MAX-PLANCK-INSTITUT FÜR INFORMATIK

A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization

Peter Barth

MPI–I–95–2-003 January 1995

INFORMATIK

Im Stadtwald
D 66123 Saarbrücken
Germany

Author's Address

Peter Barth,
Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany
barth@mpi-sb.mpg.de

Acknowledgements

The author is grateful to Alexander Bockmayr for his valuable comments and fruitful discussions. The author is also grateful to John N. Hooker for his comments on the variable selection heuristics.

This work was supported by the ESPRIT Basic Research Project ACCLAIM (contract EP 7195) and the ESPRIT Working Group CCL (contract EP 6028).

Abstract

The Davis-Putnam enumeration method (DP) has recently become one of the fastest known methods for solving the clausal satisfiability problem of propositional calculus. We present a generalization of the DP-procedure for solving the satisfiability problem of a set of linear pseudo-Boolean (or 0-1) inequalities. We extend the method to solve linear 0-1 optimization problems, i.e. optimize a linear pseudo-Boolean objective function w.r.t. a set of linear pseudo-Boolean inequalities. The algorithm compares well with traditional linear programming based methods on a variety of standard 0-1 integer programming benchmarks.

Keywords

0-1 Integer Programming; Propositional Calculus; Enumeration

Contents

1 Introduction

The Davis-Putnam enumeration method (DP) is widely used in the theorem proving community for solving the clausal satisfiability problem of propositional calculus (SAT) [DP60, Lov78]. In recent years, DP based algorithms have evolved to the fastest known methods for solving SAT [JW90, HHT94, Hoo93, Zha93]. This is due to proper implementation techniques and the development of good variable selection heuristics [JW90, HV94]. We present a generalization of the DP-procedure for solving linear pseudo-Boolean (0-1) optimization problems and so use results from theorem proving on typical operations research problems. The method compares well with traditional linear programming based methods on a variety of standard 0-1 integer programming problems found in MIPLIB [BBI92].

The generalized DP method for solving linear pseudo-Boolean optimization problems is essentially an *implicit enumeration* method based on the logical structure of pseudo-Boolean problems. Logic-based methods have been rejected in favor of linear programming based methods, exploiting the polyhedral structure of the problem, years ago. Recently there has been some renewed interest in logic-based methods. Hooker [Hoo94] relates the logic and polyhedral view of pseudo-Boolean problems. He points out that, for instance, cutting planes are a special class of logical implications and the linear programming relaxation can be replaced by a discrete relaxation. Choosing a discrete relaxation together with an appropriate logic-cut generation method [Hoo92, Hoo94, Bar94] yields a logic-based *branch-and-cut* algorithm. In branch-and-cut methods two tasks have to be done. First, a relaxation of the problem has to be solved and the problem has to be split (branch). Second, a problem has to be reformulated (strengthened) with an appropriate method if possible (cut).

In this paper we concentrate on solving a discrete relaxation of linear pseudo-Boolean problems, the basic part of any system including branching, and its application inside a pure implicit enumeration method. We consider a weak discrete relaxation, *pseudo-Boolean unit relaxation*, which can be efficiently computed, and present a branching algorithm based on this relaxation for determining the satisfiability of a set of linear pseudo-Boolean inequalities. The method then is used for optimizing a linear pseudo-Boolean function w.r.t. a set of linear pseudo-Boolean inequalities. We show that a pure branching algorithm based on a discrete relaxation compares well with linear programming based branch-and-bound methods.

The paper is organized as follows. In Section 2 we give basic definitions and present a normal form for linear pseudo-Boolean inequalities. The DP-procedure for solving SAT is recalled in Section 3. Its generalization to pseudo-Boolean constraints and the application to linear pseudo-Boolean optimization is presented in Section 4 and Section 5 respectively. Implementation issues are mentioned in Section 6. We briefly discuss variable selection heuristics in Section 7. Computational results are given in Section 8 followed by the conclusion.

2 Preliminaries

Let $\mathcal{B} := \{X_1, X_2, \ldots, A, B, \ldots\}$ be a finite set of *Boolean variables*, that is the domain of the variables is $\{0, 1\}$. A *literal* L_i is either a Boolean variable X_j (positive literal) or its negation $\overline{X_j} = 1 - X_j$ (a negative literal). Let $\mathcal{L}$ be the set of all literals. The negation of a negative literal $\overline{\overline{X_j}}$ is always simplified to X_j. We denote by $\mathsf{Var}(X_j) = \mathsf{Var}(\overline{X_j}) = X_j$ the variable of a literal. A *linear pseudo-Boolean term* $c_1 L_1 + \cdots + c_n L_n$ (abbreviated by cL) is a sum of products $c_i L_i$, where $c_i \in \mathbb{Z}$ and $L_i \in \mathcal{L}$. We view cL also as a set of products. For a set of products cL we denote by $\sum c := c_1 + \cdots + c_n$ the sum of the integer coefficients of cL. Let $\mathcal{T}$ be the set of all linear pseudo-Boolean terms. An *assignment* is a mapping $\alpha : \mathcal{B} \to \{0, 1\}$. An assignment can also be seen as a 0-1 vector of dimension $|\mathcal{B}|$. Assignments are naturally extended to a mapping $\alpha : \mathcal{T} \to \mathbb{Z}$.

A *linear pseudo-Boolean inequality* is of the form $cL \geq d$. An assignment α satisfies a linear pseudo-Boolean inequality $cL \geq d$ if $\alpha(cL) \geq d$. If there is no assignment satisfying $cL \geq d$, we simplify $cL \geq d$ to the contradiction $\perp$. If every assignment satisfies $cL \geq d$ then $cL \geq d$ is a tautology and we simplify it to $\top$. An assignment satisfies a set S of linear pseudo-Boolean inequalities if it satisfies each linear pseudo-Boolean inequality in S. The extension $\mathsf{Ext}(S)$ of S is the set of assignments satisfying S. A set S of linear pseudo-Boolean inequalities *(strictly) dominates* a set of linear pseudo-Boolean inequalities S' if $\mathsf{Ext}(S)$ is a (proper) subset of $\mathsf{Ext}(S')$. A linear pseudo-Boolean inequality $cL \geq d$, where $c_i = 1$ for all $1 \leq i \leq n$ is called an *extended clause* and abbreviated by $L \geq d$. If additionally $d = 1$, then we call the linear pseudo-Boolean inequality a *classical clause*. Note that $L_1 + \cdots + L_n \geq 1$ is equivalent to the disjunction of the literals $L_1 \vee \ldots \vee L_n$. Deciding whether there is an assignment satisfying a set of classical clauses is the *propositional satisfiability problem* (SAT). Next, we define a normal form for linear pseudo-Boolean inequalities.

