. We call such optimization problems $PL(PB)$ optimization problems. $PL(PB)$ optimization problems generalize the well known pseudo-boolean optimization problems where each constraint is a single pseudo-boolean constraint. We propose a method that solves $PL(PB)$ optimization problems via a stochastic local search $PL(PB)$ solver called $wsat(plpb)$. Our method iteratively runs $wsat(plpb)$ to improve the value of the objective functions. In addition to the linear search, which most methods that solve pseudo-boolean optimization problems use, our method provides an option that uses a combination of linear and binary search, which reduces the number of time $wsat(plpb)$ is executed. We perform experimental study on our implementation. We compare our method to existing pseudo-boolean optimizers on a set of instances. Transformation is needed as those optimizers do not accept boolean combinations of pseudo-boolean constraints. The result shows that, except for one instance, our method is uniformly better than existing methods on the rest of the instances.

1 Introduction

We propose a method that computes optimal (or sub-optimal) solutions to optimization problems with boolean combinations of pseudo-boolean ($PL(PB)$ for short) constraints.

Informally, an optimization problem consists of an objective function and a set of constraints. To solve an optimization problem, we need to maximize or minimize the objective function subject to the constraints. Problems such as finding a minimal dominating set in a graph are optimization problems.

In our setting, we represent the constraints of an optimization problem as boolean combinations of pseudo-boolean constraints, a formalism proposed by [1]. By a *pseudo-boolean constraint* (or a PB constraint for short) we mean an integer programming constraint with only 0-1 variables. By a *boolean combination*, we mean a disjunction of PB constraints.

Optimization problems with only PB constraints (no disjunctions of PB constraints) have received much attention during the past decade. Recently many optimizers that

solve PB optimization problems have emerged, including $wsat(oip)$ [2], $minisat+$ [3], $pb2sat + zchaff$ [4], $bsolo$ [5], $pueblo$ [6] and PBS [7].

To obtain an optimal solution, the optimizers we listed above rely on a series of queries to programs, called *model generators*, that compute (generate) models of sets of PB constraints. Most PB optimizers perform the linear search on the value of the objective function. Other approaches include the binary search ($pb2sat + zchaff$) and a SAT-based branch and bound method ($bsolo$).

In the linear search, the optimizer first queries the model generator for a solution to the set of PB constraints, disregarding the objective function. Then the optimizer iteratively improves the value of the objective function, each time introducing a new PB constraint saying that the value of the objective function should be less than the one found in the previous step. When the query to the model generator finally results in the "failed-to-satisfy" answer, the process terminates and the previous value of the objective function is returned. If the model generator is complete and the "failed-to-satisfy" answer is the result of a normal termination without finding a model, the returned value is optimal. If the model generator terminates due to exhausting the CPU time allocated (incomplete model generators are covered by this case), the "failed-to-satisfy" message does not mean that the instance is unsatisfiable. In such cases, the value returned is only an approximation of the optimal one.

The main reason behind the use of linear search instead of binary search, is that, deciding unsatisfiability of a set of PB constraints typically takes longer than deciding satisfiability. In the first case, the whole search space must be considered, in the second one we can stop as soon as the first model is found. The linear search terminates with only one "failed-to-satisfy" result. The binary search often needs significantly fewer iterations to terminate but many of these iterations may return the "failed-to-satisfy" message, often requiring much more time.

In this paper, we propose an algorithm that computes solutions of $PL(PB)$ optimization problems. Our optimizer uses a $PL(PB)$ model generator $wsat(plpb)$ [1]. Since $wsat(plpb)$ is a local-search algorithm, solutions returned by our optimizer are not guaranteed to be optimal. For the search component that successively improves on the quality of a solution, our optimizer provides two options: (1) the linear search (most often used in other optimizers), and (2) a novel combination of the linear search with a variant of the binary search. As we noted above, the linear search executes exactly one call to the model generator which results in the "failed-to-satisfy" message. Our hybrid method is designed so that there are exactly two calls to the model generator returning the "failed-to-satisfy" message, a significant improvement over the straightforward binary search approach.

We perform an experimental study on the algorithm we propose and compare it to existing optimizers for PB optimization problems. Results show that our method performs better than PB optimizers in the benchmark instances we use, except for just one instance that is highly structured.

