On Solving Boolean Multilevel Optimization Problems

Josep Argelich
INESC-ID
Lisbon
josep@sat.inesc-id.pt

Inês Lynce
INESC-ID/IST
Technical University of Lisbon
ines@sat.inesc-id.pt

Joao Marques-Silva
CASL/CSI
University College Dublin
jpms@ucd.ie

Abstract

Many combinatorial optimization problems entail a number of hierarchically dependent optimization problems. An often used solution is to associate a suitably large cost with each individual optimization problem, such that the solution of the resulting aggregated optimization problem solves the original set of optimization problems. This paper starts by studying the package upgradeability problem in software distributions. Straightforward solutions based on Maximum Satisfiability (MaxSAT) and pseudo-Boolean (PB) optimization are shown to be ineffective, and unlikely to scale for large problem instances. Afterwards, the package upgradeability problem is related to multilevel optimization. The paper then develops new algorithms for Boolean Multilevel Optimization (BMO) and highlights a number of potential applications. The experimental results indicate that algorithms for BMO allow solving optimization problems that existing MaxSAT and PB solvers would otherwise be unable to solve.

1 Introduction

Many real problems require an optimal solution rather than any solution. Whereas decision problems require a yes/no answer, optimization problems require the best solution, thus differentiating the possible solutions. In practice, there must be a classification scheme to determine how one solution compares with the others. Such classification may be seen as a way of establishing preferences that express cost or satisfaction.

A special case of combinatorial optimization problems may require a set of optimization criteria to be observed, for which is possible to define a hierarchy of importance. Not only you establish a hierarchy in your preferences, but also the preferences are defined in such a way that the set of potential solutions gets subsequently reduced. Such kind of problems are present not only in your daily life but also in many real applications.

Clearly, the problems we target can be encoded as a constraint optimization problem, making use of the available technology for dealing with preferences. Preference handling is a current hot topic in AI with active research lines in constraint satisfaction and optimization [Rossi et al., 2008]. Broadly, preferences over constraints may be expressed quantitatively or qualitatively. Soft constraints model quantitative preferences by associating a level of satisfaction with each of the solutions [Meseguer et al., 2006], whereas CP-nets model qualitative preferences by expressing preferential dependencies with pairwise comparisons [Boutilier et al., 2004]. Furthermore, preference-based search algorithms can be generalized to handle multi-criteria optimization [Junker, 2004].

A straightforward approach to solve a special case of a constraint optimization problem, for which there is a total ranking of the criteria, would be to establish a lexicographic ordering over variables and domains, such that optimal solutions would come first in the search tree [Freuder et al., In Press]. But this has the potential disadvantage of producing a thrashing behavior whenever assignments that are not supported by any solution are considered, as a result of decisions made at the first nodes of the search tree [Junker, 2004].

Maximum satisfiability (MaxSAT) naturally encodes a constraint optimization problem over Boolean variables where constraints are encoded as clauses. A solution to the MaxSAT problem maximizes the number of satisfied clauses. Weights may also be associated with clauses, in which case the sum of the weights of the satisfied clauses is to be maximized. The use of the weighted MaxSAT formalism allows to solve a set of hierarchically dependent optimization problems. Pseudo-Boolean (PB) optimization may also be used to solve this kind of problems, given that weighted MaxSAT problem instances can be translated to PB [Heras et al., 2008]. Each clause is extended with a relaxation variable that is then included in the cost function, jointly with the respective weight.

Boolean satisfiability (SAT) and PB have been extended in the past to handle preferences. For example, SAT-based planning has been extended to include conflicting preferences [Giunchiglia and Maratea, 2007], to which weights are associated, thus requiring the use of an objective function involving the preferences and their weights. In addition, al-

*The authors thank Nic Wilson and Jorge Orestes for insightful comments. This work is partially funded by European projects Mancoosi (FP7-ICT-214898) and Coconut (FP7-ICT-217069), and by FCT project Bsolo (PTDC/EIA/76572/2006).
algorithms for dealing with multi-objective PB problems have been developed [Lukasiewycz et al., 2007], in contrast to traditional algorithms that optimize a single linear function.

