Ann Math Artif Intell
DOI 10.1007/s10472-011-9233-2

Boolean lexicographic optimization: algorithms & applications

Joao Marques-Silva · Josep Argelich · Ana Graça · Inês Lynce

Abstract Multi-Objective Combinatorial Optimization (MOCO) problems find a wide range of practical application problems, some of which involving Boolean variables and constraints. This paper develops and evaluates algorithms for solving MOCO problems, defined on Boolean domains, and where the optimality criterion is lexicographic. The proposed algorithms build on existing algorithms for either Maximum Satisfiability (MaxSAT), Pseudo-Boolean Optimization (PBO), or Integer Linear Programming (ILP). Experimental results, obtained on problem instances from haplotyping with pedigrees and software package dependencies, show that the proposed algorithms can provide significant performance gains over state of the art MaxSAT, PBO and ILP algorithms. Finally, the paper also shows that lexicographic optimization conditions are observed in the majority of the problem instances from the MaxSAT evaluations, motivating the development of dedicated algorithms that can exploit lexicographic optimization conditions in general MaxSAT problem instances.

J. Marques-Silva
CSI/CASL, University College Dublin, Dublin, Ireland
e-mail: jpms@ucd.ie

J. Marques-Silva · I. Lynce (✉)
INESC-ID/IST, Technical University of Lisbon, Lisbon, Portugal
e-mail: ines@sat.inesc-id.pt

J. Argelich
DIEI, Universitat de Lleida, Lleida, Spain
e-mail: jargelich@diei.udl.cat

A. Graça
Engineering Faculty, Portuguese Catholic University, Lisbon, Portugal
e-mail: agraca@fe.lisboa.ucp.pt

Published online: 05 May 2011

Keywords Boolean optimization · Lexicographic optimization · Maximum satisfiability · Pseudo-Boolean optimization · Haplotyping with pedigrees · Software package dependencies

Mathematics Subject Classifications (2010) 90C27 · 90C29 · 68T20

1 Introduction

Real-world optimization problems often involve multiple objectives, that can represent conflicting purposes. There has been a large body of work on solving multi-objective combinatorial optimization (MOCO) problems, see for example [22, 23, 55, 60]. MOCO problems have natural Boolean formulations in some application domains, e.g. 0–1 multiobjective knapsack problems or the problems studied in this paper, and so Boolean-based optimization solutions could be expected to represent effective alternative solutions [22, 26, 62]. This paper addresses MOCO problems where the variables are Boolean, the constraints are represented by linear inequalities (or clauses), and the optimization criterion is lexicographic. Given a sequence of cost functions, an optimization criterion is said to be lexicographic whenever there is a preference in the order in which the cost functions are optimized. There are many examples where optimization is expected to be lexicographic. For example, suppose that instead of requiring a balance between price, horsepower and fuel consumption for choosing a new car, you have made a clear hierarchy in your mind: you have a strict limit on how much you can afford, then you will not consider a car with less than 150 hp and after that the less the fuel consumption the better. Not only you establish a priority in your preferences, but also each optimization criterion is 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, and representative examples can be found in recent surveys [22, 23, 55].

This paper develops and evaluates algorithms for Boolean lexicographic optimization problems, and has four main contributions. First, the paper formalizes Boolean Lexicographic Optimization. Second, the paper shows that a significant percentage of problem instances from the Maximum Satisfiability (MaxSAT) evaluations exhibit different forms of lexicographic optimization. Third, the paper describes practical algorithms for solving Boolean Lexicographic Optimization, either based on pseudo-Boolean optimization (PBO), 0–1 Integer Linear Programming (ILP), or MaxSAT algorithms. Fourth, the paper illustrates the practical usefulness of the proposed algorithms. The experimental evaluation focuses on two concrete applications, namely haplotyping with pedigree information [30] and software package dependencies [40]. Nevertheless, the techniques proposed in this paper are general, and can be used in other contexts.

The paper is organized as follows. Section 2 overviews MaxSAT, PBO, and Lexicographic Optimization. Moreover, Section 2 overviews Boolean Multilevel Optimization (BMO), as proposed in [8], but extends BMO with less restrictive conditions, thus increasing its applicability. Section 3 shows that lexicographic optimization conditions are encoded in the majority of problem instances from recent MaxSAT evaluations [7]. Afterwards, Section 4 describes four alternative approaches for solving lexicographic optimization problems. Three of these approaches

have been studied before in restricted settings of lexicographic optimization [8]; the fourth approach is novel. Section 5 conducts a detailed experimental evaluation on hard problem instances from haplotyping with pedigree information, allowing a detailed comparison of state of the art MaxSAT, PBO and ILP solvers against the algorithms proposed in this paper. Section 5 also summarizes the results of recent software package dependencies competitions [42], a practical application where the use of BLO techniques is essential. Section 6 summarizes related work and Section 7 concludes the paper.

2 Preliminaries

This section overviews the Maximum Satisfiability (MaxSAT) problem and its variants, as well as the Pseudo-Boolean Optimization (PBO) problem. The main approaches used by state of the art solvers are briefly summarized, including recent unsatisfiability-based MaxSAT algorithms. To conclude, this section provides a brief overview of MOCO focusing on lexicographic optimization.

In the remainder of this paper, standard definitions of Boolean Satisfiability (SAT), and related areas are assumed. Extensive treatment of these topics can be found in recent references (e.g. [14, 29, 36, 47, 53, 57]). A detailed account of 0–1 ILP can be found in [58, 64].

2.1 Maximum satisfiability

Given a CNF formula $\mathcal{C}$, the Maximum Satisfiability (MaxSAT) problem consists in finding an assignment that maximizes the number of satisfied clauses. Well-known variants of the MaxSAT problem include weighted MaxSAT, partial MaxSAT and weighted partial MaxSAT [36]. The partial variants of MaxSAT distinguish between *hard* and *soft* clauses, where hard clauses must be satisfied, and the objective is to maximize the sum of the weights of satisfied soft clauses. For the weighted variants of MaxSAT, soft clauses are associated a weight, whereas for the unweighted versions, soft clauses have weight 1. All these formulations find a wide range of practical applications (e.g. [46]). The general weighted partial MaxSAT problem formulation assumes a CNF formula $\mathcal{C}$, where each clause $c \in \mathcal{C}$ is associated a weight w, and where clauses that must be satisfied have weight $w = \top$. The optimization problem is to find a truth assignment such that the sum of the weights of the satisfied clauses is maximized.

The last decade has seen a large number of alternative algorithms for MaxSAT. These can be broadly categorized as branch-and-bound with lower bounding, decomposition-based, translation to pseudo-Boolean (PB) constraints and unsatisfiability based. Branch-and-bound algorithms integrate lower bounding and inference techniques, and represent the more mature solutions, i.e., which have been studied more extensively in the past. Examples of branch-and-bound algorithms include: MaxSatz [37], IncMaxSatz [38], WMaxSatz [5], and MiniMaxSAT [34]. A well-known example of translation to PB constraints is SAT4J-MaxSAT [12]. Examples of decomposition-based solvers include Clone [52] and sr(w) [54]. A recent alternative are unsatisfiability-based algorithms, that build on the success of modern SAT solvers, and which have been shown to perform well on problem instances

from practical applications [29, 47]. In recent years, several unsatisfiability-based MaxSAT algorithms have been proposed. A first approach was outlined in [25]. This work has been extended in a number of different ways, and recent solvers include MSUnCore [45, 48–50], WBO [45], WPM1 and PM2 [3], and WPM2 [4].

2.2 Pseudo-Boolean Optimization

Pseudo-Boolean Optimization (PBO) is an extension of SAT where constraints are linear inequalities, with integer coefficients and Boolean variables. The objective in PBO is to find an assignment to problem variables such that all problem constraints are satisfied and the value of a linear objective function is optimized. The PBO normal form [10] is defined as follows:

$$\begin{array}{ll} \text{minimize} & \sum\limits_{j \in N} v_j \cdot l_j \\ \text{subject to} & \sum\limits_{j \in N} a_{ij} l_j \geq b_i, \\ & l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0,1\}, a_{ij}, b_i, v_j \in \mathbf{N}_0^+ \end{array} \tag{1}$$

Observe that *any* pseudo-Boolean formulation can be translated into a normal form [57].

Modern PBO algorithms generalize the most effective techniques used in modern SAT solvers. These include unit propagation, conflict-driven learning and conflict-directed backtracking [15, 44]. Despite a number of common techniques, there are several alternative approaches for solving PBO. The most often used approach is to conduct a linear search on the value of the objective function. In addition, the use of binary search has been suggested and evaluated in the recent past [25, 57]. SAT algorithms can be generalized to deal with pseudo-Boolean constraints [10] natively and, whenever a solution to the problem constraints is identified, a new constraint is created such that only solutions corresponding to a lower value of the objective function are allowed. The algorithm terminates when the solver cannot improve the value of the cost function. Another often used solution is based on branch-and-bound search, where lower bounding procedures to estimate the value of the objective function are used, and the upper bound is iteratively refined. Several lower bounding procedures have been proposed over the years, e.g. [18, 44]. There are also algorithms that encode pseudo-Boolean constraints into propositional clauses [9, 21, 63] and solve the problem by subsequently using a SAT solver. This approach has been proved to be very effective for several problem sets, in particular when the clause encoding is not much larger than the original pseudo-Boolean formulation.

Although MaxSAT and PBO are different formalisms, there are well-known mappings from MaxSAT to PBO and vice-versa [2, 32, 34]. The remainder of the paper uses both formalisms interchangeably. A set of clauses or constraints is denoted by $\mathcal{C}$. Without loss of generality, linear constraints are assumed to represent clauses, thus representing instances of the Binate Covering Problem [18]. For the general case where linear constraints represent PB constraints, there are well-known mappings from PB constraints to CNF formulas [21, 57, 63], which could be used if necessary.

