Local-search Techniques for Boolean Combinations of Pseudo-boolean Constraints

Lengning Liu and **Mirosław Truszczyński**
Department of Computer Science, University of Kentucky,
Lexington, KY 40506-0046, USA

Abstract

Some search problems are most directly specified by boolean combinations of pseudo-boolean constraints. We study a logic $PL(PB)$ whose formulas are of this 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 CNF theory $cl(T)$ logically equivalent to T. We show 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. Since $cl(T)$ is often much larger than T, running search based on T promises performance gains. Our experimental results confirm this expectation.

Introduction

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

Recent advances in the performance of SAT solvers make them effective in solving search problems that can be reduced to finding a model of a certain CNF theory. SAT solvers fall in two camps: *complete* and *incomplete* ones. Complete solvers find a model of an input theory, when the theory is satisfiable. Otherwise, they generate a message that no models exist. Incomplete solvers either return a model of an input theory or terminate with no output. In the latter case, the satisfiability of the theory remains unknown.

In this paper, we focus on incomplete solvers, specifically, stochastic local search solvers (SLS solvers, for short). Although incomplete solvers do not guarantee to find a model when there is one, their ability to compute models of large satisfiable theories, which are often beyond the power of complete solvers, makes them attractive.

A drawback of SAT solvers is that they require an input theory to be in CNF. Constraints defining search problems of practical importance often do not have a direct representation as a single clause and in many cases require large sets of clauses to be faithfully described. Constraints involving numeric values, typically modeled as linear inequalities are such constraints. Large sizes of CNF theories representing search problems limit the effectiveness of SAT solvers.

To circumvent this issue, researchers studied constraints that are more general than propositional clauses and are attuned to constraints commonly appearing in applications. Certain integer programming constraints, called *pseudo-boolean*, received particular attention (Benhamou, Sais, & Siegel 1994; Barth 1995; Dixon & Ginsberg 2002; Hooker 2000). This research resulted in several solvers of pseudo-boolean constraints (Walser 1997; Preswitch 2002; Aloul *et al.* 2003; Manquinho & Roussel 2005).

We argue here that in some applications constraints are most directly stated as *boolean combinations* of pseudo-boolean constraints. We define a logic to describe such constraints and propose SLS solvers to compute models of theories in this logic. Specifically, our contributions are:

1. We propose a general formalism, called the propositional logic with pseudo-boolean constraints (or $PL(PB)$ for short), for modeling search problems. This logic subsumes both propositional logic and the formalism of pseudo-boolean constraints. We present examples of search problems, where combinations of pseudo-boolean constraints appear naturally.
2. We generalize the concepts of the *break-* and *make-counts* and develop methods to estimate them. We apply these results to design SLS solvers for arbitrary $PL(PB)$-theories, extending approaches from (Selman, Kautz, & Cohen 1994; Hoos 1999; Liu & Truszczyński 2003).
3. We demonstrate experimentally that our solvers are competitive with $wsat(oip)$(Walser 1997) on theories consisting of pseudo-boolean constraints and significantly faster on problems with constraints most directly stated as disjunctions of pseudo-boolean constraints.

Technical preliminaries

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

$$l \leq w_1x_1 + \ldots + w_kx_k \leq u, \qquad (1)$$

where x_i are integer variables, each with the domain $\{0, 1\}$, w_i are integers, which we will refer to as *weights*, and l and u are integers called the *bounds*. An assignment v of 0s and 1s to $x_i's$ is a *model* of (or *satisfies*) the constraint (1) if $l \leq w_1v(x_1) + \ldots + w_kv(x_k) \leq u$ holds.

If one of the bounds in the constraint (1) is missing, we call it a *strict* pb-constraint. Typically only strict pb-constraints are considered as every pb-constraint is equivalent to a set of two strict 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 $l[w_1 x_1, \ldots, w_k x_k]u$. We will omit the appropriate bound for strict pb-constraints. If all weights w_i are equal to 1, we drop them from the notation and write $l[x_1, \ldots, x_k]u$ instead of $l[w_1 x_1, \ldots, w_k x_k]u$. We refer to such pb-constraints as *cardinality constraints*.