This paper is organized as follows. The next section describes the problem of package upgradeability in software systems. Section 3 introduces multilevel optimization and relates it with a variety of problems. Afterwards, specific multilevel optimization algorithms are proposed, being based on MaxSAT and PB. Experimental results show the effectiveness of the new algorithms.

2 A Practical Example

We have all been through a situation where the installation of a new piece of software turns out to be a nightmare. These kinds of problems may occur because there are constraints between the different pieces of software (called packages). Although these constraints are expected to be handled in a consistent and efficient way, current software distributions are developed by distinct individuals. This is opposed to traditional systems which have a centralized and closed development. Open systems also tend to be much more complex, and therefore some packages may become incompatible. In such circumstances, user preferences should be taken into account.

The constraints associated with each package can be defined by a tuple \((p, D, C)\), where \(p\) is the package, \(D\) are the dependencies of \(p\), and \(C\) are the conflicts of \(p\). \(D\) is a set of dependency clauses, each dependency clause being a disjunction of packages. \(C\) is a set of packages conflicting with \(p\). Previous work has applied SAT-based tools to ensure the consistency of both repositories and installations, as well as to upgrade consistently package installations. SAT-based tools have first been used to support distribution editors [Mancinelli et al., 2006]. The developed tools are automatic and ensure completeness, which makes them more reliable than ad-hoc and manual tools. Recently, Max-SAT has been applied to solve the software package installation problem from the user point of view [Argelich and Lynce, 2008]. In addition, the OPIUM tool [Tucker et al., 2007] uses PB constraints and optimizes a user provided single objective function. One modeling example could be preferring smaller packages to larger ones.

The encoding of these constraints into SAT is straightforward: for each package \(p_i\) there is a Boolean variable \(x_i\) that is assigned to true iff package \(p_i\) is installed, and clauses are either dependency clauses or conflict clauses (one clause for each pair of conflicting packages).

Example 1 Given a set of package constraints \(S = \{p_1, \{p_2, p_5 \lor p_6\}, \emptyset, \{p_2, \emptyset, \{p_3\}, \{p_3, \{p_4\}, \{p_1\}\}, \{p_4, \emptyset, \{p_5, p_6\}\}\}\), its encoded CNF instance is the following:

\[
\neg x_1 \lor x_2 \quad \neg x_3 \lor x_4 \\
\neg x_1 \lor x_3 \lor x_6 \quad \neg x_3 \lor \neg x_1 \\
\neg x_2 \lor \neg x_3 \quad \neg x_4 \lor \neg x_5 \\
\neg x_4 \lor \neg x_6 
\]

The problem described above is called software installability problem. The possibility of upgrading some of the packages (or introducing new packages) poses new challenges as existing packages may eventually be deleted. The goal of the software upgradeability problem is to find a solution that satisfies user preferences by minimizing the impact of introducing new packages in the current system, which is a reasonable assumption. Such preferences may be distinguished establishing the following hierarchy: (1) constraints on packages cannot be violated, (2) required packages should be installed, (3) packages that have been previously installed by the user should not be deleted, (4) the number of remaining packages installed (as a result of dependencies) should be minimized.

The software upgradeability problem can be naturally encoded as a weighted partial MaxSAT problem. In weighted MaxSAT, each clause is a pair \((C, w)\) where \(C\) is a CNF clause and \(w\) is its corresponding weight. In weighted partial MaxSAT, hard clauses must be satisfied, in contrast to the remaining soft clauses that should be satisfied. Hard clauses are associated with a weight that is greater than the sum of the weights of the soft clauses. A solution to the weighted partial MaxSAT problem maximizes the sum of the weights of the satisfied clauses.

