Inverse QUICKXPLAIN vs. MAXSAT — A Comparison in Theory and Practice

Rouven Walter[1] and **Alexander Felfernig**[2] and **Wolfgang Küchlin**[1]

Abstract. We compare the concepts of the INVQX algorithm for computing a Preferred Minimal Diagnosis vs. Partial Weighted MAXSAT in the context of Propositional Logic. In order to restore consistency of a Constraint Satisfaction Problem w.r.t. a strict total order of the user requirements, INVQX identifies a diagnosis. Partial Weighted MAXSAT aims to find a set of satisfiable clauses with the maximum total weight. It turns out that both concepts have similarities, i.e., both deliver a correction set. We point out these theoretical commonalities and prove the reducibility of both concepts to each other, i.e., both problems are $\mathrm{FP}^{\mathrm{NP}}$-complete, which was an open question. We evaluate the performance on problem instances based on real configuration data of the automotive industry from three different German car manufacturers and we compare the time and quality tradeoff.

1 Introduction

Constraint programming is successfully applied in many different areas, e.g., planning, scheduling, and configuration. Besides the usage of Constraint Satisfaction Problem (CSP) based knowledge bases, the usage of the more restrictive Propositional Logic has been successfully established in the context of automotive configuration and verification [11].

In many practical use cases the knowledge base can become over-constrained, e.g., by overly restrictive rules or user requirements. In the context of automotive configuration the knowledge base can become over-constrained, too [23]. A typical situation would be a customer configuring a car up to his wishes conflicting with the knowledge base. Another typical situation would be an engineer given the task that new features should be constructable for an existing type series by now which were not constructable before. In both situations we would like to have an automatic reasoning procedure for assistance in order to restore consistency.

One approach to restore consistency is to guide the user by computing minimal unsatisfiable cores (conflicts), which can be considered as a problem explanation. However, more than one conflict is involved in general. Another approach is to try to satisfy as many of the constraints as possible, i.e., finding a maximal satisfiable subset (MSS) or, the opposite, finding a minimum correction subset (MCS) which can be considered as a repair suggestion. The constraints of an MCS have to be removed or altered in order to restore consistency.

An MCS can be calculated in different ways: (i) MAXSAT is a generalization of the well-known SAT problem and computes the MCS of minimum cardinality; (ii) the Inverse QUICKXPLAIN (INVQX) algorithm (also denoted as FASTDIAG) [4] delivers a preferred minimal diagnosis w.r.t. a total order on the user requirements. Both approaches can be considered as an optimal repair suggestion w.r.t. their definition of optimum. In this paper, we study both approaches by giving the following contributions:

1. We point out theoretical similarities and suggest an improvement for INVQX.
2. We show that both problems, the computation of a preferred minimal diagnosis (INVQX) and the computation of an MCS of minimum cardinality (MaxSAT), are reducible to each other and that both are $\mathrm{FP}^{\mathrm{NP}}$-complete.
3. We provide experimental evaluations based on real automotive configuration data.

To the best of our knowledge, it has not been proven before, that the computation of a preferred minimal diagnosis in the context of Propositional Logic is $\mathrm{FP}^{\mathrm{NP}}$-hard.

The remainder of the paper is structured as follows: Section 2 introduces the formal background. Section 3 discusses related work. In Section 4 and Section 5 we introduce both approaches (MINUNSAT and INVQX) and give an overview of solving techniques, respectively. Section 6 points out the theoretical relationships and Section 7 shows how to reduce one problem to the other. In Section 8 we present experimental evaluations. Finally, Section 9 concludes the paper.

2 Preliminaries

Within the scope of this paper we focus on Propositional Logic over the standard operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ with constants $\bot$ and $\top$, representing false and true, respectively. For a Boolean formula φ we denote its evaluation by $\|\varphi\|_v \in \{0, 1\}$ for a variable assignment v. A Boolean formula is in CNF normal form iff it consists of a conjunction of clauses, where a clause is a disjunction of literals. A literal is a variable or its negation. A formula in CNF can be interpreted as a set of clauses and further a clause can be interpreted as a set of literals. The NP-complete SAT problem asks whether a Boolean formula is satisfiable or not.

Definition 1. *(MSS/MCS) Let φ be a set of clauses. A set $\psi \subseteq \varphi$ is a* Maximal Satisfiable Subset *(MSS) iff ψ is satisfiable and every $\psi' \subseteq \varphi$ with $\psi \subset \psi'$ is unsatisfiable.*

A set $\psi \subseteq \varphi$ is a Minimal Correction Subset *(MCS) iff $\varphi - \psi$ is satisfiable and for all $\psi' \subseteq \varphi$ with $\psi' \subset \psi$ the set difference $\varphi - \psi'$ is unsatisfiable.*

[1] Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de

[2] Institute for Software Technology, Graz University of Technology, Graz, Austria, www.felfernig.eu, email: alexander.felfernig@ist.tugraz.at

Juha Tiihonen, Andreas Falkner and Tomas Axling, Editors
Proceedings of the 17th International Configuration Workshop
September 10-11, 2015, Vienna, Austria

The definition of an MSS (resp. MCS) can be naturally extended by taking into account a set of hard clauses which have to be satisfied.

Clearly, for a given MSS (resp. MCS) ψ of φ, the complement $\varphi - \psi$ is an MCS (resp. MSS) of φ.

Analogous to [16] we introduce the following definitions:

Definition 2. *(L- and A-Preference) Let $<$ be a strict total order over a set $\varphi = \{c_1, \ldots, c_m\}$ of clauses with $c_i < c_{i+1}$ for $1 \leq i < m$, i.e., clause c_i is* preferred *to clause c_{i+1}.*

We define the lexicographical order $<_{\text{lex}}$ as follows: For two sets $\psi_1, \psi_2 \subseteq \varphi$ we say set ψ_1 is lexicographically preferred to ψ_2, denoted as $\psi_1 <_{\text{lex}} \psi_2$, iff $\exists_{1 \leq k \leq m} : c_k \in \psi_1 - \psi_2$ and $\psi_1 \cap \{c_1, \ldots, c_{k-1}\} = \psi_2 \cap \{c_1, \ldots, c_{k-1}\}$.

Furthermore, we define the anti-lexicographical order $<_{\text{antilex}}$ as follows: For two sets $\psi_1, \psi_2 \subseteq \varphi$ we say set ψ_1 is anti-lexicographically preferred to ψ_2, denoted as $\psi_1 <_{\text{antilex}} \psi_2$, iff $\exists_{1 \leq k \leq m} : c_k \in \psi_2 - \psi_1$ and $\psi_1 \cap \{c_{k+1}, \ldots, c_m\} = \psi_2 \cap \{c_{k+1}, \ldots, c_m\}$.

