Satisfiability Testing of Boolean Combinations of Pseudo-Boolean Constraints using Local-search Techniques*

Lengning Liu Mirosław Truszczyński

February 13, 2007

Abstract

Some search problems are most directly specified by conjunctions of (sets of) disjunctions of pseudo-Boolean (PB) constraints. We study a logic PL^{PB} whose formulas are of such form, and design local-search methods to compute models of PL^{PB} theories. In our approach we view a PL^{PB} theory T as a data structure, a concise representation of a certain propositional conjunctive normal form (CNF) theory $cl(T)$ logically equivalent to T. The key idea is an observation that parameters needed by local-search algorithms for CNF theories, such as $walksat$, can be estimated on the basis of T without the need to compute $cl(T)$ explicitly.

We compare our methods to a local-search algorithm $wsat(oip)$. The experiments demonstrate that our approach performs better. In order for $wsat(oip)$ to handle arbitrary PL^{PB} theories, it is necessary to represent disjunctions of PB constraints by *sets* of PB constraints, which often increases the size of the theory dramatically. A better performance of our method underscores the importance of developing solvers that work directly on PL^{PB} theories.

1 Introduction

We propose a stochastic local-search solver for theories in a version of propositional logic, in which formulas are Boolean combinations of pseudo-Boolean constraints.

In recent years, propositional logic has attracted considerable attention as a general-purpose modeling and computational tool well suited for solving search problems. For instance, to solve a graph k-coloring problem for an undirected graph G, we construct a propositional theory T so that (1) models of T correspond to k-colorings of G, and (2) there is a polynomial-time method to reconstruct a k-coloring of G from a model of T. Once we have such a theory T, we can use a satisfiability solver to compute a model of T, and then reconstruct from it the corresponding k-coloring of G. If T has no models, G has no k-colorings.

Instances of many other search problems can be encoded as propositional theories in a similar way. This modeling capability of propositional logic has been known for a

*This paper combines and extends results included in conference papers [14, 15].

long time. With recent dramatic advances in propositional satisfiability [23, 12, 26, 19, 20, 7], state-of-the-art SAT solvers can often decide satisfiability of sets of hundreds of thousands and even millions of clauses. That makes the approach to solving search problems outlined above computationally viable and of substantial practical interest.

There are however limitations. The repertoire of operators available for building formulas to represent problem constraints is restricted to Boolean connectives. Moreover, since satisfiability solvers usually require CNF theories as input, for the most part the only formulas one can use to express constraints are clauses. Since clausal representations of even quite simple problem constraints are often very large, the use of SAT solvers as a general problem solving tool is hindered.

Recognizing this problem researchers proposed extensions to the basic language with the equivalence operator [11], with cardinality atoms [3, 5] and with pseudo-Boolean constraints [2]. They also developed more general solvers capable of computing models of theories in the enriched syntax.

The extension most relevant to our work is that of *pseudo-Boolean constraints* (PB constraints, for short), that is, expressions representing integer constraints over binary domains[1]. PB constraints generalize propositional clauses and sets of PB constraints, or PB *theories*, generalize propositional CNF theories. Since PB constraints frequently appear in specifications of search problems, it is important to design fast methods to compute models of PB theories. To the best of our knowledge, the first such solver was introduced in [2], followed with solvers described in [25, 1, 21]. More recently several new solvers for PB theories were developed; we refer to [16, 18] for details and references.

In this paper, we focus on an extension of the formalism of PB theories. Specifically, we consider theories in which formulas are *Boolean combinations*[2] of PB constraints. We call this formalism the *propositional logic with pseudo-Boolean constraints* and denote it by PL^{PB}. Under the restriction to Boolean combinations of the so called *cardinality constraints* this logic was first considered in [5] and a complete solver for theories in this logic was described there, as well.

In this paper, we drop the restriction from [5] and consider Boolean combinations of arbitrary PB constraints. In our main contribution we propose a local-search algorithm to compute models for theories in the logic PL^{PB}. In our work we built on ideas first used in $walksat$, one of the most effective local-search satisfiability solvers for propositional logic [23]. In particular, as in $walksat$, we proceed by executing a prespecified number of *tries*. Each try starts with a random truth assignment and consists of a sequence of local modification steps called *flips*. Each flip is determined by an atom selected from an *unsatisfied* clause. We base the choice on a measure of how much the corresponding flip changes the degree to which the clauses in the theory are violated. In designing such measures, we are 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.

A straightforward extension of these concepts to the case of PL^{PB} makes each

[1] We give a more formal definition later in the paper.

[2] We only focus on conjunctions of disjunctions of PB constraints. Every general PL^{PB} formula has an equivalent representation as conjunctions of disjunctions of PB constraints.

PL^{PB} clause contribute at most 1 to the $break\text{-}count$ or $make\text{-}count$ of an atom. Our implementation of this method does not perform well in experiments[3]. Therefore, we present and study here an alternative approach, in which we modify the definitions of the *break-count* and *make-count* by exploiting the fact that PB constraints have equivalent representations by means of propositional theories. Let T be a PL^{PB} theory and let T' be its propositional-logic equivalent (with all PB constraints "compiled" away). We define the break-count of an atom a in T as the number of clauses in T' that become unsatisfied after we flip a. We define the make-count of a similarly. An important point is that we do not compute T' explicitly in order to determine the break-count and the make-count of a. We estimate these measures directly on the basis of T alone. We note that this basic idea was proposed and used in [22] in an effort to extend local-search methods from CNF to arbitrary propositional theories. Our setting of PL^{PB} theories is much more complex. For instance, PB-constraints are not formulas in the syntax of logic. Thus, the implementation of the idea developed in [22] does not apply in the case we consider here.

In the paper, we describe two basic implementations of our general approach and discuss experimental studies of their performance on several search problems: the graph vertex-cover problem, the traveling salesman problem, the bounded spanning-tree problem, a variant of the graph dominating-set problem, and the weighted n-queens problem.

We studied the performance of our algorithms and compared it to that of $wsat(oip)$ [25]. $Wsat(oip)$ was designed to compute models of PB theories. To use it on arbitrary PL^{PB} theories, we proposed and implemented a transformation that replaces PL^{PB} clauses with sets of PB constraints. Whenever necessary (that is, whenever an input PL^{PB} theory contains disjunctions of PB constraints), we used this transformation to preprocess input PL^{PB} theories prior to invoking $wsat(oip)$.

The encodings of the problems we chose fall into two classes: one that does not require Boolean combinations of PB constraints in a clause, and one that does. For the first class of encodings, $wsat(oip)$ and our solvers accept the same input. Encodings in the second class are preprocessed for $wsat(oip)$ by means of the transformation we mentioned above. In this case, the comparison is not direct since the solvers work on different inputs.

Our experiments show that our methods (at least one of them) and $wsat(oip)$ perform similarly on instances from the first class. However, our solvers perform significantly better on instances from the second class, underscoring the importance of developing solvers that work directly on PL^{PB} theories.

2 Logic PL^{PB}

A *pseudo-Boolean* constraint (*PB constraint*, for short) is an integer-programming constraint of the form

$$l \leq w_1 x_1 + \ldots + w_k x_k \leq u, \tag{1}$$

[3]One of the reasons may be that the simple treatment of PL^{PB} clauses does not take into account the structure of PL^{PB} clauses and its components.

where x_i are integer variables, each with the domain $\{0, 1\}$, w_i are integers called *weights*, and l and u are integers called the *lower bound* and the *upper bound*, respectively. An assignment v of 0s and 1s to $x_i's$ is a *model* of (or *satisfies*) the constraint (1) if

$$l \leq w_1 v(x_1) + \ldots + w_k v(x_k) \leq u$$

holds.

If one of the bounds in the constraint (1) is missing, we call it a *normal* PB constraint. Typically only normal PB constraints are considered as every PB constraint is equivalent to a set of two normal PB constraints. For us it will be more convenient to consider (general) PB constraints, as we defined them above.

To simplify the notation, we will write a PB constraint (1) as

$$l[w_1 x_1, \ldots, w_k x_k]u. \tag{2}$$

We omit the appropriate bound for normal PB constraints. If $w_i = 1$, or -1, we simplify the notation further and write x_i and $-x_i$ for $1x_i$ and $-1x_i$, respectively, when specifying the PB constraint (2). In particular, if all weights w_i are equal to 1, we write the PB constraint (2) as

$$l[x_1, \ldots, x_k]u.$$

We refer to PB constraints with all weights equal to 1 as *cardinality constraints*.

By establishing the correspondence between integer values 0 and 1 on the one hand, and truth values **f** (false) and **t** (true), respectively, on the other, we can view integer 0-1 variables as propositional atoms. Furthermore, we view PB constraints as representations of propositional formulas. Specifically, we say that a constraint (1) represents a propositional formula φ, built of the same variables x_i interpreted as propositional atoms, if (1) and φ have the same models (modulo the correspondence between $\{0, 1\}$ and $\{\mathbf{f}, \mathbf{t}\}$). In particular, a (normal) PB constraint

$$(1 - m)[x_1, \ldots, x_k, -y_1, \ldots, -y_m] \tag{3}$$

represents a propositional clause

$$x_1 \vee \ldots \vee x_k \vee \neg y_1 \vee \ldots \vee \neg y_m. \tag{4}$$

In this way, PB constraints generalize clauses, and sets of PB constraints generalize propositional CNF theories. As a special case, a propositional atom a is equivalent to the PB constraint $1[a]$ and a negated propositional atom $\neg a$ is equivalent to the PB constraint $0[-a]$.