The following example shows a weighted partial MaxSAT formula for the upgradeability problem.

Example 2 Given a set of package constraints \(S = \{p_1, \{p_2, p_5\}, \{p_4\}, \{p_2 \lor p_4\}, \emptyset, \{p_4, \emptyset, \{p_5, \emptyset\}\}\), the set of packages the user wants to install \(I = \{p_1\}\), and the current set of installed packages in the system \(A = \{p_2\}\), its weighted partial MaxSAT instance is the following:

\[
\neg x_3, 1 \quad (x_2, 4) \quad (\neg x_1 \lor x_2, 16) \\
(\neg x_4, 1) \quad (x_1, 8) \quad (\neg x_1 \lor x_5, 16) \\
(\neg x_5, 1) \quad (\neg x_1 \lor \neg x_4, 16) \\
(\neg x_3 \lor x_2 \lor x_4, 16) 
\]

This example uses a weight distribution that gives priority to the user preferences over all the other packages, and also gives priority to the current installation profile over the remaining packages. The minimum weight (with value 1) is assigned to clauses encoding packages being installed as a result of dependencies, whose number should be minimized. A medium weight (with value 4, resulting from the sum of the weights of the previous clauses plus 1) is assigned to clauses encoding packages currently installed in our system, in order to minimize the number of removed packages. A maximum weight (with value 8) is assigned to the packages the user wants to install. Finally, we assign a hard weight (with value 16) to clauses encoding the dependencies and conflicts.

3 Multilevel Optimization

The software upgradeability problem described in the previous section can be viewed as a special case of the more general problem of Multilevel Optimization [Colson et al., 2007]. Multilevel optimization can be traced back to the early 70s [Bracken and McGill, 1973], when researchers focused on mathematical programs with optimization problems in the constraints. Multilevel optimization represents a hierarchy of optimization problems, where the outer optimization...
problem is subject to the outcome of each of the enclosed optimization problems in order. In part motivated by the practical complexity of the multilevel optimization, most work in the recent past has addressed the special case of bilevel optimization [Colson et al., 2007]. Moreover, and for the special case of integer or Boolean variables, existing work is still preliminary [Denegre and Ralphs, 2009]. It should also be observed that the general problems of bilevel and multilevel optimization find a wide range of applications [Colson et al., 2007], examples of which can be represented with integer or Boolean variables [Marcotte et al., 2004].

One can conclude that the software upgradeability problem can be viewed as a special case of multilevel optimization, where the constraints are clauses, and the variables have Boolean domain. The least constrained (or outer) optimization problem represents the problem of minimizing the number of newly installed packages due to dependencies, whereas the inner optimization problem represents the problem of maximizing the installation of packages in the user preferences.

This paper focuses on the special case of multilevel optimization where the constraints are propositional clauses and the variables have Boolean domain. This problem will be referred to as Boolean Multilevel Optimization (BMO). For BMO, the hierarchy of optimization problems can be captured by associating suitable weights with the clauses, as illustrated for the package upgradeability problem.

More formally, consider a set of clauses \( C = C_1 \cup C_2 \cup \ldots \cup C_m \), where \( C_1, C_2, \ldots, C_m \) form a partition of \( C \). Moreover, consider the partition of \( C \) as a sequence of sets of clauses:

\[
\langle C_1, C_2, \ldots, C_m \rangle
\]

(1)

Where a weight is associated with each set of clauses:

\[
\langle w_1, w_2, \ldots, w_m \rangle
\]

(2)

As with MaxSAT, clauses \( C_m \), each with weight \( w_m \), are required to be satisfied, and so are referred to as hard clauses. The associated optimization problem is to satisfy clauses in \( C_1 \cup C_2 \cup \ldots \cup C_{m-1} \) such that the sum of the weights of the satisfied clauses is maximized.