An MSS/MCS ψ_1 is L-preferred (resp. A-preferred) if for all MSS/MCS $\psi_2 \neq \psi_1$, $\psi_1 <_{\text{lex}} \psi_2$ (resp. $\psi_1 <_{\text{antilex}} \psi_2$).

The lexicographical order appears to be the more intuitive one. Whereas the most L-preferred set includes the most preferred clauses, the most A-preferred set excludes the most non-preferred clauses. We denote the inverse order of $<$ by $<^{-1}$.

If ψ is an L-preferred (resp. A-preferred) MSS/MCS of φ w.r.t. to the order $<$, then $\varphi - \psi$ is an A-preferred (resp. L-preferred) MSS/MCS of φ w.r.t. to the inverse order $<^{-1}$ (see Proposition 12 in [16]). Therefore, algorithms for the computation of an L-preferred MSS/MCS can also be used for the computation of the corresponding A-preferred MCS/MSS.

Note that we use the standard notation for the complexity class FP^{NP} (resp. $\text{FP}^{\text{NP}[\log n]}$), the class of function problems solvable in deterministic polynomial time using a polynomial (resp. logarithmic) number of calls to an NP oracle [19].

3 Related Work

The authors of [14] improve the computation of an MCS by newly introduced techniques, i.e., usage of backbone literals, disjoint unsatisfiable cores and satisfied clauses. Not all techniques can be applied to the computation of an A-preferred MCS, i.e., only the usage of backbone literals can be adopted. Hence the proposed enhanced version of INVQX (also denoted as FASTDIAG [4]) of the cited work can not be adopted for A-preferred MCS computation. The newly proposed MCS algorithm, called CLAUSED, exploits the fact that a falsified clause does not contain complementary literals, but it can not be adopted, either.

The INVQX algorithm (also known as FASTDIAG) [4] is based on the idea of divide-and-conquer that has been successfully exploited in the QUICKXPLAIN algorithm [9]. Whereas QUICKXPLAIN computes a preferred explanation (a minimal unsatisfiable subset (MUS) in the context of Propsitional Logic), the INVQX algorithm computes a preferred diagnosis (an MCS in the context of Propositional Logic), which can be interpreted as the *inverse* of QUICKXPLAIN.

The complexity of computing preferred sets is studied in [16]. Furthermore, the authors give an overview of established algorithms which can or can not be adopted to involve preferences.

The authors of [17] introduce improvements on computing an MCS in the context of CSP, i.e., they adopt the CLAUSED algorithm of [14]. The adopted variant is also not applicable for A-preferred MCS computation.

4 Preferred Minimal Diagnosis

The definition of a Preferred Minimal Diagnosis (PMD) used in [4] is in the context of a Constraint Satisfaction Problem (CSP). We will recap the essential definitions briefly. Let C_{KB} and C_{R} be sets of CSP constraints. The set C_{KB} (resp. C_{R}) represents the constraints of the knowledge base (resp. the user requirements).

Definition 3. *(CR Diagnosis) Let $(C_{\text{KB}}, C_{\text{R}})$ be a CR diagnosis problem. A set $\Delta \subseteq C_{\text{R}}$ is a* CR Diagnosis *if $C_{\text{KB}} \cup (C_{\text{R}} \setminus \Delta)$ is consistent. Furthermore, Δ is called* minimal *if no diagnosis $\Delta' \subset \Delta$ exists where $C_{\text{KB}} \cup (C_{\text{R}} \setminus \Delta')$ is consistent.*

In the context of Propositional Logic a CR diagnosis corresponds to the MCS problem with an additional hard part. The definitions for L- and A-Preference can analogously be defined for CSP. We will not write them out here.

Finally, we can define a preferred minimal diagnosis:

Definition 4. *(Preferred Minimal Diagnosis) Let $(C_{\text{KB}}, C_{\text{R}} = \{c_1, \ldots, c_m\})$ be a CR diagnosis problem with a strict total order s.t. $c_1 < \ldots < c_m$. A minimal diagnosis Δ is called a* Preferred Minimal Diagnosis *(PMD) if Δ is A-preferred w.r.t. the order $<^{-1}$.*

The strict total order in [4] is defined the other way round, i.e., if $c_i < c_j$ then constraint c_j is preferred to c_i. Our definition here is consistent with [16].

Remark 1. *In the context of Propositional Logic we assume, without loss of generality, that C_{KB} and C_{R} are clause sets:*

1. *C_{R}: Let s_i be a fresh variable. For each constraint $c_i \in C_{\text{R}}$ we add the clause set $\text{CNF}(s_i \rightarrow c_i)$ to the hard part and the unit clause $\{s_i\}$ replaces c_i within C_{R} (cf. [2, 7]).*
2. *C_{KB}: For any non-clausal constraint $c \in C_{\text{KB}}$ we apply the Tseitin-/Plaisted-Greenbaum Transformation [20, 21] which takes polynomial time and space. Furthermore, the constraint c and the resulting formula share the same models w.r.t. the original variables. The search space between both remains the same.*

Therefore we can interpret the problem of computing a PMD as the problem of computing an A-preferred MCS.

The computation of an A-preferred MCS is in FP^{NP} [16]. The question arises whether computing an A-preferred MCS is also FP^{NP}-hard? This is actually the case as we will show in Theorem 1.

4.1 Algorithms

A straightforward approach is Linear Search. We iterate in descending order through all constraints and check whether they conflict with the hard constraints and the previously added constraints or not. If there is a conflict, the constraint is part of the A-preferred MCS. Otherwise, the constraint will be added. The complexity of Linear Search in terms of the number of consistency checks is $\mathcal{O}(m)$, where $m = |C|$.

Algorithm 1 shows the procedure presented in [4]. The worst case complexity of INVQX in terms of the number of consistency checks is $\mathcal{O}(2d \cdot \log_2(\frac{m}{d}) + 2d)$, where d is the minimal diagnosis set size and $m = |C|$.