Mappings from soft clauses to cost functions and vice-versa are also well-known [34]. For example, suppose the cost function $\min \sum_j v_j \cdot x_j$. A set of soft clauses can replace this cost function: for each x_j create a soft clause $(\bar{x}_j)$ with cost v_j. Similarly,

a set of soft clauses can be represented with a cost function. Suppose a set of soft clauses $\mathcal{C}_a$, where each clause $c_j \in \mathcal{C}_a$ is associated a weight w_j. Replace each clause c_j with $c'_j = c_j \vee \bar{s}_j$, where s_j is a relaxation variable, and create the cost function $\min \sum_j w_j \cdot s_j$.

2.3 Boolean Multilevel Optimization

Boolean Multilevel Optimization (BMO) [8] is a restriction of weighted (partial) MaxSAT, with an additional condition on the clause weights. This section presents the original definition of BMO [8], and outlines extensions to the original BMO definition in Sections 2.3.2 and 2.3.3.

2.3.1 Complete BMO

BMO is defined on a set of sets of clauses $C = C_0 \cup C_1 \cup \ldots \cup C_m$, where $\{C_0, C_1, \ldots, C_m\}$ forms a partition of C, and a weight is associated with each set of clauses: $\langle w_0 = \top, w_1, \ldots, w_m \rangle$, such that w_i is associated with each clause c in each set C_i.

C_0 represents the *hard* clauses, each with weight $w_0 = \top$. Although C_0 may be empty, it is assumed that $C_i \neq \emptyset, i = 1, \ldots, m$.

Definition 1 (Complete BMO) An instance of Weighted (Partial) Maximum Satisfiability is an instance of (complete) BMO iff the following condition holds:

$$w_i > \sum_{i+1 \leq j \leq m} w_j \cdot |C_j| \qquad i = 1, \ldots, m-1 \tag{2}$$

Example 1 Consider the following set of sets of clauses and clause weights:

$$\langle C_0 = \{c_1, c_2\}, \quad C_1 = \{c_3, c_4, c_5\}, \quad C_2 = \{c_6, c_7\}, \quad C_3 = \{c_8, c_9\} \rangle$$
$$\langle w_0 = 40 = \top, \quad w_1 = 9, \quad w_2 = 3, \quad w_3 = 1 \rangle$$

The basic complete BMO condition holds for all i, $1 \leq i \leq 2$:

$$w_2 = 3 > \sum_{3 \leq j \leq 3} w_j \cdot |C_j| = 1 \cdot 2 = 2$$

$$w_1 = 9 > \sum_{2 \leq j \leq 3} w_j \cdot |C_j| = 3 \cdot 2 + 1 \cdot 2 = 8$$

BMO can be viewed as a technique for identifying lexicographic optimization conditions in MaxSAT and PBO problem instances. The existence of lexicographic conditions allows solving the original problem instance with iterative algorithms. Hence, a more complex problem instance is solved by iteratively solving (possibly) easier problem instances. The relationship between BMO and lexicographic optimization is further highlighted in the following sections.

2.3.2 Complete BMO condition using upper bounds

There are a number of refinements that can be made to the basic complete BMO condition proposed in Section 2.3.1 and in [8]. Suppose one knows an upper bound

on the number of clauses that can be satisfied for each C_i, with $1 \leq i \leq m$. Let the upper bound be represented by $\text{UB}(C_i)$. Then, the BMO condition can be refined as follows.

Definition 2 (Complete BMO with upper bounds) An instance of Weighted (Partial) Maximum Satisfiability is an instance of (complete) BMO with upper bounds iff the following condition holds:

$$w_i > \sum_{i+1 \leq j \leq m} w_j \cdot \text{UB}(C_j) \qquad i = 1, \ldots, m-1 \tag{3}$$

Example 2 Consider the following sequences of sets of clauses, clause weights and upper bounds on the number of satisfied clauses:

$$\langle C_0 = \{c_1, c_2\}, \quad C_1 = \{c_3, c_4\}, \quad C_2 = \{c_5, c_6, c_7, c_8\}, \quad C_3 = \{c_9, c_{10}\}\rangle$$
$$\langle w_0 = 30, \quad w_1 = 12, \quad w_2 = 3, \quad w_3 = 1\rangle$$
$$\langle UB(C_1) = 2, \quad UB(C_2) = 3, \quad UB(C_3) = 2\rangle$$

The complete BMO condition (taking upper bounds into account) holds for all i, $1 \leq i \leq 2$:

$$w_2 = 3 > \sum_{3 \leq j \leq 3} w_j \cdot \text{UB}(C_j) = 1 \cdot 2 = 2$$

$$w_1 = 12 > \sum_{2 \leq j \leq 3} w_j \cdot \text{UB}(C_j) = 3 \cdot 3 + 1 \cdot 2 = 11$$

Clearly, the basic complete BMO condition using the number of clauses in each set would not hold because of weight $w_1 = 12$.

One additional straightforward optimization is that one can compute more accurate upper bounds $\text{UB}(C_j)$ by taking into account the hard clauses in C_0 for each C_j; clearly the hard clauses need to be satisfied when computing the upper bound for each C_j. Moreover, there are a number of alternative solutions for computing upper bounds on the number of satisfied clauses. One solution is to use an unsatisfiability-based MaxSAT solver to compute an upper bound on the number of satisfied clauses in a given time bound [3, 45, 48, 50]. Another solution consists of solving the problem exactly, with an existing MaxSAT solver.

2.3.3 Partial BMO condition

For some problem instances, it is not possible to validate the BMO condition for all sets of clauses. However, it may still be possible to use BMO in a more restricted form. Instead of considering all of the sets in $C - C_0$, the new condition considers only a subset of the sets in $C - C_0$, defined by a sequence of integers $\langle k_1, k_2, \ldots, k_l\rangle$, where $k_1 < k_2 < \ldots < k_l$ and $l < m, k_l < m$. The resulting BMO definition is given below.

Definition 3 (Partial BMO) An instance of Weighted (Partial) Maximum Satisfiability is an instance of partial BMO (with upper bounds) iff there exists

$\langle k_1, k_2, \ldots, k_l \rangle$, with $k_1 < k_2 < \ldots < k_l$ and $l < m, k_l < m$, such that the following condition holds:

$$w_{k_i} > \sum_{k_i+1 \leq j \leq m} w_j \cdot \mathrm{UB}(C_j) \qquad i = 1, \ldots, l \tag{4}$$

The new BMO condition (4) is only required to hold for some of the sets in $C - C_0$. For the cases where the condition holds, a dedicated BMO algorithm will need to manipulate subsets of set $C - C_0$. Using the above notation, a dedicated algorithm would first compute the optimum solution for the set of clauses $C_0, C_1, \ldots, C_{k_1}$. This optimum solution is then used to filter the set of candidates for optimum assignments, by requiring this set of sets of clauses to satisfy its optimum solution. Given the computed result, the algorithm would then analyze $C_{k_1+1}, C_{k_1+2}, \ldots, C_{k_2}$, and would take into consideration the sets of clauses $C_0, C_1, \ldots, C_{k_1}$, as well as their already computed solution. The process would be iterated, in order, for each set of sets of clauses considered in (4).

Example 3 Consider the following sequences of sets of clauses, clause weights, and upper bounds associated with each set of clauses:

$$\begin{array}{llll} \langle C_0 = \{c_1, c_2\}, & C_1 = \{c_3, c_4\}, & C_2 = \{c_5, c_6\}, & C_3 = \{c_7, c_8, c_9\} \rangle \\ \langle w_0 = 25, & w_1 = 8, & w_2 = 2, & w_3 = 1 \rangle \\ \langle UB(C_1) = 2, & UB(C_2) = 2, & UB(C_3) = 3 \rangle & \end{array}$$

The complete BMO condition fails for $i = 2$:

$$w_2 = 2 < \sum_{3 \leq j \leq 3} w_j \cdot \mathrm{UB}(C_j) = 1 \cdot 3 = 3$$

However, partial BMO can be applied by considering sequence $\langle k_1 = 1 \rangle$, such that:

$$w_1 = 8 > \sum_{2 \leq j \leq 3} w_j \cdot \mathrm{UB}(C_j) = 2 \cdot 2 + 1 \cdot 3 = 7$$

2.4 Lexicographic optimization

Multi-Objective Combinatorial Optimization (MOCO) [22, 23, 55] is a well-known area of research, with many practical applications, including Operations Research and Artificial Intelligence. Lexicographic optimization represents a specialization of MOCO, where the optimization criterion is lexicographic. Motivated by the wide range of practical applications, Lexicographic Optimization is also often referred to Preemptive Goal Programming or Lexicographic Goal Programming [55]. This section introduces Boolean Lexicographic Optimization (BLO), a restriction of lexicographic optimization, where variables are Boolean, all cost functions and

constraints are linear, and the optimization criterion is lexicographic. The notation and definitions in this section follow [22], subject to these additional constraints.

A set X of variables is assumed, with $X = \{x_1, \ldots, x_n\}$. The domain of the variables is $\mathcal{X} = \{0, 1\}^n$. A point in $\mathcal{X}$ is represented as $\mathbf{x} \in \mathcal{X}$ or $(x_1, \ldots, x_n) \in \mathcal{X}$. A set of p linear functions is assumed, all of which are defined on Boolean variables, $f_k : \{0, 1\}^n \to \mathbb{Z}, 1 \leq k \leq p$:

$$f_k(x_1, \ldots, x_n) = \sum_{1 \leq j \leq n} v_{k,j} \cdot l_j \tag{5}$$

where $l_j \in \{x_j, \bar{x}_j\}$, and $v_{k,j} \in \mathbb{N}_0^+$. The p cost functions capturing the optimization problem represent a multi-dimensional function: $\mathbf{f} : \{0, 1\}^n \to \mathbb{Z}^p$, with $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_p(\mathbf{x}))$.

