Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

TIMO BERTHOLD STEFAN HEINZ MARC E. PFETSCH

Solving Pseudo-Boolean Problems with SCIP

Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 08-12 (July 2009)

Solving Pseudo-Boolean Problems with SCIP*

Timo Berthold, Stefan Heinz, and Marc E. Pfetsch

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
{berthold,heinz,pfetsch}@zib.de

Abstract. Pseudo-Boolean problems generalize SAT problems by allowing linear constraints and a linear objective function. Different solvers, mainly having their roots in the SAT domain, have been proposed and compared, for instance, in Pseudo-Boolean evaluations. One can also formulate Pseudo-Boolean models as integer programming models. That is, Pseudo-Boolean problems lie on the border between the SAT domain and the integer programming field.
In this paper, we approach Pseudo-Boolean problems from the integer programming side. We introduce the framework SCIP that implements constraint integer programming techniques. It integrates methods from constraint programming, integer programming, and SAT-solving: the solution of linear programming relaxations, propagation of linear as well as nonlinear constraints, and conflict analysis. We argue that this approach is suitable for Pseudo-Boolean instances containing general linear constraints, while it is less efficient for pure SAT problems.
We present extensive computational experiments on the test set used for the Pseudo-Boolean evaluation 2007. We show that our approach is very efficient for optimization instances and competitive for feasibility problems. For the nonlinear parts, we also investigate the influence of linear programming relaxations and propagation methods on the performance. It turns out that both techniques are helpful for obtaining an efficient solution method.

1 Introduction

Over the past decade, SAT-solvers have grown increasingly more efficient. Since they allow to solve large SAT instances in a consistent and fast manner, also new fields of application have been sought. One such field are Pseudo-Boolean (PB) problems, in which SAT-models are extended by linear and nonlinear constraints. Several PB-solvers have been proposed and compared during the Pseudo-Boolean evaluations, see Manquinho and Roussel [13–16].

One way to solve PB-problems is by transformation to a SAT problem, see, e.g., Eén and Sörensson [9]; this approach is used, for instance, in the solver MINISAT+ [8, 9]. Another way is to handle PB-constraints directly in the solver, see, e.g., PBS [5]. Some solvers use a constraint programming approach, for example, ABSCONPSEUDO [11].

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

Pseudo-Boolean problems can also be formulated as an integer program (IP), in which the nonlinear constraints are linearized. This idea is used by the solver GLPPB, which applies GLPK [10] for solving the IPs. The solver BSOLO [12] combines integer programming techniques with SAT-solving. A comparison of the SAT versus integer programming approaches is given in Aloul et al. [4].

In this paper, we approach PB-problems from a *constraint integer programming* (CIP) point of view. CIP is a combination of integer and constraint programming (CP) methods. We use the framework SCIP that is based on a branch-and-cut method as commonly used for the solution of IPs; see Achterberg [2] for details. Hence, SCIP performs a branch-and-bound algorithm to decompose the problem into subproblems, solves a linear relaxation, and, in order to strengthen the relaxation, it possibly adds additional inequalities (cutting planes). It also incorporates methods from SAT-solving like conflict analysis and restarts. Furthermore, SCIP applies techniques from CP like constraint propagation.

Besides introducing a new PB-solver, the main contribution of this paper is the evaluation of extensive computations on the instances of the Pseudo-Boolean evaluation 2007 [16]. It turns out that SCIP is a very effective solver.

The structure of the paper is as follows. In Section 1.1, we define linear and nonlinear Pseudo-Boolean problems. In Section 2, we introduce the concept of constraint integer programming. Section 2.1 gives a brief account of the different techniques incorporated into SCIP. The computations are discussed in Section 3. We provide an overall summary, a comparison to the results of the PB evaluation 2007, and more details for different problem groups. We also investigate the behavior of SCIP on the problems including nonlinear constraints more thoroughly. We summarize the outcomes in Section 4 and discuss some future challenges.

One reason for the success of our approach is that SCIP is also a very fast CIP solver. It can be used free of charge for academic purposes [21].

1.1 Problem Definition

A *linear Pseudo-Boolean problem* is an optimization problem over n binary (Boolean) variables $x_1, \ldots, x_n$ in the following form:

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x \\ & Ax \geq b \\ & x \in \{0,1\}^n, \end{aligned} \tag{1}$$

where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$. The term $c^{\mathrm{T}} x$ is called the *objective function*. The inequalities in $Ax \geq b$ are called *linear constraints*.

The above format is quite general. First, expressions that involve *literals* $\ell_j \in \{x_j, \overline{x}_j\}$ can be transformed into the above form by using the relation $x_j = 1 - \overline{x}_j$, i.e., if $\ell_j = \overline{x}_j$, we replace ℓ_j by $1 - x_j$, otherwise by x_j. Maximization problems can be transformed to minimization problems by multiplying the objective function coefficients by -1. Similarly, "$\leq$" constraints can be multiplied by -1 to obtain "$\geq$" constraints. Equations can be replaced by two opposite inequalities. Inequalities or objective functions involving rational coefficients can

be multiplied with the least common multiple of the denominators of all coefficients to yield integer numbers.

Integer programs are extensions of linear Pseudo-Boolean instances, in which the variables may also assume arbitrary integer values. Integer programs may be further extended by allowing some variables to take continuous values, which yields *mixed integer programs* (MIPs).