The paper is organized as follows: Section 2 gives definitions and examples for $PL(PB)$ optimization problems. Section 3 describes our method of computing optimal solutions to $PL(PB)$ optimization problems. Section 4 shows the experimental results. Section 5 concludes the paper and gives possible future research directions.

2 Preliminaries

A *pseudo-boolean (or PB) constraint* is an integer inequality of the form

$$\sum a_i x_i \geq b,$$

where a_i's and b are integers and x_i's are 0-1 variables. A PB *theory* is a finite collection of PB constraints. A value assignment v that assigns 0 or 1 to all variables in the PB constraint *satisfies* (or is a *model* of) the constraint if $\sum a_i \times v(x_i) \geq b$ holds. An expression

$$5x_1 + (-6)x_2 + 2x_3 \geq 2$$

is an example of a PB constraint. One can verify that $x_1 = 1, x_2 = 0, x_3 = 1$ is a satisfying value assignment of the PB constraint. A value assignment *satisfies* a PB theory if it satisfies all PB constraints in the theory.

We now define a formalism [1] that generalizes PB theories. A $PL(PB)$ *constraint* is a disjunction of PB constraints. A $PL(PB)$ *theory* is a finite collection of $PL(PB)$ constraints. A value assignment *satisfies* a $PL(PB)$ constraint if it satisfies at least one PB constraint in the disjunction. A value assignment *satisfies* a $PL(PB)$ theory if it satisfies all $PL(PB)$ constraints in the theory. As pointed out in [1], many practical constraints involve disjunctions of numerical properties on sets of weighted elements. Thus allowing boolean combinations of PB constraints facilitates modeling this type of constraint.

A $PL(PB)$ *optimization problem* is a pair (O, P), where O is an objective function of the form $\sum a_i x_i$, a_i's are integers x_i's are 0-1 variables, and P is a $PL(PB)$ theory. An *optimal solution* to a $PL(PB)$ optimization problem is a value assignment that minimizes the value of the objective function while satisfying the $PL(PB)$ theory.

A $PL(PB)$ optimization problem (O, P) is a PB *optimization problem* if P consists of PB constraints (is a PB theory).

We can encode many practical optimization problems into $PL(PB)$ optimization problems so that optimal solutions to the $PL(PB)$ optimization problem correspond to optimal solutions to the original problem. In other words, we can view $PL(PB)$ optimization as a modeling formalism that captures other optimization problems. For example, we consider the following variant of the dominating set problem:

Example 1. Let $G = (V, E)$ be a directed graph, where V is the set of vertices and E is the set of directed edges. Each edge (u, v) in E has an associated integer weight $w(u, v)$. A k*-dominating set* U is a subset of V such that for every $v \in V$, one of the following three conditions hold:

1. $v \in U$
2. $\sum_{u \in U:(u,v) \in E} w(u, v) \geq k$
3. $\sum_{u \in U:(v,u) \in E} w(v, u) \geq k$.

The objective is to find a minimal k-dominating set for a given weighted graph G.

In order to represent the minimal k-dominating set problem for a graph G as a $PL(PB)$ optimization problem, we use 0-1 variables U_v, where $v \in V$. Intuitively, by assigning 1 to U_v we represent the fact that vertex v belongs to a k-dominating set.

The objective function in this $PL(PB)$ optimization problem is:

$$\sum_{v \in V} U_v$$

For a vertex $v \in V$, we define three PB constraints:

$$W_v^1 = \ U_v \geq 1,$$

$$W_v^2 = \sum_{w:(w,v)\in E} U_w \geq k$$

and

$$W_v^3 = \sum_{w:(v,w)\in E} U_w \geq k,$$

The $PL(PB)$ constraints $W_v^1 \vee W_v^2 \vee W_v^3$, $v \in V$, capture the defining constraints for a k-dominating set, with the three disjuncts representing the conditions (1), (2) and (3), respectively.

We note that if only PB constraints were allowed, the defining constraints of k-dominating sets would require a more complex representation.