The optimization problem is defined on these p functions, subject to satisfying a set of constraints:

$$\begin{array}{ll} \text{lexmin} & (f_1(x_1, \ldots, x_n), \ldots, f_p(x_1, \ldots, x_n)) \\ \text{subject to} & \sum\limits_{j \in N} a_{ij} l_j \geq b_i, \\ & l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i \in \mathbb{N}_0^+ \end{array} \tag{6}$$

Any point $\mathbf{x} \in \{0, 1\}^n$ which satisfies the constraints is called a *feasible* point.

For any two vectors $\mathbf{y}^1, \mathbf{y}^2 \in \mathbb{Z}^p$, with $\mathbf{y}^1 = (y_1^1, \ldots, y_p^1)$ and $\mathbf{y}^2 = (y_1^2, \ldots, y_p^2)$, the lexicographic comparison ($<_{\text{lex}}$) is defined as follows: $\mathbf{y}^1 <_{\text{lex}} \mathbf{y}^2$ iff $y_q^1 < y_q^2$, where $q = \min\{k : y_k^1 \neq y_k^2\}$. For example, $\mathbf{y}^1 = (1, 2, 3, 2) <_{\text{lex}} \mathbf{y}^2 = (1, 2, 4, 1)$, because the coordinate with the smallest index where $\mathbf{y}^1$ and $\mathbf{y}^2$ differ is the third coordinate, with $y_3^1 = 3 < y_3^2 = 4$.

Definition 4 (Lexicographic optimality) A feasible point $\hat{\mathbf{x}} \in \{0, 1\}^n$ is lexicographically optimal if there exists no other $\mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) <_{\text{lex}} \mathbf{f}(\hat{\mathbf{x}})$.

This section concludes by showing that it is possible to relate the BMO conditions with BLO. Essentially, BMO models Weighted Partial MaxSAT problems where the weights of the clauses capture a lexicographic optimization condition. Any of the BMO conditions implicitly captures a sequence of Boolean functions, such that the optimization process is to be performed in order, giving preference to the function associated with the clauses with the largest weight, then to the clauses with the second largest weight, and so on. As a result, problem instances modelling some lexicographic optimization problem can be represented as explicit instances of lexicographic optimization, or as instances of MaxSAT (or PB), where the costs used respect one of the BMO conditions.

Example 4 The optimization problem from Example 3 can be formulated as an instance of lexicographic optimization. The first step consists of adding a relaxation variable s_j to each soft clause. This can then be used for creating two cost functions,

which capture the lexicographic optimization condition. The resulting formulation is defined as follows:

$$\begin{array}{ll} \text{lexmin} & (f_1(s_3, \ldots, s_9),\ f_2(s_3, \ldots, s_9)) \\ \text{subject to} & c_1 \land c_2 \land \bigwedge_{j=3}^{9}(\neg s_j \lor c_j) \\ & s_j, x_i \in \{0, 1\},\ j \in \{3, \ldots, 9\}, i \in N \end{array} \tag{7}$$

where $f_1(s_3, \ldots, s_9) = -(s_3 + s_4)$, $f_2(s_3, \ldots, s_9) = -2 \times (s_5 + s_6) - (s_7 + s_8 + s_9)$, and all clauses are defined over the x_i variables.

3 Boolean lexicographic optimization in practice

This section overviews the use of Boolean lexicographic optimization in practical applications. One example is the installation of software packages. Initial models represented multiple criteria with BMO [8]. More recently [42, 43], the existence of multiple criteria in software package installation is explicitly represented with lexicographic optimization. More concretely, three different linear functions model different criteria, and the optimum is defined lexicographically. Another area of application is haplotyping with pedigree information [30, 31]. Two criteria are considered for haplotyping with pedigree information, namely Minimum Recombinant Haplotyping Configuration (MRHC), which minimizes the number of recombinant events within a pedigree, and the Haplotype Inference by Pure Parsimony (HIPP), that aims at finding a solution with a minimum number of distinct haplotypes within a population. Both criteria are represented with linear functions, and the optimum is defined lexicographically.

Although the previous two example applications are significant, there are many other examples. Somewhat surprisingly, many publicly available MaxSAT benchmarks can be shown to encode some form of lexicographic optimization. This observation is a direct consequence of the relationship between BLO and (partial) BMO

Table 1 2008 MaxSAT evaluation statistics

Class	#I	%BMO	%MinR	%MedR	%MaxR
Weighted/crafted/KeXu	15	100.00	100.00	100.00	100.00
Weighted/crafted/RAMSEY	48	58.33	0.11	0.50	25.00
Weighted/crafted/WMAXCUT/DIMACS_MOD	62	0.00	0.00	0.00	0.00
Weighted/crafted/WMAXCUT/RANDOM	40	0.00	0.00	0.00	0.00
Weighted/crafted/WMAXCUT/SPINGLASS	5	100.00	0.10	0.27	1.25
Weightedpartial/crafted/AUCTIONS/AUC_PATHS	88	95.45	1.49	2.50	10.53
Weightedpartial/crafted/AUCTIONS/AUC_REGIONS	84	96.43	0.46	0.72	2.97
Weightedpartial/crafted/AUCTIONS/AUC_SCH...	84	29.76	7.14	11.11	25.00
Weightedpartial/crafted/PSEUDO/factor	186	100.00	100.00	100.00	100.00
Weightedpartial/crafted/PSEUDO/miplib	16	37.50	0.78	2.56	7.69
Weightedpartial/crafted/QCP	25	0.00	0.00	0.00	0.00
Weightedpartial/crafted/WCSP/PLANNING	71	18.31	0.80	7.14	50.00
Weightedpartial/crafted/WCSP/SPOT5/DIR	21	80.95	25.00	33.33	66.67
Weightedpartial/crafted/WCSP/SPOT5/LOG	21	80.95	25.00	33.33	66.67
Weightedpartial/industrial/PROTEIN_INS	12	100.00	50.00	100.00	100.00

Table 2 2009 MaxSAT evaluation statistics

Class	#I	%BMO	%MinR	%MedR	%MaxR
Weightedpartial_crafted/KnotPipatsrisawat	191	97.91	0.06	0.14	1.27
Weightedpartial_crafted/min-enc/planning	56	16.07	0.80	7.14	33.33
Weightedpartial_crafted/min-enc/warehouses	18	100.00	0.04	0.08	5.26

outlined in the previous section. Moreover, these results suggest that lexicographic optimization is often implicitly represented when modelling optimization problems from practical applications. As the results below demonstrate, this is usually achieved with different forms of the BMO condition: either complete BMO, complete BMO with upper bounds, and partial BMO.

In order to evaluate the existence of lexicographic optimization conditions in commonly used Boolean optimization benchmarks, we evaluated the existence of one of the BMO conditions in weighted (partial) MaxSAT instances from the 2008 and 2009 evaluations [7, 33].

Table 1 summarizes the results for the (non-random) weighted and weighted partial classes of the 2008 MaxSAT Evaluation [7, 33]. `Class` represents the class of problem instances. `#I` represents the number of problem instances in a given class. `%BMO` represents the percentage of instances where one of the BMO conditions was successfully identified. `%MinR` denotes the smallest fraction of weights where the BMO condition can be applied, over the total number of distinct weights for a given problem instance, over all the problem instances of the class. For example, if `%MinR` is 25, then there exists a problem instance where the BMO condition was identified in 25% of the weights, and no other problem instance has a smaller percentage of weights where the BMO condition was identified. `%MedR` denotes the median fraction of weights where the BMO condition can be applied, over the total number of distinct weights for a given problem instance, over all the problem instances of the class. Finally, `%MaxR` denotes the largest fraction of weights where the BMO condition can be applied, over the total number of distinct weights for a given problem instance, over all the problem instances of the class. Observe that, if both `MinR` and `MaxR` are 100%, then the complete BMO condition (2) (or alternatively (3)) applies.

From the table it can be concluded that, for a universe of 778 problem instances from the (non-random) weighted and weighted partial classes of the 2008 MaxSAT Evaluation [7, 33], the existence of one of the BMO conditions was observed in 489 instances, representing close to 63% of the instances analyzed. These results were obtained by using a trivial upper bound for each set C_i, i.e. $|C_i|$. Moreover, the use of accurate upper bounds would only serve to improve these results.

Table 2 shows the results obtained for the classes of problem instances new in the 2009 MaxSAT Evaluation.[1] For the 2009 problem instances, one of the BMO conditions is observed for 214 out of a total of 265 instances, which represents more than 80% of the total number of instances. As expected, all instances from the software upgradeability problem instances respect the (complete) BMO condition.

[1] http://www.maxsat.udl.cat/09/.

Table 3 Example Mancoosi [8, 41] benchmark suites

Class	#I	%BMO	%MinR	%MaxR
MANCOOSI test	2100	100.00	100.00	100.00
MANCOOSI #3—live	2913	100.00	100.00	100.00

Finally, we analyzed problem instances from the problem of software package dependencies [8, 41]. These results are shown in Table 3. As can be observed, all problem instances exhibit BMO conditions.

We have also checked all the instances using more accurate upper bounds instead of using $|C_i|$ for each set C_i as the trivial upper bound. The solver we have used is WMaxSatz [6], and we have computed the exact upper bound for each set C_i. Table 4 shows the results obtained. Classes without improvements or classes that take more than 30 min to compute the exact upper bound of an instance are not shown in the table. We can see that in the two classes for which improvements are observed (out of a total of 18 classes), now all the instances are identified as BMO problems. Furthermore, we can identify some of the instances of the class AUC_SCHEDULING as complete BMO problems with a 100% of MaxR instead of the previous 25%.

As can be concluded, and according to existing problem instances, BMO conditions tend to occur frequently in weighted and weighted partial MaxSAT problem instances from representative application domains, including combinatorial auctions [35] (class AUC_...), optimal planning [17] (class PLANNING), observation satellite management [11] (class SPOT5), protein alignment [59] (class PROTEIN_INS), spin glass problems [19] (class SPINGLASS), and software upgradeability problems [8]. The BMO conditions also occur in some artificially generated instances, including number factorization [9] (class factor), Ramsey numbers [65] (class RAMSEY), and mixed integer programming [1] (class miplib). As can also be observed, only for few classes of instances no BMO conditions were identified: KeXu, RANDOM (MAXCUT), DIMACS_MOD and QCP [7].