Definition 2.1 A linear pseudo-Boolean inequality $cL \geq d$ is in (pseudo-Boolean) *normal form* if

$$d \geq c_1 \geq \ldots \geq c_n \geq 1 \text{ and } \mathsf{Var}(L_i) \neq \mathsf{Var}(L_j) \text{ for all } 1 \leq i < j \leq n \ . \tag{1}$$

We assume that $d \geq 1$ because otherwise the pseudo-Boolean inequality in normal form is a tautology, that is it is valid for every assignment α and therefore need not be considered.

Proposition 2.2 *[HR68] For each linear pseudo-Boolean inequality, which is not a tautology, there exists an equivalent linear pseudo-Boolean inequality in normal form.*

We constructively describe how to obtain the pseudo-Boolean normal form of a linear pseudo-Boolean inequality. We begin with an arbitrary linear pseudo-Boolean inequality

$$e_1 L'_1 + \cdots + e_m L'_m \geq d' \ . \tag{2}$$

First, we apply several arithmetic equivalence transformations. We rewrite (2) such that literals containing the same variable are grouped together and obtain

$$a_1 X_1 + b_1 \overline{X_1} + \cdots + a_n X_n + b_n \overline{X_n} \geq d' \ ,$$

where the X_i are pairwise different. For each i such that $a_i = b_i$ we can simplify $a_i X_i + b_i \overline{X_i}$ to the constant a_i and move a_i to the right-hand side. So let us assume that $a_i \neq b_i$ for all $1 \leq i \leq n$. Next, we replace $a_i X_i + b_i \overline{X_i}$ by $c'_i L_i + c''_i$ for all $1 \leq i \leq n$, where

$$c'_i L_i + c''_i := \begin{cases} (a_i - b_i) X_i + b_i & \text{if} \quad a_i > b_i \\ (b_i - a_i) \overline{X_i} + a_i & \text{if} \quad b_i > a_i \ . \end{cases}$$

Note that the coefficients c'_i are all positive. Bringing the constants c''_i to the right-hand side gives us the new right-hand side $d = d' - \sum_{i=1}^{n} c''_i$. After re-indexing according to the ordering restriction we have brought the linear 0-1 inequality into the form

$$c'_1 L_1 + \cdots + c'_n L_n \geq d \ , \tag{3}$$

where $c'_1 \geq \cdots \geq c'_n \geq 1$ and $\mathsf{Var}(L_i) \neq \mathsf{Var}(L_j)$ for all $1 \leq i < j \leq n$. Note that $d \geq 1$, since otherwise (2) is a tautology. So far we have only applied arithmetic equivalence transformations, hence an assignment α satisfies (2) if and only if α satisfies (3). When constructing the pseudo-Boolean normal form of a linear pseudo-Boolean inequality we can detect at this point whether it is a tautology or not.

Suppose that $c'_i > d$ for some i, then every assignment α with $\alpha(L_i) = 1$, maps the left-hand side of (3) to an integer greater than d and satisfies (3). For all assignments α with $\alpha(L_i) = 0$, the value of the left-hand side is independent of c'_i. Hence, we can safely replace each c'_i by d if $c'_i > d$. Formally, we define

$$c_i := \begin{cases} c'_i & \text{if} \quad c'_i \leq d \\ d & \text{if} \quad c'_i > d \end{cases} \tag{4}$$

for all $1 \leq i \leq n$, and thereby obtain the pseudo-Boolean normal form

$$c_1 L_1 + \cdots + c_n L_n \geq d \tag{5}$$

of (2) with $d \geq c_1 \geq \cdots \geq c_n \geq 1$. Obviously, an assignment α satisfies (2) if and only if α satisfies (3) if and only if α satisfies (5). The last step of the normalization process as described by (4) is also called *coefficient reduction* [CJP83].

Note that a linear pseudo-Boolean inequality in normal form $cL \geq d$ is satisfiable if and only if $\sum c \geq d$, because $c_i > 0$ for all $1 \leq i \leq n$. Hence, $cL \geq d$ is unsatisfiable if and only if $\sum c < d$, and we can easily decide whether $cL \geq d$ is $\bot$. From now on we assume that all linear pseudo-Boolean inequalities are in pseudo-Boolean normal form.

In *optimization problems* we want to maximize (resp. minimize) a linear pseudo-Boolean term cL w.r.t. a set S of linear pseudo-Boolean inequalities, i.e. we search for a satisfying assignment α of S such that $\alpha(cL) \geq \alpha'(cL)$ (resp. $\alpha(cL) \leq \alpha'(cL)$) for all satisfying assignments α' of S.

3 The Classical Davis-Putnam Procedure

First, we recall the Davis-Putnam enumeration method (DP) for solving the satisfiability problem SAT for a set of classical clauses.

A classical clause $L \geq 1$ with $|L| = 1$, i.e. there is only one literal, is called a *unit clause.* The literal L_i of a unit clause is called a *unit literal* and we know that in all satisfying assignments α of a SAT-problem containing a unit clause $L_i \geq 1$ we have $\alpha(L_i) = 1$. Given a unit literal L_i, the basic step of the DP-procedure is to replace L_i by 1 and $\overline{L_i}$ by 0 in all classical clauses followed by a simplification step. Such a step is called a *unit resolution step.* Formally, we define

$$\mathtt{ures}(L_i \geq 1, L \geq 1) := \begin{cases} L \setminus \{\overline{L_i}\} \geq 1 & \text{if} \quad \overline{L_i} \in L, \\ \top & \text{if} \quad L_i \in L, \\ L \geq 1 & \text{otherwise} \;. \end{cases} \tag{6}$$

Then $\mathtt{ures}(L_i \geq 1, L \geq 1)$ is the classical clause obtained from $L \geq 1$ after replacing L_i by 1 and $\overline{L_i}$ by 0 followed by the simplification step. Since $\alpha(L_i) = 1$ for all solutions α of a SAT-problem containing $L_i \geq 1$, we know that each solution of $\{\mathtt{ures}(L_i \geq 1, L \geq 1), L_i \geq 1\}$ is a solution of $L \geq 1$. Therefore, $L \geq 1$ is dominated by $\{\mathtt{ures}(L_i \geq 1, L \geq 1), L_i \geq 1\}$ and can be replaced by $\mathtt{ures}(L_i \geq 1, L \geq 1)$ in a SAT-problem containing $L_i \geq 1$. For a set of classical clauses S we define

$$\mathtt{ures}(L_i \geq 1, S) := \{\mathtt{ures}(L_i \geq 1, L \geq 1) \mid L \geq 1 \in S \text{ and } \mathtt{ures}(L_i \geq 1, L \geq 1) \neq \top\} \;. \tag{7}$$

Note that there are no tautologies $\top$ in $\texttt{ures}(L_i \geq 1, S)$. When applying ures to a set of classical clauses, we may generate further unit clauses and ures may be applicable again. *Unit resolution*, or *clausal chaining*, for a set of classical clauses S means to apply ures as long as there is a unit clause in S.