By establishing the correspondence between integer values 0 and 1 on the one hand, and truth values $\mathbf{false}$ and $\mathbf{true}$, respectively, on the other, we can view integer 0-1 variables as propositional atoms. Furthermore, we can 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, but now interpreted as propositional atoms) if (1) and φ have the same models (modulo the correspondence between $\{0, 1\}$ and $\{\mathbf{false}, \mathbf{true}\}$). In particular, a (strict) pb-constraint

$$1 - m \leq x_1 + \ldots + x_k - y_1 - \ldots - y_m$$

represents a propositional clause

$$x_1 \vee \ldots \vee x_k \vee \neg y_1 \vee \ldots \vee \neg y_m.$$

Thus, pb-constraints generalize clauses, and sets of pb-constraints generalize propositional CNF theories.

Many practical search and optimization problems have concise encodings in terms of pb-constraints. To solve such problems by means of SAT solvers — an approach that received much attention lately due to advances in the performance of SAT solvers — each of the pb-constraints involved in the problem statement must be compiled first into a set of propositional clauses. However, the resulting CNF theory is often much larger than the original set of pb-constraints, which hinders the effectiveness of SAT solvers. Hence, researchers started extending techniques developed for and implemented in SAT solvers to handle collections of pb-constraints *directly* (Walser 1997; Aloul *et al.* 2003).

In this paper we are interested in an even broader class of theories, namely theories consisting of constraints (formulas) that are boolean combinations of propositional literals and pb-constraints (viewed as propositional formulas). We refer to the formalism we are about to describe as *propositional 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 and theories.

A *$PL(PB)$-clause* (or, simply, a *clause*) is an expression of the form

$$l_1 \vee \ldots \vee l_m \vee W_1 \vee \ldots \vee W_n, \qquad (2)$$

where l_i's are propositional literals and W_i's are pb-constraints. We call $l_1 \vee \ldots \vee l_m$ the *propositional* disjunct of the clause (2). A *$PL(PB)$-theory* is any set of $PL(PB)$-clauses.

The notions of *satisfiability* and a *model* extend in a standard way to $PL(PB)$-clauses and $PL(PB)$-theories. We will write $I \models E$, when I is a model of a $PL(PB)$-clause or $PL(PB)$-theory E.

The following problem, a slight generalization of the dominating-set problem in graphs (Garey & Johnson 1979), illustrates the usefulness of $PL(PB)$-clauses in modeling.

Weighted dominating-set problem. Let $G = (V, E)$ be a directed graph with each edge (x, y) assigned an integer weight $w_{x,y} \geq 0$. Given an integer w, a set $D \subseteq V$ of vertices of G is *w-dominating* for G if for every vertex $x \in V$ at least one of the conditions listed below holds.

1. $x \in D$
2. the sum of weights of edges "from x to D" is at least w: $w \leq \sum_{(x,y)\in E, y\in D} w_{x,y}$
3. the sum of weights of edges "from D to x" is at least w: $w \leq \sum_{(z,x)\in E, z\in D} w_{z,x}$.

The following $PL(PB)$-theory encodes the problem of the existence of a w-dominating set with at most k vertices. In the encoding we use atoms in_x, $x \in V$, with the intended meaning: *vertex x is in a w-dominating set.* The clauses of the theory are:

1. $in_x \vee W_1 \vee W_2$, for every $x \in V$, where
   $W_1 = w[w_{x,y} in_y \colon (x, y) \in E]$, and
   $W_2 = w[w_{z,x} in_z \colon (z, x) \in E]$.
   These clauses enforce the defining constraint for a w-dominating set.
2. $[in_x \colon x \in V]k$.
   This clause guarantees that a selected subset has at most k vertices.