The results above provide ample evidence that problem instances from a wide range of application domains exhibit BMO conditions, and so can be solved with algorithms that exploit BMO conditions, e.g. BLO algorithms. Nevertheless, for problem instances not originating from industrial settings, i.e. for crafted and random instances, existing algorithms can exploit BMO conditions to improve lower or upper bounds on the optimum solution, but are still unable to compute the optimum in most cases. The following justifications can be offered for these results (i) these instances do not exhibit the structural properties that can exploited by modern SAT solvers; and (ii) MaxSAT algorithms based on iterative SAT solving are still recent, and still being actively improved upon. The situation is totally different for problem instances originating from industrial settings. For these instances, BLO algorithms

Table 4 MaxSAT evaluation statistics using upper bounds

Class	#I	%BMO	%MinR	%MaxR
Weightedpartial/crafted/AUCTIONS/AUC_PATHS	88	100.00	1.49	10.53
Weightedpartial/crafted/AUCTIONS/AUC_SCHEDULING	84	100.00	5.26	100.00

can exploit BMO conditions to provide remarkable performance improvements. This is illustrated with two concrete applications in Section 5.

4 Algorithms for Boolean lexicographic optimization

This section describes four different algorithmic approaches for solving Boolean Lexicographic Optimization problems as defined in Section 2.4. Each algorithm uses the most suitable problem representation, i.e. either PBO or MaxSAT. As a result, Section 4.2 assumes a PBO formulation, whereas Sections 4.3 and 4.4 assume a MaxSAT formulation. Moreover, since MaxSAT problems can be mapped to PBO and vice-versa [2, 34, 45], the algorithms described in this section can be applied to either class of problems. The notation used follows earlier sections. $\mathcal{C}_i$ denotes a set of clauses or PB constraints. μ_k denotes the optimum solution for some iteration k.

4.1 Aggregated cost function

A simple solution for solving Lexicographic Optimization problems is to aggregate the different functions into a single weighted cost function [51]. In this case, any MaxSAT or PBO algorithm can be used for solving BLO problems. The aggregation is organized as follows. Let $u_k = \sum_j v_{k,j}$ denote the upper bound on the value of f_k. Then define $w_p = 1$, and $w_i = 1 + \sum_{k=i+1}^{p} w_k \cdot u_k$. The aggregated cost function becomes:

$$\min \sum_{k=1}^{p} w_k \cdot \left(\sum_{j=1}^{n} v_{k,j} \cdot l_j \right) \tag{8}$$

subject to the same constraints. Alternatively, the cost function can be represented as a set of weighted soft clauses. In this case, the problem instance can be mapped to PBO, with a unique cost function, where the weights are modified as outlined above.

Example 5 Consider the following BLO cost function:

$$\begin{aligned} \text{lexmin} \quad & (\overbrace{2x_1 + \bar{x}_2}^{f_1}, \overbrace{2x_2 + \bar{x}_3}^{f_2}, \overbrace{x_3}^{f_3}) \\ \text{s.t.} \quad & (x_1 = 1) \end{aligned}$$

The aggregated cost function for this problem is:

$$8 f_1 + 2 f_2 + f_3 = 8 \times (2x_1 + \bar{x}_2) + 2 \times (2x_2 + \bar{x}_3) + x_3,$$

and the corresponding soft clauses are $(\neg x_1, 16)$, $(x_2, 8)$, $(\neg x_2, 4)$, $(x_3, 2)$, $(\neg x_3, 1)$. Finally, the complete set of clauses is: $\{(x_1, \top), (\neg x_1, 16), (x_2, 8), (\neg x_2, 4), (x_3, 2), (\neg x_3, 1)\}$. Observe that other sets of clauses could be considered, e.g. by observing that $\bar{x}_i = 1 - x_i$. For example, another aggregated cost function would be $16 \times x_1 + 4 \times x_2 + x_3$.

The main drawback of using an aggregated cost function is that exponentially large weights need to be used in the aggregated cost function. For a MaxSAT approach, this results in large weights being associated with some of the soft clauses.

Observe that the BMO conditions described in Section 2.3 (see also [8]) allow identifying problem instances where lexicographic optimization is represented with an aggregated cost function. Despite being an apparently naive modelling solution, the data from Section 3, demonstrates that the majority of the weighted (partial) MaxSAT instances from past MaxSAT evaluations [7] respect one of the BMO conditions. These results prove that these instances are essentially using an aggregated cost function to model a naturally occurring lexicographic optimization problem.

4.2 Iterative pseudo-Boolean solving

An alternative solution for Boolean lexicographic optimization was proposed in the context of BMO [8]. A formalization of this approach is shown in Algorithm 1, and essentially represents an instantiation of the standard approach for solving lexicographic optimization problems [22]. The algorithm executes a sequence of p calls to a PBO solver, in decreasing lexicographic order. At each iteration, the solution of the PBO problem instance is recorded, and a new constraint is added to the set of constraints, requiring the cost function k to be equal to the computed optimum value. After the p iterations, the algorithm identified each of the optimum values for each of the cost functions in the lexicographically ordered cost function. One important remark is that *any* PBO or ILP solver can be used.

Algorithm 1: PBO-based BLO algorithm

Input : $f_1, f_2, \ldots, f_p, \mathcal{C}$
Output: Lexicographic Optimum Solution

for $k \leftarrow 1$ **to** p **do**
  $\mu \leftarrow \texttt{PBO}(\min f_k, \mathcal{C})$
  $\mathcal{C}_k \leftarrow (f_k = \mu)$ // Cost function k must equal μ
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_k$ // Update set of constraints
  $\mu_k \leftarrow \mu$ // Record min cost for k
return $(\mu_1, \ldots, \mu_p)$

Example 6 Consider the BLO problem from Example 5:

$$\begin{aligned} \text{lexmin} \quad & (\overbrace{2x_1 + \bar{x}_2}^{f_1}, \overbrace{2x_2 + \bar{x}_3}^{f_2}, \overbrace{x_3}^{f_3}) \\ \text{s.t.} \quad & (x_1 = 1). \end{aligned}$$

The PBO-based BLO algorithm works iteratively as follows. The first iteration finds the optimum with respect to f_1, which corresponds to solving the following PBO instance:

$$\begin{aligned} \min \quad & (2x_1 + \bar{x}_2) \\ \text{s.t.} \quad & (x_1 = 1). \end{aligned}$$

A PBO solver is used to conclude that the optimum solution is 2. The second iteration find the optimum with respect to f_2, which corresponds to solving the following PBO instance:

$$\begin{aligned} \min \quad & (2x_2 + \bar{x}_3) \\ \text{s.t.} \quad & (2x_1 + \bar{x}_2 = 2) \wedge (x_1 = 1). \end{aligned}$$

The optimal solution for this problem is 2. Finally, function f_3 is minimized,

$$\begin{aligned} &\min && (x_3) \\ &\text{s.t.} && (2x_2 + \bar{x}_3 = 2) \land (2x_1 + \bar{x}_2 = 2) \land (x_1 = 1). \end{aligned}$$

The optimal solution for this PBO problem is 1. Hence, the lexicographic optimum solution for the BLO problem is (2,2,1). Given the selected weights in Example 5, the overall cost is 21. Observe that it would also be possible to convert the soft clauses to a set of five functions (see Example 5), that would then be optimized lexicographically. In this case the solution vector would be (1, 0, 1, 0, 1), which would also represent an overall aggregated cost of 21. Finally, other lexicographic cost functions could be considered, allowing different ways of computing the optimum solution.

The main potential drawbacks of the PB-based approach are: (i) PB constraints resulting from the cost functions need to be handled natively, or at least encoded to CNF; (ii) each iteration of the algorithm yields only one new constraint on the value of the cost function k. Clearly, clause reuse could be used, but that would require a tighter integration with the underlying solver.

4.3 Iterative MaxSAT solving with weight rescaling

Another alternative approach is inspired on branch-and-bound MaxSAT algorithms, and consists of iteratively rescaling the weights of the soft clauses [8]. Algorithm 2 shows this approach. At each one of the p steps, it finds the optimum value μ_k of the current set $\mathcal{C}_k$. The function `RescaleWeights` computes the weights for the clauses taking into account the previous solutions for each one of the sets. For example, if set $\mathcal{C}_{0<i<k}$ has μ_i unsatisfied clauses, the weight for the set $\mathcal{C}_{i-1}$ can be $w_i \cdot (\mu_i + 1)$, which can be lower than $w_i \cdot (|\mathcal{C}_i| + 1)$. The function `GetMinCost` translates the optimum solution given by the MaxSAT solver, that involves all the sets of clauses up to $\mathcal{C}_k$, to the number of unsatisfied clauses of current set $\mathcal{C}_k$ associated to μ_k. The weights returned by the algorithm may affect the original weights, such that $\mu_i \leq w_i$. The same holds for the weight associated with hard clauses as it depends on the weights given to soft clauses.