A PB *theory* is a set of PB constraints. Many search and optimization problems have concise encodings as PB theories, often significantly smaller than their respective CNF representations. Hence, researchers started extending techniques developed for and implemented in SAT solvers to handle PB theories *directly* [2, 25, 1, 4, 21, 5].

In this paper we are interested in an even broader class of theories, namely propositional theories consisting of Boolean combinations of PB constraints, viewed as propositional formulas. We refer to the formalism we are about to describe as *propositional*

Algorithm 1 $wsat(plpb)$-$generic(T)$.

INPUT: T — a PL^{PB} theory
OUTPUT: I — a satisfying assignment of T, or no output
BEGIN
1. **For** $i \leftarrow 1$ **to** Max-$Tries$, **do**
2. $I \leftarrow$ randomly generated truth assignment;
3. **For** $j \leftarrow 1$ **to** Max-$Flips$, **do**
4. **If** $I \models T$ **then return** I;
5. $C \leftarrow$ randomly selected unsatisfied clause;
6. $a \leftarrow Heuristic(T, I, C)$;
7. $I \leftarrow Flip(I, a)$;
8. **End for** of j
9. **End for** of i

END

logic with PB constraints (or PL^{PB}, for short). While it can be given a more general treatment, in this paper we focus only on a certain class of formulas.

A PL^{PB} *clause* (or, simply, a *clause*) is an expression of the form

$$W_1 \vee \ldots \vee W_n, \tag{5}$$

where W_i's are PB constraints. A PL^{PB} *theory* is any set of PL^{PB} clauses.

The notions of *satisfiability* and a *model* of a PB constraint extend in the natural way to PL^{PB} clauses and theories. We write $I \models E$, when I is a model of a PL^{PB} clause or a PL^{PB} theory E.

3 Local-search Algorithms for the Logic PL^{PB}

In this section we describe a local-search algorithm $wsat(plpb)$-$generic$ designed to test satisfiability of theories in the logic PL^{PB}. It follows a general pattern of $walksat$ [23]. The algorithm executes Max-$Tries$ independent *tries*. Each try starts in a randomly generated truth assignment and consists of a sequence of up to Max-$Flips$ *flips*, that is, local changes to the current truth assignment. The algorithm terminates with a truth assignment that is a model of the input theory, or with no output at all (even though the input theory may in fact be satisfiable). We provide a detailed description of the algorithm $wsat(plpb)$-$generic$ in Algorithm 1.

The procedure $Heuristic$ picks an atom from clause C that was chosen in the previous step. A typical heuristic function selects between a greedy choice and a random choice based on some probability p called the *noise ratio*. It is known that the existence of a random move can guarantee the local-search solvers to escape from local minima.

To obtain a concrete implementation of the algorithm $wsat(plpb)$-$generic$, we need to define a *heuristic* for the choice of an atom in an unsatisfied clause. In this paper, we adapt to our needs two effective $walksat$ heuristics: *SKC* [23] and *RNovelty+* [9, 8, 10]. Both heuristics are defined for inputs given in terms of CNF theories and rely on two

Algorithm 2 Function $SKC(T, I, C)$.

INPUT: T - a PL^{PB} theory
I - a truth assignment of T
C - an "unsat" clause from which an atom is chosen
OUTPUT: a - an atom chosen by the heuristic function
BEGIN
1. **For each** atom x **in** C, compute $break\text{-}count(x)$;
2. **If** any of these atoms has $break\text{-}count$ 0 **then**
3. randomly choose an atom with break-count 0 and return it;
4. **Else**
5. with probability p, return an atom x with minimum $break\text{-}count(x)$;
6. with probability $1 - p$, return a randomly chosen atom in C;
7. **End If**

END

important quantities assigned to each atom given a truth assignment. They are the *break-count* of an atom and the *make-count* of an atom. The *break-count* of an atom denotes the number of clauses that will become unsatisfied if the atom's value is flipped to its dual value. Similarly, the *make-count* of an atom denotes the number of clauses that will become satisfied if the atom's value is flipped to its dual value.

It is straightforward to generalize these two concepts to the case of an arbitrary PL^{PB} theory T as we described in Section 1. However, the resulting heuristics for choosing an atom to flip do not perform well in our experiments. Therefore, we propose a different approach based on the *virtual break-count* and the *virtual make-count* of an atom in T. These two quantities are defined as the break-count and the make-count, respectively, of an atom in a certain CNF theory T' equivalent to T. The term *virtual* is to underline that we do not construct T' explicitly (its size would often be too large) but compute the two counts directly from T.

We discuss how to compute virtual counts later in this section. Assuming that we have the two counts for each atom, we defined two instantiations of the function $Heuristic$, SKC and $RNovelty+$, presented in Algorithms 2 and 3. In $RNovelty+$, we use the *age* of an atom, which records when the atom was lastly flipped. An atom that has the maximum age is the atom that was flipped most recently[4].

Finally, one consideration in implementing these algorithms is that it is time consuming to calculate the break-count and the make-count of an atom. Therefore, researchers often use cached break-count and make-count of an atom in the heuristic function. Initially, the break-count and the make-count of each atom is computed with respect to the initial truth assignment using their definitions and stored in the cache. Then, within each try, the cache will be updated each time after an atom x is flipped. To be precise, we update, in the cache, the two counts of an atom by their changes, $\Delta break\text{-}count(x)$ and $\Delta make\text{-}count(x)$, assuming x is the atom flipped.

[4]In some other implementations, the atom with the minimum age is the one that was flipped most recently.

Algorithm 3 Function $RNovelty+(T, I, C)$.

INPUT: T - a PL^{PB} theory
I - a truth assignment of T
C - an "unsat" clause from which an atom is chosen
OUTPUT: a - an atom chosen by the heuristic function
BEGIN
1. With probability wp, return a random atom from C;
2. **For each** atom x **in** C, $w(x) \leftarrow break\text{-}count(x) - make\text{-}count(x)$;
3. $age_{max} \leftarrow$ the maximum age of atoms in C;
4. $best \leftarrow$ the list of atoms x with the least $w(x)$;
5. $second \leftarrow$ the list of atoms x with the second least $w(x)$;
6. $diff \leftarrow w(x) - w(y)$, where $x \in best$ and $y \in second$;
7. **If** $\exists a \in best$ such that its age $< age_{max}$, return a;
8. **If** $diff > 1$, **then**
9. with probability $min\{2 - 2p, 1\}$, return a random atom from $best$;
10. with probability $1 - min\{2 - 2p, 1\}$, return a random atom from $second$;
11. **End If**
12. With probability $max\{1 - 2p, 0\}$, return a random atom from $best$;
13. With probability $1 - max\{1 - 2p, 0\}$, return a random atom from $second$;

END

3.1 Virtual break-count and make-count

These two concepts depend on a particular representation of a PL^{PB} theory T as a *multiset* of propositional clauses, $cl(T)$. We allow repetitions of clauses in sets and repetitions of literals in clauses, as by doing so we simplify some calculations.

We recall that we view PB constraints as propositional formulas. Given a PB constraint W, by T_W we denote a certain CNF formula (which we also view as a multiset of its clauses) such that T_W is logically equivalent to W. We will specify T_W later.

Let us consider a PL^{PB} clause C of the form (5). We define $cl(C)$ to be the multiset of propositional clauses that are disjuncts in the CNF formula obtained by replacing in C each PB constraint W_i with the CNF formula T_{W_i} and by applying the distributivity law. For a PL^{PB} theory T we then set

$$cl(T) = \bigcup \{cl(C) \colon C \in T\}.$$

Let I be a truth assignment. We define the *virtual break-* and *make-counts* of an atom x in a PL^{PB} theory T with respect to I as the break- and make-counts of x in $cl(T)$ with respect to I. We denote these two quantities as $break\text{-}count_T(x)$ and $make\text{-}count_T(x)$, respectively (we drop the reference to I from the notation, as I is always determined by the context). It follows that

$$break\text{-}count_T(x) = break\text{-}count_{cl(T)}(x) = \sum \{break\text{-}count_{cl(C)}(x) \colon C \in T\}.$$

Similarly,

$$make\text{-}count_T(x) = make\text{-}count_{cl(T)}(x) = \sum\{make\text{-}count_{cl(C)}(x) \colon C \in T\}.$$

We now estimate $break\text{-}count_{cl(C)}(x)$ and $make\text{-}count_{cl(C)}(x)$. To this end we need more notation. Let W be a PB constraint, I an interpretation and x a propositional atom. By $I^{\bar{x}}$ we denote the truth assignment obtained from I by flipping the truth value of x. Next, we define three sets of clauses that are relevant for $break\text{-}count_{cl(C)}(x)$ and $make\text{-}count_{cl(C)}(x)$ (we once again omit I in the notation):

1. $E_W(x) =$ the set of clauses in T_W that are satisfied by I but not by $I^{\bar{x}}$
2. $F_W(x) =$ the set of clauses in T_W that are not satisfied by I but are satisfied by $I^{\bar{x}}$
3. $G_W(x) =$ the set of clauses in T_W that are not satisfied by I nor by $I^{\bar{x}}$.

