# Applications of MaxSAT in Automotive Configuration

**Rouven Walter** and **Christoph Zengler** and **Wolfgang Küchlin**[*]

## Abstract

We give an introduction to possible applications of MaxSAT solvers in the area of automotive (re-)configuration. Where a SAT solver merely produces the answer "unsatisfiable" when given an inconsistent set of constraints, a MaxSAT solver computes the maximum subset which can be satisfied. Hence, a MaxSAT solver can compute repair suggestions, e.g. for non-constructible vehicle orders or for inconsistent configuration constraints. We implemented different state-of-the-art MaxSAT algorithms in a uniform setting within a logic framework. We evaluate the different algorithms on (re-)configuration benchmarks generated from problem instances of the automotive industry from our collaboration with German car manufacturer BMW.

## 1 Introduction

The well-known NP-complete SAT problem of propositional logic—is a given propositional formula satisfiable—has many practical applications; see [Marques-Silva, 2008] for an overview. Küchlin and Sinz [Küchlin and Sinz, 2000] pioneered the application of SAT solving for the verification of the configuration constraints and the bill-of-materials in the product documentation of the automotive industry on the example of Mercedes-Benz. A standard problem to be solved there is the following: Given a (sub-)set $O = \{o_1, \ldots, o_n\}$ of equipment options and a set $C = \{c_1, \ldots, c_m\}$ of configuration constraints whose variables are all options, is it possible to configure a car with the options in $O$ such that $C$ is satisfied? This gives us the SAT problem SAT($C \cup O$), where the options form unit clauses. If the answer is *true*, then the partial configuration $O$ is *valid* and can be extended to a full *valid* configuration $F$ which satisfies $C$, and $F$ can be readily obtained from the SAT solver.

For the unsatisfiable case, two main questions arise: (1) Which constraints (or clauses for a CNF formula $C$) of the input formula caused the unsatisfiability? (2) How many (and which) clauses can be maximally satisfied?

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

The first question can be answered with proof tracing techniques [Zhang and Malik, 2003; Asín *et al.*, 2010]. Here a CDCL SAT solver records a trace while solving the formula. From this trace, a resolution based proof can be deduced, which shows the clauses involved in the unsatisfiable core. An unsatisfiable core is also called conflict.

The answer to the second question can be of important practical use, too. For example, a customer may want to know a maximal *valid* subset of an invalid $O$. Similarly, the car manufacturer may want to know which maximal subset of $C$ is still satisfied by a currently invalid, but frequently desired option set. This optimization problem can be answered with *MaxSAT*, a generalization of the SAT problem (see Chapter 19 in [Biere *et al.*, 2009]). Instead of deciding the satisfiability of a propositional formula, MaxSAT computes the maximum number of satisfiable clauses in an unsatisfiable formula in CNF. The *Partial MaxSAT* variant splits the clause set into hard and soft clauses in a way that the number of satisfied soft clauses is maximized while all the hard clauses have to be satisfied. In the *weighted* variant of MaxSAT, clauses may carry an additional weight, such as the price of an option $o$.

Some modern MaxSAT algorithms use SAT solvers as sub-routines by reducing the problem to several SAT solver calls [Fu and Malik, 2006; Marques-Silva and Planes, 2008; Ansótegui *et al.*, 2009]. With this approach, we can make use of all modern techniques (such as clause learning, non-chronological backtracking, or watched literals) of state-of-the-art SAT solvers, which are not generally applicable to MaxSAT solvers.

MaxSAT can be used to answer further questions of practical use. For example: (1) After choosing components with priorities, what is the maximum sum of priorities that can be achieved for a valid configuration? (2) When considering the price of each component, how much is the minimal cost of a valid configuration?

*Reconfiguration* is of high practical relevance in the automotive industry [Manhart, 2005]. The after-sales business asks for extensions, replacements, or removal of components of a valid configuration with minimal effort. For example, when replacing the alarm system with a newer one, or when moving a vehicle from the U.S. to Europe, we would like to keep the maximal number of already installed components. One approach for reconfiguration uses answer set programming (ASP), which is a decidable fragment of first-order logic

[Friedrich *et al.*, 2011]. In this paper, we will describe a MaxSAT based approach for reconfiguration.

This paper is organized as follows. Section 2 defines the MaxSAT variants and notations. In Section 3 we give a short introduction to automotive configuration based on SAT, followed by a complete example. In Section 4 we describe our approach to use MaxSAT for automotive configuration to solve the above questions followed by detailed complete examples. Section 6 shows experimental proof-of-concept results based on different modern MaxSAT solvers. Section 8 concludes the paper.

