Logic Programming with Pseudo-Boolean Constraints

Alexander Bockmayr

MPI–I–91–227 April/December 1991

Author's Address

Alexander Bockmayr, Max-Planck-Institut für Informatik, Im Stadtwald, D-6600 Saarbrücken, bockmayr@mpi-sb.mpg.de

Publication Notes

The present report has been submitted for publication elsewhere and will be copyrighted if accepted.

Acknowledgements

I would like to thank Peter Barth, David Basin and Harald Ganzinger for their comments on a preliminary version of this paper.

Abstract

Boolean constraints play an important role in various constraint logic programming languages. In this paper we consider pseudo-Boolean constraints, that is equations and inequalities between pseudo-Boolean functions. A pseudo-Boolean function is an integer-valued function of Boolean variables and thus a generalization of a Boolean function. Pseudo-Boolean functions occur in many application areas, in particular in problems from operations research. An interesting connection to logic is that inference problems in propositional logic can be translated into linear pseudo-Boolean optimization problems. More generally, pseudo-Boolean constraints can be seen as a particular way of combining two of the most important domains in constraint logic programming: arithmetic and Boolean algebra.

In this paper we define a new constraint logic programming language *CLP(PB)* for logic progamming with pseudo-Boolean constraints. The language is an instance of the general constraint logic programming language scheme *CLP(X)* and inherits all the typical semantic properties. We show that any pseudo-Boolean constraint has a most general solution and give variable elimination algorithms for pseudo-Boolean unification and unconstrained pseudo-Boolean optimization. Both algorithms subsume the well-known Boolean unification algorithm of Büttner and Simonis.

Keywords

Constraint Logic Programming, Operations Research, Pseudo-Boolean Unification, Pseudo-Boolean Optimization, 0-1 Programming

Contents

1 Introduction

Boolean constraints play an important role in various constraint logic programming languages [Col87, DvHS$^+$88, ASS$^+$88]. In this paper we consider *pseudo-Boolean constraints*, that is equations and inequalities between pseudo-Boolean functions. A *pseudo-Boolean function* is an integer-valued function $f : \{0,1\}^n \to \mathcal{Z}$ and thus generalizes the notion of a Boolean function $f : \{0,1\}^n \to \{0,1\}$.

Pseudo-Boolean functions have been studied for a long time [HR68]. There exist many interesting methods and results on the solution of pseudo-Boolean constraints that have been developed mainly in the area of operations research. It seems very natural to apply these techniques in the context of constraint logic programming. A constraint logic programming language can be considerably enhanced by allowing pseudo-Boolean constraints. Many problems from various application areas can be expressed very naturally using pseudo-Boolean functions.

Consider for example the following knapsack problem. Suppose there is a vessel with capacity w and goods g_i with weight w_i and value v_i for $i = 1, \ldots, 12$. We introduce a Boolean variable $X_i, i = 1, \ldots, 12$, which indicates whether or not the good g_i is loaded on the vessel. The possible cargos not exceeding the capacity of the vessel can be determined by the following Horn clause involving a pseudo-Boolean constraint

```
cargo(X1,...,X12) :-
                w1*X1 +...+ w12*X12 ≤ w.
```

If we want to find the most valuable cargo, this can be done using a metapredicate max/2 that maximizes a pseudo-Boolean function subject to some pseudo-Boolean constraint.

```
most-valuable-cargo(X1,...,X12) :-
                          max(v1*X1 +...+ v12*X12, cargo(X1,...,X12)).
```

An interesting connection between logic and pseudo-Boolean optimization is that inference problems in propositional logic can be translated into linear pseudo-Boolean optimization problems.

For example, suppose we want to know whether X_2 is implied by the set of clauses

$$\begin{array}{ccccccc}
X_1 & \vee & X_2 & \vee & X_3, & & \\
\overline{X}_1 & \vee & X_2 & \vee & & & \overline{X}_4, \\
\overline{X}_1 & \vee & & & X_3, & & \\
\overline{X}_1 & \vee & & & \overline{X}_3 & \vee & X_4, \\
X_1 & \vee & & & \overline{X}_3 & & \quad ?
\end{array}$$

In order to solve this problem we may minimize X_2 subject to

$$\begin{array}{lllll}
+X_1 & +X_2 & +X_3 & & \geq 1, \\
-X_1 & +X_2 & & -X_4 & \geq 1-2, \\
-X_1 & & +X_3 & & \geq 1-1, \\
-X_1 & & -X_3 & +X_4 & \geq 1-2, \\
+X_1 & & -X_3 & & \geq 1-1
\end{array}$$

The logical implication holds if and only if the minimum of X_2 is equal to 1.

There are many remarkable parallels between theorem proving and mathematical programming. The logical concepts of resolution, extended resolution, input and unit refutation, the Davis-Putnam-Procedure, and drawing of inferences pertinent to a given topic are closely related to the mathematical concepts of cutting planes, Chvátal's method, elementary closure, branch and bound, and projection of a polytope, respectively (see [Hoo88] for a survey).

More generally, pseudo-Boolean constraints can be seen as a particular way of combining two of the most important domains in constraint logic programming: arithmetic and Boolean algebra.

So far we considered only linear pseudo-Boolean functions. In many applications however one has to deal with non-linear functions and non-linear constraints. For example, suppose that we want to model the interaction of n objects $ob_1, ob_2, \ldots, ob_n$, each of which can be chosen ($X_i = 1$) or not ($X_i = 0$). For any pair of objects (ob_i, ob_j) let a_{ij} measure the interaction between ob_i and ob_j. If the global interaction is given by the sum of the interactions between all pairs of chosen objects, then it can be written as a quadratic pseudo-Boolean function $\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} X_i X_j$ of the n variables $X_1, \ldots, X_n$. Maximizing or minimizing the global interaction means maximizing or minimizing this pseudo-Boolean function.

Typical applications of non-linear pseudo-Boolean functions include sequencing problems, time-table scheduling, coding theory, plant location [HR68], inter-city traffic [Rhy70], kinetic energy in spin-glass models [KGV83], or supply support of space stations [FGGB66].

The organization of the paper is as follows: We start in Section 2 with some basic facts about Boolean and pseudo-Boolean functions. In Section 3 we define a new constraint logic programming language *CLP(PB)* for logic progamming with pseudo-Boolean constraints which is an instance of the general constraint logic programming language scheme *CLP(X)*. In Section 4 we present some of the ideas that can be used in order to solve pseudo-Boolean constraints. We show that any system of pseudo-Boolean constraints has a most general solution and give a variable elimination algorithm for computing most general pseudo-Boolean unifiers. In Section 5 we consider the problem of optimizing pseudo-Boolean functions. We present a variable elimination algorithm for pseudo-Boolean optimization and discuss its relationship to the well-known Boolean unification algorithm of Büttner and Simonis [BS87]. In Section 6 we give a typical example for the application of our concepts. Finally, Section 7 contains the conclusion and a discussion of further work.

2 Boolean and Pseudo-Boolean Functions

We begin with some basic facts about Boolean and pseudo-Boolean functions. At some places we will use the terminology of equational logic (see [HO80] for additional information).

Definition 1 A *Boolean function* is a mapping $f : \{0,1\}^n \to \{0,1\}$.

A *pseudo-Boolean function* is a mapping $g : \{0,1\}^n \to \mathcal{Z}$ where $\mathcal{Z}$ denotes the ring of integer numbers.