In this paper we describe algorithms as a *transition system*, i.e. a set of *transition rules* of the form

$$\frac{\mathsf{State}_i}{\mathsf{State}_{i+1}} \text{ if } C \ , \tag{8}$$

which say that we replace a state State_i by a state State_{i+1} if the condition C holds. States State_i are typically tuples of the form $\langle X, Y\rangle$. A state State_n is a *normal form* of State_1 if State_n can be obtained by applying a set of transition rules to State_1 and for State_n no rule is applicable (i.e. for each transition rule its condition C does not hold). In our algorithms we take care that at most one transition rule may apply to a state. Hence, if there exists a normal form, i.e. the algorithm is terminating, the normal form is unique.

Unit resolution for a set of classical clauses S is described by the following transition rule, where S is a set of classical clauses and U is a set of literals.

$$\frac{\langle S, U\rangle}{\langle \texttt{ures}(L_i \geq 1, S), U \cup \{L_i\}\rangle} \text{ if } L_i \geq 1 \in S \tag{9}$$

We denote by $\langle \texttt{ur}(S), \texttt{ul}(S)\rangle$ the normal form obtained by applying the transition rule (9) as long as possible starting with $\langle S, \emptyset\rangle$. The set $\texttt{ul}(S)$ then contains the set of all literals that occurred in a unit clause during the application of unit resolution. Moreover, $\texttt{ur}(S)$ does not contain a unit clause and we have

$$\mathsf{Ext}(\texttt{ur}(S) \cup \{\texttt{ul}(S) \geq |\texttt{ul}(S)|\}) = \mathsf{Ext}(S) \ . \tag{10}$$

If $\texttt{ur}(S)$ contains the empty clause $\bot$, S is unsatisfiable and we say that the *unit relaxation* of S is unsatisfiable. On the other hand, we know that S is satisfiable if and only if $\texttt{ur}(S)$ is satisfiable, since each solution α of S is a solution of $\texttt{ur}(S)$ and $\alpha(L_i) = 1$ for all $L_i \in \texttt{ul}(S)$. Note that $\mathsf{Var}(L_i) \neq \mathsf{Var}(L_j)$ for all different L_i and L_j in $\texttt{ul}(S)$. It is well known that the unit relaxation of S is unsatisfiable if and only if the linear programming relaxation of S is unsatisfiable [BJL86].

Given a set S of classical clauses, the DP-procedure searches a solution α of S by exploring a search tree, where each node of the tree represents a SAT-problem. Next, we define a transition system that operates on states $\langle P, Sol\rangle$. Here, P is a set of tuples $\langle S, U\rangle$, where S is a SAT-problem and U is a set of literals that have been fixed so far, representing the nodes that still need to be explored. We collect the sets of fixed literals that generate an empty SAT-problem as a set of sets of literals Sol.

$$\texttt{dp\_clash:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\langle P, Sol\rangle} \text{ if } \bot \in S$$

$$\texttt{dp\_sol:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\langle P, Sol \cup \{U\}\rangle} \text{ if } S = \emptyset$$

$$\texttt{dp\_split:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\left\langle \left\{ \begin{array}{c} \langle \texttt{ur}(S'), \texttt{ul}(S') \cup U\rangle, \\ \langle \texttt{ur}(S''), \texttt{ul}(S'') \cup U\rangle \end{array} \right\} \cup P, Sol\right\rangle} \text{ if } \begin{array}{l} \bot \notin S, S \neq \emptyset, \\ L_i = \texttt{select\_literal(S)}, \\ S' = S \cup \{L_i \geq 1\} \text{ and} \\ S'' = S \cup \{\overline{L_i} \geq 1\} \end{array}$$

In $\texttt{dp\_split}$ we select a *branching literal* L_i occurring in some clause of the current SAT-problem by a procedure $\texttt{select\_literal}$. Applying the transition system defined by the three rules $\texttt{dp\_clash}$, $\texttt{dp\_sol}$ and $\texttt{dp\_split}$ as long as possible on $\langle\{\langle \texttt{ur}(S), \texttt{ul}(S)\rangle\}, \emptyset\rangle$, where S is a set of classical clauses, yields the normal form $\langle \emptyset, \mathit{Sol}\rangle$. Note that unit resolution applies at least once for each of the two subproblems introduced by $\texttt{dp\_split}$, since we add a unit clause. Therefore, the number of literals in each of the two subproblems is smaller than the number of literals in the selected problem. Hence, the above defined transition system always terminates. We know that S is satisfiable if and only if $\mathit{Sol} \neq \emptyset$. From each element $U \in \mathit{Sol}$ we can construct a solution $\alpha \in \mathsf{Ext}(S)$ by defining $\alpha(L_i) := 1$ for all $L_i \in U$ and arbitrary otherwise. The set of all solutions of S is obtained by building all solutions α for each $U \in \mathit{Sol}$.

For efficiency reasons the search tree is typically explored depth first, i.e. the set of subproblems is implemented as a "Last In, First Out" (LIFO) data structure. Since we are only interested in whether the SAT-problem is satisfiable or not, we stop as soon as the first solution is found. We define

$$\texttt{dp}(S) := \begin{cases} (\top, U) & \text{if} \qquad S \text{ is satisfiable and } U \in \mathit{Sol} \\ (\bot, \emptyset) & \text{otherwise} \ , \end{cases} \tag{11}$$

which yields $(\top, U)$ if and only if S is satisfiable and U is the first set of literals that has been added to Sol.

4 Davis-Putnam for Linear Pseudo-Boolean Inequalities

We obtain a DP like procedure for solving the satisfiability problem of a set of linear pseudo-Boolean inequalities by generalizing unit resolution to *pseudo-Boolean unit resolution*. The key idea of the unit resolution procedure is to fix unit literals to their only possible value and then to make obvious inferences, i.e. fixing other literals and detection of (un)satisfiability. Therefore, we first need to determine whether a linear pseudo-Boolean inequality implies the fixing of a literal.