We note that $PL(PB)$-clauses of the first type are disjunctions of a propositional atom and two pb-constraints. It is not obvious how to concisely rewrite the set of these clauses as a set of pb-constraints without using auxiliary variables. If one introduces auxiliary variables, such representations can be found but they are also of larger size than the original set. This underscores the potential of $PL(PB)$-theories in modeling and shows that it is important to design solvers that can find models of $PL(PB)$-theories directly without rewriting.

Local-search algorithms for logic *PL(PB)*

In this section we describe a family of SLS algorithms designed to compute models of $PL(PB)$-theories. The general structure of the algorithms follows that of *walksat* (Selman, Kautz, & Cohen 1994) and $vb\text{-}WSAT^{cc}$, the latter proposed in (Liu & Truszczyński 2003) for a fragment of logic $PL(PB)$, in which formulas are built of cardinality constraints. Briefly, the algorithms execute $Max\text{-}Tries$ independent *tries*. Each try starts in a randomly generated truth assignment and consists of a sequence of up to $Max\text{-}Flips$ *flips*, that is, local changes to the current truth assignment. A flip usually selects an atom in an input theory and changes its truth value in the current truth assignment to its dual. The algorithms terminate 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).

Algorithms that implement this general structure differ in heuristics they use to select an atom for a flip. Experiments demonstrated that for standard CNF theories two heuristics are particularly effective: the *SKC* heuristics (Selman, Kautz, & Cohen 1994) and the *RNovelty+* heuristics (Hoos 1999). The SKC heuristics takes into account the *break-count* of an atom, that is, the number of clauses that are satisfied by the current assignment but become unsatisfied once we flip the atom. The $RNovelty+$ heuristics, in addition to the break-count, also considers the *make-count* of an atom, that is, the number of clauses that are not satisfied by the current assignment but become satisfied after we flip the atom. If S is a CNF formula (or a set of clauses in this formula), I is a truth assignment and x is an atom, we denote the corresponding break- and make-counts by $bct_S(x)$ and $mct_S(x)$, respectively. Since I is always determined by the context, we do not explicitly refer to I in the notation.

(Liu & Truszczyński 2003) extended the SKC heuristics to $PL(PB)$-theories, in which every pb-constraint is a cardinality constraint. The key idea behind this extension is that of the *virtual break-count*. (Liu & Truszczyński 2003) proposed a way to compile a theory T, with cardinality constraints, into an equivalent set of *propositional clauses*, $cl(T)$, and defined the virtual break-count of an atom x in T to be the break-count of x in $cl(T)$. (Liu & Truszczyński 2003) then showed that virtual break-count can be computed directly from the input theory, without the need to actually produce the set $cl(T)$. This observation is fundamental as $cl(T)$ is often exponentially larger than T.

In the remainder of the paper, we extend the translation $T \mapsto cl(T)$ to *arbitrary* $PL(PB)$-theories. We then use this translation to define the *virtual break-* and *make-counts*. For both concepts we develop fast methods to estimate them directly on the basis of T and not requiring that $cl(T)$ be computed explicitly. We apply the virtual break-count and make-count to extend SKC and $RNovelty+$ heuristics to the case of arbitrary $PL(PB)$-theories and obtain in this way two SLS solvers for computing models of $PL(PB)$-theories, $wsat(plpb)$-skc and $wsat(plpb)$-rnp, respectively.

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 will allow repetitions of clauses in sets and repetitions of literals in clauses, as by doing so we simplify some technical 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 will 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 (2). 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 will denote these two quantities as $bct_T(x)$ and $mct_T(x)$, respectively (we again drop the reference to I from the notation). It follows that

$$bct_T(x) = bct_{cl(T)}(x) = \sum\{bct_{cl(C)}(x)\colon C \in T\}.$$

Similarly,

$$mct_T(x) = mct_{cl(T)}(x) = \sum\{mct_{cl(C)}(x)\colon C \in T\}.$$

We now estimate $bct_{cl(C)}(x)$ and $mct_{cl(C)}(x)$. 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 $bct_{cl(C)}(x)$ and $mct_{cl(C)}(x)$ (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 $E_{W,x} = F_{W,x} = \emptyset$ if x does not appear in W. We set $e = |E_{W,x}|$, $f = |F_{W,x}|$ and $g = |G_{W,x}|$.