If $g : \{0,1\}^n \to \mathcal{Z}$ is a pseudo-Boolean function, then the Boolean function $g^{\neq} : \{0,1\}^n \to \{0,1\}$ defined by

$$g^{\neq}(X_1, \ldots, X_n) = \begin{cases} 1 & \text{if } g(X_1, \ldots, X_n) \neq 0 \\ 0 & \text{if } g(X_1, \ldots, X_n) = 0 \end{cases}$$

is called the *reduct* of g.

Next we introduce a signature that allows to express Boolean and pseudo-Boolean functions as terms.

Definition 2 Consider the order-sorted signature with two sort symbols *bool* $\subseteq$ *integer* and the sets of function symbols

$$\begin{aligned}
F_{bool} &= \{0,1 :\to bool, \bar{\cdot} : bool \to bool \\
&\quad \oplus, \wedge, \vee : bool \times bool \to bool\} \\
F_{psbool} &= \{0,1 :\to integer, \\
&\quad +, * : integer \times integer \to integer, \\
&\quad - : integer \to integer\}
\end{aligned}$$

Let

$$V_{bool} = \{X, Y, Z, X_1, X_2, \ldots\}$$

$$V_{integer} = \{x, y, z, x_1, x_2, \ldots\}$$

denote countably infinite sets of variables of sort *bool* and *integer*. The terms in the term algebra $\mathcal{T}(F_{bool}, V_{bool})$ are called *Boolean terms* and those in $\mathcal{T}(F_{psbool}, V_{bool})$ *pseudo-Boolean terms.*

By interpreting $\bar{\cdot}$ as negation, $\oplus$ as sum modulo 2, $\wedge$ as conjunction, and $\vee$ as disjunction, any Boolean term $t \in \mathcal{T}(F_{bool}, V_{bool})$ containing n variables *generates* a n-ary Boolean function $\widehat{t} : \{0,1\}^n \to \{0,1\}$. Similarly, by interpreting the function symbols $+, *, -$ as the usual arithmetical operations on integer numbers any pseudo-Boolean term $t \in \mathcal{T}(F_{psbool}, V_{bool})$ containing n variables *generates* a n-ary pseudo-Boolean function $\widehat{t} : \{0,1\}^n \to \mathcal{Z}$.

For example, the Boolean term $X \oplus Y \oplus X \wedge Y$ generates the Boolean function $f : \{0,1\}^2 \to \{0,1\}, f(0,0) = 0, f(0,1) = f(1,0) = f(1,1) = 1$. The pseudo-Boolean term $X+Y+X*Y$ generates the pseudo-Boolean function $g : \{0,1\}^2 \to \mathcal{Z}, g(0,0) = 0, g(0,1) = g(1,0) = 1, g(1,1) = 3$.

Conversely, any Boolean or pseudo-Boolean function has a canonical term representation, the *polynomial normal form*, which is introduced in the next proposition.

Proposition 3 *1. For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ there is up to associativity and commutativity of $\oplus, \wedge$ a unique Boolean term of the form*

$$\bigoplus_{I \in \Omega} \bigwedge_{i \in I} X_i, \tag{1}$$

with a collection Ω of subsets of $\{1, \ldots, n\}$, that generates f.

*2. For any pseudo-Boolean function $g : \{0,1\}^n \to \mathcal{Z}$ there is up to associativity and commutativity of +, * a unique pseudo-Boolean term of the form*

$$\sum_{I \in \Omega} c_I \prod_{i \in I} X_i, \tag{2}$$

with a collection Ω of subsets of $\{1, \ldots, n\}$ and coefficients $c_I \in \mathcal{Z} \setminus \{0\}$, that generates g.

Proof: See [HR68] □

Example 4 The function $g : \{0,1\}^3 \to \mathcal{Z}$ defined by the table

X	Y	Z	g(X,Y,Z)
0	0	0	-5
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	-3
1	0	1	8
1	1	0	0
1	1	1	6

has the polynomial normal form

$$g(X,Y,Z) = -5 + 2X + 5Y + 5Z - 2XY + 6XZ - 5YZ$$

Note that there exist also other representations, for example

$$g(X,Y,Z) = 2X\overline{Y} + 6XZ - 5\overline{Y}\,\overline{Z},$$

where $\overline{X}$ is an abbreviation for $1 - X$.

In order to obtain a syntactic characterization of those terms that generate the same function we need equational theories for Boolean and pseudo-Boolean terms. An equational theory is a congruence relation $\equiv_E$ on the term algebra which is generated by a set of equations E. For two terms s, t we have $s \equiv_E t$ if and only if s and t can be proven equal using the equations in E [HO80].

Definition 5 1. The *equational theory* $\equiv_B$ *of Boolean terms* is generated by the set of equations

$$\begin{array}{rcl} B = \{X \oplus (Y \oplus Z) & \doteq & (X \oplus Y) \oplus Z, \\ X \oplus 0 & \doteq & X, \\ X \oplus X & \doteq & 0, \\ X \oplus Y & \doteq & Y \oplus X, \\ X \wedge (Y \wedge Z) & \doteq & (X \wedge Y) \wedge Z, \\ X \wedge 1 & \doteq & X, \\ X \wedge Y & \doteq & Y \wedge X, \\ X \wedge (Y \oplus Z) & \doteq & (X \wedge Y) \oplus (X \wedge Z), \\ X \wedge X & \doteq & X, \\ \overline{X} & \doteq & X \oplus 1, \\ X \vee Y & \doteq & (X \oplus Y) \oplus X \wedge Y\} \end{array}$$

2. The *equational theory* $\equiv_{PB}$ *of pseudo-Boolean terms* is generated by the set of equations

$$\begin{array}{rcl} PB = \{x + (y + z) & \doteq & (x + y) + z, \\ x + 0 & \doteq & x, \end{array}$$

$$\begin{aligned}
x + (-x) &\doteq 0, \\
x + y &\doteq y + x, \\
x * (y * z) &\doteq (x * y) * z, \\
x * 1 &\doteq x, \\
x * y &\doteq y * x, \\
x * (y + z) &\doteq (x * y) + (x * z), \\
X * X &\doteq X\}
\end{aligned}$$

The next proposition shows that two terms denote the same function iff they are congruent modulo the equational theory $\equiv_{PB}$ or $\equiv_B$ respectively.

Proposition 6 *1. Two Boolean terms $s, t \in \mathcal{T}(F_{bool}, V_{bool})$ generate the same Boolean function iff they are congruent modulo the theory B, that is $\widehat{s} = \widehat{t}$ iff $s \equiv_B t$.*

2. Two pseudo-Boolean terms $s, t \in \mathcal{T}(F_{psbool}, V_{bool})$ generate the same pseudo-Boolean function iff they are congruent modulo the theory PB, that is $\widehat{s} = \widehat{t}$ iff $s \equiv_{PB} t$.

Proof: We prove only the second part of the theorem. The first part is completely analogous.

"$\Longleftarrow$": If $s \equiv_{PB} t$, then by the well-known theorem of Birkhoff [HO80] the equation $s \doteq t$ holds in all models of the set of equations PB, in particular in the algebra $(\{0,1\}, \mathcal{Z})$. This means that for any variable assignment $\sigma : V_{bool} \to \{0,1\}$ the terms s and t have the same value. This is equivalent to say that the pseudo-Boolean functions generated by s and t are the same.