3 Local search based optimizer for $PL(PB)$ optimization problems

To solve $PL(PB)$ optimization problems, we use a method that iteratively improves the quality of models of the $PL(PB)$ theory with respect to the objective function, until no further improvements are possible. To be precise, our method consists of a $PL(PB)$ model generator S and a search algorithm. Given a $PL(PB)$ optimization problem (O, P), we use the $PL(PB)$ model generator to compute a model (or, as we will also say, a *feasible solution*) of P (ignoring the objective function). If no model is found, we terminate the search (either there are no feasible solutions or, if the model generator is incomplete, it fails to find any). Otherwise, we take the feasible solution computed as the starting point and use the search algorithm organized in a series of iterations to find feasible solutions with successively better objective-function values. When improvement is no longer possible, we stop and return the most recently computed feasible solution.

The method we implemented following this general pattern differs from existing PB optimizers in two ways. First, it deals with $PL(PB)$ optimization problems, which are syntactically more general than PB optimization problems[1]. Second, in addition to linear search it also supports a new search strategy, LBS. The LBS algorithm is obtained by combining the linear search and a variant of the binary search. We describe all these methods below.

[1] Formally, the two formalisms have the same expressive power. That is, there exist polynomial-time algorithms to convert a $PL(PB)$ optimization problem into an equivalent PB optimization problem and vice versa.

The linear and binary search methods differ in the way they improve on the feasible solution found in the first call to the model generator (as pointed out above, if no feasible solution is found by that call, the whole process terminates).

Let us assume (O, P) is the $PL(PB)$ optimization problem and v is the current feasible solution found by the model generator. We write $v(O)$ to denote the value of O under the assignment v.

In the linear search, in each iteration we add to P the $PL(PB)$ constraint

$$-O \geq -\lfloor v(O) - 1 \rfloor,$$

where v is the present feasible solution. With exception of the first iteration, this constraint subsumes the one added in the previous one, which now can be removed. The new constraint ensures that all feasible solutions found in the future will have a smaller objective-function value than v. When the model generator fails, we terminate the search and return the last feasible solution computed.

In the binary search, we maintain both the lower and the upper bounds on the objective-function values for feasible solutions. The upper bound is provided by $v(O)$, where v is the present feasible solution. We denote the lower bound by $l(O)$. Initially, we set $l(O)$ to the sum of all negative coefficients in O (if there are none, $l(O) = 0$).

If $l(O) = v(O)$, we terminate the search. Otherwise, we have $l(O) < v(O)$ and we add to P the constraint

$$-O \geq -\lfloor l(O) + c \times (v(O) - l(O)) \rfloor,$$

where c is a real number between 0 and 1. It is easy to check that

$$l(O) \leq \lfloor l(O) + c \times (v(O) - l(O)) \rfloor < v(O).$$

We then run the model generator. If it fails, we update the lower bound to $\lfloor l(O) + c \times (v(O) - l(O)) \rfloor + 1$ and continue. Otherwise, we continue with the new feasible solution replacing the old one.

For both the linear and binary search methods, the objective-function value of each next solution is better than that of the previous one. If a complete model generator is used in the search, the returned feasible solution is an optimal one. If an incomplete model generator is used, it is not guaranteed to be an optimal one.

The reason why the linear search is might be preferred to the binary search is that establishing that a PB (or $PL(PB)$) theory is unsatisfiable often requires much more CPU time than establishing that a PB (or $PL(PB)$) theory is satisfiable. Indeed, in the first case, we must explore the entire search space to make sure no models exist, while the latter task can be completed as soon as the first solution is found (which may happen quite early in the search).

Let (O, P) be a $PL(PB)$ optimization problem (O, P) and let v_0 be a feasible solution to the search problem P[2]. Let $n = v_0(O) - l(O)$. In such case, the binary search queries a model generator $log(n)$ times to find the optimal solution, while the

[2] If P has no solutions, the optimization problem has no solutions either, and all methods perform in the same way. Thus, we will consider only optimization problems where solutions exist.

linear search, in the worst case may require as many as n queries. However, all model-generator calls made by the binary search after v_0 is found may (in the worst case) involve unsatisfied instances. On the other hand, if we use linear search, then only the last call requires the model generator to run on an unsatisfiable instance. This is the reason why most optimizers employ the linear search.

Our search algorithm LBS combines the linear search and the binary search and balances between the number of iterations and the CPU time needed during each iteration. LBS starts with the binary search algorithm. The binary-search phase stops immediately after the model generator fails to find a feasible solution for the first time. Then LBS switches to the linear search phase, starting with the best feasible solution found in the binary-search phase. This method guarantees that at most two unsuccessful calls to the model generator are made in the process.