Lemma 4.1 *A linear pseudo-Boolean inequality $cL \geq d$ dominates $L_i \geq 1$ if and only if $c_iL_i \in cL$ and $\sum c - c_i < d$, where $\sum c$ denotes the sum over all coefficients of c.*

If a linear pseudo-Boolean inequality $cL \geq d$ dominates $L_i \geq 1$, we call L_i a *unit literal* of $cL \geq d$.

Lemma 4.2 *If a linear pseudo-Boolean inequality $cL \geq d$ dominates $L_i \geq 1$ with $c_iL_i \in cL$, then $cL \geq d$ dominates $L_j \geq 1$ for all $c_jL_j \in cL$ with $c_j \geq c_i$.*

Using Lemma 4.2, we see that a linear pseudo-Boolean inequality $cL \geq d$ dominates $L_i \geq 1$ for some L_i if and only if $cL \geq d$ dominates $L_1 \geq 1$, since L_1 is a literal with the largest coefficient in a linear pseudo-Boolean inequality $cL \geq d$ in normal form. We define

$$\texttt{fixed}(cL \geq d) := \begin{cases} L_1 & \text{if } \sum c - c_1 < d \\ \bot & \text{otherwise} \ . \end{cases} \tag{12}$$

Then $\texttt{fixed}(cL \geq d) = L_1$ if and only if $cL \geq d$ dominates $L_1 \geq 1$. Moreover, if $\texttt{fixed}(cL \geq d) = \bot$, then $cL \geq d$ does not dominate any unit clause. Given a unit clause $L_i \geq 1$, we can simplify a linear pseudo-Boolean inequality $cL \geq d$ by replacing L_i by 1 and $\overline{L_i}$ by 0. Resulting constants on the left-hand side are brought to the right-hand side. Special cases arise when the linear pseudo-Boolean inequality becomes tautologous after fixing ($\top$) or unsatisfiable ($\perp$). Formally, we define

$$\mathtt{fix}(L_i, cL \geq d) := \begin{cases} \top & \text{if } d - c_i \leq 0 \text{ and } c_i L_i \in cL, \\ cL \setminus \{c_i L_i\} \geq d - c_i & \text{if } d - c_i > 0 \text{ and } c_i L_i \in cL, \\ \perp & \text{if } \sum c - c_i < d \text{ and } c_i \overline{L_i} \in cL, \\ cL \setminus \{c_i \overline{L_i}\} \geq d & \text{if } \sum c - c_i \geq d \text{ and } c_i \overline{L_i} \in cL, \\ cL \geq d & \text{if neither } L_i \text{ nor } \overline{L_i} \text{ in } L \ . \end{cases} \tag{13}$$

Then $\mathtt{fix}(L_i, cL \geq d)$ denotes the linear pseudo-Boolean inequality obtained by fixing a unit literal L_i in $cL \geq d$ followed by the simplification step. We have

$$\mathsf{Ext}(\{cL \geq d, L_i \geq 1\}) = \{\mathtt{fix}(L_i, cL \geq d), L_i \geq 1\} \ .$$

Note that fixing a literal L_i in a linear pseudo-Boolean inequality $cL \geq d$ may produce a tautology only if $L_i \in L$ and may produce a contradiction only if $\overline{L_i} \in L$. Given a set S of linear pseudo-Boolean inequalities, we denote by $\mathtt{fix}(L_i, S)$ the set of all linear pseudo-Boolean inequalities $\mathtt{fix}(L_i, cL \geq d) \neq \top$ with $cL \geq d \in S$. Note that no tautologies are in $\mathtt{fix}(L_i, S)$. Next, we define *pseudo-Boolean unit resolution* for a set of linear pseudo-Boolean inequalities S, the generalization of the unit resolution procedure for classical clauses to the pseudo-Boolean case.

$$\frac{\langle S, U\rangle}{\langle \mathtt{fix}(L_i, S), U \cup \{L_i\}\rangle} \ \text{if } cL \geq d \in S \text{ and } L_i = \mathtt{fixed}(cL \geq d) (\neq \perp) \tag{14}$$

We denote by $\langle \mathtt{pbur}(S), \mathtt{pbul}(S)\rangle$ the normal form obtained by applying the transition rule (14) as long as possible starting with $\langle S, \emptyset\rangle$. The set $\mathtt{pbul}(S)$ then contains the set of all unit literals detected by $\mathtt{fixed}$ during the application of pseudo-Boolean unit resolution. Moreover, $\mathtt{pbur}(\mathrm{S})$ does not contain a linear pseudo-Boolean inequality $cL \geq d$ with $\mathtt{fixed}(cL \geq d) \neq \perp$ and we have

$$\mathsf{Ext}(\mathtt{pbur}(S) \cup \{\mathtt{pbul}(S) \geq |\mathtt{pbul}(S)|\}) = \mathsf{Ext}(S) \ . \tag{15}$$

The *pseudo-Boolean unit relaxation* of S is the set of all assignments α such that $\alpha(L_i) = 1$ for all $L_i \in \mathtt{pbul}(U)$ if $\perp \notin \mathtt{pbur}(S)$ and $\emptyset$ otherwise. If $\perp \in \mathtt{pbur}(S)$, then we say that the pseudo-Boolean unit relaxation of S is unsatisfiable. On the other hand, S is satisfiable if and only if $\mathtt{pbur}(S)$ is satisfiable, since each solution α of S is a solution of $\mathtt{pbur}(S)$ and $\alpha(L_i) = 1$ for all $L_i \in \mathtt{pbul}(S)$. If S contains only classical clauses, then $\mathtt{ur}(S) = \mathtt{pbur}(S)$ and therefore $\perp \in \mathtt{pbur}(S)$ if and only if $\perp \in \mathtt{ur}(S)$. If the pseudo-Boolean unit relaxation of S is unsatisfiable, then the linear programming relaxation of S is unsatisfiable. The converse no longer holds. For example, consider

$$S = \{1 \cdot A + 1 \cdot B + 1 \cdot C \geq 2, 1 \cdot \overline{A} + 1 \cdot \overline{B} + 1 \cdot \overline{C} \geq 2\} \ . \tag{16}$$

Since $\mathtt{fixed}(1 \cdot A + 1 \cdot B + 1 \cdot C \geq 2) = \mathtt{fixed}(1 \cdot \overline{A} + 1 \cdot \overline{B} + 1 \cdot \overline{C} \geq 2) = \perp$, we have $\mathtt{pbur}(S) = S$ and $\mathtt{pbul}(S) = \emptyset$, thus $\perp \notin \mathtt{pbur}(S)$. On the other hand, the sum of both linear pseudo-Boolean inequalities simplifies to $3 \geq 4$ and therefore the linear programming relaxation is unsatisfiable.