"$\Longrightarrow$": Using the equations in PB we may transform the pseudo-Boolean terms s, t to pseudo-Boolean terms $s{\downarrow} \equiv_{PB} s$ and $t{\downarrow} \equiv_{PB} t$ such that $s{\downarrow}$ and $t{\downarrow}$ are of the form (2). From "$\Longleftarrow$" we deduce that $\widehat{s} = \widehat{s{\downarrow}}$ and $\widehat{t} = \widehat{t{\downarrow}}$. From the preceding proposition and the assumption $\widehat{s} = \widehat{t}$ we know that $s{\downarrow}$ and $t{\downarrow}$ are the same up to associativity and commutativity of $+, *$. This implies $s{\downarrow} \equiv_{PB} t{\downarrow}$ and consequently $s \equiv_{PB} t$. □

Boolean functions can be described by both Boolean and pseudo-Boolean terms. A canonical transformation between the two representations is given in the next definition.

Definition 7 The *pseudo-Boolean form* $t^P \in \mathcal{T}(F_{psbool}, V_{bool})$ of a term $t \in \mathcal{T}(F_{bool} \cup F_{psbool}, V_{bool})$ is obtained by normalizing t using the canonical term rewrite system

$$\begin{aligned}
P \;=\; \{ & X \oplus Y \to X + Y - (X * Y + X * Y), \\
& X \wedge Y \to X * Y, \\
& \overline{X} \to 1 - X, \\
& X \vee Y \to X + Y - X * Y\}
\end{aligned}$$

Lemma 8 *For any Boolean term $t \in \mathcal{T}(F_{bool}, V_{bool})$ the functions generated by t and t^P agree, that is $\widehat{t} = \widehat{t^P}$.*

Proof: The equations $X \oplus Y \doteq X + Y - (X * Y + X * Y), X \wedge Y \doteq X * Y, \overline{X} \doteq 1 - X, X \vee Y \doteq X + Y - X * Y$ hold in the algebra $(\{0,1\}, \mathcal{Z})$. This implies $(\{0,1\}, \mathcal{Z}) \models t \doteq t^P$ and we get $\widehat{t} = \widehat{t^P}$. □

Lemma 9 *Let $s, t \in \mathcal{T}(F_{bool}, V_{bool})$ be two Boolean terms. Then $s \equiv_B t$ iff $s^P \equiv_{PB} t^P$.*

Proof: By proposition 6 we have $s \equiv_B t$ iff $\widehat{s} = \widehat{t}$. By the preceding lemma $\widehat{s} = \widehat{s^P}$ and $\widehat{t} = \widehat{t^P}$. Again by proposition 6, $s^P \equiv_{PB} t^P$ iff $\widehat{s^P} = \widehat{t^P}$. Altogether this yields $s \equiv_B t$ iff $\widehat{s} = \widehat{t}$ iff $\widehat{s^P} = \widehat{t^P}$ iff $s^P \equiv_{PB} t^P$. □

We close this section by an easy lemma that we will need several times in the sequel.

Lemma 10 *Suppose $f, g : \{0,1\}^n \to \mathcal{Z}$ are pseudo-Boolean functions and $u : \{0,1\}^n \to \{0,1\}$ is a Boolean function. Then $f * u + g * \overline{u} = 0$ implies $f * u = g * \overline{u} = 0$ and $f * g = 0$.*

Proof: Assume $f * u + g * \overline{u} = 0$. Multiplying this equation with u yields $0 = f * u * u + g * \overline{u} * u = f * u$. In the same way, multiplication with $\overline{u}$ yields $0 = f * u * \overline{u} + g * \overline{u} * \overline{u} = g * \overline{u}$. Finally, $f * g = f * g * (u + \overline{u}) = f * g * u + f * g * \overline{u} = 0 + 0 = 0$. □

3 Logic Programming with Pseudo-Boolean Constraints

We now introduce a constraint logic programming language *CLP(PB)* over an algebraic structure PB that allows us to handle equations and inequalities between pseudo-Boolean functions. The language is an instance of the constraint logic programming language scheme *CLP(X)* of [JL86, JL87].

The algebraic structure in question is the order-sorted algebra PB over the signature $\Sigma = (S, F, P)$ with

$$S = \{bool \subseteq integer \subseteq tree\}$$

$$F = F_{bool} \cup F_{psbool} \cup \{f_i : tree^{n_i} \to tree \mid i = 1, \ldots, k, k \geq 0, n_i \geq 0\}$$

$$P = \{\doteq : tree \times tree \to tree, \leq, <, \geq, > : integer \times integer \to integer\}$$

where the interpretation of the sort, function and predicate symbols is the usual one.

$V = V_{bool} \cup V_{integer} \cup V_{tree}$ denotes a collection of countably infinite sets of variables of the different sorts.

An *atomic pseudo-Boolean constraint* is of the form

$$s \doteq t, u \leq v, u < v, u \geq v, \text{ or } u > v$$

with terms $s, t \in \mathcal{T}(F, V)$ and pseudo-Boolean terms $u, v \in \mathcal{T}(F_{psbool}, V_{bool})$. A *pseudo-Boolean constraint* is a possibly empty finite set of atomic constraints.

An *atom* is of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol distinct from $\doteq, \leq, <, \geq, >$ and $t_i \in \mathcal{T}(F, V), i = 1, \ldots, n, n \geq 0$, are terms.

A *CLP(PB) constraint logic program* is defined over a signature $\Sigma^\star = (S, F, P^\star)$ where $P^\star \cap P = \emptyset$. It consists of a finite set of *constraint rules* each being of the form

$$H \text{ :- } c \parallel B_1, \ldots, B_n$$

where c is a possibly empty pseudo-Boolean constraint and H, B_i, $i = 1, \ldots, n$, $n \geq 0$, are atoms over $\Sigma^\star$.

A *CLP(PB) goal* is of the form

$$? - c \parallel B_1, \ldots, B_n$$

where again c is a possibly empty pseudo-Boolean constraint and B_i, $i = 1, \ldots, n$, $n \geq 0$, are atoms over $\Sigma^\star$.

The structure PB is solution compact because it has no limit elements. From the general results of [JL86, JL87] we can derive for the language *CLP(PB)*:

- the existence of a canonical domain of computation
- the existence of a least and greatest model semantics
- the existence of a least and greatest fixpoint semantics
- soundness and completeness results for successful derivations of the underlying implementation model
- soundness and completeness results for finitely failed derivations of the underlying implementation model
- soundness and completeness results for negation as failure

4 Solving Pseudo-Boolean Constraints

In this section we describe how we may solve arbitrary pseudo-Boolean constraints. A detailed account of this subject is beyond the scope of this paper. We will restrict ourselves mainly to the case of equality constraints. In addition, we show how arbitrary pseudo-Boolean constraints can be reduced to pseudo-Boolean equations. However, such a reduction is mainly of theoretical interest. In practice, it will be more efficient to handle non-equality constraints by a specific solving procedure.

4.1 A Variable Elimination Algorithm for Pseudo-Boolean Unification

Definition 11 A *pseudo-Boolean unification problem* is a system of equations

$$S : s_1 \doteq t_1, \ldots, s_n \doteq t_n$$

with pseudo-Boolean terms $s_i, t_i \in \mathcal{T}(F_{psbool}, V_{bool}), i = 1, \ldots, n$. A *pseudo-Boolean unifier of S* is a substitution $\tau : V_{bool} \to \mathcal{T}(F_{bool}, V_{bool})$ such that

$$\tau(s_i)^P \equiv_{PB} \tau(t_i)^P, \text{ for all } i = 1, \ldots, n.$$

A *most general pseudo-Boolean unifier of S* is a substitution $\sigma : V_{bool} \to \mathcal{T}(F_{bool}, V_{bool})$ such that for all pseudo-Boolean unifiers τ of S there is a substitution $\lambda : V_{bool} \to \mathcal{T}(F_{bool}, V_{bool})$ with

$$\lambda(\sigma(X)) \equiv_B \tau(X), \text{ for all } X \in V_{bool}.$$

Now we want to give a constructive proof that, similar to syntactic unification in classical logic programming, any two unifiable pseudo-Boolean terms have a most general pseudo-Boolean unifier. In the terminology of unification theory this means that pseudo-Boolean terms form a unitary theory. The next theorem is a generalization of the well-known Boolean unification method of [BS87] to the pseudo-Boolean case. For the proof we use previous results by P. Hammer [Ham64a].