## 2 Preliminaries: SAT and MaxSAT variants

A Boolean *assignment* $v$ is a mapping from a set of Boolean variables $X$ to $\{0, 1\}$. If a propositional formula $\varphi$ evaluates to true under an assignment $v$ (denoted as $v \models \varphi$), we call $v$ a *satisfying assignment* or *model* for $\varphi$, otherwise an *unsatisfying assignment*. The SAT problem of propositional logic is the question whether such a satisfying assignment $v$ exists for a given formula $\varphi$ or not.

A literal is a variable or its negation. A clause is a disjunction of literals. Given a propositional formula $\varphi = \bigwedge_{i=1}^{m} \psi_i$ in conjunctive normal form (CNF) over $n$ variables, where $\psi_i$ is a clause for all $1 \leq i \leq m$ and $m \in \mathbb{N}_{\geq 0}$, the solution to the Maximum Satisfiability problem (MaxSAT) is the maximal number of clauses which can be satisfied by an assignment $v$. Equation (1) shows a formal definition.

$$\mathrm{MaxSAT}(\varphi) := \max \left\{ \sum_{j=1}^{m} \|\psi_i\|_v \middle| v \in \{0,1\}^n \right\} \quad (1)$$

Where $\|\psi_i\|_v = 1$, if $v \models \psi_i$, otherwise $\|\psi_i\| = 0$.

We notice that for the corresponding MinUNSAT problem whose solution is the minimum number of unsatisfied clauses, equation (2) holds.

$$\mathrm{MaxSAT}(\varphi) + \mathrm{MinUNSAT}(\varphi) = m \quad (2)$$

Equation (2) also holds for the same resulting model. As a consequence, we only have to compute one problem to directly get the optimum and the corresponding model for both problems.

There are two extensions of the MaxSAT problem, called Weighted MaxSAT (WMaxSAT) and Partial MaxSAT (PMaxSAT). As the name suggests, in a weighted MaxSAT instance each clause $\psi_i$ has a weight $w_i \in \mathbb{N}_{\geq 0}$ (denoted by the tuple $(\psi_i, w_i)$). The Weighted MaxSAT problem then asks for the maximal sum of weights of satisfied clauses. Furthermore, in a partial MaxSAT instance, the clauses are divided into disjoint hard and soft clauses sets: $\mathrm{Hard} \,\dot\cup\, \mathrm{Soft}$. An optimal solution satisfies all hard clauses and a maximal number of soft clauses. Both extensions can be combined to Partial Weighted MaxSAT (PWMaxSAT).

The relationship of equation (2) also holds for each MaxSAT variant.

## 3 Automotive Configuration with SAT

Automotive configuration can be represented as a constraint satisfaction problem (c. f. [Astesana *et al.*, 2010]) and also as a CNF formula in propositional logic, where each satisfying assignment is called a *valid configuration* of a car. The latter approach was investigated in [Küchlin and Sinz, 2000].

We will give a simplified and short introduction into this representation: (1) Each component (option) $c$ is represented by a separate variable $x_c$; the component will be used in the final configuration assignment $v$ if and only if $v(x_c) = 1$; (2) components of a family (e.g. different steering wheels) will be restricted by cardinality constraints [Sinz, 2005; Bailleux *et al.*, 2009] to choose exactly one (or at most one, if the component is an optional feature); (3) dependencies between components are expressed as clauses (e.g. the implication $(x_a \wedge x_b) \rightarrow (x_c \vee x_d)$ means "If components $a$ and $b$ are chosen, then component $c$ or $d$ has to be chosen (or both)"; in clause form $(\neg x_a \vee \neg x_b \vee x_c \vee x_d)$).

The resulting formula in CNF is:

$$\varphi_{car} := \varphi_{cc} \wedge \varphi_{dep} \quad (3)$$

Where $\varphi_{cc}$ are the clauses of the families' cardinality constraints and $\varphi_{dep}$ are the clauses of the dependencies between the components. With this representation, we can answer the following questions using a SAT solver:

1. **Validation of a partial configuration.**
2. **Forced component:** A component, which is used in every valid configuration.
3. **Redundant component:** A component, which can never be used in any valid configuration.

### 3.1 Example: SAT based Configuration

We consider the families of components with their limitations listed in Table 1.