We observe that if x does not appear in W, $E_W(x) = F_W(x) = \emptyset$. We set $e(x) = |E_W(x)|$, $f(x) = |F_W(x)|$ and $g(x) = |G_W(x)|$.

We now have the following theorem.

Theorem 1. *Let C be a PL^{PB} clause of the form (5). Let cl be the transformation we define above. Let $e_i(x)$, $f_i(x)$, and $g_i(x)$'s be the cardinalities of the three sets $E_{W_i}(x)$, $F_{W_i}(x)$ and $G_{W_i}(x)$ with respect to an atom x and cl. Then we have the following equations:*

$$break\text{-}count_{cl(C)}(x) = \prod_{i=1}^{n}(e_i(x) + g_i(x)) - \prod_{i=1}^{n} g_i(x). \tag{6}$$

and

$$make\text{-}count_{cl(C)}(x) = \prod_{i=1}^{n}(f_i(x) + g_i(x)) - \prod_{i=1}^{n} g_i(x). \tag{7}$$

Proof. Equation (6). Every clause in $cl(C)$ is of the form $D_1 \vee \ldots \vee D_n$, where $D_i \in T_{W_i}$, $1 \leq i \leq n$. For such a clause to get "unsatisfied" with the flip of x, all D_i's in $cl(C)$ must satisfy the following two conditions:

1. each D_i is chosen from $E_{W_i}(x) \cup G_{W_i}(x)$; and
2. at least one D_i is chosen from $E_{W_i}(x)$.

Since $E_{W_i}(x) \cap G_{W_i}(x) = \emptyset$, the number of clauses in $cl(C)$ such that each D_i is chosen from $E_{W_i}(x) \cup G_{W_i}(x)$ is given by

$$\prod_{i=1}^{n}(e_i(x) + g_i(x)).$$

Among these clauses, the ones in which each D_i is chosen from $G_{W_i}(x)$ do not contribute to the break-count of x. The number of such clauses is given by

$$\prod_{i=1}^{n} g_i(x).$$

Thus, the equation (6) follows. The proof of the equation (7) is similar. □

3.1.1 Estimating e, f and g

To make formulas (6) and (7) complete, we need to specify a CNF representation T_W of a PB constraint W and, given this representation and a truth assignment I, for each atom x find formulas for e, f and g (in this section, we omit x in $e(x)$, $f(x)$ and $g(x)$ as x always exists in the context).

We first consider the case of a PB constraint W:

$$W = l[w_1 a_1, \ldots, w_k a_k]u,$$

where all w_i are *non-negative*. For each atom a_i we introduce new atoms a_i^j, $1 \leq j \leq w_i$. We then define a cardinality constraint

$$W' = l[a_1^1, \ldots, a_1^{w_1}, \ldots, a_k^1, \ldots, a_k^{w_k}]u,$$

and a set of formulas

$$EQ = \{a_i \equiv a_i^j : 1 \leq i \leq k, 1 \leq j \leq w_i\}.$$

The PB constraint W and $\{W'\} \cup EQ$ are equivalent in the following sense. There is a one-to-one correspondence between models of W and models of $\{W'\} \cup EQ$. The corresponding models coincide on the set $\{a_1, \ldots, a_k\}$. In the case of the theory $\{W'\} \cup EQ$, the part of the model contained in $\{a_1, \ldots, a_k\}$ determines the rest, as models of $\{W'\} \cup EQ$ must satisfy formulas in EQ.

Let set S consist of the following clauses:

$$\neg x_{i_1} \vee \ldots \vee \neg x_{i_{u+1}} \tag{8}$$

for every $(u+1)$-element subset $\{x_{i_1}, \ldots, x_{i_{u+1}}\}$ of $\{a_1^1, \ldots, a_1^{w_1}, \ldots, a_k^1, \ldots, a_k^{w_k}\}$, and

$$x_{i_1} \vee \ldots \vee x_{i_{K-l+1}} \tag{9}$$

for every $(K-l+1)$-element subset $\{x_{i_1}, \ldots, x_{i_{K-l+1}}\}$ of $\{a_1^1, \ldots, a_1^{w_1}, \ldots, a_k^1, \ldots, a_k^{w_k}\}$, where $K = \sum w_i$.

The following theorem is an easy consequence of Proposition 4.7 from [3], It states that the cardinality constraint W' is equivalent to S.

Theorem 2. *Let W' be the cardinality constraint and S the set of clauses we defined above. Then a truth assignment I satisfies W' if and only if I satisfies S.*

Thus, W is equivalent to $S \cup EQ$ (in the same sense as before). Consequently, W is equivalent (has the same models) as the multiset of clauses obtained from S by replacing each atom a_i^j with a_i. We define T_W to be this multiset. We also note that clauses in this multiset may contain multiple occurrences of the same literals.

We do not simplify T_W further (that is, we do not eliminate duplicate clauses nor duplicate occurrences of literals in clauses) since the multiset form of T_W makes it easier to compute the cardinalities e, f and g of the sub-multisets $E_{W,x}$, $F_{W,x}$ and $G_{W,x}$ of T_W and their cardinalities e, f and g. Namely, we have the following formulas for e, f and g:

$$e = \begin{cases} 0 & \text{case 1} \\ \binom{N+w}{K-l+1} - \binom{N}{K-l+1} & \text{case 2} \\ \binom{P+w}{u+1} - \binom{P}{u+1} & \text{otherwise} \end{cases} \tag{10}$$

$$f = \begin{cases} 0 & \text{case 1} \\ \binom{P}{u+1} - \binom{P-w}{u+1} & \text{case 2} \\ \binom{N}{K-l+1} - \binom{N-w}{K-l+1} & \text{otherwise} \end{cases} \tag{11}$$

$$g = \begin{cases} \binom{N}{K-l+1} + \binom{P}{u+1} & \text{case 1} \\ \binom{N}{K-l+1} + \binom{P-w}{u+1} & \text{case 2} \\ \binom{P}{u+1} + \binom{N-w}{K-l+1} & \text{otherwise.} \end{cases} \tag{12}$$

Case 1 covers all situations when x does not occur in W. Case 2 covers situations when x occurs in W and $I \models x$. In these formulas we use the notation $K = \sum w_i$, $P = \sum_{I \models a_i} w_i$, $N = \sum_{I \not\models a_i} w_i$, and write w for the weight of atom x in W (if x occurs in W).

We now provide arguments for each case of (10), (11), and (12). In the following, let I be a truth assignment. Let us assume $\mathcal{N} = \{a_i^j : 1 \leq i \leq k,\ 1 \leq j \leq w_i,\ I \not\models a_i\}$ and $\mathcal{P} = \{a_i^j : 1 \leq i \leq k,\ 1 \leq j \leq w_i,\ I \models a_i\}$. It is clear that $N = |\mathcal{N}|$ and $P = |\mathcal{P}|$.

Case 1. Since x does not occur in W, by the definition of sets $E_{W,x}$ and $F_{W,x}$, $e = f = 0$. Therefore, (10) and (11) are correct in this case. By the definition of $G_{W,x}$, clauses in $G_{W,x}$ are those satisfied by neither I nor $I^{\bar{x}}$. Since x does not occur in W, all clauses that are not satisfied by I form precisely the set G. That is, a clause $C \in G_{W,x}$ if and only if

1. C is obtained from a clause C' in S of the form (8) such that C' contains only atoms from $\mathcal{P}$; or
2. C is obtained from a clause C' in S of the form (9) such that C' contains only atoms from $\mathcal{N}$.

There are $\binom{P}{u+1}$ clauses of the first type and $\binom{N}{K-l+1}$ clauses of the second type in S. Since we do not remove duplicate clauses when we generate T_W from S, the number of clauses in $G_{W,x}$ is precisely $\binom{N}{K-l+1} + \binom{P}{u+1}$. Thus, case 1 of (12) holds.

Case 2. We first look at the case 2 of the equation (10). In this case, x occurs in W and $I \models x$. Let us assume that $x = a_1$. Since $I \models a_1$, $\{a_1^1, \ldots, a_1^{w_1}\} \cap \mathcal{N} = \emptyset$. The definitions of S and $E_{W,x}$ imply that $C \in E_{W,x}$ if and only if C is obtained from a clause C' in S of the form (9) such that C' contains at least one atom a_1^p, $1 \leq p \leq w_1$, and for every other disjunct y of C', $y \in \mathcal{N}$. Since $N = |\mathcal{N}|$, there are $\binom{N+w_1}{K-l+1} - \binom{N}{K-l+1}$ such clauses C'. Since when generating T_W from S we do not remove any clauses, the formula (10), case 2, follows.

With the same assumption, a clause C belongs to $F_{W,x}$ if and only if C is obtained from a clause C' in S of the form (8) such that C' contains at least one atom a_1^p, for some $1 \leq p \leq w_1$, and for every other disjunct y of C', $y \in \mathcal{P}$. Since $P = |\mathcal{P}|$, there are $\binom{P}{u+1} - \binom{P-w_1}{u+1}$ such clauses C', which is also the number of clauses C. Thus the formula (11), case 2, holds.