Theorem 12 *Pseudo-Boolean unification is unitary, that is any two pseudo-Boolean terms are either not unifiable modulo PB or they have a most general pseudo-Boolean unifier.*

Proof: It is sufficient to consider the pseudo-Boolean unification problem $u \doteq 0$, with a pseudo-Boolean term $u \in \mathcal{T}(F_{psbool}, V_{bool})$. Let $Var(u) = \{X_1, \ldots, X_{n_u}\}$ be the set of variables occurring in u. The proof uses induction on n_u and provides a decision procedure for the unifiability of u and 0 as well as an algorithm for computing a most general unifier.

To simplify our notation we will identify throughout the proof a Boolean or pseudo-Boolean term t with the corresponding function $\widehat{t}$. In particular, we will write $s = t$ instead of $\widehat{s} = \widehat{t}$. This is justified by the considerations in Section 2.

First let $n_u = 0$. If $u = 0$ then the empty substitution is a most general pseudo-Boolean unifier. and if $u \neq 0$ then the equation is unsolvable.

Suppose now that the theorem is true for pseudo-Boolean terms which contain at most n variables, $n \geq 0$, and let u be a pseudo-Boolean term with $n+1$ variables. Select a variable X_{n+1} in u and write u in the form

$$u = v * X_{n+1} + w * \overline{X_{n+1}}$$

where $\{X_1, \ldots, X_n\}, n \geq 0$, are the only variables occurring in the pseudo-Boolean terms v, w.

Let $\tau : V_{bool} \to \mathcal{T}(F_{bool}, V_{bool})$ be a pseudo-Boolean unifier of u and 0. Then we have

$$\tau(v) * \tau(X_{n+1}) + \tau(w) * \overline{\tau(X_{n+1})} = 0.$$

From Lemma 10 we deduce that $\tau(v) * \tau(w) = \tau(v * w) = 0$ or that τ is a pseudo-Boolean unifier of $v * w$ and 0.

Conversely, if $\rho : V_{bool} \to \mathcal{T}(F_{bool}, V_{bool})$ is a unifier of $v * w$ and 0, then

$$\begin{aligned} \sigma(X_i) &\overset{\text{def}}{=} \rho(X_i), \text{ for } i = 1, \ldots, n \\ \sigma(X_{n+1}) &\overset{\text{def}}{=} \overline{\rho(v^{\neq})} \wedge Y \oplus \rho(w^{\neq}) \wedge \overline{Y} = \overline{\rho(v^{\neq})} * Y + \rho(w^{\neq}) * \overline{Y}, \end{aligned}$$

with a new variable $Y \in V_{bool}$, defines a pseudo-Boolean unifier of u and 0. In fact, we have

$$\begin{aligned} \sigma(u) &= \sigma(v) * \sigma(X_{n+1}) + \sigma(w) * \overline{\sigma(X_{n+1})} \\ &= \rho(v) * (\overline{\rho(v^{\neq})} * Y + \rho(w^{\neq}) * \overline{Y}) + \rho(w) * \overline{(\overline{\rho(v^{\neq})} * Y + \rho(w^{\neq}) * \overline{Y})} \\ &= \rho(v) * \overline{\rho(v^{\neq})} * Y + \rho(v) * \rho(w^{\neq}) * \overline{Y} + \rho(w) * (1 - \overline{\rho(v^{\neq})} * Y - \rho(w^{\neq}) * \overline{Y}) \\ &= 0 + 0 + \rho(w) - \rho(w) * \overline{\rho(v^{\neq})} * Y - \rho(w) * \rho(w^{\neq}) * \overline{Y} \\ &= \rho(w) - \rho(w) * (1 - \rho(v^{\neq})) * Y - \rho(w) * (1 - Y) \\ &= \rho(w) - \rho(w) * Y + \rho(w) * \rho(v^{\neq}) * Y - \rho(w) + \rho(w) * Y = 0 \end{aligned}$$

Here, we have used the identities $\rho(v) * \overline{\rho(v^{\neq})} = 0$, $0 = \rho(v) * \rho(w) = \rho(v^{\neq}) * \rho(w) = \rho(v) * \rho(w^{\neq}) = \rho(v^{\neq}) * \rho(w^{\neq})$, and $\rho(w) * \rho(w^{\neq}) = \rho(w)$.

Solved	$\dfrac{\{u \doteq 0\}}{true}$
	if $u \equiv_{PB} 0$
Unsolvable	$\dfrac{\{u \doteq 0\}}{fail}$
	if $u \not\equiv_{PB} 0$ and $Var(u) = \emptyset$
Variable Elimination	$\dfrac{\{u \doteq 0\}}{\{v * w \doteq 0\} \cup \{X \doteq \overline{v^{\neq}} * Y + w^{\neq} * \overline{Y}\}}$
	if $u \not\equiv_{PB} 0$, $u \equiv_{PB} v * X + w * \overline{X}$, $X \notin Var(v, w)$, and Y is a new variable

Figure 1: Variable Elimination for Pseudo-Boolean Unification

It remains to show that if ρ is a most general pseudo-Boolean unifier of $v * w \doteq 0$, then σ is a most general pseudo-Boolean unifier of $u \doteq 0$. Let τ be an arbitrary unifier of u and 0. The restriction τ' of τ to the variables $\{X_1, \ldots, X_n\}$ unifies $v * w$ and 0. Consequently, there exists a substitution λ' such that

$$\tau'(X_i) = \lambda'(\rho(X_i)), \text{ for all } i = 1, \ldots, n.$$

We define a substitution λ by

$$\begin{aligned} \lambda(X) &\overset{\text{def}}{=} \lambda'(X), \text{ for } X \in \bigcup_{i=1,\ldots,n} Var(\rho(X_i)) \\ \lambda(Y) &\overset{\text{def}}{=} \tau(X_{n+1}). \end{aligned}$$

Now we get

$$\begin{aligned} \lambda(\sigma(X_{n+1})) &= \overline{\lambda(\rho(v^{\neq}))} * \lambda(Y) + \lambda(\rho(w^{\neq})) * \overline{\lambda(Y)} \\ &= \overline{\lambda'(\rho(v^{\neq}))} * \tau(X_{n+1}) + \lambda'(\rho(w^{\neq})) * \overline{\tau(X_{n+1})} \\ &= \overline{\tau(v^{\neq})} * \tau(X_{n+1}) + \tau(w^{\neq}) * \overline{\tau(X_{n+1})} \\ &= (1 - \tau(v^{\neq})) * \tau(X_{n+1}) + \tau(w^{\neq}) * \overline{\tau(X_{n+1})} \\ &= \tau(X_{n+1}) - \tau(v^{\neq}) * \tau(X_{n+1}) + \tau(w^{\neq}) * \overline{\tau(X_{n+1})} \\ &= \tau(X_{n+1}) - 0 + 0 = \tau(X_{n+1}) \end{aligned}$$

The last equality holds, because τ is a unifier of u and 0. This means $\tau(v) * \tau(X_{n+1}) + \tau(w) * \overline{\tau(X_{n+1})} = 0$ which by Lemma 10 implies $\tau(v^{\neq}) * \tau(X_{n+1}) = \tau(w^{\neq}) * \overline{\tau(X_{n+1})} = 0$. Altogether we have shown that σ is a most general pseudo-Boolean unifier of u and 0. □

From the proof of the preceding theorem we derive a variable elimination algorithm for pseudo-Boolean unification which subsumes the Boolean unification algorithm [BS87]. Given a pseudo-Boolean term u, it computes a most general pseudo-Boolean unifier of u and 0 if it exists and fails otherwise (see Figure 1).