The following formula computes $bct_{cl(C)}(x)$ (we label e, f and g with indices i of pb-constraints W_i occurring in (2). We write L for the propositional disjunct $l_1 \vee \ldots \vee l_m$ of C):

$$bct_{cl(C)}(x) = \begin{cases} 0 & \text{case 1} \\ \prod_{i=1}^{n}(e_i + g_i) & \text{case 2} \\ \prod_{i=1}^{n}(e_i + g_i) - \prod_{i=1}^{n} g_i & \text{o/w} \end{cases} \tag{3}$$

where case 1 occurs when $I^{\bar{x}} \models L$ and case 2 occurs when case 1 does not hold and $I \models L$.

Indeed, every clause in $cl(C)$ is of the form $L \vee D_1 \vee \ldots \vee D_n$, where $D_i \in T_{W_i}$, $1 \leq i \leq n$. In case 1, all such clauses are satisfied in $I^{\bar{x}}$. Thus, $bct_{cl(C)}(x) = 0$. In case 2, all clauses in $cl(C)$ are satisfied in I. In order not to be satisfied in $I^{\bar{x}}$, every disjunct D_i must be an element of $E_{W_i,x} \cup G_{W_i,x}$. Thus, there are $\prod_{i=1}^{n}(e_i + g_i)$ such clauses in $cl(C)$. The argument for the last case is similar.

Reasoning as above, we also obtain a formula for $mct_{cl(C)}(x)$:

$$mct_{cl(C)}(x) = \begin{cases} 0 & \text{case 1} \\ \prod_{i=1}^{n}(f_i + g_i) & \text{case 2} \\ \prod_{i=1}^{n}(f_i + g_i) - \prod_{i=1}^{n} g_i & \text{o/w} \end{cases} \tag{4}$$

where case 1 occurs when $I \models L$ and case 2 occurs when case 1 does not hold and $I^{\bar{x}} \models L$.

To make these formulas 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 our discussion, we assume that W contains no negative weights. This assumption simplifies the discussion but is not essential.

Let us then consider a pb-constraint W:

$$W = l[a_1w_1, \ldots, a_kw_k]u,$$

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

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

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.

One can check that the cardinality constraint W' is equivalent to the set S consisting of the following clauses:

$$\neg x_{i_1} \vee \ldots \vee \neg x_{i_{u+1}} \tag{5}$$

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

$$x_{i_1} \vee \ldots \vee x_{i_{K-l+1}} \tag{6}$$

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 $K = \sum w_i$.

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:

$$e = \begin{cases} 0 & \text{case 1} \\ \binom{N+w}{K-l+1} - \binom{N}{K-l+1} & \text{case 2} \\ \binom{P+w}{u+1} - \binom{P}{u+1} & \text{otherwise} \end{cases} \tag{7}$$

$$f = \begin{cases} 0 & \text{case 1} \\ \binom{P}{u+1} - \binom{P-w}{u+1} & \text{case 2} \\ \binom{N}{K-l+1} - \binom{N-w}{K-l+1} & \text{otherwise} \end{cases} \tag{8}$$

$$g = \begin{cases} \binom{N}{K-l+1} + \binom{P}{u+1} & \text{case 1} \\ \binom{N}{K-l+1} + \binom{P-w}{u+1} & \text{case 2} \\ \binom{P}{u+1} + \binom{N-w}{K-l+1} & \text{otherwise.} \end{cases} \tag{9}$$

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 provide an argument for the case 2 of (7). In this case, x occurs in W and $I \models x$. Let us assume that $x = a_1$ and let $\mathcal{N} = \{a_i^j \colon 1 \leq i \leq k,\ 1 \leq j \leq w_i,\ I \not\models a_i\}$. 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 (6) 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 (7), case 2, follows.