For the formula (12), again, a clause C belongs to $G_{W,x}$ if and only if one of the two conditions listed in case 1 is true. Since this time $x \in W$ and $I \models x$, the number of the first type of the clauses becomes $\binom{P-w_1}{u+1}$, while the number of the second type of clauses does not change. Therefore, case 2 of the formula (12) holds.

Case 3. The reasoning is similar to that in case 2. This time x occurs in W and $I \not\models x$. Again we assume that $x = a_1$. Since $I \not\models a_1$, $\{a_1^1, \ldots, a_1^{w_1}\} \cap \mathcal{P} = \emptyset$. The definitions of S and $E_{W,x}$ imply that $C \in E_{W,x}$ if and only if C is obtained from a clause C' in S of the form (8) such that C' contains at least one atom a_1^p, $1 \leq p \leq w_1$, and for every other disjunct y of C', $y \in \mathcal{P}$. Since $P = |\mathcal{P}|$, there are $\binom{P+w_1}{u+1} - \binom{P}{u+1}$ such clauses C'. Since when generating T_W from S we do not remove any clauses, the formula (10), case 3, follows.

With the same assumption, a clause C belongs to $F_{W,x}$ if and only if C is obtained from a clause C' in S of the form (9) such that C' contains at least one atom a_1^p, for some $1 \leq p \leq w_1$, and for every other disjunct y of C', $y \in \mathcal{N}$. Since $N = |\mathcal{N}|$, there are $\binom{N}{K-l+1} - \binom{N-w_1}{K-l+1}$ such clauses C', which is also the number of clauses C. Thus the formula (11), case 3, holds.

For the formula (12), a clause C belongs to $G_{W,x}$ if and only if one of the two conditions listed in case 1 is true. Since this time $x \in W$ and $I \not\models x$, the number of the second type of the clauses becomes $\binom{N-w_1}{K-l+1}$, while the number of the first type of clauses does not change. Therefore, case 3 of the formula (12) holds. □

To deal with negative weights a PB constraint may contain, we follow a standard and well known normalization method. We briefly recall it here. Let us consider a PB constraint:

$$W = l[w_1 a_1, \ldots, w_m a_m, w_{m+1} a_{m+1}, \ldots, w_k a_k]u \tag{13}$$

where $w_i < 0$, $1 \leq i \leq m$, and $w_i \geq 0$, $m+1 \leq i \leq k$. Let

$$W' = (l+d)[w_1 a'_1, \ldots, w_m a'_m, w_{m+1} a_{m+1}, \ldots, w_k a_k](u+d) \tag{14}$$

where $d = \sum_{i=1,\ldots,m} w_i$ and a'_i, $1 \leq i \leq m$, are new 0-1 variables. We now define T_W to be the CNF theory consisting of all clauses in $T_{W'}$, in which we replace a'_i with $\neg a_i$ and $\neg a'_i$ with a_i.

One can check that W is equivalent to the set of PB constraints consisting of W' and PB constraints $a'_i + a_i = 1$, $1 \leq i \leq m$. Thus, since W' is equivalent to $T_{W'}$, it follows that W is equivalent to T_W as defined above.

Moreover, it follows that e, f and g can be computed using the formulas we developed earlier with the following modifications: (1) the bounds in the formulas must be replaced with $l+d$ and $u+d$, where d is specified above, and (2) if a_i is assigned a negative weight, the condition $I \models a_i$, must be replaced with $I \not\models a_i$ (and *vice versa*). In this way, the computation of virtual break- and make-count for arbitrary PB constraints can be accomplished without any overhead.

We will now illustrate how to compute the virtual break- and make-count of an atom.

Example. Let $C = W_1 \vee W_2 \vee W_3$, where $W_1 = 2[a_1, a_2, a_3]2$, $W_2 = 4[2a_2, 1a_3, 4a_4]5$, and $W_3 = 3[10a_5, 3a_3, 8a_6]10$.

Let $I = \{a_1, a_3, a_4, a_5\}$ be the truth assignment. We first note that:

1. In W_1, $K_1 = 3, N_1 = 1, P_1 = 2$
2. In W_2, $K_2 = 7, N_2 = 2, P_2 = 5$
3. In W_3, $K_3 = 21, N_3 = 8, P_3 = 13$.

Now let us suppose we flip atom a_2. We recall that $I \not\models a_2$. Based on the formulas for e, f, g, we have the following result:

1. Since $a_2 \in W_1$, case 3 applies to W_1 and so:

$$e_1 = \binom{P_1 + w_1^2}{u_1 + 1} - \binom{P_1}{u_1 + 1} = \binom{2+1}{2+1} - \binom{2}{2+1} = 1$$

$$f_1 = \binom{N_1}{K_1 - l_1 + 1} - \binom{N_1 - w_1^2}{K_1 - l_1 + 1} = \binom{1}{3-2+1} - \binom{1-1}{3-2+1} = 0$$

$$g_1 = \binom{P_1}{u_1 + 1} + \binom{N_1 - w_1^2}{K_1 - l_1 + 1} = \binom{2}{2+1} + \binom{1-1}{3-2+1} = 0$$

2. Since $a_2 \in W_2$, case 3 applies to W_2 and so:

$$e_2 = \binom{P_2 + w_2^2}{u_2 + 1} - \binom{P_2}{u_2 + 1} = \binom{5+2}{5+1} - \binom{5}{5+1} = 7$$

$$f_1 = \binom{N_2}{K_2 - l_2 + 1} - \binom{N_2 - w_2^2}{K_2 - l_2 + 1} = \binom{2}{7-4+1} - \binom{2-2}{7-4+1} = 0$$

$$g_1 = \binom{P_2}{u_2 + 1} + \binom{N_2 - w_2^2}{K_2 - l_2 + 1} = \binom{5}{5+1} + \binom{2-2}{7-4+1} = 0$$

3. Since $a_2 \notin W_3$, case 1 applies to W_3 and so:

$$e_3 = f_3 = 0$$

$$g_3 = \binom{P_3}{u_3 + 1} + \binom{N_3}{K_3 - l_3 + 1} = \binom{13}{10+1} + \binom{8}{21-3+1} = 78$$

We can now compute the break- and make-count of a_2 using the formulas (6) and (7):

$$\textit{break-count}_C(a_2) = \prod_{i=1}^{3}(e_i + g_i) - \prod_{i=1}^{3} g_i = (1+0)\times(7+0)\times(0+78) - 0\times 0\times 78 = 546$$

$$\textit{make-count}_C(a_2) = \prod_{i=1}^{3}(f_i + g_i) - \prod_{i=1}^{3} g_i = (0+0)\times(0+0)\times(0+78) - 0\times 0\times 78 = 0$$

△

We used the formulas (6) and (7) for the break- and make-count in the heuristics SKC and $RNovelty+$ given in Algorithms 2 and 3, respectively. We used these two

heuristics with our generic local-search algorithm and obtained its two instantiations $wsat(plpb)$-skc and $wsat(plpb)$-rnp, respectively.

We point out that, in the formulas we have derived, we use values of the form $\binom{n}{k}$. Such values exceed the maximum integer that can be represented in a (typical) computer even for relatively small values of n, if k is close to $n/2$. In our implementations, we replaced each occurrence of such overflow with a certain fixed large integer. However, we observed that, even though overflow occurs quite common when we ran $wsat(plpb)$-skc and $wsat(plpb)$-rnp in our experiments, it is rarely the case that $wsat(plpb)$-skc or $wsat(plpb)$-rnp chose the "best" atoms (according to their heuristics) whose $break$-$count$ or $make$-$count$ computation involves overflows. We will illustrate this phenomenon in the following section. Thus the overflow does not cause much problem to $wsat(plpb)$-skc and $wsat(plpb)$-rnp in those instances. It is possible to use floating point arithmetic and Stirling's approximation to approximate $\binom{n}{k}$. We did implement $wsat(plpb)$-skc and $wsat(plpb)$-rnp following this direction. With floating point numbers, we can represent significantly larger numbers in the computer than integer representation. However, we still observed that overflow happened in some cases, with much less frequency. We also observed that, there is a dual problem associated with floating point arithmetic, which is the loss of precision. With a single-precision 32-bit floating point number, only 23 bits are used to represent the mantissa. That means, we have less than 10 decimal digits in the mantissa and, consequently, we often cannot distinguish between two large numbers of the same order.

Finally, we comment here that there is more than one way to represent a PB constraint as an equivalent CNF theory. In particular, our approach is not the most concise representation. Actually, the size of our representation is exponential to the size of the PB constraint. We could adopt other more concise representations. However, those representations usually rely on auxiliary new atoms and more complex equivalence formulas to reduce the exponential blow-up in size. Such structure hinders the performance of wsat-like local search algorithms [14]. Therefore, we did not use those representations to compute virtual counts.

4 Experiments, Results and Discussion

We performed experimental studies of the effectiveness of our local-search algorithms. For testing we selected five families of instances generated from five search problems.

4.1 Benchmark problems and instances

We describe first the problems we considered, and illustrate how they give rise to specific PL^{PB}-theories that can be used in testing.