Table 1: Component families with limitations

| family | alternatives | limit |
|---|---|---|
| engine | $E_1, E_2, E_3$ | $= 1$ |
| gearbox | $G_1, G_2, G_3$ | $= 1$ |
| control unit | $C_1, C_2, C_3, C_4, C_5$ | $= 1$ |
| dashboard | $D_1, D_2, D_3, D_4$ | $= 1$ |
| navigation system | $N_1, N_2, N_3$ | $\leq 1$ |
| air conditioner | $AC_1, AC_2, AC_3$ | $\leq 1$ |
| alarm system | $AS_1, AS_2$ | $\leq 1$ |
| radio | $R_1, R_2, R_3, R_4, R_5$ | $\leq 1$ |

Furthermore, we consider the dependencies between the components listed in Table 2.

Table 2: Component dependencies

| premise | conclusion |
|---|---|
| $G_1$ | $E_1 \vee E_2$ |
| $N_1 \vee N_2$ | $D_1$ |
| $N_3$ | $D_2 \vee D_3$ |
| $AC_1 \vee AC_3$ | $D_1 \vee D_2$ |
| $AS_1$ | $D_2 \vee D_3$ |
| $R_1 \vee R_2 \vee R_5$ | $D_1 \vee D_4$ |

For example, the implication "$G_1 \rightarrow E_1 \vee E_2$" means "If gearbox $G_1$ is chosen, then engine $E_1$ or $E_2$ has to be chosen".

With the resulting formula $\varphi_{car}$ from the above specifications, we consider two customer cases:

1. A customer chooses engine $E_1$ and control unit $C_1$ for the car. But she does not want the air conditioner $AC_2$. We test Formula (4) for satisfiability.

$$\varphi_{car} \wedge x_{G_1} \wedge x_{C_1} \wedge \neg x_{AC_2} \tag{4}$$

   The result is **true**. Derived from the resulting model, we can choose the components $D_1, C_1, G_1, E_1$ to get a complete valid configuration assignment.

2. A customer chooses the components $E_1, G_2, C_2, D_3$ and $N_2, AC_1, AS_1, R_2$. The result is **false**.

   The question now is, which maximal subset of the original choice will lead to a valid configuration?

## 3.2 Advantage of the MaxSAT based approach

With the SAT based configuration, two main problems arise. First, if the configuration is not valid, it is not possible to know which components cause the conflict. Second, even if we know the components causing the conflict, we do not know, which components to omit to get a valid configuration with a maximal number of components we wanted originally. The example 2 of Subsection 3.1 shows such a case.

As mentioned in the introduction, the first problem can be handled with proof tracing to explain a conflict for an invalid configuration. The second problem can be handled with MaxSAT and its extensions. We explain this approach in the next section in detail.

## 4 Automotive Configuration with MaxSAT

For the representation of automotive configuration as a MaxSAT instance we consider the Partial MaxSAT problem. We use the SAT based specification $\varphi_{car}$ of Section 3 and divide the clauses into hard and soft ones. First, all cardinality constraints are marked as hard clauses, because they have to be satisfied (e.g. it is not possible to configure a car with more than one steering wheel). Second, it is possible that the dependencies between components do not necessarily have to be satisfied (e.g. a dependency could have been created due to marketing reasons; "No black seats for all Japanese cars"). On the other hand, technical dependencies have to be satisfied (e.g. a conflict between an engine and a gearbox). For simplicity reasons, we also mark all dependencies as hard clauses.

With the representation above, we can consider the following advanced use cases and answer the new arising questions with the help of a Partial (Weighted) MaxSAT solver:

1. **(Maximization of chosen components)** A customer chooses components $c_1, \ldots, c_n$ which lead to an invalid configuration. We can answer the question, what the maximal number of the chosen components for a valid configuration is, by solving Formula (5) with a Partial MaxSAT solver.

$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{x_{c_1} \wedge \ldots \wedge x_{c_n}}_{\text{soft clauses}} \tag{5}$$

2. **(Maximization of priorities)** We can generalize the use case 1 by attaching priorities to the components: A customer chooses components $c_1, \ldots, c_n$ which lead to an invalid configuration. Additionally, the customer has priorities $p_1, \ldots, p_n$, $p_i \in \mathbb{N}_{>0}$, for each component. We can answer the question, which sum of priorities can be maximally reached for a valid configuration by solving Formula (6) with a Partial Weighted MaxSAT solver.

$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(x_{c_1}, p_1) \wedge \ldots \wedge (x_{c_n}, p_n)}_{\text{soft clauses}} \tag{6}$$

3. **(Reconfiguration)** We can use the introduced techniques in the use cases 1 and 2 for reconfiguration. Let us assume a customer wants to add, replace, or remove components of her existing car. She chooses the components $c_1, \ldots, c_k$ with priorities $p_1, \ldots, p_k \in \mathbb{N}_{>0}$. If the priority or partial state (hard or soft) of a clause of an originally chosen component has changed, the original clause will be replaced by the new partial weighted clause. Otherwise, the clause will be kept. We solve Formula (7) with a Partial Weighted MaxSAT solver to reach the maximal sum of priorities.