Remark 2. *(Exploiting inc-/decremental SAT interface) Modern SAT solvers often provide an inc-/decremental interface for adding clauses and removing them afterwards. This can be useful, e.g., when*

Algorithm 1: INVQX Algorithm

Input: $C \subseteq AC$, $AC = \{c_1, \ldots, c_t\}$
Output: Preferred minimal diagnosis Δ
if isEmpty(C) or inconsistent$(AC - C)$ **then**
  return $\emptyset$
else
  return FD$(\emptyset, C, AC)$

func FD$(D, C = \{c_1, \ldots, c_q\}, AC)$: diagnosis Δ
if $D \neq \emptyset$ and consistent(AC) **then**
  return $\emptyset$
if singleton(C) **then**
  return C
$k = \frac{q}{2}$; $C_1 = \{c_1, \ldots, c_k\}$; $C_2 = \{c_{k+1}, \ldots, c_q\}$
$D_1 = \mathrm{FD}(C_1, C_2, AC - C_1)$
$D_2 = \mathrm{FD}(D_1, C_1, AC - D_1)$
return $D_1 \cup D_2$

we have a huge formula representing the configuration model and small test instances to check against the configuration model.

Therefore we can improve Algorithm 1 by adding all constraints AC − C *first and do the consistency checks by using the inc-/decremental interface. Another improvement can be made for the second recursive call* $D_2 = \mathrm{FD}(D_1, C_1, AC - D_1)$*. All constraints in* $C_2 - D_1$ *have to be satisfied for this call. We add all constraints* $C_2 - D_1$ *before and remove them afterwards. We evaluated this improvement, see Section 8.*

5 Partial Weighted MINUNSAT

In the context of this paper, we will only focus on the Partial Weighted MINUNSAT problem. See [13] for analogous definitions of (Partial) (Weighted) MAXSAT.

Definition 5. *(*MINUNSAT*) Let* Hard *be a set of propositional clauses. Let* $\mathrm{Soft} = \{(c_1, w_1), \ldots, (c_m, w_m)\}$ *a set of tuples where* c_i *is a propositional clause and* $w_i \in \mathbb{N}_{\geq 1}$ *is a weight for all* $i = 1, \ldots, m$*. Let* $\varphi = (\mathrm{Hard}, \mathrm{Soft})$ *and* n *be the number of variables in* Hard ∪ Soft*. The* Partial Weighted Minimum Unsatisfiable Problem (MINUNSAT) *is defined as follows:*

$$\mathrm{MINUNSAT}(\varphi) := \min \left\{ \sum_{i=1}^{m} w_i (1 - \|c_i\|_v) \middle| v \in \{0,1\}^n \right\}$$

Partial Weighted MINUNSAT and Partial Weighted MAXSAT are closely connected: $\mathrm{MaxSAT}(\mathrm{Hard}, \mathrm{Soft}) = \sum_{i=1}^{m} w_i - \mathrm{MINUNSAT}(\mathrm{Hard}, \mathrm{Soft})$. A solution for one problem directly leads to a solution for the other one and vice verca. The Partial Weighted MaxSAT (resp. MINUNSAT) problem is $\mathrm{FP}^{\mathrm{NP}}$-complete [19]. The *unweighted* (partial) MAXSAT (resp. MINUNSAT) problem is $\mathrm{FP}^{\mathrm{NP}[\log n]}$-complete [10].

In the context of this paper, we will refer to Partial Weighted MINUNSAT just as MINUNSAT to simplify reading. Please note that in the literature often the name MAXSAT is used to refer to MINUNSAT. But we will use its original name to make the distinction clear.

5.1 Algorithms

Different approaches to solve the MINUNSAT problem have been developed: Branch-and-Bound for general optimization problems has been adopted to MINUNSAT, e.g., [8, 12]. In recent years, many MINUNSAT solvers make use of SAT solvers as a black box. A basic approach is to add a blocking variable to each soft clause, iteratively checking the instance for satisfiability and restricting the blocking variables further each time. Also binary search is possible this way. Nowadays solvers make usage of SAT solvers delivering an unsatisfiable core, which was first presented in [5] and extended by weights in [1]. Our list is not complete, see [18] for an overview. As an example, Algorithm 2 shows the WPM1 algorithm. If φ_c is not necessarily an MUS, the worst case complexity of WPM1 in terms of the number of consistency checks is $\mathcal{O}(d)$, where d is the minimal sum of weights of unsatisfied clauses, i.e., only costs of 1 are added in each iteration [6].

Algorithm 2: WPM1 Algorithm

Input: $(\mathrm{Hard}, \mathrm{Soft} = \{(c_1, w_1) \ldots, (c_m, w_m)\})$
Output: Sum of minimal unsatisfied weights: $cost$
if UNSAT(Hard) **then**
  return *No solution*
$cost \leftarrow 0$
while $true$ **do**
  $(st, \varphi_c) \leftarrow \mathrm{SAT}(\mathrm{Hard} \cup \{(c_i, w_i) \in \mathrm{Soft}\})$
  if $st = \mathrm{SAT}$ **then**
    return $cost$
  $\mathrm{BV} \leftarrow \emptyset$
  $w_{\min} \leftarrow \min\{w_i \mid c_i \in \varphi_c \wedge\ c_i \text{ is soft}\}$
  foreach $c_i \in \varphi_c \cap \mathrm{Soft}$ **do**
    $b_i \leftarrow$ fresh blocking variable
    Soft ←
    $\mathrm{Soft} - \{(c_i, w_i)\} \cup \{(c_i, w_i - w_{\min})\} \cup \{(c_i \vee b_i, w_{\min})\}$
    $\mathrm{BV} \leftarrow \mathrm{BV} \cup \{b_i\}$
  $\mathrm{Hard} \leftarrow \mathrm{Hard} \cup \mathrm{CNF}(\sum_{b \in \mathrm{BV}} b = 1)$
  $cost \leftarrow cost + w_{\min}$

6 Relationship

We want to identify and discuss similarities of the PMD problem and the MINUNSAT problem. As we have already shown in Remark 1 in the context of Propositional Logic the PMD problem can be interpreted as an A-preferred MCS problem. The hard parts of both problems are equal in terms of expressive power, since both are sets of clauses.

The interesting part is the set C_{R} with its strict total ordering $<$ and the set Soft of weighted clauses. Both can be defined as a special case of an MCS problem: The PMD problem can be interpreted as an A-preferred MCS problem (cf. Remark 1):

$$\min_{<^{-1}_{\text{antilex}}} \{S \mid S \text{ is MCS of } C_{\mathrm{R}} \text{ w.r.t. } C_{\mathrm{KB}}\}$$

Whereas the MINUNSAT problem is also an MCS with the minimum sum of weights:

$$\min_{\leq_{\sum_{c \in S} \mathrm{weight}(c)}} \left\{ S \middle| S \text{ is MCS of Soft w.r.t. Hard} \right\}$$

Where $\mathrm{weight}(c)$ denotes the weight of clause c.