We now use the formulas for break- and make-counts to design algorithms *wsat(plpb)-skc* and *wsat(plpb)-rnp*. The algorithm *wsat(plpb)-skc* follows the design of *walksat* (Selman, Kautz, & Cohen 1994) and *vb-WSAT*^{cc} (Liu & Truszczyński 2003), except that it uses the formulas we derived above to compute virtual break-counts of atoms. It accepts arbitrary $PL(PB)$-theories. The pseudo-code is given in Algorithm 1. We note that the algorithm decides between a random choice and a greedy choice in lines 5 and 6 according to the probability p, called the *noise ratio*.

Algorithm 1 Heuristic function $SKC(T, I, C)$ used in *wsat(plpb)-skc*

```
INPUT:   $T$ - a $PL(PB)$-theory
         $I$ - a truth assignment of $T$
         $C$ - an unsat clause
OUTPUT:  $a$ - an atom (to be flipped)
BEGIN
1.  For each atom $x$ in $C$, compute $bct(x)$;
2.  If exist atoms whose $bct = 0$ then
3.      randomly return such an atom;
4.  Else
5.      with probability $p$, return a randomly chosen atom in $C$;
6.      otherwise, return an atom $x$ with minimum $bct(x)$;
7.  End If
END
```

(Hoos 1999) introduced a different heuristics, *RNovelty+*, for choosing an atom to flip and showed it to be effective for *walksat*. It uses both the break- and the make-count values. By replacing these values with the virtual break-count and the virtual make-count, respectively, we obtain a version of *RNovelty+* that computes models of $PL(PB)$-theories. We present a pseudo-code description in Algorithm 2.

To help search escape loops, with probability wp the heuristics chooses a random atom from the input clause C to return as the next atom to flip. As in (Hoos 1999), we use $wp = 0.01$ in our implementation. Otherwise, the algorithm selects an atom to flip based on the *quality* of atoms (the quality of an atom is a difference between its virtual break- and make-counts), the *age* of atoms (the age of an atom is defined as the time, measured in flips, when the atom was last flipped; initially all atoms have age 0), and a probability p, which determines whether an atom with the best or the second best quality value is selected. Even though the role of the parameter p is different here than it is in the case of the *SKC* heuristics, we call this value the *noise ratio*, as well. We also note that if all atoms in C have the same value of *qlty*, then one is selected randomly.

Experiments

We tested the implementations of our algorithms on $PL(PB)$-theories encoding instances of four search prob-

Algorithm 2 Heuristic function $RNovelty{+}(T, I, C)$ used in $wsat(plpb)$-rnp

```
INPUT and OUTPUT as in Algorithm 1
BEGIN
1.  With probability $wp$, return a random atom from $C$;
2.  For each atom $x$ in $C$,
        $qlty(x) \leftarrow bct(x) - mct(x)$;
3.  $age_{max} \leftarrow$ the maximum age of atoms in $C$;
4.  $best \leftarrow$ atoms $x$ with the least $qlty(x)$;
5.  $second \leftarrow$ atoms $x$ with the second least $qlty(x)$;
6.  If $second = \emptyset$, return a random atom from $C$;
7.  $diff \leftarrow qlty(x) - qlty(y)$, where $x \in best$, $y \in second$;
8.  If $\exists a \in best$ such that its age $< age_{max}$, return $a$;
9.  If $diff > 1$, then
10.     with probability $min\{2 - 2p, 1\}$,
          return a random atom from $best$;
11.     otherwise, return a random atom from $second$;
12. End If
13. With probability $max\{1 - 2p, 0\}$,
        return a random atom from $best$;
14. otherwise, return a random atom from $second$;
END
```

lems. We compared the performance of our solvers to that of $wsat(oip)$ (Walser 1997). The four search problems are:

1. **Vertex-cover problem (*vcov*).** Given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, find a set $U \subseteq V$ such that $|U| \leq k$ and every edge in E has at least one of its vertices in U.
2. **Bounded spanning tree problem (*bst*).** Let $G = (V, E)$ be an undirected graph with each edge assigned an integer weight. Given an integer w 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.
3. **Weighted dominating set problem (*wdm*).** The problem is defined earlier in the paper.
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' does not exceed d.