Moreover, the hierarchy of optimization problems is captured by the condition:

\[
w_i > \sum_{1 \leq j < i} w_j \cdot |C_j| \quad i = 2, \ldots, m
\]

(3)

The above condition ensures that the solution to the BMO problem can be split into a sequence of optimization problems, first solving the optimization problem for the soft clauses with the largest weight (i.e. \( w_{m-1} \)), then for the next clause weight, and so on until all clause weights are considered. Building on this observation, the next section proposes dedicated algorithms for BMO.

4 Solving Boolean Multilevel Optimization

This section describes alternative solutions for BMO, in addition to the weight-based solution described earlier in the paper. The first solution is based on iteratively rescaling the weights of the MaxSAT formulation. The second formulation extends the standard encoding of weighted MaxSAT with PB constraints.

4.1 BMO with MaxSAT

Consider the BMO problem specified by equations (1)-(3). We will now explain how solving a sequence \((m-1, \ldots, 1)\) of MaxSAT subproblems can significantly reduce the weights associated with the clauses.

The first MaxSAT subproblem corresponds to subproblem \(m-1\) and is defined as follows:

\[
\begin{align*}
&\wedge_{c \in C_{m-1}} (c, 1) \\
&\wedge_{c \in C_m} (\theta, |C_{m-1}| + 1) \\
&\text{Initial UB: } |C_{m-1}| + 1
\end{align*}
\]

(4)

Note that \( C_m \) corresponds to the set of hard clauses and the initial upper bound (UB) is given by the weight associated with the hard clauses. In the worst case, the problem has no solution because at least one hard clause is unsatisfied.

For each MaxSAT subproblem following in the sequence, the weights of the clauses and the initial UB are rescaled, in such a way that the computed weights can be substantially smaller than the original weights.

For each subproblem \(i\), let \( u_i \) represent the minimum sum of weights of falsified clauses in \( C_i \). In case the weights of the set of clauses in \( C_i \) is 1, \( u_i \) corresponds to the minimum number of falsified clauses in \( C_i \). Also, let \( p_j \) be the weight associated with a set of clauses \( C_j \) in a subproblem. The remaining MaxSAT subproblems can be defined as follows, with \( m - 2 \leq i \leq 1 \):

\[
\begin{align*}
&\wedge_{c \in C_i} (c, 1) \\
&\wedge_{c \in C_{i+1}} (\theta, (|C_i| + 1) \cdot p_i) \\
&\wedge_{j=i+2}^{m} \wedge_{c \in C_j} (c, (u_{j-1} + 1) \cdot p_{j-1}) \\
&\text{Initial UB: } (u_{m-1} + 1) \cdot p_{m-1}
\end{align*}
\]

(5)

Observe that the values of \( p_j \) are refined for each iteration of the algorithm, as these values depend on the value of \( u_{j-1} \) and \( p_{j-1} \) computed by previous iterations, s.t. \( p_j = (u_{j-1} + 1) \cdot p_{j-1} \) with \( p_1 = 1 \). Note that \( u_{j-1} \) can be smaller than \( |C_{j-1}| \), thus reducing the values of the computed weights.

Finally, the MaxSAT solution for the original problem is obtained as follows:

\[
\sum_{i=1}^{m-1} w_i \cdot (|C_i| - u_i)
\]

(6)

Proposition 1 The value obtained with (6), where the different \( u_i \) values are obtained by the solution of the solution of the (4) and (5) MaxSAT problems, yields the correct solution to the BMO problem.

4.2 BMO with PB

The efficacy of the rescaling method of the previous section is still bound by the weights used. Even though the rescaling method is effective at reducing the weights that need to be considered, for very large problem instances the challenge of

---

2We are considering the MinUNSAT problem, instead of the equivalent MaxSAT problem.
large clause weights can still be an issue. An alternative approach is described in this section, which eliminates the need to handle large clauses weights. This approach is based on solving the BMO problem as a sequence of PB problems.