Remark 3. *The same similarities hold for the corresponding opposite problems, i.e., the* MAXSAT *problem and the L-preferred MSS problem.*

6.1 Approximation of MINUNSAT

Solvers for the A-preferred MCS problem can be used to approximate the MINUNSAT problem. Let π be a permutation of the indices $1, \ldots, m$ such that the weights are sorted, i.e., if $i < j$ then $w_{\pi(i)} < w_{\pi(j)}$. We set:

$$\begin{aligned} C_{\mathrm{KB}} &:= \mathrm{Hard} \\ C_{\mathrm{R}} &:= \mathrm{Soft} \\ < &:= c_{\pi(1)}, \ldots, c_{\pi(m)} \end{aligned}$$

This transformation is actually an approximation: An A-preferred MCS will prefer to satisfy a clause with a high weight value more than to satisfy multiple clauses with low weight values. Whereas a MINUNSAT solution will minimize the sum of the weights of unsatisfied clauses in total. The transformation can be done in polynomial time: We sort the clauses w.r.t. their weights, which can be done in $\mathcal{O}(m \log m)$ where m is the number of soft clauses. We evaluated this approximation, see Section 8.

Example 1. *Consider* $\mathrm{Hard} = \emptyset$ *and* $\mathrm{Soft} = \{(x, 6), (\neg x \vee y, 5), (\neg y, 4), (\neg x, 3)\}$. *With a strict total ordering relying on the weights of the soft clauses (see above), the A-preferred MCS is* $\Delta = \{(\neg y, 4), (\neg x, 3)\}$ *resulting in costs of 7. But the* MINUNSAT *solution* $\{(x, 6)\}$ *has costs of 6.*

6.2 Approximation of A-preferred MCS

We can also use MINUNSAT to approximate the A-preferred MCS problem. The crucial question is which weights to assign to the soft clauses. We can assume C_{R} to be a set of clauses, see Remark 1. We set:

$$\begin{aligned} \mathrm{Hard} &:= C_{\mathrm{KB}} \\ \mathrm{Soft} &:= C_{\mathrm{R}} \\ \mathrm{weight}(c_i) &:= m - (i - 1) \end{aligned}$$

With this weight assignment, the most preferred clause is assigned to weight m, the next one to weight $m - 1$ and so on. The lexicographical order will be imitated somewhat but not sufficiently to be exact. The greater the distances of the weights of consecutive constraints c_i and c_{i+1} the better the approximation gets. Above we set the distance to 1. In Subsection 7.1 we will show how to determine distances that help to reach an exact reduction. The transformation can be done in polynomial time, too.

Example 2. *Consider* C_R *with* $x \vee y < \neg x < \neg y < x < z$. *The A-preferred MCS is* $\Delta = \{\neg y, x\}$, *but* $\mathrm{MINUNSAT}(\{(x \vee y, 5), (\neg x, 4), (\neg y, 3), (x, 2), (z, 1)\}) = 4$, *because* $\neg x$ *with weight 4 is less than clauses* x *and* $\neg y$ *with weights* $2 + 3 = 5$.

7 Reduction

In this section we will show how to polynomially reduce one problem to each other and finally see that both problems are $\mathrm{FP}^{\mathrm{NP}}$-complete and are therefore equally hard to solve.

7.1 From A-preferred MCS to MINUNSAT

Let (C_{KB}, C_R) be an A-preferred MCS problem with $C_R = \{c_1, \ldots, c_m\}$ and a total strict order s.t. $c_1 < \ldots < c_m$. We can assume C_{R} to be a set of clauses, see Remark 1. We can reduce the problem to a MINUNSAT problem by:

$$\begin{aligned} \mathrm{Hard} &:= C_{\mathrm{KB}} \\ \mathrm{Soft} &:= C_{\mathrm{R}} \end{aligned}$$

with weight w_i defined recursively:

$$w_i := \left(\sum_{j=i+1}^{m} w_j \right) + 1$$

With these weights assigned we achieve two important properties: (1) the ascending order of the constraints c_i and more important (2) the lexicographical ordering, because constraint c_i with weight $\left(\sum_{j=i+1}^{m} w_j\right) + 1 = w_{i+1} + \ldots + w_m + 1$ is greater than the sum of weights of all previous and less preferred constraints $c_{i+1}, \ldots, c_m$.

Each step requires polynomial time. But the downside of the above reduction is the exponential growth of the search space. It can be shown by induction that $(\sum_{j=i+1}^{m} w_j) + 1 = 2^{m-i}$. The most preferred clause has weight 2^{m-1}, so the weights are in $\mathcal{O}(2^m)$.

7.2 From MINUNSAT to A-preferred MCS

Firstly, we show how we can reduce the MINUNSAT problem easily to the A-preferred MCS problem if the weights comply with the following property:

Proposition 1. *Let* $\varphi = (\mathrm{Hard}, \mathrm{Soft})$ *be a* MINUNSAT *problem as defined in Definition 5 with* $\mathrm{Soft} = \{(c_1, w_1), \ldots, (c_m, w_m)\}$. *If there exists a permutation* π *of the indices* $\{1, \ldots, m\}$ *such that:*

$$w_{\pi(i)} > \sum_{j=i+1}^{m} w_{\pi(j)}$$

then the strict total ordering $<_\pi$ *with* $c_{\pi(1)} < \ldots < c_{\pi(m)}$ *with* $C_{\mathrm{KB}} = \mathrm{Hard}$ *and* $C_{\mathrm{R}} = \{c_{\pi(1)}, \ldots, c_{\pi(m)}\}$ *is a reduction to the A-preferred MCS problem.*

Proof. The reduction is correct since (1) it preserves the order of the soft clauses w.r.t. their weights and (2) the weights are in such a relation that for each clause $c_{\pi(i)}$ a MINUNSAT solution will try to satisfy $c_{\pi(i)}$ before satisfying all clauses $c_{\pi(i+1)}, \ldots, c_{\pi(m)}$ and so will a solution of A-preferred MCS due to the strict total clause ordering. □

We can check whether such a permutation can be found by: (1) Sorting soft clauses $c_1, \ldots, c_m$ in ascending order w.r.t. their weights for complexity $\mathcal{O}(m \log m)$, (2) Iterating through the sorted list and checking each clause c_i whether the inequality holds for complexity $\mathcal{O}(m)$.

The property of Proposition 1 is *sufficient* but *not necessary*. There are other classes of MINUNSAT instances without this property which are reducible in polynomial time, too.