We expect that LBS will outperform the linear search when the range of the objective function is large and the quality improvement in each iteration is small. On the other hand, when the range of the objective function is small or the improvement of the quality of the feasible solutions is large, the linear search may outperform LBS. Our, still preliminary and non-comprehensive experiments, support this expectation. We come back to these issues in the experimentation section.

The pseudo code of LBS is given in Figure 1.

Algorithm 1 LBS

INPUT: P - a $PL(PB)$ theory
O - an objective function
S - a $PL(PB)$ model generator
OUTPUT: v - a value assignment that optimizes f subject to T
BEGIN
1. Call S with P; **If** S fails, return "unsatisfiable";
2. **While** S returns a value assignment v;
3. Let m be $l(O) + c \times (v(O) - l(O))$;
4. Let P' be $P \cup \{-O \geq -\lfloor m \rfloor\}$;
5. Call S with P';
6. **End While**
7. Let v be the last value assignment S returns;
8. **Do**
9. Let m be $v(O) - 1$;
10. Let P' be $P \cup \{-O \geq -\lfloor m \rfloor\}$;
11. Call S with P';
12. **While** S returns a value assignment v;
13. return the last value assignment S returns;
END

Line 1 says when S fails the optimizer will halt and report the set of $PL(PB)$ constraints alone is unsatisfiable. In the case when S is an incomplete solver, as is in our implementation, this message means S fails to find a model given the amount of

resource allocated to S. It may be the case that T is indeed satisfiable. This limitation comes from the fact that S is incomplete.

From line 2 to line 6, we first perform a variant of the binary search with the constant value c set between 0 and 1. In practice, we set c to $2/3$. From line 8 to line 12, we perform a linear search, starting with the value of f found from the binary search step. It is clear that LBS search needs to test exactly two unsatisfiable instances: one at the end of the binary search and the other at the end of the linear search.

In our implementation, as the model generator we use the only existing $PL(PB)$ solver that we are aware of, $wsat(plpb)$ [1]. Since $wsat(plpb)$ is an incomplete solver, our optimizer does not guarantee optimality of solutions it returns. However, our method is general in that it works with any model generator for the sets of $PL(PB)$ constraints. When used with a complete $PL(PB)$ solver (and without any time-out limits on calls to the model generator), our method returns optimal solutions.

4 Experimental results

Our implementation has two options: LBS-$wsat(plpb)$ and LS-$wsat(plpb)$, using the LBS search and pure linear search respectively (our experiments showed that the pure binary search often performs worse than the linear and the LBS methods). We compare the performance of the two implementations to existing PB optimizers, $minisat+$ [3] and $bsolo$ [5], on a set of $PL(PB)$ optimization problems. Since the PB optimizers do not accept $PL(PB)$ constraints, we apply the transformation from $PL(PB)$ constraints to PB constraints proposed in [1].

We now describe the $PL(PB)$ optimization problem instances used in our experiments. We considered three categories of instances generated for the traveling salesperson problem, the minimum k-dominating set problem, and a variant of the NQueens problem, respectively.

Traveling salesman problem (*tsp*). Given a complete undirected graph $G = (V, E)$, where each edge $(u, v) \in E$ has an associated weight $w(u, v)$, the goal is to find a Hamiltonian cycle in G such that the sum of the weights of the edges in the cycle is minimized. For testing, we randomly generated 10 weighted complete graphs with 70 vertices. The edge weight ranges from 1 to 9.

Minimum k-dominating set problem (*dms*). We have defined this problem in Section 2. To generate $PL(PB)$ optimization instances for testing, we randomly generated 10 weighted graphs of 500 vertices and 2000 edges. The range of the edge weight is $[1..19]$. Finally, we set w to 4.

Weighted n-queens problem (*wnq*). Squares of an $n \times n$ chess-board have integer weights. Given two integers w and d, find an arrangement of n queens on the board so that 1) no two queens attack each other; 2) the sum of weights of the squares with queens does not exceed w; and 3) for each queen Q, there is at least one queen Q' in a neighboring row or column such that the Manhattan distance between Q and Q' exceeds d. For testing, we randomly generated 10 weighted chessboards. The weights of the blocks on the chessboards range from 1 to 29.

The first problem yields PB optimization instances while the other two yield general $PL(PB)$ optimization instances. We use the transformation proposed in [1] to con-

vert the $PL(PB)$ instances into equivalent PB instances when we test the PB optimizers.