Next, we generalize DP to the pseudo-Boolean case. Again, we define a transition system that operates on states $\langle P, \mathit{Sol}\rangle$, but here P is a set of tuples $\langle S, U\rangle$, where S is a set of linear pseudo-Boolean inequalities. For the sake of completeness we include the transition system, which is almost identical to the one for classical clauses.

$$\texttt{pbdp\_clash:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\langle P, Sol\rangle} \text{ if } \perp \in S$$

$$\texttt{pbdp\_sol:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\langle P, Sol \cup \{U\}\rangle} \text{ if } S = \emptyset$$

$$\texttt{pbdp\_split:} \quad \frac{\langle\langle S, U\rangle \uplus P, Sol\rangle}{\langle \left\{ \begin{array}{l} \langle \texttt{pbur}(S'), \texttt{pbul}(S') \cup U\rangle, \\ \langle \texttt{pbur}(S''), \texttt{pbul}(S'') \cup U\rangle \end{array} \right\} \cup P, Sol\rangle} \text{ if } \begin{array}{l} \perp \notin S, S \neq \emptyset, \\ L_i = \texttt{select\_literal(S)}, \\ S' = S \cup \{L_i \geq 1\} \text{ and} \\ S'' = S \cup \{\overline{L_i} \geq 1\} \end{array}$$

Only the unit resolution procedure is replaced by its generalization to the pseudo-Boolean case. Applying the transition system defined by the three rules `pbdp_clash`, `pbdp_sol` and `pbdp_split` as long as possible on $\langle\{\langle \texttt{pbur(S)}, \texttt{pbul(S)}\rangle\}, \emptyset\rangle$, where S is a set of linear pseudo-Boolean inequalities, yields the normal form $\langle \emptyset, Sol\rangle$ and Sol represents the set of all solutions of S. We define

$$\texttt{pbdp}(S) := \left\{ \begin{array}{lll} (\top, U) & \text{if} & S \text{ is satisfiable and } U \in Sol \\ (\perp, \emptyset) & \text{otherwise} , & \end{array} \right. \tag{17}$$

which yields $(\top, U)$ if and only if S is satisfiable and U is the first set of literals that is added to Sol when exploring the search tree depth first and stopping after the first solution has been found.

5 Optimizing with Pseudo-Boolean Davis-Putnam

Optimizing a linear pseudo-Boolean term cL subject to a set of linear pseudo-Boolean inequalities S can be done by solving a sequence of pseudo-Boolean satisfiability problems. We consider the problem of maximizing cL subject to S. Minimization works in a similar way.

The goal is to find a solution α of S such that $\alpha(cL) \geq \alpha'(cL)$ for all solutions α' of S. We will find such an assignment by solving a sequence of satisfiability problems of the form $S_i := S \cup \{cL \geq \max_i\}$, where only $\max_i$ differs from problem to problem. We call $cL \geq \max_i$ the *objective function inequality.* Suppose that $\max_0$ is such that $cL \geq \max_0$ is a tautology. Solving S_0 yields a solution α_0 of S_0 and therefore of S. A lower bound of the optimum then is $\alpha_0(cL)$. We define $\max_{i+1} := \max_i +1$ and so exclude assignments yielding no better lower bounds than the current one. Obviously, we have $\alpha_{i+1}(cL) > \alpha_i(cL)$ for all satisfying assignments α_i of S_i and α_{i+1} of S_{i+1}. If S_i is satisfiable and S_{i+1} is unsatisfiable, then $\alpha_i(cL)$ is the desired maximum.

We can incorporate this idea into `pbdp` just by replacing the rule `pbdp_sol`. Instead of adding a computed solution that fixes the literals U, we construct a solution α such that $\alpha(L_i) = 1$ for all $L_i \in U \cup (L \setminus \overline{U})$, where $\overline{U} := \{\overline{L_i} \mid L_i \in U\}$ is the set of all negated literals in U. Then $\alpha(cL)$ is the maximal value of max: cL subject to $\{U \geq |U|\}$. We add to the remaining satisfiability problems the linear pseudo-Boolean inequality $cL \geq \alpha(cL) + 1$. Thus, we ensure that for each further solution the value of the objective function is larger. If there are no remaining nodes, then the last computed $\alpha(cL)$ was optimal.

Suppose that cL is a linear pseudo-Boolean term with $c_i > 0$ for all $c_i \in c$. We define a transition system that operates on states $\langle P, \max\rangle$, where P is a set of tuples $\langle S, U\rangle$, S is a set of linear pseudo-Boolean inequalities, and U is a set of literals that are fixed so far. The current lower bound of the maximization problem is max.

$$\texttt{opbdp\_clash:} \quad \frac{\langle\langle S, U\rangle \uplus P, \max\rangle}{\langle P, \max\rangle} \quad \text{if } \perp \in S$$

$$\texttt{opbdp\_climb:} \quad \frac{\langle\langle S, U\rangle \uplus P, \max\rangle}{\langle \texttt{propagate}(P, \max' + 1), \max'\rangle} \quad \text{if} \quad \begin{array}{l} S = \emptyset, \\ \forall L_i \in U \cup (L \setminus \overline{U}) : \alpha'(L_i) = 1 \text{ and} \\ \max' = \alpha'(cL) \end{array}$$

$$\texttt{opbdp\_split:} \quad \frac{\langle\langle S, U\rangle \uplus P, \max\rangle}{\langle \left\{ \begin{array}{l} \langle \texttt{pbur}(S'), \texttt{pbul}(S') \cup U\rangle, \\ \langle \texttt{pbur}(S''), \texttt{pbul}(S'') \cup U\rangle \end{array} \right\} \cup P, \max\rangle} \quad \text{if} \quad \begin{array}{l} \perp \notin S, S \neq \emptyset, \\ L_i = \texttt{select\_literal(S)}, \\ S' = S \cup \{L_i \geq 1\} \text{ and} \\ S'' = S \cup \{\overline{L_i} \geq 1\} \end{array}$$

It remains to define `propagate`, which updates the remaining satisfiability problems such that only better lower bounds are generated.