Example 3. $\mathrm{Soft} = \{(x_1, m), \ldots, (x_m, 1)\}$, *i.e., a descending weight for each clause. We assume* $\mathrm{atMost}(x_1, \ldots, x_m) \subseteq \mathrm{Hard}$, *i.e., at most one soft clause is allowed to be true.* MINUNSAT *will try to satisfy the clause with the highest weight. The A-preferred MCS problem with the strict total ordering* $x_1 < \ldots < x_m$ *will be a diagnosis which contains clauses except for the most preferred one in the ordering which can be satisfied under* Hard. *Since at most one of the clauses can be true, the result is the same.*

Next we will show the newly result that actually any MINUNSAT instance is polynomially reducible to the A-preferred MCS problem, i.e., the A-preferred MCS is $\mathrm{FP}^{\mathrm{NP}}$-hard.

Theorem 1. *The A-preferred MCS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-hard.*

Proof. Consider the *Maximum Satisfying Assignment* (MSA) problem: For a Boolean formula φ over the variables $x_1, \ldots, x_n$ find a satisfying assignment with the lexicographical maximum of the word $x_1 \cdots x_n \in \{0,1\}^n$ or 0 if not satisfiable. This problem is $\mathrm{FP}^{\mathrm{NP}}$-complete as proved in [10].

We can polynomially reduce the MSA problem to the A-preferred MCS problem:

$$C_{\mathrm{KB}} := \mathrm{Tseitin}(\varphi)$$
$$C_{\mathrm{R}} := \{\{x_1\}, \ldots, \{x_n\}\}$$
$$< := x_1, \ldots, x_n$$

Since the Tseitin-transformed formula $\mathrm{Tseitin}(\varphi)$ has the same models on the set of the original variables $x_1, \ldots, x_n$ as the original formula ψ (see Remark 1), our reduction is sound. Let $\mathrm{APreMCS}(C_{\mathrm{KB}}, C_{\mathrm{R}})$ be the solution of the constructed A-preferred MCS problem w.r.t. the order $<^{-1}$. Using Proposition 12 of [16], the corresponding L-preferred MSS, which is $\varphi - \mathrm{APreMCS}(C_{\mathrm{KB}}, C_{\mathrm{R}})$ w.r.t. the order $<$, is the solution for the MSA problem. Therefore, the A-preferred MCS problem is $\mathrm{FP}^{\mathrm{NP}}$-hard. □

Corollary 1. *The A-preferred MCS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-complete.*

Proof. The A-preferred MCS problem is $\mathrm{FP}^{\mathrm{NP}}$-hard (Theorem 1) and in $\mathrm{FP}^{\mathrm{NP}}$ [16]. □

Remark 4. *With similar arguments one can prove that the L-preferred MSS problem is* $\mathrm{FP}^{\mathrm{NP}}$*-complete, too. Assuming* $\mathrm{P} \neq \mathrm{NP}$*, then* $\mathrm{FP}^{\mathrm{NP}[\log n]} \subset \mathrm{FP}^{\mathrm{NP}}$ *holds [10]. Hence* $\mathrm{FP}^{\mathrm{NP}}$*-complete problems are strictly harder than problems in* $\mathrm{FP}^{\mathrm{NP}[\log n]}$.

Theorem 1 negatively answers the open question whether computing L-preferred MSSes and A-preferred MCSes could be in $\mathrm{FP}^{\mathrm{NP}[\log n]}$ *or not stated in Remark 1 in [16].*

A practical encoding of a MINUNSAT problem instance as an A-preferred MCS problem could be to encode each soft clause as an unit clause with variable s_i (similiar to Remark 1) and to build the binary representation of the sum:

$$\sum_i w_i \cdot s_i = 2^{l+1} \cdot c_{l+1} + \ldots + 2^0 \cdot c_0$$

Where w_i is the weight of the unit soft clause $\{s_i\}$. Then, design an adder-network of this sum with the output variables $c_{l+1}, \ldots, c_0$ and set $C_{\mathrm{R}} := \{c_{l+1}, \ldots, c_0\}$. The strict total order is given by the order of the coefficients of the binary representation from the most significant bit c_{l+1} to the least significant bit c_0. Set C_{KB} is the union of the set of the hard clauses, the encoding of the soft clauses as unit clause and the encoding of the adder-network of the sum. We will not go into more detail in this work.

8 Experimental Evaluation

We evaluate both problems, the A-preferred MCS problem and the MINUNSAT problem, with real industrial datasets from the automotive domain. In the next subsections we describe the benchmark data and the considered use cases in more detail, afterwards we describe the used solver settings and finally discuss the results.

Table 1. Statistics about the considered POFs

		Rules		Families	
POF	#Var.	Qty.	Avg. #Var.	Qty.	Avg. #Var.
M1_1	996	**11,627**	5.9	**188**	6.3
M1_2	612	4,465	5.3	174	5.3
M2_1	483	495	4.3	60	8.7
M3_1_1	1,772	2,074	33.8	35	56.7
M3_1_2	1,586	1,496	4.6	35	48.9
M3_1_3	1,993	2,281	32.9	35	**61.6**
M3_2_1	**2,087**	2,430	34.0	42	57.2
M3_3_1	880	1,137	19.6	31	29.3
M3_3_2	884	1,121	**55.1**	31	29.4
M3_3_3	885	1,198	47.8	31	29.4

8.1 Benchmark Data

For our benchmarks we use test instances based on real automotive configuration data from three different major German car manufacturers. The configuration model of constructable cars is given as a product overview formula (POF) in Propositional Logic [11]. A POF is satisfiable and each satisfying assignment represents a valid configurable car. If the POF is over-constrained (unsatisfiable), then no cars are constructable. We denote a POF by `Mx_y_z`, where each `x` is a different car manufacturer, each `y` is a different type series and each `z` is a different model type. Each satisfying variable assignment is a constructable car configuration. Table 1 shows statistics about each used POF instance. Column #Var. shows the total variable number of each instance. Rules are Boolean formulas (not necessarily clauses) describing the dependencies between components. Column Qty. shows the number of rules and column Avg. #Var. shows the average number of variables of a rule. Families are groups of components where usually one element has to be chosen, e.g., one motor has to be chosen of the family of motors. But the condition of a family depends on the car manufacturer. Column Qty. shows the number of families and column Avg. #Var. shows the average number of components of a family.