Consider the BMO problem specified by equations (1), (2) and (3). Each set of clauses $C_i$ can be modified by adding a relaxation variable to each clause. The resulting set of relaxed clauses is $C'_j$, and the set of relaxation variables used is denoted by $Y_j$. For example, if $c_j \in C_i$, the resulting clause is $c_j, r = c_j \cup y_j$, and $y_j \in Y_j$. Solving MaxSAT by adding relaxation variables to clauses is a standard technique [Amgoud et al., 1996; Aloul et al., 2002].

The next step is to solve a sequence of PB problems. The first PB problem is defined as:

$$\min \sum_{y \in Y_{m-1}} y$$  
$$\text{s.t.} \bigwedge_{c \in C_m} c, \bigwedge_{c_r \in C'_r \cap C_{m-1}} c_r$$  

(7)

Let the optimum solution be $v_{m-1}$. $v_{m-1}$ represents the largest number of clauses with weight $w_{m-1}$ that can be satisfied, independently of the other clause weights.

Moreover, the remaining PB problems can then be defined as follows:

$$\min \sum_{y \in Y_i} y$$  
$$\text{s.t.} \bigwedge_{c \in C_m} c, \bigwedge_{j=i}^{m-1} \left( \bigwedge_{c_r \in C'_r \cap C_{m-1}} c_r \right)$$  

(8)

With $1 < i < m - 1$, and where the optimum solution is $v_i$. In this case, $v_i$ represents the largest number of clauses with weight $w_i$ that can be satisfied, taking into account that for larger weights, the number of satisfied clauses must be taken into account. The last problem to be considered corresponds to $i = 1$, for the clauses with the smallest weight.

Finally, given the definition of $v_i$, the PB-based BMO solution is obtained as follows:

$$\sum_{i=1}^{m-1} w_i \cdot v_i$$  

(9)

As can be concluded, the proposed PB-based approach can solve the BMO problem without directly manipulating any clause weights.

**Proposition 2** The value obtained with (9), where the different $v_i$ values are obtained by the solution of the (7) and (8) PB problems, yields the correct solution to the BMO problem.

## 5 Experimental Evaluation

This section describes the experimental evaluation conducted to show the effectiveness of the new algorithms described above. With this purpose, we have generated a comprehensive set of problem instances of the software upgradeability problem. In a first step, a number of off-the-shelf MaxSAT and PB solvers have been run. In a second step, these MaxSAT and PB solvers have been adapted to perform BMO approaches. In what follows we will use $\text{BMO}^{rsc}$ to denote weight rescaling BMO with MaxSAT and $\text{BMO}^{ipb}$ to denote BMO with iterative pseudo-Boolean optimization.

The problem instances of the upgradeability problem have been obtained from the Linux Debian distribution archive \(^3\), where Debian packages are daily archived. Each daily archive is a repository. Two repositories corresponding to a snapshot with a time gap of 6 months have been selected. From the first repository, the packages for a basic Debian installation have been picked, jointly with a set of other packages. From the second repository, a set of packages to be upgraded have been picked. This set of packages is a subset of the installed packages. Each problem instance is denoted as $i < x > u < y >$ where $x$ is the number of installed packages (apart from the 826 packages of the basic installation) and $y$ is the number of packages to be upgraded. In the following experiments the number of $x$ packages ranges from 0 to 4000 and the number of $y$ packages is 98. The $y$ packages correspond to the subset of packages of the basic installation that have been updated from one repository to the other.

The four MaxSAT solvers used for the evaluation are: IncWMaxSatz [Lin et al., 2008], MiniMaxSat [Heras et al., 2008], Sat4jMaxsat 4 and WMaxSat [Argelich and Manyà, 2007]. The four PB solvers used for the evaluation are: B solo [Manquinho and Marques-Silva, 2004], Minisat+ [Éen and Sörensson, 2006], PBS4 [Aloul et al., 2002] and Sat4jPB. Other solvers could have been used, even tough we believe that these ones are some of the most competitive and overall implement different techniques which affect performance differently. For each solver, a set of instances were run with the default solver and $\text{BMO}^{rsc}$ or $\text{BMO}^{ipb}$. In order to study the scalability as the number of packages to install increases, an additional number of instances has been run for the best performing solver.

