

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.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

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

$$l \leq w_1x_1 + \dots + w_kx_k \leq u, \quad (1)$$

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 $l \leq w_1v(x_1) + \dots + w_kv(x_k) \leq u$ 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 $l[w_1x_1, \dots, w_kx_k]u$. 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 $l[x_1, \dots, x_k]u$ instead of $l[w_1x_1, \dots, w_kx_k]u$. 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 **false** and **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

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

represents a propositional clause

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

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

$$l_1 \vee \dots \vee l_m \vee W_1 \vee \dots \vee W_n, \quad (2)$$

where l_i 's are propositional literals and W_i 's are pb-constraints. We call $l_1 \vee \dots \vee l_m$ 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
 $W_1 = w[w_{x,y}in_y : (x, y) \in E]$, and
 $W_2 = w[w_{z,x}in_z : (z, x) \in E]$.
 These clauses enforce the defining constraint for a *w-dominating set*.
2. $[in_x : x \in V]k$.
 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-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-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. 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)-rmp*, 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

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

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

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

Similarly,

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

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 $l_1 \vee \dots \vee l_m$ of C :

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

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 $L \vee D_1 \vee \dots \vee D_n$, 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 $mct_{cl(C)}(x)$:

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

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 :

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