We also test all optimizers on an instance, normalized-fast0507, from the *pseudo-boolean evaluation 05* [8]. The objective function of this instance has a range $[0, 122411]$. Instances like this one can be used to study our conjecture on the cases when LBS performs better than the linear search.

We use the following parameters for $wsat(plpb)$, the underlying $PL(PB)$ solver of our optimizers: 1 restart, 200000 flips per restart. We use default values for the other parameters required by $wsat(plpb)$ [1].

All experiments are conducted on machines with P4 3.2GHz CPUs, 1GB memory, and running Linux with kernel version 2.6.15. We allocate 200 seconds to each optimizer on each instance.

We write LS-$wsat(plpb)$ and LBS-$wsat(plpb)$ to denote our linear search optimizer and LBS search optimizer respectively. Both optimizers use $wsat(plpb)$ as the $PL(PB)$ model generator. None of the optimizers we tested (even those based on complete solvers $minisat+$ and $bsolo$) can prove the optimality of the solutions they find within the 200-second time limit. Therefore, we only report the best value for the objective function found by each optimizer in the following tables.

We first present the results on the tsp problem in Figure 1. We observe that both of

tsp	LS-$wsat(plpb)$	LBS-$wsat(plpb)$	$minisat+$	$bsolo$
I_1	**138**	140	245	299
I_2	**133**	138	255	256
I_3	142	**136**	262	279
I_4	135	**132**	248	302
I_5	**133**	135	246	291
I_6	**143**	144	272	251
I_7	**140**	144	267	292
I_8	150	**133**	272	266
I_9	**140**	144	240	225
I_{10}	**139**	143	275	274

Fig. 1. Best value found — tsp

our optimizers find significantly better values for the objective functions than the PB optimizers in this category of instances. Among our two optimizers, LS-$wsat(plpb)$ performs better than LBS-$wsat(plpb)$.

Figure 2 shows the results on the dms problem instances. In this category of instances, LS-$wsat(plpb)$ is a clear winner. LBS-$wsat(plpb)$ loses to $minisat+$ in one instance, but is the second best optimizer in the rest of the instances. We examined the log file of the LBS-$wsat(plpb)$ in this experiment and found that LBS-$wsat(plpb)$ did not even start the linear search phase when the 200-second time limit was reached. This result shows that the performance of local-search solvers may be improved if multiple tries are used.

dms	$LS\text{-}wsat(plpb)$	$LBS\text{-}wsat(plpb)$	$minisat+$	$bsolo$
I_1	**98**	115	121	127
I_2	**93**	112	119	118
I_3	**95**	111	119	127
I_4	**96**	121	119	125
I_5	**96**	118	121	128
I_6	**95**	116	120	118
I_7	**94**	117	123	122
I_8	**95**	119	123	124
I_9	**97**	112	122	126
I_{10}	**98**	112	122	127

Fig. 2. Best value found — dms

Figure 3 shows the results from the wnq problem. In this category of instances,

wnq	$LS\text{-}wsat(plpb)$	$LBS\text{-}wsat(plpb)$	$minisat+$	$bsolo$
I_1	161	**137**	275	303
I_2	190	**175**	314	386
I_3	228	**162**	308	365
I_4	250	**162**	301	360
I_5	200	**190**	268	431
I_6	176	**110**	316	350
I_7	188	**174**	282	325
I_8	242	**224**	302	438
I_9	146	**119**	279	324
I_{10}	242	**131**	312	359

Fig. 3. Best value found — wnq

we observe that $LBS\text{-}wsat(plpb)$ performs better than all the other optimizers. The $LS\text{-}wsat(plpb)$ is the second best among these optimizers.

Finally, in Figure 4 we present the result of testing all four optimizers on the instance from [8]. As we mentioned earlier, this instance has an objective function with the range from 0 to 122411. The instance consists of 63009 0-1 variables and 490 PB constraints.

It is clear that the LBS search performs much better than the linear search in this instance. Within similar amount of time, the LBS improves the value of the objective function to 8038, almost 7.5 times better than what the linear search could achieve. $minisat+$ caused a segmentation fault on this instance. Therefore we could not report the result of $minisat+$.