$$\begin{array}{lcl} \texttt{propagate}(\langle S, U\rangle \uplus P, \max) & := & \{\langle \texttt{pbur}(S \cup \{cL \geq \max\}), \texttt{pbul}(S \cup \{cL \geq \max\}) \cup U\rangle\} \\ & & \cup\ \texttt{propagate}(P, \max) \\ \texttt{propagate}(\emptyset, \max) & := & \emptyset \end{array}$$

Applying the transition system defined by the three rules `opbdp_clash`, `opbdp_climb`, and `opbdp_split` as long as possible starting with $\langle\{\langle \texttt{pbur}(S), \texttt{pbul}(S)\rangle\}, -1\rangle$ yields the normal form $\langle\emptyset, \max\rangle$, where $\max \geq \alpha(cL)$ for all $\alpha \in \mathsf{Ext}(S)$ and $\max = -1$ if and only if S is unsatisfiable. We define the optimization procedure $\texttt{opbdp}(S, cL) := \max$, where max is the value computed by the above given transition system.

It is no restriction to require that cL contains only positive coefficients. Each term $c_i L_i$ with $c_i < 0$ is equivalent to $c_i + |c_i|\overline{L_i}$, since

$$\begin{array}{rcl} c_i L_i & = & c_i - c_i + c_i L_i \\ & = & c_i - c_i \cdot (1 - L_i) \\ & = & c_i - c_i \overline{L_i} \ . \end{array}$$

Optimizing a linear pseudo-Boolean term $\sum_{i=1}^{n} c_i L_i$ containing also negative coefficients is done by optimizing $\sum_{i=1}^{n} |c_i| L_i'$, where L_i' is L_i if $c_i > 0$ and $\overline{L_i}$ otherwise, and adding the sum over the negative coefficients c_i to the positive result of `opbdp`.

6 Implementation

Proper implementation and the use of appropriate data structures are important factors for the efficiency of the DP-procedure [Hoo93, HHT94, Zha93]. In a naive implementation of `dp`, a satisfiability problem has to be copied at each node which may require excessive storage. We use the following data structure for implementing `dp`. Given a set S of classical clauses and the set V of variables occurring in S, we compute in advance the following index data structures. Each classical clause can be referenced by a unique identifier cid. The set of literals in a classical clause with identifier cid can be referenced by lids(cid). We can obtain a list of clause identifiers where a literal L_i occurs by $\mathsf{cids}(L_i)$. We represent a sub-problem of S by a list of *active variables* av, i.e. variables that are not yet fixed, and a list of *clause states*, where a clause state is a clause identifier and the number of literals nl currently active in that classical clause. Applying unit resolution for the literal L_i is done by

- deleting each element in the clause state list, where the identifier in the clause state is in $\mathsf{cids}(L_i)$
- and decreasing nl by one if the identifier in the clause state is in $\mathsf{cids}(\overline{L_i})$.

We obtain a unit clause if nl is set to 1. The new unit literal then is determined by $\mathsf{lids}(cid) \cap (\mathsf{av} \cup \overline{\mathsf{av}})$ and its variable is deleted from av. The empty clause is derived if nl is set to 0. Copying a sub-problem is now done by just copying the active clause list and the clause state list. The advantage of this data structure is that the index data structure does not change while exploring the search tree. The representation of the sub-problem in a node is very compact and detection of unit literals is straightforward.

For pbdp each linear pseudo-Boolean inequality can be referenced by a unique inequality identifier iid. Given a literal L_i, we can obtain a list of inequality identifiers where L_i occurs by $\mathsf{iids}(L_i)$. We need an additional index data structure to access the coefficient $c_i = \mathsf{coeff}(L_i, \mathsf{iid})$ of a given literal L_i and an inequality identifier *iid*. A sub-problem is represented by a list of active variables av and a list of inequality states. An inequality state consists of an inequality identifier iid, a current right-hand side d of the linear pseudo-Boolean inequality and a current sum $\sum c$ over all coefficients of the active literals of the linear pseudo-Boolean inequality. For applying pseudo-Boolean unit resolution for the literal L_i we do the following for all inequality states where the inequality identifier iid is in $\mathsf{iids}(L_i)$:

- Delete the inequality state if $\mathsf{coeff}(L_i, \mathsf{iid}) \geq d$ (eliminate tautologies).
- Otherwise decrease the current right-hand side d and the current sum over all coefficients $\sum c$ by $\mathsf{coeff}(L_i, \mathsf{iid})$.

For all inequality states, where the inequality identifier iid is in $\mathsf{iids}(\overline{L_i})$ we do the following:

- Decrease the sum over all coefficients $\sum c$ by $\mathsf{coeff}(\overline{L_i}, \mathsf{iid})$. If $\sum c$ is smaller than the current right-hand side, then the sub-problem is unsatisfiable.
- Compute the active literal $L_j \in \mathsf{av}$ for which $\mathsf{coeff}(L_j, \mathsf{iid})$ is maximal. L_j is a new unit literal, i.e. it can be fixed, if $\sum c - \mathsf{coeff}(L_j, \mathsf{iid}) < d$.

For large linear pseudo-Boolean inequalities, computing the active literal $L_j \in \mathsf{av} \cup \overline{\mathsf{av}}$ for which $\mathsf{coeff}(L_j, \mathsf{iid})$ is maximal will often yield the same literal. One improvement is to store the maximal coefficient with its literal in the inequality state and to update it only if this literal is fixed.

We implement opbdp as a slight variation of pbdp. When a satisfiable solution is found (i.e an empty inequality state list) we do not return $\top$, but calculate the maximal value max of the objective function under the current assignment. We then change destructively all copied right-hand sides d of the objective function inequality states to the new value $d - (\max +1 - \max')$ where $\max'$ is the previous calculated maximal value. The current node then is unsatisfiable and we backtrack until the objective function constraint is no longer violated. Then standard pbdp depth first search is continued. We follow roughly the ideas presented by Hooker [Hoo93] for incrementally solving SAT.

7 Heuristics

The selection of the branching literal [JW90, HHT94, HV94] is the other important factor for the efficiency of the DP-procedure. Good literal selection heuristics can reduce the number of explored nodes by an order of magnitude. Hooker et. al [HV94] investigate several branching heuristics for the clausal case. The "Two-Sided Jeroslow-Wang" rule has been justified by a Markov chain analysis of a simplification model that selects the literal L_i when $\texttt{pbur}(S \cup \{L_i \geq 1\})$ is the *mostly simplified* problem [HV94]. The rule says that we should branch on the variable X_i which maximizes $J(X_i) + J(\overline{X_i})$, with

$$J(L_i) := \sum_{L \in S : L_i \in L} 2^{-|L|} \quad . \tag{18}$$

If $J(X_i) \geq J(\overline{X_i})$, then X_i should be selected, otherwise $\overline{X_i}$. The intention of the rule is that we branch on a variable that occurs often in short clauses and so get a small sub-problem with an increasing probability that unit literals occur.