Algorithm 2: MaxSAT-based BLO algorithm with weight rescaling

Input : Sets of clauses $\langle \mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_p \rangle$ with corresponding weights $\langle \top, w_1, \ldots, w_p \rangle$
Output: Lexicographic Optimum Solution

for $k \leftarrow 1$ **to** p **do**
  $\mathcal{C} \leftarrow \mathcal{C}_0 \cup \cdots \cup \mathcal{C}_k$ `// Current set of clauses`
  $\mathcal{C}_{rsc} \leftarrow$ `RescaleWeights`$(\mathcal{C}, \langle \mu_1, \ldots, \mu_k - 1 \rangle)$ `// Update clause weights`
  $o \leftarrow$ `MaxSAT`$(\mathcal{C}_{rsc})$
  $\mu_k \leftarrow$ `GetMinCost`$(\mathcal{C}_{rsc}, o, \langle \mu_1, \ldots, \mu_k - 1 \rangle)$ `// Record min cost for` k
return $(\mu_1, \ldots, \mu_p)$

Example 7 Consider again the BLO problem from Example 5, but represented as a weighted partial MaxSAT problem:

$$\mathcal{C} = \{ \underbrace{(x_1, \top)}_{C_0}, \underbrace{(\neg x_1, 16)}_{C_1}, \underbrace{(x_2, 8)}_{C_2}, \underbrace{(\neg x_2, 4)}_{C_3}, \underbrace{(x_3, 2)}_{C_4}, \underbrace{(\neg x_3, 1)}_{C_5} \}.$$

First, observe that the BMO condition holds. Hence, the iterative MaxSAT with weight rescaling algorithm can be used. At each step k, the formulae for weight rescaling are $w_k = 1$, $w_{k-1} = |C_k| + 1$ and $w_{i-1} = w_i \cdot (\mu_i + 1)$, for $1 < i < k$.

In the first step, $k = 1$, the weights are rescaled such that $w_1 = 1$ and $w_0 = 1 \times (1 + 1) = 2 = \top$, so that the set of weighted clauses is

$$\mathcal{C} = \{ \underbrace{(\neg x_1, 1)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The minimum unsatisfiable cost for set $\mathcal{C}$ is $\mu_1 = 1$.

In the second step, $k = 2$, the weights are rescaled such that $w_2 = 1$, $w_1 = 1 \times (1 + 1) = 2$ and $w_0 = 2 \times (1 + 1) = 4 = \top$, and, therefore, the set of clauses is

$$\mathcal{C} = \{ \underbrace{(x_2, 1)}_{C_2}, \underbrace{(\neg x_1, 2)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The number of unsatisfiable clauses introduced in this step is 0 and, hence, $\mu_2 = 0$.

In the third step, the weights are rescaled such that $w_0 = 1$, $w_1 = 1 \times (1 + 1) = 2$, $w_2 = 2 \times (0 + 1) = 2$ and $w_3 = 2 \times (1 + 1) = 4 = \top$ and the set of clauses is

$$\mathcal{C} = \{ \underbrace{(\neg x_2, 1)}_{C_3}, \underbrace{(x_2, 2)}_{C_2}, \underbrace{(\neg x_1, 2)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \},$$

Because clauses C_0 and C_1 cannot be satisfied simultaneously, the minimal unsatisfiable cost for this step is $\mu_3 = 1$.

According to the same procedure, in the fourth step, the set of clauses is

$$\mathcal{C} = \{ \underbrace{(x_3, 1)}_{C_4}, \underbrace{(\neg x_2, 2)}_{C_3}, \underbrace{(x_2, 4)}_{C_2}, \underbrace{(\neg x_1, 4)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}$$

and $\mu_4 = 0$. In the fifth step, the set of clauses becomes

$$\mathcal{C} = \{ \underbrace{(\neg x_3, 1)}_{C_5}, \underbrace{(x_3, 2)}_{C_4}, \underbrace{(\neg x_2, 2)}_{C_3}, \underbrace{(x_2, 4)}_{C_2}, \underbrace{(\neg x_1, 4)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}$$

and $\mu_5 = 1$.

The minimal unsatisfiable solution is given by $\sum_{i=1}^{p} w_i \cdot \mu_i = 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 0 + 1 \times 1 = 21$. Observe that, as before, this value can be converted to the actual values of each of the original cost functions (see Example 5).

Although the rescaling method is effective at reducing the weights that need to be considered, for very large problem instances the challenge of large clause weights can still be an issue. This is in contrast with iterative pseudo-Boolean solving which, for the cases corresponding to the complete BMO condition, weights are never used.

4.4 Iterative unsatisfiability-based MaxSAT solving

Our final approach for solving BLO problems is based on unsatisfiability-based MaxSAT algorithms [3, 4, 25, 45, 48–50]. A possible organization is shown in Algorithm 3.

Algorithm 3: Unsatisfiability-based MaxSAT BLO algorithm

Input : Sets of clauses $\langle \mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_p \rangle$ with corresponding weights $\langle \top, w_1, \ldots, w_p \rangle$
Output: Lexicographic Optimum Solution

```
1 C_H ← C_0                                              // Initial hard clauses
2 for k ← 1 to p do
3     C_S ← C_k                                          // Current soft clauses
4     (μ, {C_H^r, C_S^r}) ← UnsatMaxSAT({C_H, C_S})
5     μ_k ← μ                                            // Record min cost for k
6     C_H ← C_H^r ∪ C_S^r                                // Harden soft clauses
7 return (μ_1, ..., μ_p)
```

Similarly to the organization of the other algorithms, Algorithm 3 executes p iterations, and each cost function is analyzed separately, in order. At each step a (unsatisfiability-based) partial (weighted) MaxSAT solver is called on a set of hard and soft clauses. The result corresponds to the minimum unsatisfiability cost for the set of clauses C_k, given that an optimum solution is also computed for the sets of clauses having weight larger than those in C_k. In contrast with previous algorithms, the CNF formula is modified in each iteration. Clauses relaxed by the unsatisfiability-based MaxSAT algorithm are kept and become *hard* clauses for the next iterations. The hardening of soft clauses after each iteration can be justified by associating sufficiently large weights with each cost function. When analyzing cost function k, relaxing clauses associated with cost functions 1 to $k-1$ are irrelevant for computing the optimum value at iteration k. The set of clauses that become hard depends on the MaxSAT algorithm used [3, 4, 45, 48–50]. The correctness of the algorithm builds on the following: (i) unsatisfiability-based MaxSAT algorithms are correct [3, 45]; and (ii) the transformations induced by unsatisfiability-based MaxSAT algorithms are parsimonious, i.e. the number of models remains unchanged in between iterations of the MaxSAT algorithm used. This guarantees that possible solutions remain viable for subsequent iterations of the top-level algorithm. As noted above, the unsatisfiability-based MaxSAT solver used can either solve partial MaxSAT or partial weighted MaxSAT. The former applies in the case of complete BMO, whereas the latter applies in the case of partial BMO.

The unsatisfiability-based lexicographic optimization approach inherits some of the drawbacks of unsatisfiability-based MaxSAT algorithms. One example is that, if the minimum unsatisfiability cost is large, then the number of iterations may render the approach ineffective. Another drawback is that, when compared to previous algorithms, a tighter integration with the underlying MaxSAT solver is necessary.

Interestingly, a well-known drawback of unsatisfiability-based MaxSAT algorithms is addressed by the lexicographic optimization approach. Unsatisfiability-based MaxSAT algorithms iteratively refine lower bounds on the minimum unsatisfiability cost. Hence, in case the available computational resources are exceeded, the algorithm terminates without providing an approximate solution to the original problem. In contrast, the lexicographic optimization approach allows obtaining intermediate solutions, each representing an *upper bound* on the minimum

unsatisfiability cost. Each intermediate solution μ_k can assume some solution for the remaining instances, e.g. by either assuming all clauses unsatisfied or by using the computed model to obtain a better estimate. Subsequent intermediate solutions will refine this solution, but all represent upper bounds on the actual optimum solution.

Example 8 Consider once more the BLO problem from Example 5, but represented as a weighted partial MaxSAT problem:

$$\mathcal{C} = \{ \underbrace{(x_1, \top)}_{C_0}, \underbrace{(\neg x_1, 16)}_{C_1}, \underbrace{(x_2, 8)}_{C_2}, \underbrace{(\neg x_2, 4)}_{C_3}, \underbrace{(x_3, 2)}_{C_4}, \underbrace{(\neg x_3, 1)}_{C_5} \}.$$

The algorithm starts by solving the following partial MaxSAT problem:

$$\mathcal{C} = \{ \underbrace{(\neg x_1, 1)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

where $\mathcal{C}_1$ is soft and $\mathcal{C}_0$ is hard. The unsatisfiability-based MaxSAT solver returns cost 1, and the modified set of clauses:

$$\mathcal{C} = \{ \underbrace{(\neg x_1 \vee s_1, 1)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

where, s_1 is a fresh relaxation variable, that in this case is not constrained. Clause $\mathcal{C}_1$ is made hard for the second iteration, where the set of clauses becomes:

$$\mathcal{C} = \{ \underbrace{(x_2, 1)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The MaxSAT solver returns cost 0 (i.e. all clauses can be satisfied). For the third iteration, the set of clauses becomes:

$$\mathcal{C} = \{ \underbrace{(\neg x_2, 1)}_{C_3}, \underbrace{(x_2, \top)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The MaxSAT solver returns cost 1, and the modified set of clauses:

$$\mathcal{C} = \{ \underbrace{(\neg x_2 \vee s_2, 1)}_{C_3}, \underbrace{(x_2, \top)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

As before, s_2 is a fresh relaxation variable, which in this example is unconstrained. For the fourth iteration, the set of clauses becomes:

$$\mathcal{C} = \{ \underbrace{(x_3, 1)}_{C_4}, \underbrace{(\neg x_2 \vee s_2, \top)}_{C_3}, \underbrace{(x_2, \top)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The MaxSAT solver returns cost 0 (i.e. all clauses can be satisfied). For the fifth iteration, the set of clauses becomes:

$$\mathcal{C} = \{ \underbrace{(\neg x_3, 1)}_{C_5}, \underbrace{(x_3, \top)}_{C_4}, \underbrace{(\neg x_2 \vee s_2, \top)}_{C_3}, \underbrace{(x_2, \top)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

The MaxSAT solver returns cost 1, and the modified set of clauses:

$$\mathcal{C} = \{ \underbrace{(\neg x_3 \vee s_3, 1)}_{C_5}, \underbrace{(x_3, \top)}_{C_4}, \underbrace{(\neg x_2 \vee s_2, \top)}_{C_3}, \underbrace{(x_2, \top)}_{C_2}, \underbrace{(\neg x_1 \vee s_1, \top)}_{C_1}, \underbrace{(x_1, \top)}_{C_0} \}.$$

As before, s_3 is a fresh relaxation variable, that is unconstrained (as the other relaxation variables for this problem). The final MaxSAT solution is $\mu = (1, 0, 1, 0, 1)$, representing an aggregated cost of $21 = 16 \times 1 + 4 \times 1 + 1 \times 1$.

