Wildcat: a Local Search PB Optimizer

Lengning Liu and Mirosław Truszczyński

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

1 Introduction

$Wildcat$ is an optimizer designed for the *pseudo-boolean optimization* (or PBO for short) problem. PBO problems form a special class of the *optimization problems with boolean combinations of pseudo-boolean constraints*. For detailed discussions of this more general type of optimization problem and our algorithms designed for these problems, we refer to the following papers: [1–4].

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 set of constraints. Problems such as finding a minimal dominating set in a graph are optimization problems.

In our setting, we represent the constraints in an optimization problem as pseudo-boolean constraints, a formalism rooted in integer programming and operations research. By a *pseudo-boolean constraint* (or a PB constraint for short), we mean an integer programming constraint with only 0-1 variables.

Optimization problems with PB constraints have received much attention during the past decade. Recently many optimizers that solve PBO problems have emerged, including $wsat(oip)$ [5], $minisat+$ [6], $pb2sat + zchaff$ [7], $bsolo$ [8], $pueblo$ [9], PBS [10] and so on.

To obtain the optimal solution, the optimizers we list above rely on a series of queries to PB constraint model generators. A model generator, on the input of a set of PB constraints, computes a value assignment that satisfies all the PB constraints. Most PB optimizers we listed above use linear search on the value of the objective function. There are two exceptions: $pb2sat + zchaff$ uses a binary search and $bsolo$ uses a SAT-based branch and bound method [11].

In the linear search, the optimizer first queries the model generator for a solution to the set of the PB constraints, disregarding the objective function. Then the optimizer iteratively improves the value of the objective function by introducing a new PB constraint each time, 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 receives an "unsatisfiable" answer, the previous value of the objective function is thus the optimal one.

The main reason behind the use of linear search instead of binary search, is that, deciding a set of PB constraints is unsatisfiable is harder than deciding a set of PB constraints is satisfiable. The binary search often needs fewer iterations than the linear search before the solutions converge to the optimal ones. However, the binary search may also need to test unsatisfiability more often than the linear search does. Actually,

the linear search only needs to test the unsatisfiability once before it finds the optimal solution.

$Wildcat$ computes sub-optimal solutions of PBO problems since it uses a local search PB model generator, which specializes the solver proposed in [2]. For the search component that successively improves on the quality of a solution, $wildcat$ provides two options: (1) the linear search (most often used in other optimizers), and (2) a 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 on an unsatisfiable instance. Our hybrid method is designed so that to execute the model generator on an unsatisfiable instances exactly twice, a significant improvement over the straightforward binary search approach.

2 Preliminaries

A *pseudo-boolean (or PB) constraint* is an integer inequality[1] of the form

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

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 we say v is a model of) the constraint if

$$\sum a_i \times v(x_i) \geq b$$

holds. An example of PB constraint is given as follows:

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

We can verify that $x_1 = 1, x_2 = 1, x_3 = 1$ 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.

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

3 *Wildcat* algorithm

To solve PBO problems, we use a method that iteratively improve the quality of models of the PBO theory, with respect to the objective function, until no further improvements are possible. To be precise, our method consists of a PB model generator and a search algorithm. The PB model generator is able to compute models of a PB theory. Given a

[1] We only use the $\geq$ operator in defining a PB constraint, following the requirement of the *Pseudo-boolean Evaluation 2006* format.

PBO problem (O, P), we use the PB model generator to generate a model of P, which we call a *feasible solution*, as the starting point. Then we use the search algorithm to iteratively improve the quality of the feasible solutions found by the model generator. In each iteration, the search algorithm produces a new PB constraint and adds it to P, which guarantees that all the feasible solutions of the new PB theory are better than the previously found feasible solutions. Then the search algorithm queries the model generator, asking for another feasible solution.

In $wildcat$, we apply a stochastic local search PB solver, $wsat(plpb)$, first develop for a more general setting, which allows disjunctions of PB constraints [2]. Solver $wsat(plpb)$ uses a $wsat$-like algorithm. In particular, as in $walksat$, $wsat(plpb)$ proceeds by executing a pre-specified number of *tries*. Each try starts with a random value assignment and consists of a sequence of local modification steps called *flips*. Each flip is determined by an atom selected from an *unsatisfied* PB constraint. $Wsat(plpb)$ bases the choice on a measure of how much the corresponding flip changes the degree to which the PB constraints in the theory are violated. In designing such measures, $wsat(plpb)$ is guided by the concepts of the *break-count* and *make-count* used in $walksat$, which are defined as the numbers of clauses that become unsatisfied and satisfied, respectively, as a result of a flip.