For linear pseudo-Boolean inequalities we adapt the two-sided Jeroslow-Wang rule and branch on a variable X_i that maximizes $PBJ(X_i) + PBJ(\overline{X_i})$, with

$$PBJ(L_i) = \sum_{cL \geq d \in S : c_i L_i \in cL} \frac{c_i}{\sum c - d} \cdot 2^{-(\sum c/d)} \quad . \tag{19}$$

If $PBJ(X_i) \geq PBJ(\overline{X_i})$, then X_i is selected, otherwise $\overline{X_i}$. We replace $|L|$, which is a measure of the length of a clause, by $\sum c/d$, which reduces to the length if we have classical clauses and so is a straightforward generalization. The idea is that we prefer linear pseudo-Boolean inequalities, where many of the coefficients are needed in order to satisfy the linear pseudo-Boolean inequality. We weight this preference by multiplying $2^{-(\sum c/d)}$ by $c_i/(\sum c - d)$ and so take into account the relative improvement of reaching the right-hand side when fixing the literal to 1. Further analysis is required in order to obtain a more efficient branching heuristics. For `opbdp` we use as alternative a greedy-like heuristics depending only on the objective function. We simply select the literal L_i with the maximal coefficient in the objective function. The idea is that a better approximation of the optimal value is obtained earlier. A better literal selection heuristics that takes into account the goal of maximizing an objective function *and* reducing the search space needs to be found.

8 Computational Results

A prototype of `opbdp` has been implemented in C++.

System available:
`ftp.mpi-sb.mpg.de:/pub/guide/staff/barth/opbdp/opbdp.tar.Z` or
`http://www.mpi-sb.mpg.de:/guide/staff/barth/barth.html`.

In Figure 1 we present computational results for using `opbdp` on a variety of pure 0-1 integer programming problems found in MIPLIB [BBI92]. In column "Name" we give the name of the problem. In "# V" we mention the number of variables and in "# I" the number of inequalities of the problem. "Nodes" is the number of nodes explored for solving the problem and "Time" is

Name	#I	#V	DS-JW (opt) Nodes	Time	DS-JW (ver) Nodes	Time	OF (opt) Nodes	Time	OF (ver) Nodes	Time
air01	46	771	43553	184.90	70070	279.73	1549	51.26	19128	72.33
air02	100	6774	?	&	?	&	2721179	35496.44	5961252	75622.30
bm23	20	27	5002	2.71	6184	3.41	4510	0.33	7772	0.60
enigma	42	100	2401	1.06	2401	1.06	659	0.31	659	0.31
lseu	28	89	3035527	1076.04	3515755	1261.22	650193	69.16	10063415	997.28
misc01	69	82	345	0.34	1740	1.15	19740	2.79	49010	7.61
misc02	51	58	104	0.06	289	0.13	141	0.04	395	0.08
misc03	121	159	19578	21.16	32172	34.76	425	0.15	357611	112.01
misc07	245	259	1252433	1887.23	2312017	4098.65	164544	84.94	53120546	26127.40
p0033	15	33	336	0.06	916	0.15	1047	0.06	17344	1.10
p0040	23	40	346905	33.16	490410	46.83	61	0.03	542998	28.11
p0201	133	201	?	&	?	&	499	0.53	?	&
p0282	221	282	379895	281.43	385142	289.70	62573	20.80	63746	21.23
p0291	205	291	4654	3.63	5199	4.05	487	1.60	686	1.68
sentoy	30	60	223504	275.68	301374	424.40	778	0.21	58366	10.45
stein15	36	15	9	0.00	436	0.05	6	0.03	686	0.05
stein27	118	27	2587	0.31	29216	4.10	17	0.01	38324	3.28
stein45	331	45	157389	56.23	1446536	534.98	9170	1.73	3543186	774.16

& : aborted after 30000 seconds

Figure 1: 0-1 Integer Optimization Problems from MIPLIB with `opbdp`

the user cpu time used for solving the problem with `opbdp` on a SPARC-10/31. In the "DS-JW" columns we use `opbdp` with the adapted two-sided Jeroslow-Wang heuristics and in the columns "OF" we select the literal with the largest coefficient in the objective function. The columns "...(opt)" describe when the optimal value is found and the columns "...(ver)" describe when the optimal result is verified and `opbdp` terminates. In Figure 2 we solve the same 0-1 integer optimization problems with CPLEX 3.0. CPLEX is a commercial mixed integer solver based on the simplex algorithm. Integer problems are solved by branch-and-bound. CPLEX offers the possibility to additionally generate cutting planes (e.g. clique inequalities and lifted knapsack covers) at each node if possible, yielding a branch-and-cut algorithm. CPLEX does not report the time when the optimal solution was found. The other columns in Figure 2 are similar to Figure 1. We solve all problems first by branch-and-bound (B&B) and then with branch-and-cut (B&C)[1]. On some problems, e.g. "p0201", the cutting plane generation gives a significant speedup.

The presented `opbdp` procedure compares well w.r.t. the linear programming based solver CPLEX. Especially "enigma" and the "stein"-problems are solved much faster by `opbdp`. On other problems, i.e. "lseu" and the "air"-problems, the linear programming based method is faster. Problems where the optimal value of the linear programming relaxation is already near the integer optimal value, but the number of variables is very high (the "air"-problems), are better attacked by linear programming based methods. It is interesting to note that exploiting the logical structure of a problem, i.e. using `opbdp`, yields good performance on problems where exploiting the polyhedral structure seems to be inefficient and vice versa. Hence, there is no "better" approach, but problem specific characteristics that can be explored. An integration of both approaches and systems that exploit either structure need to be investigated and may lead

[1] For B&B we disallowed clique and cover generation (option -1). For B&C we forced clique and cover generation (option 1).