We consider two use cases of re-configuration problems (see [23] for a detailed description of use cases regarding optimization and reconfiguration in the context of automotive configuration):

- **Re-Configuration of Selections:** The constraints of the POF are considered as hard constraints. We choose soft user requirements (variables) at random.
  Since not all user requirements are consistent w.r.t. the POF in general, such a random selection easily gets inconsistent. The goal is to optimize the user selections.
  By this use case we try to realistically imitate a user behavior when configuring a custom car.
- **Re-Configuration of Rules:** The constraints of the POF are considered as soft constraints. We choose user requirements (variables) at random.
  Similarly to the previous use case, such a random user selection easily leads to inconsistency. But in contrast to the previous use case, we want to optimize the POF constraints instead of the user selections.
  By this use case we try to realistically imitate, for example, an engineering situation where new hard requirements are given and the corresponding engineer wants to be guided by an optimized repair suggestion to adjust the rules.

Table 2. Results of MINUNSAT use case "Re-Configuration of Selections" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	3.19	1.33	0.51	**0.47**	0.48	t/o	4.97	0.50	0.55	**0.49**	t/o	47.96	0.51	0.62	**0.50**
M1_2	2.13	**0.04**	0.24	0.23	0.20	0.77	**0.05**	0.23	0.23	0.22	5.20	**0.08**	0.25	0.24	0.24
M2_1	0.10	0.53	0.09	**0.07**	0.09	0.11	0.75	0.08	**0.07**	0.08	0.12	1.08	0.08	**0.07**	0.08
M3_1_1	1.13	**0.53**	0.89	1.01	0.89	1.19	**0.75**	0.95	1.00	1.00	1.24	1.08	**0.91**	0.98	1.04
M3_1_2	0.12	**0.05**	0.09	0.07	0.09	0.11	**0.07**	0.08	**0.07**	0.08	0.12	**0.08**	**0.08**	**0.08**	**0.08**
M3_1_3	1.34	**0.63**	1.16	1.08	1.17	1.36	**0.96**	1.19	1.09	1.17	1.43	1.30	1.18	**1.11**	1.15
M3_2_1	1.38	**0.71**	1.13	1.03	1.16	1.39	**0.92**	1.21	1.07	1.47	1.48	1.40	1.43	**1.05**	1.24
M3_3_1	0.88	**0.46**	1.00	0.75	1.16	0.87	**0.45**	0.88	0.75	0.94	0.90	**0.61**	0.89	0.79	0.89
M3_3_2	1.59	**0.78**	1.60	1.43	1.62	1.57	**0.90**	1.41	1.46	1.45	1.69	**1.41**	1.59	1.76	1.46
M3_3_3	1.34	**0.63**	1.22	1.23	1.18	1.25	**0.93**	1.18	1.23	1.18	1.31	**0.88**	1.20	1.22	1.21

Table 3. Results of MINUNSAT use case "Re-Configuration of Rules" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	10.98	t/o	80.45	13.01	**4.97**	48.15	t/o	79.33	22.16	**8.01**	31.69	t/o	79.26	25.25	**9.32**
M1_2	3.24	35.08	7.43	1.78	**0.72**	4.24	89.95	7.36	2.47	**0.89**	5.88	64.91	7.10	3.23	**1.13**
M2_1	0.20	**0.11**	0.25	0.15	0.16	0.19	0.19	0.25	0.18	**0.15**	0.23	0.34	0.25	0.19	**0.16**
M3_1_1	1.88	13.35	9.36	1.97	**1.85**	2.17	41.86	9.49	2.30	**2.04**	**1.89**	14.00	9.13	2.42	2.18
M3_1_2	0.39	0.59	1.48	0.41	**0.27**	0.50	4.98	1.54	0.55	**0.33**	**0.33**	2.15	1.56	0.51	0.34
M3_1_3	2.11	23.69	12.95	2.25	**2.08**	2.44	59.11	12.32	2.75	**2.37**	**2.17**	32.55	12.51	2.84	2.57
M3_2_1	2.40	36.57	11.47	2.56	**2.22**	**2.37**	71.15	14.16	2.99	2.50	**2.29**	65.02	13.95	3.17	2.70
M3_3_1	**1.10**	2.07	5.14	1.24	1.22	**1.18**	10.80	5.24	1.49	1.36	**1.09**	5.26	5.18	1.44	1.36
M3_3_2	**2.04**	5.92	7.78	2.08	2.10	**2.03**	14.19	8.76	2.38	2.27	3.08	8.11	8.54	**2.30**	2.48
M3_3_3	**1.58**	4.62	6.41	1.79	1.78	**1.61**	5.54	6.72	1.89	1.83	**1.57**	6.41	6.73	2.00	1.98

Additionally, we considered three different levels of the user selections: 30%, 50% and 70%. Each level represents the percentage of components chosen from the families, where 100% are all families. By this distinction we want to compare the performance impact of less configured cars in contrast to highly configured cars. A more detailed description of these use cases can be found in [22].

Table 4. Avg. approx. quality of "Re-Config. of Selections"

Problem	30%	50%	70%
M1_1	93.39 %	93.60 %	91.78 %
M1_2	95.79 %	95.94 %	95.76 %
M2_1	100 %	99.47 %	98.79 %
M3_1_1	99.43 %	99.29 %	97.61 %
M3_1_2	100 %	100 %	97.49 %
M3_1_3	99.67 %	100 %	96.62 %
M3_2_1	100 %	98.21 %	96.43 %
M3_3_1	99.57 %	99.44 %	99.17 %
M3_3_2	100 %	97.56 %	98.85 %
M3_3_3	100 %	95.64 %	97.62 %
Avg.	98.79 %	97.92 %	97.01 %

We consider two problem categories:

- **Category I**: MINUNSAT
- **Category II**: A-preferred MCS

For MINUNSAT the weights were chosen between 1 to 10 by random and for A-preferred MCS the order was chosen by random. We assume a uniform distribution of the soft formulas to simulate the interaction of the user. For each POF, each use case and each percentage level we created 10 instances to get a reasonable distribution.

8.2 Implementation Techniques

On the MINUNSAT side we used the following solvers for our evaluation:

- **MSU4:** An unsat core-guided approach with iterative SAT calls using a reduced number of blocking variables [15].
- **WPM1**: An unsat core-guided with iterative SAT calls, see Algorithm 2 and [1]. In each iteration a new blocking variable will be added to each soft clause within the unsat core.

Whereas for the A-preferred MCS side we used:

- **LSB**: Linear search with backbone improvement plus usage of the in-/decremental SAT-Solving interface.
- **IQB**: Basic INVQX with backbone improvement plus usage of the first part of Remark 2.
- **IQBO**: Optimized INVQX with backbone improvement plus full usage of Remark 2.