The experiments were performed on an Intel Xeon 5160 server (3GHz, 1333Mhz FSB, 4MB cache) running Red Hat Enterprise Linux WS 4. JRE 1.6.0.0.07 was used for Sat4j. Each instance was given the timeout of 900 seconds.

Table 1 shows the CPU time required by MaxSAT solvers to solve a set of given problem instances. **Column Default** shows the results for the off-the-shelf solver and column **BMO^{rsc}** shows the results for the weight rescaling approach specially designed for solving BMO problems with MaxSAT. For each instance the best result is highlighted in bold.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Default</th>
<th>BMO^{rsc}</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = 98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x = 4000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = 98</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As can be concluded, the proposed PB-based approach can solve the BMO problem without directly manipulating any clause weights.

**Proposition 2** The value obtained with (9), where the different $v_i$ values are obtained by the solution of the (7) and (8) PB problems, yields the correct solution to the BMO problem.

3http://snapshot.debian.net
4http://www.sat4j.com
same instances. BMO$^{ipb}$ boosts the solvers performance, with Sat4jPB being the only exception (for some instances it improves, for some other it does not). For the remaining solvers, the improvements are significant: most of the instances aborted by the default solver are now solved with BMO$^{ipb}$. Although there is no dominating solver, in contrast to what happens with IncWMaxSat in the MaxSAT solvers, Bsolo is the only solver able to solve all the instances with BMO$^{ipb}$. Also, despite an observable trend of increasing run times as the size of the instances increase, there are a few outliers. This also contrasts with MaxSAT solvers, but it is no surprise as additional variables can degrade the solvers performance in an unpredictable way.

Finally, we have further investigated IncWMaxSat, which was the best performing solver. Figure 1 shows the scalability of the solver comparing the default performance of IncWMaxSat with its performance using BMO$^{rsc}$. (The plot includes results for additional instances, with each point corresponding to the average of 100 instances.) We should first note that the default IncWMaxSat solver is by far more competitive than any other default MaxSAT or PB solver. Its performance is not even comparable with WMaxSat, despite IncWMaxSat being an extension of WMaxSat. This is due to the features of IncWMaxSat that make it particularly suitable for these instances, namely the incremental lower bound computation and the removal of inference rules that are particularly effective for solving random instances. Nonetheless, BMO$^{rsc}$ has been able to improve its performance and to reduce the impact of the size of the instance in the performance.

### 6 Conclusions

In many practical applications, one often needs to solve a hierarchy of optimization problems, where each optimization problem is specified in terms of a sequence of nested optimization problems. Examples in AI include specific optimization problems with preferences. Another concrete example is package management in software systems, where SAT, PB and MaxSAT find increasing application. It is possible to relate these optimization problems with multilevel (or hierarchical) optimization [Bracken and McGill, 1973; Candler and Norton, 1977; Colson et al., 2007], which finds a large number of practical applications.
This paper focuses on Boolean Multilevel Optimization (BMO) and, by considering the concrete problem of package upgradability in software systems, shows that existing solutions based on either MaxSAT or PB are in general inadequate. Moreover, the paper proposes two different algorithms, one that uses MaxSAT and another that uses PB, to show that dedicated algorithms for BMO can be orders of magnitude more efficient than the best off-the-shelf MaxSAT and PB solvers.

Despite the very promising results, a number of research directions can be outlined. One is to evaluate how the proposed algorithms scale for larger problem instances. Another one is to consider other computational problems in AI that can be cast as BMO, for example in the area of preferences and in the area of SAT-based optimization.

References