$bsolo$ is the best optimizer on this instance (the only instance when our optimizers were outperformed). We think the reason is the high degree of structure in this instance. Local-search based model generators are known to perform poorly on such instance.

normalized-fast0507	LS-$wsat(plpb)$	LBS-$wsat(plpb)$	$minisat+$	$bsolo$
Best value	59947	8038	N/A	251
Time to best value	173	165	N/A	71

Fig. 4. Instance from PB competition '05

5 Conclusions and future work

Many practical optimization problems can be encoded in a concise way as $PL(PB)$ optimization problems. We propose a method to deal with problems in this class. Our method relies on a $PL(PB)$ model generator and a search algorithm that iteratively improves the quality of the feasible solutions found by the model generator. Our method differs from the existing PB optimizers in two ways: (1) it accepts optimization problems whose constraints are encoded as disjunctions of PB constraints; and (2) our method uses a combination of the linear and the binary search to improve the quality of the feasible solutions.

Our experimental results show that, our optimizers, perform uniformly better than existing PB optimizers we tested, except for one instance which is highly structured. We believe there are two reasons for this. First, we use a local-search model generator while the PB optimizers use DPLL-based model generators. Selman *et al.* [9] and Hoos *et al.* [10] have shown that local-search based methods often scale better than DPLL methods in solving propositional satisfiability problems. Since the satisfiability testing of $PL(PB)$ and PB theories is closely related to propositional satisfiability testing, this phenomenon also appears in the case of $PL(PB)$ and PB model generators. Second, two optimization problems we test involve disjunctions of constraints. Therefore, it is natural to encode them using $PL(PB)$ theories. In order to test PB solvers on those instances, one needs to transform these $PL(PB)$ theories into equivalent PB theories, which makes the theories larger and PB optimizers less effective.

However, our preliminary implementation of the LBS method is not uniformly better than the linear search method. In fact, the linear search method won in a slightly larger number of instances than LBS.

There are several research directions we intend to investigate in the future:

1. Refinement of the LBS strategy. We feel it can be improved significantly by allowing dynamic adjustments in the selection of the constant c. Furthermore, for easy problems, linear search has an advantage as it may happen that all iterations are fast except for the last one. The cost of two calls to the model generator failing to find a solution in the LBS-search method may negate all the savings coming from the fewer number of iterations it makes. We will study dynamic strategies to select between the linear and binary search depending on the instance and the characteristics of search, possibly using machine learning approaches.
2. We will study how to generate random $PL(PB)$ optimization problems and the distribution of the optimal solutions of these random instances. Experimenting with random instances will provide us with insights into the properties of solutions of such problems and may give us methods to guide the selection of the constant c.

3. A comparison of the linear search with the LBS with complete model generators. Since local-search based model generators are incomplete their running time may show a significant variability from run to run even on the same instance. We intend to integrate LBS with DPLL model generators and systematically compare the the linear search with the LBS method for this class of solvers.

References

1. Liu, L., Truszczyński, M.: Local search techniques for boolean combinations of pseudo-boolean constraints. In: Proceedings of The Twentieth National Conference on Artificial Intelligence (AAAI-06), AAAI Press (2006) to appear
2. Walser, J.: Solving linear pseudo-boolean constraints with local search. In: Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-97), AAAI Press (1997) 269–274
3. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 1–25
4. Bailleux, O., Boufkhad, Y., Roussel, O.: A translation of pseudo boolean constraints to sat. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 191–200
5. Manquinho, V., Marques-Silva, J.: Effective lowerbounding techniques for pseudo-boolean optimization. In: Proceedings of the Design and Test in Europe Conference. (2005) 660–665
6. Sheini, H., Sakallah, K.: Pueblo: a modern pseudo-boolean sat solver. In: Proceedings of the Design and Test in Europe Conference. (2005) 684–685
7. Aloul, F., Ramani, A., Markov, I., Sakallah, K.: PBS v0.2, incremental pseudo-boolean backtrack search SAT solver and optimizer (2003) `http://www.eecs.umich.edu/~faloul/Tools/pbs/`.
8. Manquinho, V., Roussel, O.: Pseudo boolean evaluation 2005 (2005) `http://www.cril.univ-artois.fr/PB05/`.
9. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-1994), Seattle, USA, AAAI Press (1994) 337–343
10. Hoos, H.H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann, San Francisco (CA), USA (2004)