4.2 Solving Arbitrary Pseudo-Boolean Constraints

We show in this section how arbitrary pseudo-Boolean constraints can be reduced to equality constraints.

Lemma 13 *The problem of solving a pseudo-Boolean inequality is reducible to that of solving a pseudo-Boolean equation.*

Proof: It is enough to consider a pseudo-Boolean inequality of the form

$$g(X_1, \ldots, X_n) \leq 0. \tag{3}$$

The following proof was first given in [Ham64b]. Let $g(X_1, \ldots, X_n) = \sum_{I \in \Omega} c_I \prod_{i \in I} X_i$ be a polynomial representation of g and let $A \overset{\text{def}}{=} \sum_{c_I < 0} c_I$. Then A is a lower bound of the pseudo-Boolean function g. Let $-A = a_0 2^0 + \cdots + a_k 2^k, k \geq 0$, be the binary representation of $-A \geq 0$. The inequality (3) is equivalent to the equation

$$g(X_1, \ldots, X_n) + y \doteq 0 \tag{4}$$

with the additional restriction $y \geq 0$. Therefore we may write $y = Y_0 2^0 + \cdots + Y_l 2^l, l \geq 0$, and since $y = -g(X_1, \ldots, X_n) \leq -A$ we get $l \leq k$. It follows that (3) and (4) are equivalent to the pseudo-Boolean equation

$$g(X_1, \ldots, X_n) + Y_0 2^0 + \cdots + Y_k 2^k \doteq 0 \tag{5}$$

where in the case $k > l$ we have $Y_{l+1} = \ldots = Y_k = 0$. □

Lemma 14 *The problem of solving a system of pseudo-Boolean equations is reducible to that of solving a single pseudo-Boolean equation.*

Proof: The system $s_1 \doteq t_1, \ldots, s_n \doteq t_n, n \geq 1$, is equivalent to the equation $\sum_{i=1}^{n} (s_i - t_i)^2 \doteq 0$. □

The two lemmas together yield

Proposition 15 *The problem of solving an arbitrary pseudo-Boolean constraint is reducible to that of solving a single pseudo-Boolean equation.*

5 Pseudo-Boolean Optimization

In the last section we have sketched a solution procedure for pseudo-Boolean constraints. In practical applications, one is often not interested in all solutions of a pseudo-Boolean constraint but only in those solutions which are optimal with respect to some objective function, which itself is a pseudo-Boolean function.

This leads to the problem of *pseudo-Boolean optimization* also known as *nonlinear 0-1 programming.*

> Maximize the pseudo-Boolean function $g(X_1, \ldots, X_n)$ subject to the pseudo-Boolean constraint $f(X_1, \ldots, X_n) \doteq 0$.

A constrained pseudo-Boolean optimization problem can be transformed to an unconstrained pseudo-Boolean optimization problem by first computing a most general solution

$$X_1 = t_1(Y_1, \ldots, Y_m), \ldots, X_n = t_n(Y_1, \ldots, Y_m)$$

of the constraint $f(X_1, \ldots, X_n) \doteq 0$ and then maximizing the pseudo-Boolean function

$$h(Y_1, \ldots, Y_m) \overset{\text{def}}{=} g(t_1(Y_1, \ldots, Y_m), \ldots, t_n(Y_1, \ldots, Y_m)).$$

The unconstrained quadratic pseudo-Boolean maximization problem $z \doteq max(g)$ with a quadratic pseudo-Boolean function $g : \{0,1\}^n \to \mathcal{Z}$ is NP-complete. As a matter of fact, most of the well-known NP-complete problems (for example the minimum cut problem, balancing a signed graph, maximum 2-satisfiability) can be very naturally formulated as unconstrained quadratic pseudo-Boolean optimization problems. The same is true for broad classes of constrained pseudo-Boolean optimization problems, such as the maximization of a linear function subject to a quadratic Boolean equation or the maximization of a quadratic function subject to a system of linear equations [HS86].

5.1 The Basic Algorithm for Pseudo-Boolean Optimization

There is an extensive literature on the solution of pseudo-Boolean optimization problems (see the survey [HS86]) that we cannot cover in this paper. Because of its close relationship to the variable elimination method for Boolean unification we present here a variant of the so-called Basic Algorithm for pseudo-Boolean optimization. This algorithm was originally proposed in [HRR63a, HRR63b] and allows to maximize an arbitrary nonlinear pseudo-Boolean function $f : \{0,1\}^n \to \mathcal{Z}$.

First we define the notion of a most-general maximizer of a pseudo-Boolean function f.

Definition 16 Let $f : \{0,1\}^n \to \mathcal{Z}$ be a pseudo-Boolean function with the maximum value $z \in \mathcal{Z}$. Let $t_f \in T(F_{psbool}, V_{bool})$ be a pseudo-Boolean term representing f. A *(most general) maximizer* of f (resp. t_f) is a (most general) pseudo-Boolean unifier of the pseudo-Boolean equation $t \doteq z$.

From Theorem 12 we can conclude that any pseudo-Boolean function $f : \{0,1\}^n \to \mathcal{Z}$ has a most general maximizer σ_f which subsumes any other maximizer τ of f. However, we cannot compute σ_f by the variable elimination algorithm for pseudo-Boolean unification, because we do not know a priori the maximum value of f. The Basic Algorithm for pseudo-Boolean optimization is a variable elimination method which computes simultaneously the maximum value z of f and a most general pseudo-Boolean unifier of the equation $f \doteq z$.

In order to formulate the algorithm we need the following definitions.

Definition 17 Let $g : \{0,1\}^n \to \mathcal{Z}$ be a pseudo-Boolean function. The pseudo-Boolean function $g^+ : \{0,1\}^n \to \mathcal{Z}$ is defined by

$$g^+(X_1, \ldots, X_n) \overset{\text{def}}{=} \begin{cases} g(X_1, \ldots, X_n), & \text{if } g(X_1, \ldots, X_n) > 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \leq 0. \end{cases}$$

Constant
$$\frac{z \doteq max(u)}{z \doteq c}$$

if $u \equiv_{PB} c = constant$

Variable Elimination
$$\frac{z \doteq max(u)}{\{z \doteq max(v^{+} + w)\} \cup \{X \doteq v^{=} * X' + v^{>}\}}$$

if $u \equiv_{PB} v * X + w$,
$X \notin Var(v, w)$,
$v \not\equiv_{PB} 0$,
and X' is a new variable

Figure 2: The Basic Algorithm for Pseudo-Boolean Optimization

Definition 18 Let $g : \{0,1\}^n \to \mathcal{Z}$ be a pseudo-Boolean function. The Boolean functions $g^{=}, g^{>} : \{0,1\}^n \to \{0,1\}$ are defined by

$$g^{=}(X_1, \ldots, X_n) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } g(X_1, \ldots, X_n) = 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \neq 0 \end{cases}$$

and

$$g^{>}(X_1, \ldots, X_n) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } g(X_1, \ldots, X_n) > 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \leq 0 \end{cases}$$

Now we can describe the Basic Algorithm by a set of inference rules (see Figure 2).