For testing, for each problem we generated 50 random instances, setting parameters so that all instances had solutions and then expressed the instances as $PL(PB)$-theories. We presented the $PL(PB)$-theory encoding an instance of the problem *wdm* earlier. Encodings for other problems can be found at (Liu & Truszczyński 2006). We note that:

1. Theories for the *vcov* problem consist only of strict pb-constraints and are accepted directly by our programs and $wsat(oip)$.
2. Theories for the *bst* problem contain formulas which are not strict pb-constraints. However, these formulas have simple representations as one or two strict pb-constraints and do not require the help of new atoms. In experiments, we use original encodings with our algorithms and transformed encodings with $wsat(oip)$.
3. Theories encoding instances of the last two problems consist of non-unary $PL(PB)$-clauses. To avoid a blow-up in the size of representation, when expressing these clauses in terms of sets of pb-constraints, we *need* to introduce new atoms. As before, we use original encodings with our algorithms and transformed encodings with $wsat(oip)$.

	$wsat(plpb)$-skc	$wsat(plpb)$-rnp	$wsat(oip)$
$vcov$	30/0/474.89	**48/44/4.92**	42/4/25.26
bst	50/10/2.03	**50/41/1.57**	50/0/5.80
wdm	49/25/0.65	**50/26/0.64**	4/0/1000
wnq	**50/39/52.24**	46/11/218.85	2/0/1000

Table 1: Summary on all instances

We could encode the instances of these four problems as CNF theories using the naive encoding defined by clauses (5) and (6). However, the sizes of the resulting theories are too large for current solvers such as WalkSAT to be effective.

Since the choice of the noise ratio p often has strong effect on the performance of $wsat(plpb)$-rnp, we tested all methods with 9 different noise ratios $0.1, 0.2, \ldots, 0.9$. For comparisons, we used results obtained with the best value of p for each method. For each instance, we allocated 1000 seconds to each method and ran it 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. We then recorded the CPU time spent by each method on each instance. All experiments were done on P4 3.2GHz machines with 1GB memory and Linux kernel version 2.6.10.

Table 1 is a summary of all experiments with entries of the form $s/w/m$, where s is the number of instances in a family solved by a solver, w is the number of instances when the solver was the fastest one, and m is the median running time (over all 50 instances in the family; the time of 1000 seconds was used whenever the solver timed out on an instance).

These results demonstrate the superiority of our methods over $wsat(oip)$ on the instances we used in experiments. Of the two methods we proposed, $wsat(plpb)$-rnp performs better in three out of four problems, with $wsat(plpb)$-skc being significantly better for the remaining one. We emphasize, that our algorithms performed better than $wsat(oip)$ even for problems that were encoded directly as sets of strict pb-constraints or required only small and simple modifications (problems *vcov* and *bst*). There is only one exception: for the problem *vcov* $wsat(oip)$ outperformed $wsat(plpb)$-skc (but was outperformed by $wsat(plpb)$-rnp).

(Hoos & Stützle 2005) argued that *run-time distribution* (or *RTD* for short) is a more reliable measure to compare the performance of SLS solvers. We now present RTD graphs for the problems *bst* and *wdm* problem. Figure 1 shows that $wsat(plpb)$-rnp performs the best.

Figure 2 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 is very similar, with $wsat(plpb)$-rnp being slightly better (in fact, it is the only

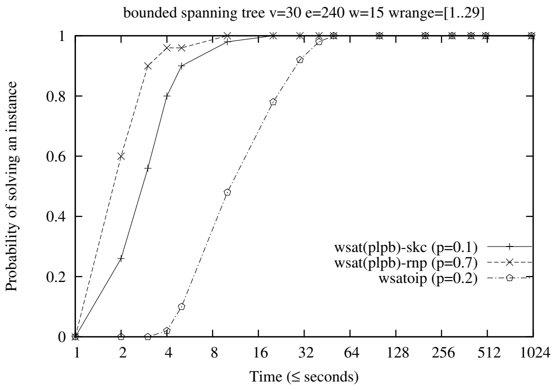

Figure 1: RTDs on the bst problem

algorithm that solved all instances in the family).