SAT problems are special cases of Pseudo-Boolean problems: A clause of a SAT formula $\ell_1 \vee \cdots \vee \ell_k$ (with literals $\ell_1, \ldots, \ell_k$) can be expressed as

$$\sum_{j=1}^{k} \ell_j \geq 1. \tag{2}$$

Then the literals are transformed as explained above. Inequalities of the type (2) are also called OR-*constraints* or *set covering constraints*. In order to state feasibility problems we can set $c = 0$.

A *nonlinear Pseudo-Boolean constraint* over literals ℓ_{ij} is defined as follows

$$\sum_{i=1}^{t} d_i \prod_{j=1}^{k} \ell_{ij} \geq r, \tag{3}$$

where $d \in \mathbb{Z}^t$, $r \in \mathbb{Z}$. Each product $z_i = \prod_{j=1}^{k} \ell_{ij} \in \{0, 1\}$ can be expressed as an AND-constraint

$$z_i = \bigwedge_{j=1}^{k} \ell_{ij}.$$

The resulting new variables z_i can be inserted into the linear constraint $d^{\mathrm{T}} z \geq r$, which is equivalent to (3). AND-constraints are nonlinear in the sense that they cannot be represented by a single linear constraint. They can either be treated directly by the solver or linearized by adding the following linear constraints.

$$\begin{aligned} z_i &\leq \ell_{ij} \qquad \text{for } j = 1, \ldots, k \\ \sum_{j=1}^{k} \ell_{ij} - z_i &\leq k - 1. \end{aligned} \tag{4}$$

These constraints suffice to describe an AND-constraint in the sense that 0/1-solutions of (4) are solutions of the corresponding AND-constraint and conversely. One can show that the polyhedron defined by (4), $z_{ij} \geq 0$, and $\ell_{ij} \leq 1$ for all i and j is integral, i.e., has only integral vertices.

In our approach we treat AND-constraints in varying levels of algorithmic effort; this is described in Section 3.5.

2 Constraint Integer Programming

The majority of the solvers for SAT, MIP, and CP work in the spirit of branch-and-bound, which means that they recursively subdivide the problem instance

yielding a so-called branch-and-bound-tree, whose nodes represent subproblems of the original instance. Although this strategy implicitly enumerates all potential solutions, the hope is that due to effective processing and bounding of the subproblems, one may prune other parts of the tree.

SAT and MIP are special cases of the general idea of CP. The power of CP arises from the possibility to model the problem directly with a huge variety of different, expressive constraints. In contrast, SAT and MIP only allow for very specific constraints: Boolean clauses for SAT and linear and integrality constraints for MIP. Their advantage, however, lies in the sophisticated techniques available to exploit the structure provided by these constraint types.

An important point for the efficiency of solving algorithms is the interaction between constraints. For instance, in SAT-solving, this takes place via propagation of the variables' domains. In MIP solving there exists a second, more complex but very powerful communication interface: the LP-relaxation.

The goal of constraint integer programming is to combine the advantages and compensate the weaknesses of CP, MIP, and SAT. To support this aim, we slightly restrict the notion of a CP, in order to be able to apply MIP and SAT-solving techniques, and especially provide an LP-relaxation without losing the high degree of freedom in modeling.

Definition. A *constraint integer program* $\mathrm{CIP} = (\mathfrak{C}, I, c)$ consists of solving

$$c^\star = \min\left\{c^{\mathrm{T}}x \;:\; \mathcal{C}_i(x) = 1 \text{ for all } i = 1,\dots,m,\; x \in \mathbb{R}^n,\; x_j \in \mathbb{Z} \text{ for all } j \in I\right\},$$

with a finite set $\mathfrak{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_m\}$ of constraints $\mathcal{C}_i : \mathbb{R}^n \to \{0,1\}$, $i = 1,\dots,m$, a subset $I \subseteq N := \{1,\dots,n\}$ of the variables, and an objective function $c \in \mathbb{Z}^n$. Defining $C := N \setminus I$, a CIP has to fulfill the following additional condition:

$$\forall\, \hat{x}_I \in \mathbb{Z}^I \;\exists\, (A', b') : \{x_C \in \mathbb{R}^C \;:\; \mathfrak{C}(\hat{x}_I, x_C)\} = \{x_C \in \mathbb{R}^C \;:\; A'x_C \le b'\} \quad (5)$$

where $A' \in \mathbb{Z}^{k \times C}$ and $b' \in \mathbb{Z}^k$ for some $k \in \mathbb{Z}_{\ge 0}$.

Restriction (5) ensures that the subproblem remaining after fixing all integer variables always is a linear program. Note that this does not forbid quadratic or more involved expressions – as long as the nonlinearity only refers to the integer variables.

Clearly, MIPs are special cases of CIPs. Hence, the same holds for PB-problems. One can also show that every CP with finite domains for all variables can be modeled as a CIP.

In the following, we will describe basic ideas for solving CIPs on the example of the CIP-framework SCIP (Solving Constraint Integer Programs). We keep the description quite brief and refer to Achterberg [2] for details.

Most MIP solvers are based on a *branch-and-cut* strategy, which is a combination of branch-and-bound and cutting plane algorithms. It is also a main component of SCIP. In every node of the tree the linear relaxation, i.e., a linear program (LP), of the current subproblem is solved. Then iteratively additional cutting inequalities are added to strengthen the relaxation. More information