4.5 Discussion

The previous sections describe four different approaches for solving Boolean lexicographic optimization problems. Some of the main drawbacks were identified, and will be evaluated in the results section. Although the proposed algorithms return a vector of optimum cost function values, it is straightforward to obtain an aggregated result cost function, e.g. using (8). Moreover, and besides serving to solve complex lexicographic optimization problems, the proposed algorithms can provide useful information in practical settings. For example, the iterative algorithms provide partial solutions (satisfying some of the target criteria) during their execution. These partial solutions can be used to provide approximate solutions in case computing the optimum value exceeds available computation resources. Given that all algorithms analyze the cost functions in order, the approximate solutions will in general be much tighter than those provided by algorithms that refine an upper bound (e.g. Minisat+).

The proposed techniques can also be used for solving already existing problem instances. Indeed, existing problem instances may encode lexicographic optimization in the cost function or in the weighted soft clauses. This information can be exploited for developing effective solutions [8]. For example, the BMO condition essentially identifies PBO or MaxSAT problem instances where lexicographic optimization is modelled with an aggregated cost function (represented explicitly or with weighted soft clauses) [8]. In some settings, this is an often used modelling solution. Hence, the BMO condition can be interpreted as an approach to identify an aggregated cost function in an optimization problem, so that it can be solved as a lexicographic optimization problem.

Finally, the algorithms proposed in the previous sections accept cost functions *implicitly* specified by soft constraints. This provides an added degree of modelling flexibility when compared with the abstract definition of (Boolean) lexicographic optimization.

5 Experimental results

This section evaluates the application of Boolean Lexicographic Optimization in two concrete practical applications: haplotyping with pedigrees [31] and package dependencies [40, 61]. Besides these two concrete applications, the algorithms proposed in previous sections were also applied to instances from the MaxSAT evaluation analyzed in Section 3. For these instances, it was in general possible to observe that MaxSAT solvers based on iterative SAT calls achieve improved lower bounds. However, the lack of structure in these problem instances still prevents them from being solved with MaxSAT solvers based on iterative SAT calls.

Table 5 Aborted problems instances (out of 500)

BLO Solution	Solver	# Aborted	
		Default	BLO
Default solvers	CPLEX	465	**464**
Iterated PBO	Minisat+	496	**56**
	BSOLO	**456**	500
	SCIP	495	**474**
	SAT4J-PB	463	**435**
Iterated MaxSAT with rescaling	SAT4J-MaxSAT	464	**404**
	WPM1	**69**	72
	MSUnCore	**84**	85
	MiniMaxSat	500	500
Iterated Unsat-based MaxSAT	MSUnCore	84	**51**

5.1 Haplotyping with pedigrees

This section evaluates existing state of the art PBO and MaxSAT solvers, as well as the algorithms described in Section 4, on lexicographic optimization problem instances resulting from haplotyping with pedigree information [30, 31]. The problem of haplotyping with pedigrees is an example of lexicographic optimization, because there are two cost functions and preference is given to one of the cost functions.

All experimental results were obtained on 3 GHz Xeon 5160 servers, with 4 GB of RAM, and running RedHat Enterprise Linux. The CPU time limit was set to 1,000 s, and the memory limit was set to 3.5 GB. For the experimentation, a well-known commercial ILP solver as well as the best performing PBO and MaxSAT solvers of the most recent evaluations[2] were considered. As a result, the following solvers were used in the experiments: CPLEX, SCIP, Minisat+, BSOLO, MiniMaxSat, MSUnCore, WPM1, SAT4J-PB, SAT4J-MaxSAT. Other well-known MaxSAT solvers (many selected among the best performing in recent MaxSAT evaluations) were also considered. However, the large size and intrinsic hardness of the problem instances resulted in these solvers being unable to provide results for any instance. Consequently, these solvers were discarded.

The instances coming from the haplotyping with pedigree information problem can be generated with different optimizations to the core model [30, 31]. For the results presented in this section, 500 of the most difficult problem instances were selected.[3] These instances are the most difficult for the best performing solver; hence, any other instances would be easier to solve by the best performing solver. The results are organized in two parts. The first part evaluates the number of instances aborted within the CPU time and physical memory limits. The second part compares the CPU times. In all cases, the focus is on evaluating the effectiveness of the proposed algorithms for solving lexicographic optimization problems. The approach considered as default corresponds to the aggregated cost function algorithm.

Table 5 shows the number of aborted instances, i.e., instances that a given solver cannot prove the optimum within the allowed CPU time limit or memory limit. The

[2] http://www.maxsat.udl.cat/, and http://www.cril.univ-artois.fr/PB09/.

[3] Problem instances available from http://sat.inesc-id.pt/publications/amai11-lexopt/.

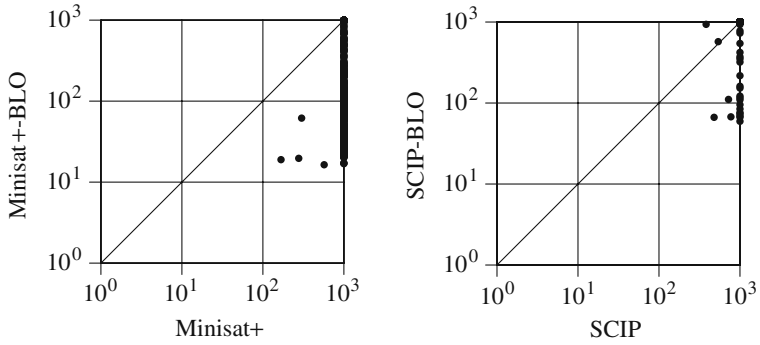

Fig. 1 Original Minisat+ and SCIP vs. iterated pseudo-Boolean solving

smallest numbers in each line are highlighted in bold. The results allow drawing several conclusions. First, for some solvers, the use of dedicated lexicographic optimization algorithms can provide remarkable performance improvements. A concrete example is Minisat+. The default solver aborts most problem instances, whereas Minisat+ integrated in an iterative pseudo-Boolean BLO solver ranks among the best performing solvers, aborting only 56 problem instances (i.e. the number of aborted instances is reduced in more than 85%). Second, for some other solvers, the performance gains are significant. This is the case with MSUnCore. For MSUnCore, the use of unsatisfiability-based lexicographic optimization reduces the number of aborted instances in close to 40%. SAT4J-PB integrated in an iterative pseudo-Boolean solver reduces the number of aborted instances in close to 6%. Similarly, SCIP integrated in an iterative pseudo-Boolean solver reduces the number of aborted instances in more than 4%. Despite the promising results of using iterative pseudo-Boolean solving, there are examples for which this approach is not effective, e.g. BSOLO. This suggests that the effectiveness of this solution depends strongly on the type of solver used and on the target problem instances. The results for the MaxSAT-based weight rescaling algorithm are less conclusive. There are several justifications for this. Given that existing branch and bound algorithms are unable to run large problem instances, the MaxSAT solvers considered are known to be less dependent on clause weights.

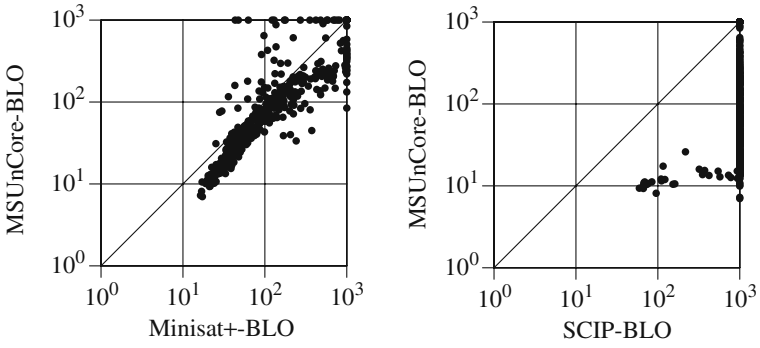

Fig. 2 Iterated unsat-based MaxSAT vs. iterated pseudo-Boolean solving

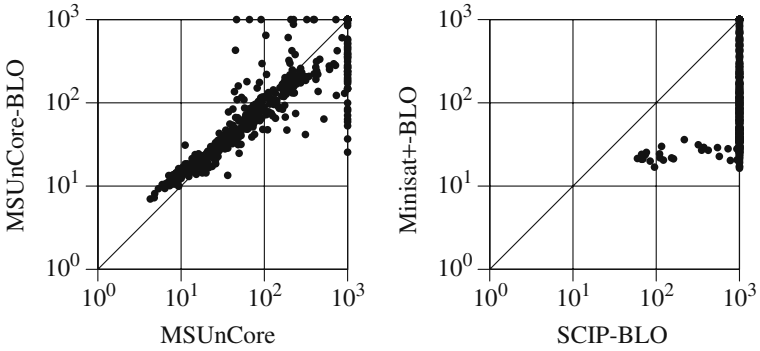

Fig. 3 Comparison of BLO solutions

Figures 1, 2, 3, 4 and 5 show scatter plots comparing the run times for different solvers on the same problem instances. Each plot compares two different approaches, where each point represents one problem instance, being the x-axis value given by one approach and the y-axis value given by the other. Again, several conclusions can be drawn. Figures 1, 2 and 3 confirm the effectiveness of the algorithms proposed in this paper, namely iterative PB solving and iterative unsatisfiability-based MaxSAT. Despite the remarkable improvements in the performance of Minisat+ when integrated in a lexicographic optimization algorithm, MSUnCore integrated in a lexicographic optimization algorithm provides the best performance in terms of aborted problem instances. Nevertheless, the use of BLO adds overhead to the solvers, and so for most instances the best performance is obtained with the default solver WPM1. These conclusions are further highlighted in Figs. 4 and 5. Although WPM1 is the best performing algorithm without lexicographic optimization support, MSUnCore with lexicographic optimization provides more robust performance, namely for the hardest problem instances.