Vertex-cover problem. Let $G = (V, E)$ be an undirected graph, where V and E are the sets of vertices and edges of G, respectively, and let k be a positive integer. The objective is to find a set $U \subseteq V$ with at most k elements and such that every edge has at least one of its vertices in U (such sets U are *vertex covers* for G).

To build a PL^{PB} theory $vcv(G,k)$ encoding the vertex-cover problem, for every vertex $v \in V$ we introduce an atom in_v to represent the statements: *vertex v is in a vertex cover*. We include in $vcv(G,k)$ the following clauses:

1. $in_u \vee in_v$, for every edge $\{u,v\} \in E$
   These clauses enforce the constraint defining vertex covers

2. $[in_v \colon v \in V]k$
   This clause guarantees that a selected subset has at most k vertices.

It is straightforward to check that models of $vcv(G,k)$ are in one-to-one correspondence with those vertex covers of G that have at most k elements. In fact, if we represent a truth assignment by a set of atoms that are true in it, models of $vcv(G,k)$ are precisely sets of the form $\{in_u \colon u \in U\}$, where U is a vertex cover of G with at most k elements.

To create instances for testing solvers, we randomly generated 50 graphs of 2000 vertices and 4000 edges. For each graph we selected a relatively small integer as the bound for the size of the vertex cover in such a way that the problem still had a solution. We used these instances to instantiate the encoding given above, and obtained the family vcv of 50 satisfiable PB theories.

Traveling salesperson problem. Let k be a non-negative integer k and G a complete undirected weighted graph, in which each edge $\{u,v\}$ is assigned an integer weight $W_{u,v}$[5]. The goal is to find a Hamiltonian cycle in G such that the sum of the weights of the edges in the cycle is at most k.

To specify the problem as a PL^{PB} theory we use propositional atoms $ord_{v,i}$ and $e_{u,v}$, where $1 \leq i \leq |V|$ and $u, v \in V$. The role of atoms $ord_{v,i}$ is to define an ordering (permutation) of V specifying a Hamiltonian cycle. Specifically, an atom $ord_{v,i}$ stands for the statement: *v is the i^{th} vertex in the cycle*. The role of atoms $e_{u,v}$ is to determine the edges $\{u,v\}$ in the graph that are included in the Hamiltonian cycle given by the atoms $ord_{v,i}$. We use the atoms $e_{u,v}$ to state the constraint bounding the length of the cycle. We define the theory $tsp(G,k)$ to consist of the following clauses:

1. $1[ord_{v,i} \colon v \in V]1$, for every i, $1 \leq i \leq |V|$
   $1[ord_{v,i} \colon 1 \leq i \leq |V|]1$, for every $v \in V$
   These clauses enforce the ordering constraints: for every i, there is exactly one vertex in the position i in the cycle; and for every vertex v there is exactly one position in the cycle containing v.

2. $\neg ord_{u,i} \vee \neg ord_{v,i+1} \vee e_{u,v}$, for every $u, v \in V$ and for every i, $1 \leq i \leq |V|$
   $\neg e_{u,v} \vee \neg ord_{u,i} \vee ord_{v,i+1}$, for every $u, v \in V$ and for every i, $1 \leq i \leq |V|$
   Note: when incrementing indices by one, we assume arithmetic modulo $|V|$ on the set $\{1, \ldots, |V|\}$; in particular, $|V| + 1 = 1$.
   These clauses define the edges $e_{u,v}$ that form the Hamiltonian cycle specified by the permutation given by atoms $ord_{v,i}$.

3. $[W_{u,v} e_{u,v} : u, v \in V]k$
   This clause enforces the bound on the length of the cycle.

[5]Since G is an undirected graph, we have $W_{u,v} = W_{v,u}$

One can verify that the atoms $e_{u,v}$ in a model of $tsp(G,k)$ form a Hamiltonian cycle of length at most k and that each such cycle is determined by a model of $tsp(G,k)$ (in fact, by $2|V|$ models as there are $|V|$ choices for the first vertex and two directions in which the cycle can be traversed).

For testing, we randomly generated 50 weighted complete graphs of 40 vertices. The weight of each edge was uniformly chosen from the range $[1..39]$. Next, for each instance we selected a relatively small integer as the TSP bound in such a way that the problem still had a solution. These graphs together with the encoding specified above yielded the family tsp of 50 satisfiable PL^{PB} theories.

Bounded spanning-tree problem. Let $G = (V, E)$ be an undirected graph with each edge $\{u, v\} \in E$ assigned an integer weight $W_{u,v}$. Further, let w be an integer. The goal is to find a spanning tree T in G such that for each vertex $x \in V$, the sum of the weights of all edges in T incident to x is at most w.

Let us assume that $|V| = n$. We observe that a set T of edges of a graph G is a spanning tree for G if and only if there is an ordering (permutation) $x_1, \ldots, x_n$ of vertices of G such that

(ST) for every $i > 1$, there is exactly one j such that $1 \leq j < i$ and $\{x_j, x_i\} \in T$

To build the PL^{PB} theory $bst(G,k)$ encoding the bounded spanning-tree problem, we use atoms $ord_{x,i}$ and $te_{x,y}$, where $1 \leq i \leq n$ and $x, y \in V$. Atoms $ord_{x,i}$ are meant to establish an ordering (permutation) of vertices in G, while atoms $te_{x,y}$ express the statement that the edge $\{x, y\} \in T$, written so that x precedes y in the ordering determined by atoms $ord_{x,i}$, is chosen into the spanning tree. To model the desired properties of $ord_{x,i}$ and $te_{x,y}$, the condition (ST), and the weight-bound constraint on vertices, we include in the theory $bst(G,k)$ the following clauses:

1. $1[ord_{x,i} : x \in V]1$, for every $i = 1, \ldots, n$
   $1[ord_{x,i} : i = 1, \ldots n]1$, for every $x \in V$
   These clauses generate an ordering of vertices in G.

2. $\neg ord_{x,i} \vee \neg ord_{y,j} \vee \neg te_{x,y}$, for every $1 \leq j < i \leq n$
   These clauses ensure that each edge selected to T is represented by the pair of its endpoints ordered according to the ordering specified by atoms of the form $ord_{x,i}$.

3. $(n-1)[te_{x,y} \colon \{x, y\} \in E](n-1)$
   This clause guarantees that exactly $n-1$ edges are selected into T. It is implied by other constraints and could be omitted.

4. $\neg ord_{y,i} \vee 1[te_{x,y} \colon \{x, y\} \in E]1$, for every $i > 1$ and $y \in V$
   These clauses model the condition (ST)

5. $[W_{x,y} te_{x,y} \colon \{x, y\} \in E; W_{y,x} te_{y,x} \colon \{x, y\} \in E]w$, for every $y \in V$
   Note: the ";" stands for the operator of concatenation of two lists of weighted propositional atoms.
   These clauses ensure the weight-bound constraint on each vertex y.

To create PL^{PB}-theories for testing we generated 50 random graphs of 30 vertices and 240 edges. The weight of each edge was uniformly chosen from the range $[1..29]$. We set w to 15, as for $w = 15$ all instances were satisfiable. The encoding of the bounded spanning-tree problem instantiated with these graphs determined the family bst of 50 satisfiable PB theories.

Weighted dominating-set problem. This is a variant of the dominating-set problem [6]. Let $G = (V, E)$ be a directed graph. Each edge $(u, v) \in E$ has an integer weight $W_{u,v}$. Let w be an integer. A set $D \subseteq V$ is *w-dominating* if for every vertex $v \in V$ one of the following three conditions holds:

1. $v \in D$;
2. the sum of weights of edges "from v to D" is at least w: $w \leq \sum_{(v,u)\in E, u\in D} W_{v,u}$
3. the sum of weights of edges "from D to v" is at least w: $w \leq \sum_{(u,v)\in E, u\in D} W_{u,v}$.

Given a weighted directed graph G, an integer w, and a positive integer k, the *weighted dominating-set problem* consists of finding a w-dominating set of G with at most k elements.

As in the other three cases, instances of this problem can be encoded as a PL^{PB} theories so that models of theories correspond to solutions to the problem. We refer to [13] for a detailed description of the encoding. Here we only mention that it utilizes PL^{PB} clauses that are disjunctions of PB constraints.

We generated 50 random graphs of 500 vertices and 2000 edges. The weight of each edge was generated uniformly from $[1..19]$. The value of w was set to 40 and k to 330. All instances we generated were satisfiable. The instances and the encoding yielded the family wdm of 50 satisfiable PL^{PB} theories.

Weighted n-queens problem with an additional separation constraint. It is a variant of the well-known n-queens problem. We are given an $n \times n$ chess board, with each square (i, j) assigned an integer weight $W_{i,j}$. Given two integers w and d, the goal is to find an arrangement of n queens on the board so that (1) no two queens attack each other (standard constraint); (2) the sum of weights of the squares with queens does not exceed w (hence the term 'weighted"); 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 (a *separation* constraint) . We refer to [13] for the encoding of this problem as a PL^{PB} theory and note only that it contains clauses that are disjunctions of PB constraints.

We generated 50 random weighted 20×20 chess boards, where the weights are uniformly chosen from range $[1..19]$. The value of w was set to 80 and d to 10. All instances we generated were satisfiable. The encoding [13] and these 50 instances of the problem gave rise to the last family, wnq, of 50 satisfiable PL^{PB} theories we used in testing.