Solver MSU4 is an external one due to [15]. All other solvers were implemented on top our uniform logic framework, which we use for commercial applications within the context of automotive configuration. Our SAT solver provides an inc-/decremental interface. We maintain two versions (Java and .NET) and decided to implement the solvers in C# using .NET 4.0.

Our experiments were run on the following settings: Processor: Intel Core i7-3520M, 2.90 GHz; Main memory: 8GB. All our .NET

Table 6. Results of A-preferred MCS use case "Re-Configuration of Selections" (in seconds)

	30%					50%					70%				
Problem	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO	WPM1	msu4	LSB	IQB	IQBO
M1_1	0.76	2.15	**0.41**	**0.41**	0.44	–	–	**0.42**	0.47	0.49	–	–	**0.52**	0.56	**0.52**
M1_2	0.36	0.64	**0.18**	0.20	0.21	–	–	**0.19**	0.22	**0.19**	–	–	0.26	0.23	**0.22**
M2_1	0.10	**0.05**	0.06	0.06	0.06	0.11	**0.04**	0.06	0.06	0.05	0.11	0.06	0.06	0.07	**0.05**
M3_1_1	1.20	**0.62**	0.75	0.81	0.72	1.34	0.88	0.82	0.89	**0.73**	1.18	1.10	**0.76**	0.91	0.85
M3_1_2	0.10	**0.05**	0.06	0.06	0.06	0.11	**0.06**	**0.06**	**0.06**	**0.06**	0.11	0.08	**0.06**	0.07	**0.06**
M3_1_3	1.27	**0.69**	0.86	0.94	0.86	1.48	**0.73**	0.87	1.00	0.85	1.42	1.23	0.93	1.03	**0.88**
M3_2_1	1.31	0.84	**0.76**	0.98	0.85	1.33	1.10	0.85	0.96	**0.84**	1.39	1.44	**0.93**	0.99	**0.93**
M3_3_1	0.86	0.84	0.59	0.64	**0.56**	0.88	**0.56**	0.65	0.65	0.62	0.90	0.64	**0.63**	0.67	0.67
M3_3_2	1.63	**0.83**	1.08	1.19	1.07	1.59	1.13	**1.07**	1.15	1.08	1.62	**1.08**	1.10	1.17	1.09
M3_3_3	1.28	**0.66**	0.91	0.93	0.88	1.26	**0.86**	**0.86**	0.96	0.92	1.27	1.18	**0.93**	0.98	0.94

Table 7. Results of A-preferred MCS use case "Re-Configuration of Rules" (in seconds)

	30%			50%			70%		
Problem	LSB	IQB	IQBO	LSB	IQB	IQBO	LSB	IQB	IQBO
M1_1	77.00	10.69	**3.65**	73.83	17.95	**6.14**	73.75	19.90	**7.31**
M1_2	6.40	1.54	**0.56**	6.30	1.81	**0.67**	5.96	2.12	**0.81**
M2_1	0.22	0.14	**0.11**	0.22	0.15	**0.12**	0.22	0.18	**0.14**
M3_1_1	8.41	1.88	**1.62**	8.10	2.05	**1.63**	8.35	2.12	**1.71**
M3_1_2	1.36	0.33	**0.20**	1.33	0.46	**0.24**	1.38	0.49	**0.28**
M3_1_3	11.03	1.95	**1.69**	11.04	2.34	**1.94**	11.53	2.35	**2.07**
M3_2_1	9.87	2.20	**1.79**	12.48	2.51	**2.11**	12.12	2.69	**2.14**
M3_3_1	4.24	1.19	**1.02**	4.29	1.31	**1.08**	4.46	1.34	**1.16**
M3_3_2	6.79	2.01	**1.66**	7.25	2.06	**1.79**	7.05	2.10	**1.95**
M3_3_3	5.41	1.61	**1.42**	5.65	1.68	**1.55**	5.70	1.79	**1.61**

Table 5. Avg. approx. quality of "Re-Configuration of Rules"

Problem	30%	50%	70%
M1_1	99.90 %	99.91 %	99.93 %
M1_2	99.94 %	99.94 %	99.86 %
M2_1	99.82 %	99.56 %	99.70 %
M3_1_1	100 %	99.98 %	100 %
M3_1_2	100 %	99.96 %	99.99 %
M3_1_3	99.98 %	99.99 %	99.99 %
M3_2_1	100 %	99.99 %	99.97 %
M3_3_1	99.95 %	99.89 %	99.99 %
M3_3_2	99.96 %	99.94 %	99.97 %
M3_3_3	99.97 %	99.98 %	99.95 %
Avg.	99.95 %	99.91 %	99.94 %

based algorithms run under Windows 7 while MSU4 runs under Ubuntu 12.04.

8.3 Results

For all of the following result tables, each table entry shows the average time in seconds a solver needed to solve 10 different instances of the considered use case and POF. We set a timeout (t/o) of 600 seconds. Each table is separated in three levels (30%, 50% and 70%) which is the percentage of families where user selections were made from.

8.3.1 Category I: Partial Weighted MINUNSAT

Table 2 and Table 3 show the results of both use cases, respectively. MSU4 performs well on the first use case, whereas IQBO performs most often best on the second use case.

Table 4 and Table 5 show the average approximation quality of the A-preferred MCS approaches used as MINUNSAT approximation (cf. Subsection 6.1), which turns out to be quite good. Percentage p is calculated as follows:

$$p = \frac{(\sum_{i=1}^{m} w_i) - \text{approxOpt}}{(\sum_{i=1}^{m} w_i) - \text{exactOpt}}$$

Where approxOpt is the optimum of the A-preferred MCS solver and exactOpt is the optimum of the MINUNSAT solver.

8.3.2 Category II: A-preferred MCS

Table 6 does not contain entries for M1_1 and M1_2 in the columns 50% and 70% because the encoding of the weights exceeds the

native *long* data type. The MaxSAT competition format[3] permits a top weight lower than 2^{63}. Table 7 is missing an evaluation for MINUNSAT solvers for the same reason.

For instances where an encoding is possible, both MINUNSAT solvers can keep up with the native A-preferred MCS solvers. IQBO performs significantly better than IQB for the second use case.

9 Conclusions and Future Work

In this work, we compared the problem of finding a minimal preferred diagnosis (A-preferred MCS in the context of Propositional Logic) with the problem of finding a diagnosis of minium cardinality (MINUNSAT in the context of Propositional Logic). We proved the $\mathrm{FP}^{\mathrm{NP}}$-hardness of the A-preferred MCS problem and therefore showed, that both problems are equally hard to solve and reducible to each other.