$Wsat(plpb)$ uses a finer definition of the *break-count* and *make-count*. The intuition is that PB constraints are more complex than clauses and breaking (or fixing) a PB constraint should have more penalty (or reward) than breaking (or fixing) a clause. To this end, $wsat(plpb)$ exploits the fact that PB constraints have equivalent representation by means of propositional theories. Let T be a PB theory and let T' be its propositional-logic equivalent (with all PB constraints "compiled" away). $Wsat(plpb)$ defines the break-count of an atom a in T as the number of clauses in T' that become unsatisfied after it flips a. $Wsat(plpb)$ defines the make-count of a similarly. An important point is that $wsat(plpb)$ does not compute T' explicitly in order to determine the break-count and the make-count of a. Both measures are estimated directly on the basis of T alone.

$Wsat(plpb)$ has two variants: $wsat(plpb)$-skc and $wsat(plpb)$-rnp, which extend the *SKC* and the *RNovelty+* heuristic for $walksat$ respectively. Therefore, we call $wildcat$ combined with $wsat(plpb)$-skc $wildcat$-skc and $wildcat$ combined with $wsat(plpb)$-rnp $wildcat$-rnp. We note that, because $wsat(plpb)$ is incomplete, $wildcat$ combined with $wsat(plpb)$ only finds sub-optimal solutions to a PBO problem.

We now present the search algorithm in $wildcat$. We provide two options here: the linear search and a combination of the linear search and a variant of the binary search.

Let us assume (O, P) is the PBO 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 and $l(O)$ to denote the smallest value O can take (the sum of all negative coefficients in O).

In the linear search, a PB constraint of the following form is added to P in each iteration:

$$-O \geq \lceil -(v(O) - 1) \rceil,$$

where $-O$ is a linear function built from O by changing the signs of the coefficients in O to their dual: $+$ to $-$ and $-$ to $+$. This constraint ensures that future feasible solutions will yield a smaller value for the objective function.

The second option of our search algorithm, denoted by 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 a variant of the binary search algorithm. The binary search phase stops immediately after it tests one unsatisfiable case. Then LBS switches to the linear search phase, starting with the best feasible solution found by the binary search phase. Our method guarantees that exactly two unsatisfiable testings are needed to compute an optimal solution.

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.

The pseudo code of LBS is given in Figure 1.

Algorithm 1 LBS

INPUT: P - a PB theory
O - an objective function
S - a 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 $P \cup \{-O \geq \lceil -m \rceil\}$;
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 $P \cup \{-O \geq \lceil m \rceil\}$;
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 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.

For the *Pseudo-boolean Evaluation 2006*, we submitted the two versions of $wildcat$, $wildcat\text{-}skc$ and $wildcat\text{-}rnp$, with the LBS search only.

References

1. Liu, L., Truszczyński, M.: Local-search techniques in propositional logic extended with cardinality atoms. In Rossi, F., ed.: Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003). Volume 2833 of LNCS., Springer (2003) 495–509 Lecture Notes in Computer Science, Springer.
2. 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
3. Liu, L., Truszczyński, M.: Solving optimization problems with boolean combinations of pseudo-boolean constraints (a preliminary report). `http://www.cs.uky.edu/~lliu1/papers/opt.pdf` (2006)
4. Liu, L.: Computational tools for solving hard search problems. PhD thesis, University of Kentucky (2006) `http://www.cs.uky.edu/~lliu1/papers/LiuDissertation.pdf`.
5. 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
6. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 1–25
7. 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
8. 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
9. Sheini, H., Sakallah, K.: Pueblo: a modern pseudo-boolean sat solver. In: Proceedings of the Design and Test in Europe Conference. (2005) 684–685
10. 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/`.
11. Manquinho, V., Roussel, O.: Pseudo boolean evaluation 2005 (2005) `http://www.cril.univ-artois.fr/PB05/`.