We conclude this subsection by pointing out that all encodings discussed here make use of PB constraints. It makes the encodings more concise than any CNF encoding

of the same problems we are aware of. For instance, just one PB constraint is needed to enforce the upper bound on the number of vertices in a vertex cover (clause (2) in the theory $vcv(G,k)$), while the best CNF representation known to us uses $\Theta(|V|\lg k)$ clauses. Moreover, some encodings use PB constraints with both the lower and the upper bounds ($tsp(G,k)$ and $bst(G,k)$). Two normal PB constraints are needed to represent each such constraint. Finally, $wdm(G,w,k)$ uses "proper" PB clauses, that is, disjunctions of PB constraints, to encode the weighted dominating-set property. Several PB constraints and additional propositional atoms are needed to capture disjunctions of PB constraints.

Thus, logic PL^{PB} yields more concise encodings not only with respect to CNF representations but also with respect to representations in terms of (normal) PB constraints. It points to its modeling effectiveness and underlines the need to develop solvers for general PB theories.

4.2 The design of the experiments

We studied our solvers $wsat(plpb)$-skc and $wsat(plpb)$-rnp and compared their performance to that of $wsat(oip)$. Since $wsat(oip)$ accepts only theories consisting of *normal* PB constraints, direct comparison is possible only for instances from the vcv family.

Instances in the family tsp consist of PB constraints that involve both lower and upper bounds. As we noted, such constraints can be represented in a direct way by two normal PB constraints, and theories obtained by applying this transformation have essentially the same structure (no new atoms needed, no essential change in the expression of the constraint, except that it is split into two inequalities, one for each bound).

Instances in the families bst, wdm, and wnq contain clauses that are disjunctions of weight atoms and $wsat(oip)$ cannot be used directly. The approach we adopted consisted of replacing such clauses with sets of normal PB constraints. This approach requires new atoms and introduces additional structure into the representation. We will now outline the specific method we used.

Given a PL^{PB} clause C, our transformation $\tau(C)$ converts it into a set of pseudo-Boolean constraints and maintains the models, modulo the new atoms introduced in the transformation. Let us consider a PL^{PB} clause C of form (5). We introduce new propositional atoms $\mathcal{W}_1,\ldots,\mathcal{W}_n$ to represent each PB constraint in the clause. With the new atoms, the clause (5) can be written as the propositional clause

$$\mathcal{W}_1 \vee \ldots \vee \mathcal{W}_n$$

and then represented by the PB constraint:

$$1[\mathcal{W}_1,\ldots,\mathcal{W}_n]. \tag{15}$$

In order for this representation to work, we must also specify PB constraints to enforce the equivalence of the newly introduced atoms $\mathcal{W}_i$ and the corresponding PB-constraints W_i, for $i=1,\ldots,n$.

To show how it can be done, let us consider a PB constraint $W = l[w_1a_1, \ldots, w_ma_m]u$. Let $\mathcal{W}^+$ and $\mathcal{W}^-$ be two additional propositional atoms. We observe that the following two PB constraints:

$$\Big[(\sum\{w_i : w_i > 0\} - u)\mathcal{W}^-, w_1a_1, \ldots, w_ma_m\Big] \sum\{w_i : w_i > 0\} \tag{16}$$

$$(u+1)\Big[(u+1-\sum\{w_i : w_i < 0\})\mathcal{W}^-, w_1a_1, \ldots, w_ma_m\Big] \tag{17}$$

are equivalent to the propositional formula

$$\mathcal{W}^- \equiv [w_1a_1, \ldots, w_ma_m]u$$

(we view the PB constraint $[w_1a_1, \ldots, w_ma_m]u$ as a propositional formula here). Similarly, the following two PB constraints:

$$\sum\{w_i : w_i < 0\}\Big[(\sum\{w_i : w_i < 0\} - l)\mathcal{W}^+, w_1a_1, \ldots, w_ma_m\Big] \tag{18}$$

$$\Big[(l-1-\sum\{w_i : w_i > 0\})\mathcal{W}^+, w_1a_1, \ldots, w_ma_m\Big](l-1) \tag{19}$$

are equivalent to the propositional formula

$$\mathcal{W}^+ \equiv l[w_1a_1, \ldots, w_ma_m].$$

Finally, the following three PB constraints:

$$0[-\mathcal{W}, \mathcal{W}^+] \tag{20}$$

$$0[-\mathcal{W}, \mathcal{W}^-] \tag{21}$$

$$-1[-\mathcal{W}^+, -\mathcal{W}^-, \mathcal{W}] \tag{22}$$

are equivalent to the formula

$$\mathcal{W} \equiv \mathcal{W} \wedge \mathcal{W}^-.$$

It follows that the formula $\mathcal{W} \equiv l[w_1a_1, \ldots, w_ma_m]u$ is equivalent to PB constraints (16), (17), (18), (19), (20), (21) and (22).

For a PL^{PB}-clause $C = W_1 \vee \ldots \vee W_n$, we define $\tau(C)$ to consist of the PB constraint (15) and of PB constraints (16), (17), (18), (19), (20), (21) and (22), constructed as described above for every PB constraint W_i in C by means of propositional atoms $\mathcal{W}_i^+$ and $\mathcal{W}_i^-$). One can verify that the size of $\tau(C)$ is linear in the size of C.

Given a PL^{PB} theory T, by $\tau(T)$ we denote the set of normal PB constraints

$$\tau(T) = \{\tau(C) \colon C \in T\}.$$

To summarize our observations made above

1. (15) is equivalent to $\mathcal{W}_1 \vee \ldots \vee \mathcal{W}_n$
2. (16) and (17) are equivalent to $\mathcal{W}^- \equiv [w_1a_1, \ldots, w_ma_m]u$

3. (18) and (19) are equivalent to $\mathcal{W}^+ \equiv l[w_1a_1, \ldots, w_ma_m]$, and

4. (20), (21) and (22) are equivalent to $\mathcal{W} \equiv \mathcal{W} \wedge \mathcal{W}^-$.

Thus, we obtain the following theorem. To state it, we write $At(T)$ to denote the set of propositional atoms that occur in a PL^{PB} or PB theory T.

Theorem 3. *Let T be a PL^{PB} theory and M a set of atoms, $M \subseteq At(T)$. Then M is a model of T in the logic PL^{PB} if and only if M has a unique extension M' by some of the new atoms in $At(\tau(T))$ such that M' is a model of the normal PB theory $\tau(T)$.*

We note that our encoding of the bst problem involves PL^{PB} clauses of the form:

$$\neg ord_{y,i} \vee 1[te_{x,y} \colon \{x, y\} \in E]1$$

In this case, we do not need to introduce a new propositional atom to represent the PB constraint $1[te_{x,y} \colon \{x, y\} \in E]1$. Instead, the following two PB constraints are equivalent to the PL^{PB} clause in the encoding of the bst problem:

$$[(|E| - 1)ord_{y,i}; te_{x,y} \colon \{x, y\} \in E](|E|)$$

which is a special case of (16), and

$$0[-ord_{y,i}; te_{x,y} \colon \{x, y\} \in E]$$

which is a special case of (18).

When comparing our solvers with $wsat(oip)$ on families bst, wdm, and wnq, we ran our solvers directly on the instances from the two families and $wsat(oip)$ on their transformations by means of the mapping τ.

We used four identical machines in all our experiments. Each of these machines is equipped with a Pentium 4 3.2GHz CPU and 1GB memory, and runs Linux with kernel version 2.6.10. We used gcc 3.3.4 as a compiler for our solvers (with two flags "-Wall -O3"). We did not have access to the source code of $wsat(oip)$ and used in the experiments the binary code obtained from the author's website [24].

We set a 1000-second run-time limit to each solver we test. There is no limitation, other than the physical one, on the memory usage.

We set the $Max\text{-}Tries$ to 1 and $Max\text{-}Flips$ to a large value so that the local search solvers will not exhaust all flips during the 1000-second time limit. Therefore, a solver in our experiments halts either because it finds a model or it reaches the time limit. We perform a systematic experimental study on the noise ratio p as it may affect local search solvers' performance. We discuss this matter in more detail in the latter part of this section.

All solvers we test report the amount of time they spend to *solve* an instance. By *solving an instance*, we mean the solver finds a model of the instance. For a fair comparison, we use the GNU time program (version 1.7) to gather the timing information of all solvers. We report or perform statistical analysis on the "user time" reported by the GNU time program, which is the CPU time spent by the solvers.

For comparison, we report the following statistics of all solvers tested for each benchmark problem. We first compare the solvers instance by instance. A solver *wins*

	$wsat(plpb)$-skc	$wsat(plpb)$-rnp	$wsat(oip)$
vcv	$30/0$	$\mathbf{48/44}$	$42/4$
tsp	$1/0$	$\mathbf{50/48}$	$50/2$
bst	$50/10$	$\mathbf{50/41}$	$50/0$
wdm	$49/25$	$\mathbf{50/26}$	$4/0$
wnq	$\mathbf{50/39}$	$46/11$	$2/0$

Table 1: $wsat(plpb)$ *versus* $wsat(oip)$: summary on all instances.

on an instance if the amount of time it uses to solve the instance is the minimum one (no other solver uses less time). We report, for each benchmark problem, in how many instances a solver wins. Then we aggregate the timing information of a solver in the family of random instances by means of the *run-time distribution* (or RTD for short) instead of the simple statistical measures such as average or standard deviation. The reason is that, experiments show that the hardness of instances generated randomly with fixed parameters varies significantly. Therefore, the run time of a solver solving an instance, which can be viewed as a random variable, varies significantly as well. In other words, the probability distribution on the run time has high variance. Moreover, the run-time of a local search solver is itself a random variable even on a single instance. In this case, simple statistics such as average run time does not provide enough information about the performance of different solvers [9].