Name	# I	# V	B&B(opt) Nodes	B&B(ver) Nodes	B&B(ver) Time	B&C(opt) Nodes	B&C(ver) Nodes	B&C(ver) Time
air01	23	771	3	3	0.35	3	3	0.35
air02	50	6774	19	20	13.78	19	20	13.52
bm23	20	27	452	452	1.88	521	590	5.80
enigma	21	100	22086	22086	214.68	22086	22086	213.85
lseu	28	89	10811	15889	82.05	1576	1932	14.95
misc01	54	83	76	721	6.97	104	439	5.95
misc02	39	59	29	60	0.60	20	44	0.55
misc03	96	160	24	527	16.77	24	428	15.40
misc07	212	260	2601	22832	1450.48	1613	11854	858.70
p0033	16	33	662	964	2.23	7	82	0.47
p0040	23	40	51	54	0.17	0	0	0.07
p0201	133	201	760	1023	25.83	478	986	45.25
p0282	241	282	?	?	&	582	749	39.65
p0291	252	291	86	97	0.58	28	39	0.80
sentoy	30	60	595	723	4.92	1129	1129	19.97
stein15	36	15	7	85	0.38	7	85	0.38
stein27	118	27	8	4061	32.87	8	4061	32.83
stein45	331	45	29373	71595	2910.18	29373	71595	2924.38

& : aborted after 30000 seconds

Figure 2: 0-1 Integer Optimization Problems from MIPLIB with CPLEX

to more powerful optimization algorithms.

Note that in all but one of the problems the optimal solution was found relatively early when using the greedy like heuristics. Unfortunately, the number of evaluated nodes does not decrease significantly when using the adapted Two-Sided Jeroslow-Wang heuristics, which indicates that better heuristics need to be developed. Special handling of equality constraints are further possibilities to improve the presented algorithm. Note that all occurring linear pseudo-Boolean equalities $cL = d$ have been replaced by the two linear pseudo-Boolean inequalities $cL \leq d$ and $cL \geq d$. Therefore, the column "# I" may differ from the column "ROWS" found in MIPLIB. Direct use of the equality constraints would reduce the problem size and might help to decrease the number of explored nodes.

We currently require that all the coefficients are integer. The results still hold when rational coefficients are allowed in the constraint inequalities and adapting the presented method is easy. This has not yet been done and therefore some problems ("mod008", "mod010", "pipex") have not been considered. Furthermore, we have not considered some problems with a very large number of variables. We believe that a version of `opbdp` using preprocessing techniques and logic-cut generation techniques [Bar94] could also be applied on larger problems.

9 Conclusion

Branching methods based on a discrete relaxation compare well in efficiency with branching methods based on the linear programming relaxation for solving linear pseudo-Boolean optimization problems. By choosing a straightforward generalization of unit relaxation in propositional calculus to the pseudo-Boolean case, implementation techniques of clausal satisfiability provers can be used. The next step towards an efficient logic-based branch-and-cut method is the incorporation of logic-cut generation methods [Hoo92, Bar94].

References

[Bar94] P. Barth. *Logic-based 0-1 constraint solving in constraint logic programming*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, December 1994. forthcoming.

[BBI92] R. E. Bixby, E. A. Boyd, and R. Indovina. MIPLIB: A Test Set of Mixed-Integer Programming Problems. *SIAM News*, 25(16), 1992.

[BJL86] C. E. Blair, R. G. Jeroslow, and J. K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 13(5):633–645, 1986.

[CJP83] H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, September 1983.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–205, 1960.

[HHT94] F. Harche, J. N. Hooker, and G. L. Thompson. A computational study of satisfiability algorithms for propositional logic. *ORSA Journal on Computing*, 6(4):423–435, 1994.

[Hoo92] J. N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[Hoo93] J. N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15:177–186, 1993.

[Hoo94] J. N. Hooker. Logic-based methods for optimization. *ORSA CSTS Newsletter*, 15(2):4–11, 1994.

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

[HV94] J. N. Hooker and V. Vinay. Branching rules for satisfiability. presented on "Third International Symposium on Artificial Intelligence and Mathematics": Fort Lauderdale; Florida (to appear in Journal of Automated Reasoning), January 1994.

[JW90] R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and AI*, 1:167–187, 1990.

[Lov78] D. W. Loveland. *Automated theorem proving : a logical basis*, volume 6 of *Fundamental studies in computer science*. North-Holland, Amsterdam, 1978.

[Zha93] H. Zhang. Sato: A decision procedure for propositional logic. *Association of Automated Reasoning Newsletters*, 22:1–3, March 1993.

INFORMATIK

Below you find a list of the most recent technical reports of the research group *Logic of Programming* at the Max-Planck-Institut für Informatik. They are available by anonymous ftp from our ftp server `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Regina Kraemer
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `kraemer@mpi-sb.mpg.de`

MPI-I-95-2-005	F. Baader, H.-J. Ohlbach	A Multi–Dimensional Terminological Knowledge Representation Language
MPI-I-95-2-002	H. J. Ohlbach, R. A. Schmidt	Functional Translation and Second–Order Frame Properties
MPI-I-95-2-001	S. Vorobyov	Proof normalization and subject reduction in extensions of Fsub
MPI-I-94-261	P. Barth, A. Bockmayr	Finite Domain and Cutting Plane Techniques in CLP($\mathcal{PB}$)
MPI-I-94-257	S. Vorobyov	Structural Decidable Extensions of Bounded Quantification
MPI-I-94-254		Report and abstract not published
MPI-I-94-252	P. Madden	A Survey of Program Transformation With Special Reference to *Unfold/Fold* Style Program Development
MPI-I-94-251	P. Graf	Substitution Tree Indexing
MPI-I-94-246	M. Hanus	On Extra Variables in (Equational) Logic Programming
MPI-I-94-241	J. Hopf	Genetic Algorithms within the Framework of Evolutionary Computation: Proceedings of the KI-94 Workshop
MPI-I-94-240	P. Madden	Recursive Program Optimization Through Inductive Synthesis Proof Transformation
MPI-I-94-239	P. Madden, I. Green	A General Technique for Automatically Optimizing Programs Through the Use of Proof Plans
MPI-I-94-238	P. Madden	Formal Methods for Automated Program Improvement
MPI-I-94-235	D. A. Plaisted	Ordered Semantic Hyper-Linking
MPI-I-94-234	S. Matthews, A. K. Simpson	Reflection using the derivability conditions
MPI-I-94-233	D. A. Plaisted	The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison
MPI-I-94-232	D. A. Plaisted	An Abstract Program Generation Logic
MPI-I-94-230	H. J. Ohlbach	Temporal Logic: Proceedings of the ICTL Workshop
MPI-I-94-229	Y. Dimopoulos	Classical Methods in Nonmonotonic Reasoning