$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(x_{c_1}, p_1) \wedge \ldots \wedge (x_{c_n}, p_n)}_{\text{soft clauses}} \tag{7}$$

   To force certain new components to be installed or old components to be kept, we can designate the corresponding clauses as hard clauses.

   To reach a valid reconfiguration for the customer, a reconfiguration scenario can be considered as a process in different steps:

   - Check for validation after the customer chooses new components with priorities as previously described.
   - If the hard clauses are unsatisfiable, check for validation after the sales division sets additional dependencies as soft clauses (with priorities).
   - If the hard clauses are unsatisfiable, check for validation after the engineering divison sets additional dependencies as soft clauses (with priorites).

   If the hard clauses are unsatisfiable after all steps, there is no valid configuration, because technical limitations are reached which can not be set as soft clauses. Otherwise, if the hard clauses are satisfiable in one step, we can compute the maximal sum of priorities of the soft clauses while satisfying the hard clauses.

4. **(Minimization of costs)** The components $c_1, \ldots, c_n$ have prices $p_1, \ldots, p_n$, $p_i \in \mathbb{N}_{>0}$. We want to know which components have to be chosen, to get a valid configuration with minimal cost. We can answer the question by solving Formula (8) with a Partial Weighted MinUNSAT solver.

$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(\neg x_{c_1}, p_1) \wedge \ldots \wedge (\neg x_{c_n}, p_n)}_{\text{soft clauses}} \tag{8}$$

   Instead of finding the minimal costs of a valid configuration, we could also compute a valid configuration of minimal weights, $CO_2$ emissions, or other interesting targets.

In all situations above, the resulting model of the solver tells us which components to choose to get the optimum.

Additionally, we can add arbitrary hard clauses to enforce certain constraints: (1) Unit clauses to enforce the in- or exclusion of a component; (2) Additional dependencies between components (e.g. "When engine $E_1$ is chosen, then choose gearbox $G_2$"; $(x_{E_1} \rightarrow x_{G_2})$); (3) Additional cardinality constraints (e.g. $x_{D_1} \vee x_{D_2}$ to ensure that one of the dashboards $D_1$ or $D_2$ will be chosen).

For example, in Situation 4 (minimization of costs), we could add unit clauses to enforce the inclusion of certain components and then compute the minimal costs of the configuration. The result is a valid configuration with minimal costs which includes our chosen components.

## 4.1 Example: MaxSAT based Configuration

We reconsider the example in Subsection 3.1.

1. In the second case, the choice of the customer was unsatisfiable. With the MaxSAT based approach of configuration we can find an assignment of a valid configuration where a maximum number of components is included. After solving Formula (5) with a Partial MaxSAT solver, we obtain the results shown in Table 3.

Table 3: Customer choices and Partial MaxSAT results

| family | choice | result |
|---|---|---|
| engine | $E_1$ | $E_1$ |
| gearbox | $G_2$ | $G_2$ |
| control unit | $C_2$ | $C_2$ |
| **dashboard** | $\boldsymbol{D_3}$ | $\boldsymbol{D_1}$ |
| navigation system | $N_2$ | $N_2$ |
| air conditioner | $AC_1$ | $AC_1$ |
| **alarm system** | $\boldsymbol{AS_1}$ | **–** |
| radio | $R_2$ | $R_2$ |

We can reach a valid configuration by changing two of the choices (bold rows in the table) and therefore, we can keep 6 of our 8 original components at most. For the alarm system, the resulting model did not set another alarm system variable to true, because this is an optional feature.

In general, the result obtained from the solver may not be the only optimum. There can be other different assignments with the same number of satisfied clauses.

2. We consider another case, where the customer chooses the components with priorities as shown in Table 4. Additionally, she wants dashboard $D_2$, $D_3$, or $D_4$. To enforce this constraint, we add the hard clause $(x_{D_2} \vee x_{D_3} \vee x_{D_4})$.