We estimate the run-time distribution of a solver over a family of randomly generated instances as follows. We run the solver on each instance and record the amount of time it takes to solve the instance. Then we estimate the probability for a solver S to solve an instance in the family F within time $0 \leq t \leq 1000$ by the ratio M_t/N, where M_t is the number of instances that are solved by S within time t and N is the total number of instances in that family.

Since the choice of the noise ratio p often has strong effect on the performance of $wsat(plpb)$-rnp, we test all methods with 9 different noise ratios $0.1, 0.2, \ldots, 0.9$. For comparisons, we use results obtained with the best value of p for each method.

For each instance, we ran each solver in one try, with the maximum number of flips set so that to guarantee the unsuccessful try does not end prior to the 1000-second limit. We set other parameters of each solver to their default settings.

4.3 Results

We first present a summary of all experiments in Table 1. It shows two values in the form s/w. Value s denotes the number of instances a solver solved in a family of instances. Value w denotes the number of instances that it solves with the shortest amount of time among all solvers we tested.

These results demonstrate the superiority of our methods over $wsat(oip)$ on the instances we use in experiments. Of the two methods we proposed, $wsat(plpb)$-rnp performs better in four out of five problems, with $wsat(plpb)$-skc being significantly better for the remaining one. We emphasize that $wsat(plpb)$-rnp algorithm performs better than $wsat(oip)$ even for problems that were encoded directly as sets of normal

PB constraints or required only small and simple modifications (splitting PB constraints into pairs of normal PB constraints in problem tsp, and rewriting disjunctions of a propositional literal and a single PB constraint into sets of normal PB constraints in problem bst, according to the method described after Theorem 3). We note that there were two ties in the experiments. The bst and the wdm families contain one instance on which $wsat(plpb)$-rnp and $wsat(plpb)$-skc terminate successfully in the same amount of time.

We now present and discuss RTD graphs for the five problems.

Figure 1(a) concerns with problem vcv. It shows that the run-time behavior of $wsat(plpb)$-rnp and $wsat(oip)$ is close with $wsat(plpb)$-rnp being better than $wsat(oip)$. $Wsat(oip)$ has a slightly higher probability of success than $wsat(plpb)$-rnp only at the point $Time \leq 128$ seconds.

Figure 1(b) shows the RTD behavior of solvers on problem tsp. Even though $wsat(plpb)$-skc and $wsat(oip)$ both have a probability 1 of success when they are given enough time (> 256 seconds), $wsat(plpb)$-skc has a clear advantage over $wsat(oip)$ for time between 4 seconds and 256 seconds.

Figure 1(c) shows that both $wsat(plpb)$-rnp and $wsat(plpb)$-skc perform uniformly better than $wsat(oip)$ in problem bst.

Figure 1(d) corresponds to problem wdm. It shows that $wsat(oip)$ is not effective. It also shows that $wsat(plpb)$-skc has a higher probability of solving easy instances (instances that can be solved in up to about 8 seconds). Then $wsat(plpb)$-rnp catches up and the performance of the two algorithms becomes similar, with $wsat(plpb)$-rnp being slightly better (in fact, it is the only algorithm that solves all instances in the family).

Figure 1(e) tells a similar story in problem wnq. $Wsat(oip)$ is not effective when the PL^{PB} theory involves general PB clauses. In this case, $wsat(plpb)$-skc performs uniformly better than $wsat(plpb)$-rnp.

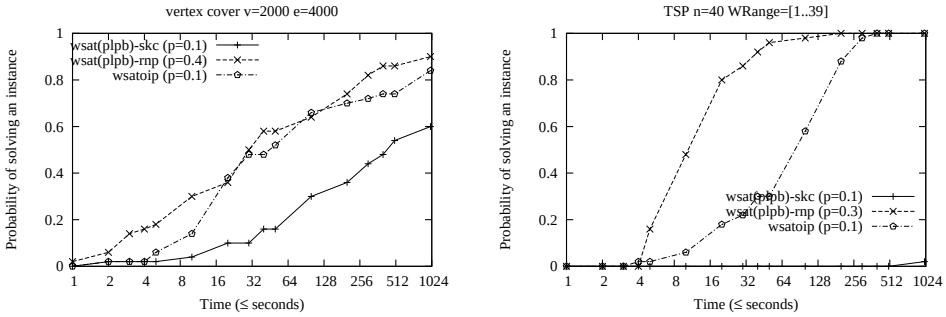
All the run-time distributions we just showed further confirm that our local search solvers designed for PL^{PB} theories perform better than $wsat(oip)$ in most of the test cases.

Finally, we discuss the *robustness* of a local search solver to the choice of the noise ratio. The noise ratio is an important parameter to all $walksat$-like solvers as it has much effect on the performance. By the range of *good behavior* we mean the range of values for the noise ratio for which a local-search algorithm performs well. A large range of good behavior is a highly desirable feature of a local-search algorithm as it makes it easier to find such values. Our graphs showing RTDs as a function of the noise ratios for the three algorithms tested (Figures 2, 3, 4, 5, and 6), show that the range of good behavior for the algorithm $wsat(plpb)$-rnp is larger than that for the other two algorithms, with one exception. For the problem wnq the range of good behavior is larger for the algorithm $wsat(plpb)$-skc.

Finally, we present our observations on the overflow problem we discussed in the previous section. We ran $wsat(plpb)$-skc in a short try (100000 flips) of one instance of each of the five benchmark problems. We recorded the total number of $\binom{n}{k}$ computation, the number of times there is an overflow, and the number of times the "best" atom chosen is involved in overflow.

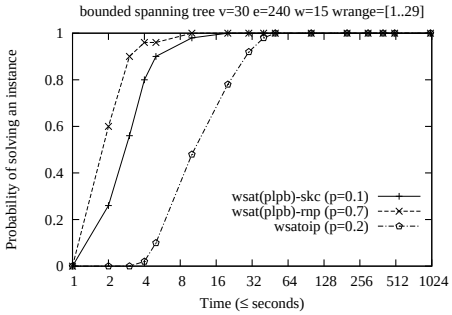
The data show that the "best" atoms are rarely the ones whose $break\text{-}count$ involves overflow. Indeed, only in the wdm problem, $wsat(plpb)\text{-}skc$ ever chose the "best" atoms that involve in overflow. Furthermore, among about 18 million of $\binom{n}{k}$ computation, out of which 3.4 million had the overflow problem, only 16 "best" atoms are the ones that involve overflow.

In addition to the five benchmark problems we considered, we also tailored our solvers to PB theories, and submitted them for the PB evaluation 2006 [17]. Our solver, called $wildcat$, won the SATUNSAT-SMALLINT category, in which it solved all 163 SAT instances. The second best solver solved 153 of the SAT instances in this category. Except for $wildcat$, all the participating solvers were complete solvers.

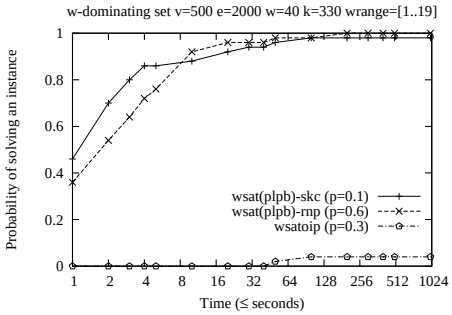


(a) vcv

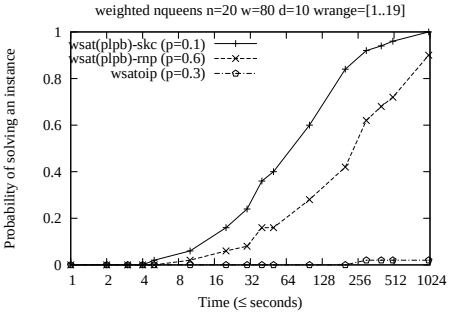
(b) tsp



(c) bst

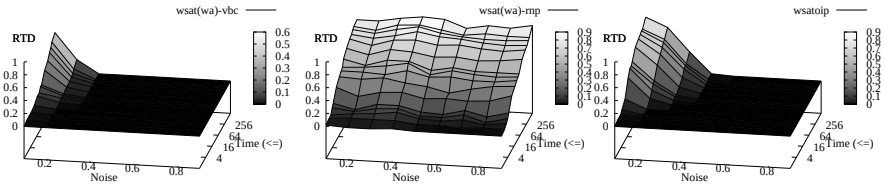


(d) wdm



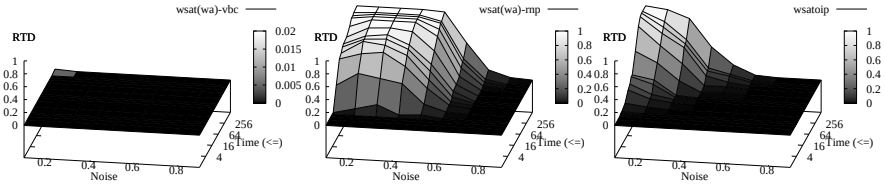
(e) wnq

Figure 1: Run-time distributions.