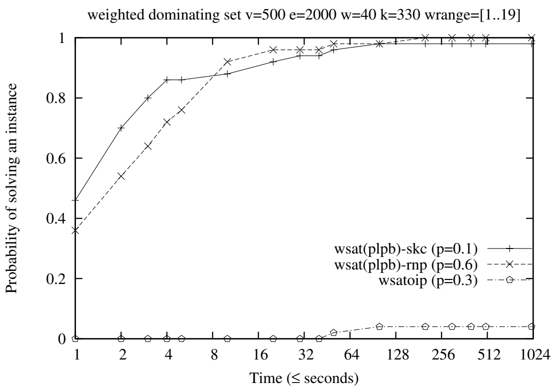

Figure 2: RTDs on the wdm problem

We do not present here the two other RTD graphs. They can be found at (Liu & Truszczyński 2006). In the case of the problem *vcov*, RTDs show that $wsat(\mathit{plpb})\text{-}\mathit{rnp}$ performs better than both $wsat(\mathit{oip})$ and $wsat(\mathit{plpb})\text{-}\mathit{skc}$. The RTD graph for the problem *wnq* shows that $wsat(\mathit{oip})$ is not effective at all (it solves only two instances), and that $wsat(\mathit{plpb})\text{-}\mathit{skc}$ performs much better than $wsat(\mathit{plpb})\text{-}\mathit{rnp}$.

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 $wsat(\mathit{oip})$.

Clearly, CNF representations of pb-constraints other than $W \mapsto T_W$ are possible 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 fi xed 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.

References

Aloul, F.; Ramani, A.; Markov, I.; and Sakallah, K. 2003. PBS v0.2, incremental pseudo-boolean backtrack search SAT solver and optimizer. `http://www.eecs.umich.edu/~faloul/Tools/pbs/`.

Liu, L., and Truszczyński, M. 2006. Experiments with algorithms $wsat(\mathit{plpb})\text{-}\mathit{skc}$ and $wsat(\mathit{plpb})\text{-}\mathit{rnp}$. `http://www.cs.uky.edu/ai/wsatcc/exp/`.

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

Benhamou, B.; Sais, L.; and Siegel, P. 1994. Two proof procedures for a cardinality based language in propositional calculus. In *Procs. of STACS-94*, vol 775, *LNCS*. Springer. 71–82.

Dixon, H., and Ginsberg, M. 2002. Inference methods for a pseudo-boolean satisfiability solver. In *Procs. of AAAI-02*, 635–640. AAAI Press.

Garey, M., and Johnson, D. 1979. *Computers and intractability. A guide to the theory of NP-completeness*. San Francisco, Calif.: W.H. Freeman and Co.

Hooker, J. 2000. *Logic-Based Methods for Optimization*. J. Wiley and Sons.

Hoos, H., and Stützle, T. 2005. *Stochastic Local Search Algorithms — Foundations and Applications*. Morgan-Kaufmann.

Hoos, H. 1999. On the run-time behaviour of stochastic local search algorithms for sat. In *Procs. of AAAI-99*, 661–666. AAAI Press.

Liu, L., and Truszczyński, M. 2003. Local-search techniques in propositional logic extended with cardinality atoms. In *Procs. of CP-03*, volume 2833 of *LNCS*. Springer. 495–509.

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

Preswitch, S. 2002. Randomised backtracking for weightless linear pseudo-boolean constraint problems. In *Procs. of CPAIOR-02*, 7–19.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Procs. of AAAI-94*, 337–343. AAAI Press.

Walser, J. 1997. Solving linear pseudo-boolean constraints with local search. In *Procs. of AAAI-97*, 269–274. AAAI Press.