The correctness proof of the Basic Algorithm depends on the following two lemmas that relate a most general maximizer of a pseudo-Boolean function $f = g * X + h$ to a most general maximizer of $g^{+} + h$.

Lemma 19 *Let $f : \{0,1\}^{n+1} \to \mathcal{Z}$ be a pseudo-Boolean function.*

Suppose

$$f(X_1, \ldots, X_{n+1}) = X_{n+1} * g(X_1, \ldots, X_n) + h(X_1, \ldots, X_n)$$

with pseudo-Boolean functions $g, h : \{0,1\}^n \to \mathcal{Z}$. Then

$$max(f) = max(g^{+} + h).$$

If $\tau : \{X_1, \ldots, X_{n+1}\} \to T(F_{bool}, V_{bool})$ is a maximizer of f then the restriction of τ to $\{X_1, \ldots, X_n\}$ is a maximizer of $g^{+} + h$.

Moreover, τ satisfies the condition $\tau(X_{n+1}) = \tau(g^{=}) * \tau(X_{n+1}) + \tau(g^{>})$.

Proof: As before, we will not distinguish a pseudo-Boolean function f and the corresponding term t_f.

From $f = X_{n+1} * g + h \leq X_{n+1} * g^{+} + h \leq g^{+} + h$ we get $max(f) \leq max(g^{+} + h)$. Let $z \in \mathcal{Z}$ denote the maximum value of $g^{+} + h$. Then there exist $a_1, \ldots, a_n \in \{0,1\}$ such that $(g^{+} + h)(a_1, \ldots, a_n) =$

z. Let $a_{n+1} \overset{\text{def}}{=} g^{>}(a_1, \ldots, a_n)$. Then $f(a_1, \ldots, a_{n+1}) = a_{n+1} * g(a_1, \ldots, a_n) + h(a_1, \ldots, a_n) = g^{+}(a_1, \ldots, a_n) + h(a_1, \ldots, a_n) = z$. This implies $max(f) \geq max(g^{+} + h)$ and altogether we get $max(f) = max(g^{+} + h) = z$.

Now let $\tau : \{X_1, \ldots, X_{n+1}\} \to T(F_{bool}, V_{bool})$ be a maximizer of f.

First we show that $\tau(X_{n+1}) = \tau(g^{=}) * \tau(X_{n+1}) + \tau(g^{>})$. Otherwise there would exist a ground substitution $\gamma : V_{bool} \to \{0, 1\}$ such that $(\gamma \circ \tau)(X_{n+1}) \neq (\gamma \circ \tau)(g^{=}) * (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(g^{>})$. Since

$$(\gamma \circ \tau)(g^{=}) * (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(g^{>}) = \begin{cases} 1, & \text{if } (\gamma \circ \tau)(g) > 0 \\ (\gamma \circ \tau)(X_{n+1}), & \text{if } (\gamma \circ \tau)(g) = 0 \\ 0, & \text{if } (\gamma \circ \tau)(g) < 0 \end{cases},$$

we get a contradiction in the case $(\gamma \circ \tau)(g) = 0$. In the two remaining cases we can derive

$$(\gamma \circ \tau)(X_{n+1}) = \begin{cases} 0, & \text{if } (\gamma \circ \tau)(g) > 0 \\ 1, & \text{if } (\gamma \circ \tau)(g) < 0 \end{cases}.$$

This is an obvious contradiction to the maximality of τ as we show now.

Define the substitution $\overline{\tau} : \{X_1, \ldots, X_{n+1}\} \to T(F_{bool}, V_{bool})$ by

$$\begin{aligned} \overline{\tau}(X_i) &\overset{\text{def}}{=} (\gamma \circ \tau)(X_i), \text{ for } i = 1, \ldots, n \\ \overline{\tau}(X_{n+1}) &\overset{\text{def}}{=} \begin{cases} 1, & \text{if } (\gamma \circ \tau)(g) > 0 \\ 0, & \text{if } (\gamma \circ \tau)(g) < 0 \end{cases}. \end{aligned}$$

Then $\overline{\tau}(f) = \overline{\tau}(g * X_{n+1} + h) = \overline{\tau}(g) * \overline{\tau}(X_{n+1}) + \overline{\tau}(h) = (\gamma \circ \tau)(g) * \overline{\tau}(X_{n+1}) + (\gamma \circ \tau)(g) > (\gamma \circ \tau)(g) * (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(h) = (\gamma \circ \tau)(f) = \gamma(z) = z$, in contradiction to the maximality of z. This allows us to conclude that $\tau(X_{n+1}) = \tau(g^{=}) * \tau(X_{n+1}) + \tau(g^{>})$.

In order to show that the restriction of τ to $\{X_1, \ldots, X_n\}$ is maximizer of $g^{+} + h$, we must prove that $\tau(g^{+} + h) = z$. Since we know that $\tau(f) = \tau(X_{n+1}) * \tau(g) + \tau(h) = z$, it is enough to show that $\tau(X_{n+1}) * \tau(g) = \tau(g^{+})$. But, $\tau(X_{n+1}) * \tau(g) = (\tau(g^{=}) * \tau(X_{n+1}) + \tau(g^{>})) * \tau(g) = \tau(g^{=}) * \tau(g) * \tau(X_{n+1}) + \tau(g^{>}) * \tau(g) = 0 + \tau(g^{+}) = \tau(g^{+})$, and the lemma is proved.

□

Lemma 20 *Let $f : \{0, 1\}^{n+1} \to \mathcal{Z}$ be a pseudo-Boolean function. Suppose*

$$f(X_1, \ldots, X_{n+1}) = X_{n+1} * g(X_1, \ldots, X_n) + h(X_1, \ldots, X_n)$$

with pseudo-Boolean functions $g, h : \{0, 1\}^n \to \mathcal{Z}$. Let $\rho : V_{bool} \to T(F_{bool}, V_{bool})$ be a most general maximizer of the pseudo-Boolean function $g^{+} + h$. Then the substitution $\sigma : V_{bool} \to T(F_{bool}, V_{bool})$,

$$\begin{aligned} \sigma(X_i) &\overset{\text{def}}{=} \rho(X_i), \textit{ for } i = 1, \ldots, n \\ \sigma(X_{n+1}) &\overset{\text{def}}{=} \rho(g^{=}) \wedge Y \oplus \rho(g^{>}) = \rho(g^{=}) * Y + \rho(g^{>}), \end{aligned}$$

with a new variable $Y \in V_{bool}$, defines a most general maximizer of f.

Proof: Again we do not distinguish a pseudo-Boolean function f and the corresponding term t_f. Let $z \in \mathcal{Z}$ be the maximum value of f. By Lemma 19, z is also the maximum value of $g^{+} + h$. This means that the substitution ρ is a most general pseudo-Boolean unifier of the equation $g^+ + h \doteq z$. In particular, we get $\rho(g^+) + \rho(h) = z$. It follows that

$$\begin{aligned}\sigma(f) &= \sigma(X_{n+1}) * \sigma(g) + \sigma(h)\\ &= (\rho(g^=) * Y + \rho(g^>)) * \rho(g) + \rho(h)\\ &= \rho(g^=) * \rho(g) * Y + \rho(g^>) * \rho(g) + \rho(h)\\ &= \rho(g^= * g) * Y + \rho(g^> * g) + \rho(h)\\ &= \rho(0) * Y + \rho(g^+) + \rho(h)\\ &= 0 + z = z.\end{aligned}$$

Here we have used the identities $g^= * g = 0$ and $g^> * g = g^+$. This shows, that σ is a maximizer of f.