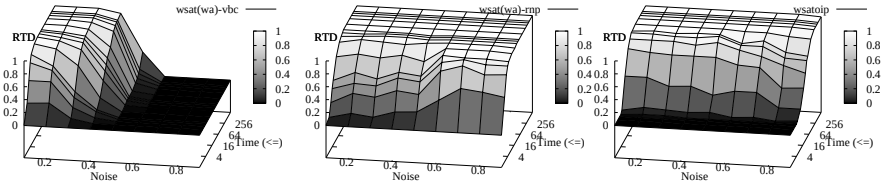
(a) $wsat(plpb)$-skc (b) $wsat(plpb)$-rnp (c) $wsat(oip)$

Figure 2: Robustness: vcv, v=2000, e=4000.



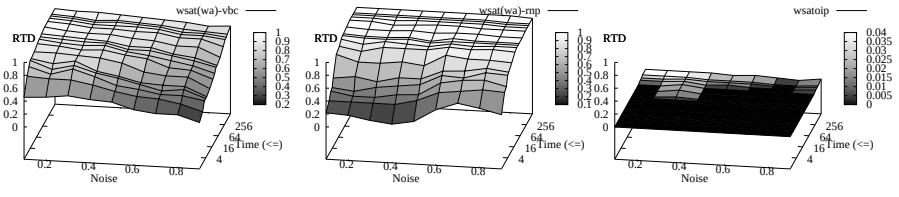
(a) $wsat(plpb)$-skc (b) $wsat(plpb)$-rnp (c) $wsat(oip)$

Figure 3: Robustness: tsp, n=40, weight range [1..39].



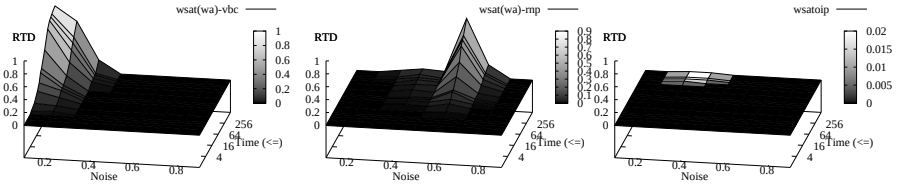
(a) $wsat(plpb)$-skc (b) $wsat(plpb)$-rnp (c) $wsat(oip)$

Figure 4: Robustness: bst, v=30, e=240, w=15, weight range [1..29].



(a) $wsat(plpb)$-skc (b) $wsat(plpb)$-rnp (c) $wsat(oip)$

Figure 5: Robustness: wdm, v=500, e=2000, w=40, k=330, weight range [1..19].



(a) $wsat(plpb)$-skc (b) $wsat(plpb)$-rnp (c) $wsat(oip)$

Figure 6: Robustness: wnq, n=20, w=80, d=10, weight range [1..19].

	# of $\binom{n}{k}$ computation	# of overflow	# of best atoms involving in overflow
vcv	152368	97204	0
tsp	8724203	7729844	0
bst	13185212	4324504	0
wdm	18118808	3403398	16
wnq	16155580	8653700	0

Figure 7: Observation on overflow problem.

5 Conclusions

We designed a family of extensible SLS algorithms for PL^{PB} theories. The key idea behind our algorithms is to view a PL^{PB} theory T as a concise representation of a certain propositional CNF theory $cl(T)$ logically equivalent to T, and to show that key parameters needed by SLS solvers developed for CNF theories can be computed on the basis of T, without the need to build $cl(T)$ explicitly. Our experiments demonstrate that our methods are superior to those relying on explicit representations of PL^{PB} clauses as sets of PB constraints and resorting to off-the-shelf local-search solvers for PB constraints such as $wsat(oip)$.

Clearly, CNF representations of PB constraints other than the one we used exist and could be used within a general approach we developed, as long as one can derive formulas (or procedures) to compute values of e, f and g. In fact, we can push this idea even further. For an arbitrary constraint (not necessarily a PB constraint), if we can evaluate e, f and g in some translation that converts it into a set of propositional clauses, our general framework yields solvers accepting theories containing such constraints.

Finally, we point out that the formulas we derived use values of the form $\binom{n}{k}$, which will overflow already for relatively small values of n, if k is close to $n/2$. In our experiments, even though in some cases overflows occurred quite often (which we replaced with a certain fixed large integer), for the atoms our solvers selected to flip the computation of virtual counts only rarely involved overflows. Still, in our future research we will study how to approximate $\binom{n}{k}$ to avoid overflows. Since we only care about the relative order of the break- and make-counts of atoms, any approximation that maintains this ordering will be appropriate.

Acknowledgments

We acknowledge the support of NSF grant IIS-0325063 and KSEF grant KSEF-1036-RDE-008. We are thankful to the reviewers of the paper for many helpful comments.

References

[1] F.A. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: a backtrack-search pseudo-Boolean solver and optimizer. In *Proceedings of the 5th International Symposium on Theory and Applications of Satisfiability*, pages 346 – 353, 2002.

[2] P. Barth. A Davis-Putnam based elimination algorithm for linear pseudo-Boolean optimization. Technical Report, Max-Planck-Institut für Informatik, 1995. MPI-I-95-2-003.

[3] B. Benhamou, L. Sais, , and P. Siegel. Two proof procedures for a cardinality based language in propositional calculus. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science (STACS-1994)*, volume 775 of *LNCS*, pages 71–82. Springer, 1994.

[4] H.E. Dixon and M.L. Ginsberg. Inference methods for a pseudo-Boolean satisfiability solver. In *The 18th National Conference on Artificial Intelligence (AAAI-2002)*, pages 635–640. AAAI Press, 2002.

[5] D. East and M. Truszczyński. Propositional satisfiability in answer-set programming. In *Proceedings of Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, volume 2174 of *LNAI*, pages 138–153. Springer, 2001.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness*. W.H. Freeman and Co., San Francisco, Calif., 1979.

[7] E. Goldberg and Y. Novikov. Berkmin: a fast and robust sat-solver. In *DATE-2002*, pages 142–149. 2002.

[8] H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of The Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 661–666, Orlando, Florida, 1999.

[9] Holger H. Hoos and Thomas Stutzle. A characterization the run-time behaviour of stochastic local search. `http://www.cs.ubc.ca/~hoos/Publ/aida-98-01.ps`, 1998.

[10] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann, San Francisco (CA), USA, 2004.

[11] C.M. Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proccedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 291–296. AAAI Press, 2000.

[12] C.M. Li and M. Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, volume 1330 of *LNCS*, pages 342–356. Springer, 1997.

[13] L. Liu. *Computational tools for solving hard search problems*. PhD thesis, University of Kentucky, 2006. `ftp://ftp.cs.uky.edu/cs/manuscripts/LiuDissertation.pdf`.

[14] L. Liu and M. Truszczyński. Local-search techniques in propositional logic extended with cardinality atoms. In F. Rossi, editor, *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP-2003)*, volume 2833 of *LNCS*, pages 495–509. Springer, 2003.

[15] L. Liu and M. Truszczyński. Local search techniques for Boolean combinations of pseudo-Boolean constraints. In *Proceedings of The Twentieth National Conference on Artificial Intelligence (AAAI-06)*, pages 98–103. AAAI Press, 2006.

[16] V. Manquinho and O. Roussel. Pseudo Boolean evaluation 2005, 2005. `http://www.cril.univ-artois.fr/PB05/`.

[17] V. Manquinho and O. Roussel. Pseudo Boolean evaluation 2006, 2006. http://www.cril.univ-artois.fr/PB06/.

[18] V.M. Manquinho and O. Roussel. The first evaluation of pseudo-Boolean solvers (pb'05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2:103–143, 2006.

[19] J.P. Marques-Silva and K.A. Sakallah. GRASP: A new search algorithm for satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[20] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th ACM IEEE Design Automation Conference*, pages 530–535. ACM Press, 2001.

[21] S.D. Prestwich. Randomised backtracking for linear pseudo-Boolean constraint problems. In *Proceedings of the 4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, (CPAIOR-2002)*, pages 7–20, 2002. http://www.emn.fr/x-info/cpaior/Proceedings/CPAIOR.pdf.

[22] R. Sebastiani, Applying GSAT to Non-Clausal Formulas (Research Note). *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[23] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-1994)*, pages 337–343, Seattle, USA, 1994. AAAI Press.

[24] J. Walser. SLS PB solver *wsatoip*, 1997. http://www.ps.uni-sb.de/~walser/wsatpb/wsatpb.html.

[25] J.P. Walser. Solving linear pseudo-Boolean constraints with local search. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-97)*, pages 269–274. AAAI Press, 1997.

[26] H. Zhang. SATO: an efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction (CADE-97)*, volume 1104 of *LNAI*, pages 308–312, 1997.