The experimental results provide useful insights about the behavior of the solvers considered. For example, Minisat+ performs poorly when an aggregated cost function is used. However, it is among the best performing solvers when integrated in the

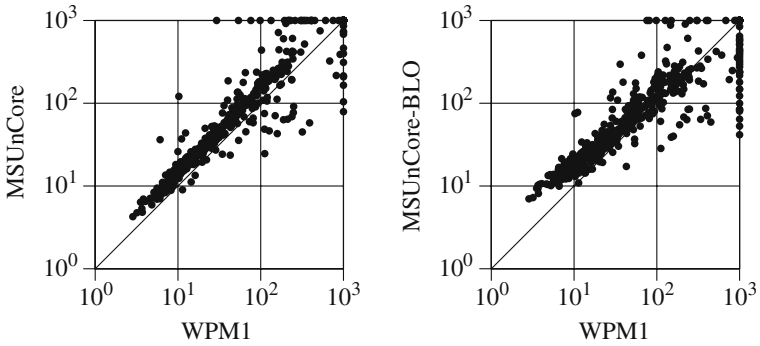

Fig. 4 BLO improvements on MSUnCore vs. WPM1

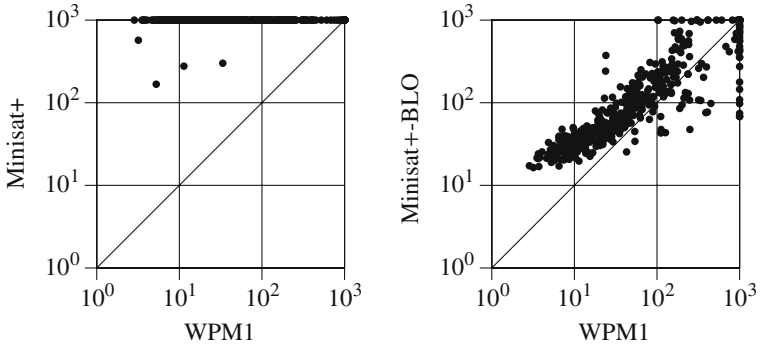

Fig. 5 BLO improvements on Minisat+ vs. WPM1

iterative pseudo-Boolean solving framework. In contrast, the improvements to SCIP are far less notable.

These performance differences are explained by the structure of the problem instances. For Minisat+, the main challenge is the relatively complex aggregated cost function. Hence, the use of iterated pseudo-Boolean solving eliminates this difficulty, and so Minisat+ is able to perform very effectively on the resulting problem instances, which exhibit hard to satisfy Boolean constraints. In contrast, SCIP is less sensitive to the cost function, and the iterative approach does not help with solving the (hard to solve) Boolean constraints. Hence, the use of iterative pseudo-Boolean solving is less effective for SCIP for the problem instances considered. The performance different between Minisat+ and BSOLO also provides relevant insights. Minisat+ and BSOLO operate in fundamentally different modes. Minisat+ conducts a linear search on the values of the cost function and BSOLO implements branch-and-bound search. The results suggest that BLO helps PBO solvers that implement a linear search of the cost function. The results for MSUnCore and iterative unsatisfiability-based MaxSAT indicate that the use of BLO solvers provides added robustness, at the cost of additional overhead for problem instances that are easy to solve.

5.2 Software package dependencies

The issue of installing, upgrading and removing software packages finds a wide range of applications, including the Eclipse ecosystem [13], and the Linux operating system [20, 40, 41], among others. This section focus on the problem of package dependencies for the Linux operating system. A number of recent competitions have been organized [42, 43], that consider a wide range of criteria when installing, removing or upgrading software packages, and that also promote the development and evaluation of new algorithmic solutions. In this section we summarize the results of the MISC #3—live competition.[4]

[4]The MISC competitions are organized by the MANCOOSI EU project. The MISC #3—live results are available from http://www.mancoosi.org/misc-live/20101126/results/.

Table 6 Summary of MISC #3—live results

Class	No BLO	BLO			
		#2,SAT4J	#2,WBO	#2,ASP	#4,MSU
Paranoid	0	1	4	3	2
Trendy	0	0	1	3	6
NRC-user track	0	0	0	0	1
CRUN-user track	0	0	1	1	2
CNRN-user track	0	0	1	1	2
Total wins (out of 29)	0	1	7	8	13
Percent wins (%)	0.00	3.45	24.14	27.59	44.83

Table 6 summarizes these results, and shows the number of wins by each type of solver for each subclass in each class of problem instances. The first column denotes each class of problem instances, which can be one of the following [42, 43]:

- The *Paranoid* track, which targets solutions that solve the user request, but also minimizes all of the following: (i) the number of packages removed in the solution; and (ii) the packages changed by the solution. The *Paranoid* class has ten subclasses of instances.
- The *Trendy* track, which targets solutions that solve the user request, but also minimizes all of the following: (i) the number of packages removed in the solution; (ii) the number of outdated packages in the solution; (iii) the number of package recommendations that are not satisfied; and finally (iv) the number of extra packages installed. The *Trendy* class has 10 subclasses of instances.
- A number of additional tracks, which solve the user request, but also aim for an optimal solution according to an optimization criterion provided by the user. The criterion is constructed from a list of utility functions each of which is taken from a fixed list of possible functions. In addition, a polarity can be specified to allow maximizing or minimizing each of the functions. The following user tracks were considered:
  - NRC-User Track: denotes `-notuptodate,-removed,-changed`.
  - CRUN-User Track: denotes `-changed,-removed,-unmet_recommends,-new`.
  - CNTN-User Track: denotes `-changed,-notuptodate,-removed,-new`.

The second column denotes the aggregated results of solutions not based on using BLO. The following columns denote solutions that implement some form of BLO, where #N denotes the number of algorithm outlined in Section 4, followed by the type of solver considered. The algorithms considered are (i) iterative Pseudo-Boolean solving (see Section 4.2), represented with #2 in the table; and (ii) iterative unsatisfiability-based MaxSAT solving (see Section 4.4), represented with #4 in the table. Example solvers include the SAT4J PBO solver [12], the clasp ASP solver [27], WBO [45] and MSUnCore (MSU) [48–50]. As can be concluded, solutions not based on dedicated algorithms for BLO perform poorly. For any of the categories in the MISC competition, the winner was a solution that implements some form of BLO. Different implementations of the iterative pseudo-Boolean solving (one of which uses ASP) win in slightly more than 50% of the categories (with native PBO

or ASP solvers being the most effective), whereas the unsatisfiability-based MaxSAT approach wins in slightly less than 50% of the categories. The results indicate that, for the sets of problem instances considered, unsatisfiability-based solvers offer the most robust performance.

6 Related work

Recent surveys on MaxSAT and PBO solvers are available in [36, 57]. Lexicographic optimization has a long history of research, with many different algorithms and applications. Examples of recent surveys are provided in [22, 23, 55]. The iterative pseudo-Boolean solving approach (see Section 4.2) is tightly related with the standard organization of lexicographic optimization algorithms [22].

Boolean Multilevel Optimization (BMO) was first proposed in [8]. The original BMO condition is referred to as *complete* BMO in this paper. Moreover, complete BMO is extended in a number of ways, allowing flexibility in the identification of conditions where BMO conditions arise, and so where dedicated algorithms can be used.

In the area of Boolean-based optimization procedures, there has been preliminary work on solving pseudo-Boolean MOCO problems [39]. Nevertheless, this work addresses exclusively Pareto optimality, and does not cover lexicographic optimization. In the area of constraints and preferences, lexicographic optimization has been the subject of recent work (for example, [24]), but the focus has been on the use of standard CSP algorithms. Recent work has proposed analyzing cost functions using their binary representation [16, 28, 56]. This can be viewed as a restricted form of BLO, where the individual Boolean functions represent the bits of the cost function representation. The BMO conditions proposed in [8] and in this paper extend this basic use of lexicographic optimization.

Some of the algorithms outlined in this paper were first proposed elsewhere, for problem instances respecting the complete BMO condition [8]. This paper extends and adapts these algorithms to all cases of BMO, and integrates them in a BLO framework. The unsatisfiability-based algorithm for BLO was first proposed in this paper.

7 Conclusions and future work

This paper formalizes the problem of Boolean lexicographic optimization (BLO), and develops algorithms for this problem. General lexicographic optimization is a well-known variant of multi-objective combinatorial optimization [22], with a large number of practical applications. The restriction considered in this paper assumes Boolean variables, linear constraints and linear cost functions. The paper formalizes Boolean Lexicographic Optimization, and demonstrates that lexicographic optimization conditions are naturally occurring, being present in the majority of problem instances from recent MaxSAT evaluations [7].

The paper outlines four different algorithmic solutions for BLO, either based on aggregating cost functions in a single cost function, iterative pseudo-Boolean solving, iterative MaxSAT with weight rescaling and iterative unsatisfiability-based MaxSAT.

The four algorithmic solutions were evaluated using complex lexicographic optimization problem instances from haplotyping with pedigree information [31]. In addition, the paper summarizes and analyzes results from recent competitions on software package dependencies [42]. The experimental evaluation allows drawing several conclusions. First, the use of a single aggregated cost function can impact performance negatively. This is demonstrated both by our own implementations for the haplotyping with pedigrees problem, but also by solvers evaluated in the software package dependencies competitions. Second, the use of iterative solutions (either based on PBO solvers or on unsatisfiability-based MaxSAT solvers) can yield significant performance gains when compared with the original solvers, resulting in remarkable reductions in the number of problem instances unsolved.