Let $\tau : V_{bool} \to T(F_{bool}, V_{bool})$ be an arbitrary maximizer of f, that is $z = \tau(f) = \tau(X_{n+1}) * \tau(g) + \tau(h)$. By Lemma 19, τ is also a maximizer of $g^+ + h$. It follows that there exists a substitution $\lambda' : V_{bool} \to T(F_{bool}, V_{bool})$ such that $\lambda'(\rho(X_i)) = \tau(X_i)$, for all $i = 1, \ldots, n$.

We define a substitution λ by

$$\begin{aligned}\lambda(X) &\overset{\text{def}}{=} \lambda'(X), \text{ for } X \in \bigcup_{i=1,\ldots,n} Var(\rho(X_i))\\ \lambda(Y) &\overset{\text{def}}{=} \tau(X_{n+1}).\end{aligned}$$

Then we get $\lambda(\sigma(X_{n+1})) = \lambda(\rho(g^=) * Y + \rho(g^>)) = \lambda(\rho(g^=)) * \lambda(Y) + \lambda(\rho(g^>)) = \tau(g^=) * \tau(X_{n+1}) + \tau(g^>) = \tau(X_{n+1})$. The last equality holds because of Lemma 19.

By definition, $\lambda(\sigma(X_i)) = \lambda'(\rho(X_i)) = \tau(X_i)$, for all $i = 1, \ldots, n$. This shows that σ is even a most general maximizer of f. □

Now we are able to prove the correctness of our algorithm.

Theorem 21 *Let $f : \{0,1\}^n \to \mathcal{Z}$ be a pseudo-Boolean function represented by the pseudo-Boolean term $u \in T(F_{psbool}, V_{bool})$.*

Starting with the problem $z \doteq max(u)$ and applying the rules of the Basic Algorithm at most $n+1$ times one obtains a solved form

$$\{z \doteq c, X_1 \doteq t_n, \ldots, X_n \doteq t_n\},$$

with $c \in \mathcal{Z}, t_i \in T(F_{psbool}, V_{bool})$, such that c is the maximum value and $\{X_1 \leftarrow t_1\} \circ \ldots \circ \{X_n \leftarrow t_n\}$ is a most general maximizer of f.

Proof: Let $Var(u) = \{X_1, \ldots, X_{n_u}\}$ denote the set of variables occurring in the pseudo-Boolean term u. The proof is by induction on n_u.

First consider the case $n_u = 0$. If u contains no variables, the function f is equivalent to a constant function $c :\to \mathcal{Z}$, the rule *Constant* applies, and we get $max(f) = z = c$.

Now suppose $n_u = n+1, n \geq 0$. We may assume that u is not congruent to a constant. Then we can choose a variable X_{n+1} in u and write u in the form

$$u = v * X_{n+1} + w,$$

Constant
$$\frac{z \doteq max(u)}{z \doteq c}$$

if $u \equiv_{PB} c = constant$

Variable Elimination
$$\frac{z \doteq max(u)}{\{z \doteq max(v^{+} + w)\} \cup \{X \doteq v^{>}\}}$$

if $u \equiv_{PB} v * X + w$,
$X \notin Var(v, w)$,
$v \not\equiv_{PB} 0$

Figure 3: The Basic Algorithm for Pseudo-Boolean Optimization (Special Version)

where $\{X_1, \dots, X_n\}$ are the only variables occurring in the terms v, w and $v \neq 0$. We apply the variable elimination rule and obtain the new problem

$$\{z \doteq max(v^{+} + w)\} \cup \{X_{n+1} \doteq v^{=} * X' + v^{>}\},$$

with a new variable X'.

Applying the induction hypothesis to $z \doteq max(v^{+} + w)$ then yields in at most n steps the solved form

$$\{z \doteq c, X_1 \doteq t_n, \dots, X_n \doteq t_n\},$$

with $c \in \mathcal{Z}, t_i \in T(F_{psbool}, V_{bool})$, where c is the maximum value and $\{X_1 \leftarrow v\} \circ \dots \circ \{X_n \leftarrow t_n\}$ is a most general maximizer of $v^{+} + w$.

Using Lemma 19 and Lemma 20 we may conclude that c is also the maximum value of f and that $\{X_1 \leftarrow v\} \circ \dots \circ \{X_n \leftarrow t_n\} \circ \{X_{n+1} \doteq v^{=} * X' + v^{>}\}$ defines a most general maximizer of f.
$\square$

5.2 The Special Version of the Basic Algorithm

Sometimes one is interested only in one special maximizer of a given pseudo-Boolean function and not in the most general maximizer. In this case, one can use a special version of the Basic Algorithm (see Figure 3).

For the special version of the Basic Algorithm an interesting complexity result was recently proved in [CHJ90]. In this paper it is shown that for some special class of pseudo-Boolean functions, namely the pseudo-Boolean functions of bounded tree-width, the special version of the Basic Algorithm runs in linear time.

Definition 22 Let $f(X_1, \dots, X_n)$ be a pseudo-Boolean function in polynomial form. The *co-occurrence graph* $G(f)$ has vertex-set $V = \{X_1, \dots, X_n\}$ and an edge between X_i and $X_j, i \neq j$, iff these variables occur simultaneously in at least one monomial of f. The *tree-width* of f is the tree-width of the co-occurrence graph $G(f)$.

The definition of the tree-width of a graph is rather technical and can be found in the appendix (see also [Arn85]).

Theorem 23 ([CHJ90]) *The Basic Algorithm has linear-time complexity when applied to pseudo-Boolean functions with bounded tree width.*

More precisely: For every fixed $k \geq 0$, the Basic Algorithm can be implemented to run in O(n) time on those pseudo-Boolean functions $f(X_1, \ldots, X_n)$, expressed in polynomial form, for which $(X_1, \ldots, X_n)$ is a k-scheme of $G(f)$.

If $g : \{0,1\}^n \to \{0,1\}$ is a *Boolean* function, then the equation $g(X_1, \ldots, X_n) \doteq 1$ has a solution iff $max(g(X_1, \ldots, X_n)) = 1$. This means that Boolean unification can be regarded as a special pseudo-Boolean optimization problem. In this case, the Basic Algorithm can be used to compute a special solution of the equation $g(X_1, \ldots, X_n) \doteq 1$.

Moreover, for Boolean functions the Basic Algorithm specializes to a variant of Büttner-Simonis' Boolean unification algorithm [BS87]. Therefore we get the following complexity result for Boolean unification.

Corollary 24 *For every fixed $k \geq 0$, Büttner/Simonis' Boolean unification algorithm can be implemented to compute in O(n) time a special solution of those Boolean unification problems $f(X_1, \ldots, X_n) \doteq 1$, for which the Boolean function $f(X_1, \ldots, X_n)$ has tree-width at most k.*

Because of the close relationship between the two algorithms, experiences that have been gained in the implementation of one of them have immediate consequences for the other one. For example, [CHJ90] compare two variable elimination orderings, which are related to the co-occurrence graph of the pseudo-Boolean function. The determination of a good elimination ordering is one of the main problems in the implementation of Büttner-Simonis' algorithm. It has a great influence on the size of the most general unifiers that are computed. In this context, the orderings proposed in [CHJ90] may be very interesting. On the other hand, there exist rather sophisticated implementations of Büttner-Simonis' algorithm. It is clear that the ideas which have been developed in this context should be used also in the pseudo-Boolean case.