Both optimization approaches (A-preferred MCS and MINUNSAT) complement each other. For use cases which can be expressed as an A-preferred MCS problem one should use INVQX as solving engine. For the other use cases, one should use MINUNSAT.

For instances where MINUNSAT is not able to find the solution in within a reasonable time or where fast responses are needed, we can fall back to INVQX if an approximated answer is reasonable for the considered use case.

We evaluated the performance of both problems with benchmarks based on real automotive configuration data. We assumed a uniform distribution of the soft formulas to simulate the interaction of the user. These benchmarks can be improved by choosing a more realistic distribution of the soft formulas.

For an exhaustive evaluation of both problems, we need to consider more complex benchmark data, e.g. unsatisfiable instances of the SAT[4] or MaxSAT[5] competitions. Also, we need to evaluate all solvers based on the same SAT solving engine, i.e. MINISAT [3], in order to build up an uniform environment.

Another interesting evaluation could be the reduction of the MINUNSAT problem to an A-preferred MCS problem by using an adder-network encoding as described in Subsection 7.2. With such an encoding, we could use every A-preferred MCS solver, like INVQX, to solve MINUNSAT problem instances.

INVQX and MINUNSAT in their original form compute only a single diagnosis. Both approaches can be extended to compute a set of all diagnoses. We plan to investigate the relationship of these extensions and to evaluate them, too.

REFERENCES

[1] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'Solving (weighted) partial MaxSAT through satisfiability testing', in *Theory and Applications of Satisfiability Testing - SAT 2009*, ed., Oliver Kullmann, volume 5584 of *Lecture Notes in Computer Science*, 427–440, Springer Berlin Heidelberg, (2009).

[2] Josep Argelich and Felip Manyà, 'Exact Max-SAT solvers for over-constrained problems.', *Journal of Heuristics*, **12**(4–5), 375–392, (September 2006).

[3] Niklas Eén and Niklas Sörensson, 'An extensible SAT-solver', in *Theory and Applications of Satisfiability Testing—SAT 2003*, eds., Enrico Giunchiglia and Armando Tacchella, volume 2919 of *Lecture Notes in Computer Science*, 502–518, Springer Berlin Heidelberg, (2004).

[3] www.maxsat.udl.cat/13/requirements/index.html

[4] http://www.satcompetition.org/

[5] www.maxsat.udl.cat

[4] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, **26**(1), 53–62, (February 2012).

[5] Zhaohui Fu and Sharad Malik, 'On solving the partial MAX-SAT problem', in *Theory and Applications of Satisfiability Testing—SAT 2006*, eds., Armin Biere and Carla P. Gomes, volume 4121 of *Lecture Notes in Computer Science*, 252–265, Springer Berlin Heidelberg, (2006).

[6] Federico Heras, António Morgado, and João Marques-Silva, 'Core-guided binary search algorithms for maximum satisfiability', in *AAAI*, eds., Wolfram Burgard and Dan Roth, pp. 36 – 41. AAAI Press, (August 2011).

[7] Federico Heras, António Morgado, and João Marques-Silva, 'An empirical study of encodings for group MaxSAT', in *Canadian Conference on AI*, eds., Leila Kosseim and Diana Inkpen, volume 7310 of *Lecture Notes in Computer Science*, pp. 85–96. Springer, (2012).

[8] Federico Heras, António Morgado, and João Marques-Silva, 'Lower bounds and upper bounds for MaxSAT', in *Learning and Intelligent Optimization – 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, eds., Youssef Hamadi and Marc Schoenauer, volume 7219 of *Lecture Notes in Computer Science*, pp. 402–407. Springer Berlin Heidelberg, (2012).

[9] Ulrich Junker, 'QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems', in *Proceedings of the 19th National Conference on Artificial Intelligence*, pp. 167–172. AAAI Press / The MIT Press, (2004).

[10] Mark W. Krentel, 'The complexity of optimization problems', *Journal of Computer and System Sciences*, **36**(3), 490–509, (June 1988).

[11] Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *Journal of Automated Reasoning*, **24**(1–2), 145–163, (2000).

[12] Adrian Kügel, 'Improved exact solver for the weighted MAX-SAT problem', in *POS-10. Pragmatics of SAT*, ed., Daniel Le Berre, volume 8 of *EasyChair Proceedings in Computing*, pp. 15–27. EasyChair, (2012).

[13] Chu Min Li and Felip Manyà, 'MaxSAT, hard and soft constraints', in *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 19, 613–631, IOS Press, (2009).

[14] João Marques-Silva, Federico Heras, Mikoláš Janota, Alessandro Previti, and Anton Belov, 'On computing minimal correction subsets.', in *IJCAI*, ed., Francesca Rossi, pp. 615–622. IJCAI/AAAI, (2013).

[15] João Marques-Silva and Jordi Planes, 'Algorithms for maximum satisfiability using unsatisfiable cores', in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pp. 408–413. IEEE, (2008).

[16] João Marques-Silva and Alessandro Previti, 'On computing preferred MUSes and MCSes', in *Theory and Applications of Satisfiability Testing – SAT 2014*, eds., Carsten Sinz and Uwe Egly, volume 8561 of *Lecture Notes in Computer Science*, pp. 58–74. Springer International Publishing, (July 2014).

[17] Carlos Mencía and João Marques-Silva, 'Efficient relaxations of over-constrained CSPs', in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pp. 725–732. IEEE, (2014).

[18] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva, 'Iterative and core-guided MaxSAT solving: A survey and assessment.', *Constraints*, **18**(4), 478–534, (2013).

[19] Christos M. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, Massachusetts, 1994.

[20] David A. Plaisted and Steven Greenbaum, 'A structure-preserving clause form translation', *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).

[21] Grigorii S. Tseitin, 'On the complexity of derivations in the propositional calculus', *Studies in Constructive Mathematics and Mathematical Logic*, **Part II**, 115–125, (1968).

[22] Rouven Walter and Wolfgang Küchlin, 'ReMax – a MaxSAT aided product configurator', in *Proceedings of the 16th International Configuration Workshop*, eds., Alexander Felfernig, Cipriano Forza, and Albert Haag, pp. 59–66, Novi Sad, Serbia, (September 2014).

[23] Rouven Walter, Christoph Zengler, and Wolfgang Küchlin, 'Applications of MaxSAT in automotive configuration', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 21–28, Vienna, Austria, (August 2013).