Future work will address tighter integration between default solvers and the algorithms for lexicographic optimization. Concrete examples include clause reuse and incremental interface with the default solvers. This can be done at different levels. For example, lexicographic optimization can exploit an incremental interface to the MaxSAT solver MSUnCore. Similarly, at present MSUnCore does not exploit the incremental interface of the underlying SAT solver (i.e. PicoSAT); this will change in future releases of the solvers. A side effect of an incremental interface to MaxSAT and SAT solvers is that reuse of learned clauses is automatically provided. Another area of research is the development of effective techniques for exploiting BMO conditions in existing problem instances from the MaxSAT evaluations [7].

Acknowledgements Claude Michel provided insights on Lexicographic Optimization. Mikoláš Janota developed the software package dependencies tools used in the MISC competition [42], which integrate the BLO-based MSU and PBO/WBO approaches described in this paper.

This work was partially funded by SFI PI Grant 09/IN.1/I2618, EU grants FP7-ICT-217069 and FP7-ICT-214898, FCT grant ATTEST (CMU-PT/ELE/0009/2009), FCT PhD grant SFRH/BD/28599/2006, CICYT Projects TIN2009-14704-C03-01 and TIN2010-20967-C04-03, and by INESC-ID multiannual funding from the PIDDAC program funds.

References

1. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. **34**(4), 361–372 (2006). Benchmarks available from http://miplib.zib.de
2. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Generic ILP versus specialized 0-1 ILP: an update. In: International Conference on Computer-Aided Design, pp. 450–457 (2002)
3. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 427–440 (2009)
4. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: AAAI Conference on Artificial Intelligence (2010)
5. Argelich, J., Li, C.M., Manyà, F.: An improved exact solver for partial Max-SAT. In: International Conference on Nonconvex Programming: Local and Global Approaches, pp. 230–231 (2007)
6. Argelich, J., Li, C.M., Manyà, F.: An improved exact solver for partial Max-SAT. In: Int. Conf. on Nonconvex Programming: Local & Global Approaches, pp. 230–231 (2007)
7. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT Evaluations. http://www.maxsat.udl.cat/
8. Argelich, J., Lynce, I., Marques-Silva, J.: On solving Boolean multilevel optimization problems. In: International Joint Conference on Artificial Intelligence, pp. 393–398 (2009)
9. Bailleux, O., Boufkhad, Y., Roussel, O.: A translation of pseudo Boolean constraints to SAT. JSAT **2**, 191–200 (2006)
10. Barth, P.: A Davis–Putnam enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max Plank Institute for Computer Science (1995)

11. Bensana, E., Lemaître, M., Verfaillie, G.: Earth observation satellite management. Constraints **4**(3), 293–299 (1999)
12. Berre, D.L.: SAT4J Library. http://www.sat4j.org
13. Berre, D.L., Rapicault, P.: Dependency management for the Eclipse ecosystem. In: International Workshop on Open Component Ecosystems, pp. 21–30 (2009)
14. Büning, H.K., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) SAT Handbook, pp. 735–760. IOS Press (2009)
15. Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver. In: Design Automation Conference, pp. 830–835 (2003)
16. Codish, M., Lagoon, V., Stuckey, P.J.: Logic programming with satisfiability. TPLP **8**(1), 121–128 (2008)
17. Cooper, M.C., Cussat-Blanc, S., de Roquemaurel, M., Régnier, P.: Soft arc consistency applied to optimal planning. In: International Conference on Principles and Practice of Constraint Programming, pp. 680–684 (2006)
18. Coudert, O.: On solving covering problems. In: Design Automation Conference, pp. 197–202 (1996)
19. de Simone, C., Diehl, M., Jünger, M., Mutzel, P., Reinelt, G., Rinaldi, G.: Exact ground states of Ising spin glasses: new experimental results with a branch-and-cut algorithm. J. Stat. Phys. **80**(1–2), 487–496 (1995)
20. EDOS European Project: URL: http://www.edos-project.org/bin/view/Main/
21. Een, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. JSAT **2**, 1–26 (2006)
22. Ehrgott, M.: Multicriteria Optimization. Springer (2005)
23. Ehrgott, M., Gandibleux, X.: A survey and annotated bibliography of multiobjective combinatorial optimization. OR-Spektrum **22**(4), 425–460 (2000)
24. Freuder, E., Heffernan, R., Wallace, R., Wilson, N.: Lexicographically-ordered constraint satisfaction problems. Constraints **15**(1), 1–28 (2010)
25. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 252–265 (2006)
26. Gandibleux, X., Frevilel, A.: Tabu search based procedure for solving the 0–1 multiobjective knapsack problem: the two objectives case. Journal of Heuristics **6**(3), 361–383 (2000)
27. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp*: a conflict-driven answer set solver. In: Logic Programming and Nonmonotonic Reasoning, pp. 260–265 (2007)
28. Giunchiglia, E., Maratea, M.: Planning as satisfiability with preferences. In: AAAI Conference on Artificial Intelligence, pp. 987–992 (2007)
29. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. In: Handbook of Knowledge Representation, pp. 89–134. Elsevier (2008)
30. Graça, A., Lynce, I., Marques-Silva, J., Oliveira, A.L.: Haplotype inference combining pedigrees and unrelated individuals. In: Workshop on Constraint Based Methods for Bioinformatics (2009)
31. Graça, A., Lynce, I., Marques-Silva, J., Oliveira, A.L.: Efficient and accurate haplotype inference by combining parsimony and pedigree information. In: Algebraic and Numeric Biology, LNCS, vol. 6479 (2010)
32. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. Computing **44**(4), 279–303 (1990)
33. Heras, F., Larrosa, J., de Givry, S., Schiex, T.: 2006 and 2007 Max-SAT evaluations: contributed instances. JSAT **4**(2–4), 239–250 (2008)
34. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: an efficient weighted Max-SAT solver. J. Artif. Intell. Res. **31**, 1–32 (2008)
35. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. In: ACM Conference on Electronic Commerce, pp. 66–76 (2000)
36. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: SAT Handbook, pp. 613–632. IOS Press (2009)
37. Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res. **30**, 321–359 (2007)
38. Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for MAX-SAT solving. In: International Joint Conference on Artificial Intelligence, pp. 2334–2339 (2007)
39. Lukasiewycz, M., Glaß, M., Haubelt, C., Teich, J.: Solving multi-objective pseudo-Boolean problems. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 56–69 (2007)

40. Mancinelli, F., Boender, J., Cosmo, R.D., Vouillon, J., Durak, B., Leroy, X., Treinen, R.: Managing the complexity of large free and open source package-based software distributions. In: Int. Conf. Automated Soft. Engineering, pp. 199–208 (2006)
41. Mancoosi European Project: URL: http://www.mancoosi.org/
42. Mancoosi European Project: Internal evaluation. URL: http://www.mancoosi.org/misc-live (2010)
43. Mancoosi European Project: International evaluation. URL: http://www.mancoosi.org/misc-2010 (2010)
44. Manquinho, V., Marques-Silva, J.: Satisfiability-based algorithms for Boolean optimization. Ann. Math. Artif. Intell. **40**(3–4), 353–372 (2004)
45. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted Boolean optimization. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 495–508 (2009)
46. Marques-Silva, J.: Practical applications of Boolean-based optimization: a bibliography. Technical Report 58, INESC-ID (2009)
47. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) SAT Handbook, pp. 131–154. IOS Press (2009)
48. Marques-Silva, J., Manquinho, V.: Towards more effective unsatisfiability-based maximum satisfiability algorithms. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 225–230 (2008)
49. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. Computing Research Repository, abs/0712.0097 (2007)
50. Marques-Silva, J., Planes, J.: Algorithms for maximum satisfiability using unsatisfiable cores. In: Design, Automation and Testing in Europe Conference, pp. 408–413 (2008)
51. Phillips, N.V.: A weighting function for pre-emptive multicriteria assignment problems. J. Oper. Res. Soc. **38**(9), 797–802 (1987)
52. Pipatsrisawat, K., Palyan, A., Chavira, M., Choi, A., Darwiche, A.: Solving weighted Max-SAT problems in a reduced search space: a performance analysis. Journal on Satisfiability Boolean Modeling and Computation **4**, 191–217 (2008)
53. Prestwich, S.: CNF encodings. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) SAT Handbook, pp. 75–98. IOS Press (2009)
54. Ramírez, M., Geffner, H.: Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In: International Conference on Principles and Practice of Constraint Programming, pp. 605–619 (2007)
55. Romero, C.: Extended lexicographic goal programming: a unifying approach. Omega **29**(1), 63–71 (2001)
56. Rosa, E.D., Giunchiglia, E., Maratea, M.: Solving satisfiability problems with preferences. Constraints **15**(4), 485–515 (2010)
57. Roussel, O., Manquinho, V.: Pseudo-Boolean and cardinality constraints. In: SAT Handbook, pp. 695–734. IOS Press (2009)
58. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1998)
59. Strickland, D., Barnes, E., Sokol, J.: Optimal protein structure alignment using maximum cliques. Oper. Res. **53**(3), 389–402 (2005)
60. Teghem, J.: Multi-objective combinatorial optimization. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, pp. 2437–2442. Springer (2009)
61. Tucker, C., Shuffelton, D., Jhala, R., Lerner, S.: OPIUM: optimal package install/uninstall manager. In: Int. Conf. Soft. Engineering, pp. 178–188 (2007)
62. Ulungu, E.L., Teghem, J.: Multi-objective combinatorial optimization problems: a survey. J. Multi-Criteria Decis. Anal. **3**, 83–104 (1994)
63. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. **68**(2), 63–69 (1998)
64. Wolsey, L.A., Nemhauser, G.L.: Integer and Combinatorial Optimization. Wiley (1999)
65. Yagiura, M.: Ramsey Number Problem Generator. Available from http://www-or.amp.i.kyoto-u.ac.jp/∼yagiura/sat/Ramsey/ (1998)