6 Example

In this section we present a typical application of pseudo-Boolean constraints [HR68] that is simple enough to be considered here.

The problem is to assemble a radio set under the following conditions:

- Any one of the three types T1, T2, T3 of tubes may be utilized, but only one.
- The box may be either of wood W, or of plastic material P. When using P, dimensionality requirements impose the choice of T2, and as there is no place for a transformer F, a special power supply S is needed.
- T1 needs F.
- T2 and T3 need S (and not F).

The prices of the above components are

Tubes T1	28 units
Tubes T2	30 units
Tubes T3	31 units
Transformer F	25 units
Special Power Supply S	23 units
Wood Box W	9 units
Plastic Material Box P	6 units

The other necessary components of the radio set cost

27 units,	if the tubes T1 are utilized
28 units,	if the tubes T2 are utilized
25 units,	if the tubes T3 are utilized

The price of the make is 10 units for each set in all the cases and a set is sold at 110 units when it is enclosed in a plastic material box, and at 105 units in the other case. Which model is to be constructed in order to maximize the profit? The following program solves this problem.

```
assemble(T1, T2, T3, F, S, W, P) :-
                 T1 + T2 + T3 = 1,
                 W + P = 1,
                 F + S = 1,
                 P ≤ T2 * S,       % P = 1 implies T2 = S = 1
                 T1 ≤ F,           % T1 = 1 implies F = 1
                 T2 ≤ S,           % T2 = 1 implies S = 1
                 T3 ≤ S.           % T3 = 1 implies S = 1

maximal-profit(T1, T2, T3, F, S, W, M) :-
        max( 110W + 105M - ( 28T1 + 30T2 + 31T3 + 25F + 23S
                           + 9W + 6P + 27T1 + 28T2 + 25T3 + 10),
        assemble(T1, T2, T3, F, S, W, M) ).
```

Asking the query

```
?- maximal-profit(T1, T2, T3, F, S, W, M).
```

yields the unique answer

```
T1 = 0, T2 = 0, T3 = 1, F = 0, S = 1, W = 1, M = 0.
```

7 Conclusion and Further Research

In this paper we have introduced a new constraint logic programming language *CLP(PB)* for logic programming with pseudo-Boolean constraints. The language is an instance of the constraint logic programming language scheme *CLP(X)* and has therefore the usual declarative and operational semantics of constraint logic programming languages. The main emphasis of the paper was on the solution of pseudo-Boolean constraints. We showed that any pseudo-Boolean constraint has a most general solution and gave an algorithm for pseudo-Boolean unification. Then we discussed the problem of optimizing pseudo-Boolean functions. We presented the Basic Algorithm for pseudo-Boolean optimization and discussed its relationship to the variable elimination method in Boolean unification. Among others, we obtained a new complexity result for Boolean unification.

Several research directions can be outlined to continue this work. On the practical side, the most important problem is to design an efficient constraint solver for pseudo-Boolean constraints. This should be done using the techniques that have been developed for the Boolean case. The next step is to extend this constraint solver by an algorithm for pseudo-Boolean optimization. On the theoretical side, it would be interesting to find further classes of Boolean and pseudo-Boolean functions that admit polynomial unification or optimization algorithms. Various such classes are already known, however it is not clear which role they play in practical applications.

References

[Arn85] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey. *BIT*, 25:2–23, 1985.

[ASS$^+$88] A. Aiba, K. Sakai, Y. Sato, D.J. Kawley, and R. Hasegawa. Constraint logic programming language CAL. In *Fifth Generation Computer Systems, Tokyo, 1988*. Springer-Verlag, 1988.

[BR89a] A. Bockmayr and H.H. Rath. Algorithms for boolean unification. In *Proc. FAW Workshop on Boolean Functions, Propositional Logic, and AI Systems*. FAW Ulm, Germany, 1989.

[BR89b] A. Bockmayr and H.H. Rath. Extended prolog with boolean unification. Technical Report 21/89, Fakultät für Informatik, Univ. Karlsruhe, 1989.

[BS87] W. Büttner and H. Simonis. Embedding boolean expressions in logic programming. *Journal of Symbolic Computation*, 4(2):191–205, 1987.

[CHJ90] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29:171 – 185, 1990.

[Col87] A. Colmerauer. Introduction to PROLOG III. In *4th Annual ESPRIT Conference, Bruxelles*. North Holland, 1987.

[DvHS$^+$88] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. The constraint logic programming language CHIP. In *Fifth Generation Computer Systems, Tokyo, 1988*. Springer-Verlag, 1988.

[FGGB66] R.J. Freeman, D.C. Gogerty, G.W. Graves, and R.B.S. Brooks. A mathematical model of supply support for space operations. *Oper. Research*, 14:1–15, 1966.

[Ham64a] P.L. Hammer. The method of successive eliminations for pseudo-boolean equations. *Bulletin de l'Academie Polonaise des Sciences-Serie des Sciences Math., Astr. et Phys.*, XII(11):681–683, 1964.

[Ham64b] P.L. Hammer. Systems of pseudo-boolean equations and inequalities. *Bulletin de l'Academie Polonaise des Sciences-Serie des Sciences Math., Astr. et Phys.*, XII(11):673–680, 1964.

[HO80] G Huet and D. C. Oppen. Equations and rewrite rules, A survey. In R. V. Book, editor, *Formal Language Theory.* Academic Press, 1980.

[Hoo88] J. N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45 – 69, 1988.

[HR68] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer-Verlag, 1968.

[HRR63a] P.L. Hammer, I. Rosenberg, and S. Rudeanu. Application of discrete linear programming to the minimization of boolean functions. *Rev. Math. Pures Appl.*, 8:459–475, 1963. (in Russian).

[HRR63b] P.L. Hammer, I. Rosenberg, and S. Rudeanu. On the determination of minima of pseudo-boolean functions. *Stud. Cerc. Mat.*, 14:359–364, 1963. (in Romanian).

[HS86] P.L. Hammer and B. Simeone. Quadratic functions of binary variables. In *Combinatorial Optimization*, volume 1403 of *Lecture Notes in Mathematics.* Springer-Verlag, 1986.

[JL86] J. Jaffar and J.-L. Lassez. Constraint logic programming. Technical Report 86/73, Department of Computer Science, Monash University, June 1986.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proc. 14th ACM Symp. Principles of Programming Languages*, Munich, 1987.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Rhy70] J. Rhys. A selection problem of shared fixed costs and networks. *Manag. Science*, 17:200–207, 1970.

A Tree-Width of a Graph

Let G be a simple and undirected graph. A vertex v is *simplicial* in G if the neighbors of v induce a complete subgraph of G. For $k \geq 0$, a *k-perfect elimination scheme* of the graph G is an ordering $(v_1, \ldots, v_n)$ of its vertices, such that v_j is simplicial and has degree k in the subgraph G_j of G induced by $\{v_j, v_{j+1}, \ldots, v_n\}$ for $j = 1, \ldots, n-k$. A graph is a *k-tree* if it has a k-perfect elimination ordering. A *partial k-tree* is any graph obtained by deleting edges from a k-tree. Any graph on n vertices is a partial n-tree. The *tree-width* of a graph is the smallest value of k for which the graph is a partial k-tree. For instance, trees and forests have tree-width at most 1, and series-parallel graphs have tree-width at most 2. Determining the tree-width of a graph is NP-hard. However, for every fixed k, there is a polynomial algorithm to decide if a graph has tree-width k.