Table 4: Customer prioritized choices and PWMaxSAT results

| family | choice | priority | result |
|---|---|---|---|
| engine | $E_1$ | 8 | $E_1$ |
| gearbox | $G_2$ | 5 | $G_2$ |
| control unit | $C_2$ | 7 | $C_2$ |
| | $\boldsymbol{D_2}$ | **8** | |
| **dashboard** | $\boldsymbol{D_3}$ | **15** | $\boldsymbol{D_4}$ |
| | $\boldsymbol{D_4}$ | **15** | |
| **navigation system** | $\boldsymbol{N_2}$ | **20** | **–** |
| **air conditioner** | $\boldsymbol{AC_1}$ | **7** | **–** |
| **alarm system** | $\boldsymbol{AS_1}$ | **2** | **–** |
| radio | $R_2$ | 15 | $R_2$ |

Table 4 shows the result, scoring 50 priority points, after solving Formula (6).

3. After the previous configuration, the customer wants to reconfigure her existing car. Table 5 shows her choice. We can imagine that for technical or financial reasons, the engine $E_1$ and gearbox $G_2$ can not be replaced. We set them as hard clauses. However, control unit $C_2$ and dashboard $D_4$ can possibly be replaced and therefore are set as soft clauses.

Table 5: Reconfiguration choice and PWMaxSAT results

| family | state | new priorities | choice | results |
|---|---|---|---|---|
| engine | $E_1$ | hard | $E_1$ | $E_1$ |
| gearbox | $G_2$ | hard | $G_2$ | $G_2$ |
| control unit | $C_2$ | (5, soft) | $C_2$ | $C_2$ |
| **dashboard** | $\boldsymbol{D_4}$ | **(2, soft)** | $\boldsymbol{D_4}$ | $\boldsymbol{D_2}$ |
| navigation system | – | (10, soft) | $N_3$ | $N_3$ |
| air conditioner | – | hard | $AC_1 \vee AC_2$ | $AC_2$ |
| alarm system | – | (5, soft) | $AS_1$ | $AS_1$ |
| **radio** | $\boldsymbol{R_2}$ | **(13, soft)** | $\boldsymbol{R_2}$ | **–** |

The results show that dashboard $D_4$ was replaced by dashboard $D_2$ and radio $R_2$ has to be removed in favor of other components.

4. Now we associate the components with prices (as shown in Table 6) and we want to know a valid configuration with a minimal total price.

Table 6: Components with prices

| family | alternatives | | | | |
|---|---|---|---|---|---|
| engine | $E_1$ | $E_2$ | $E_3$ | | |
| price (€) | 4,000 | 2,500 | 4,500 | | |
| gearbox | $G_1$ | $G_2$ | $G_3$ | | |
| price (€) | 500 | 800 | 300 | | |
| control unit | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ |
| price (€) | 800 | 2,000 | 1,500 | 1,600 | 1,200 |
| dashboard | $D_1$ | $D_2$ | $D_3$ | $D_4$ | |
| price (€) | 300 | 500 | 600 | 450 | |
| navigation system | $N_1$ | $N_2$ | $N_3$ | | |
| price (€) | 100 | 150 | 130 | | |
| air conditioner | $AC_1$ | $AC_2$ | $AC_3$ | | |
| price (€) | 180 | 100 | 90 | | |
| alarm system | $AS_1$ | $AS_2$ | | | |
| price (€) | 300 | 250 | | | |
| radio | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |
| price (€) | 100 | 80 | 200 | 180 | 150 |

For the minimal costs we solve Formula (8) with a Partial Weighted MinUNSAT solver. For the maximal costs we solve Formula (6) with a Partial Weighted MaxSAT solver by considering the prices as priorities. The results are:

- Minimal cost: € 3,900
- Maximal cost: € 8,625

Table 7 lists the components to choose to reach the minimal and maximal costs.

Table 7: Choices for minimal and maximal costs

| | choice | |
|---|---|---|
| family | minimal cost | maximal cost |
| engine | $E_2$ | $E_3$ |
| gearbox | $G_3$ | $G_2$ |
| control unit | $C_1$ | $C_2$ |
| dashboard | $D_1$ | $D_3$ |
| navigation system | – | $N_3$ |
| air conditioner | – | $AC_2$ |
| alarm system | – | $AS_1$ |
| radio | – | $R_3$ |

## 5 Algorithmic techniques

In order to give the reader an impression of how MaxSAT can be computed, we present a short incomplete overview of some algorithmic techniques.

**Branch-and-Bound** The general branch and bound approach to explore the search tree of optimization problems can also be used for solving MaxSAT and its extensions. Each node of the tree represents a variable of the instance and has two children for the two values the variable can be assigned to. Tree pruning is used as soon as a partial solution becomes worse than the best solution found elsewhere in the tree. Heuristics have been developed e.g. by Wallace and Freuder to narrow the search space predicting the final value of partial solutions [Wallace and Freuder, 1993].

**Basic SAT-based** Given an unsatisfiable SAT problem $\varphi = \{C_1, \ldots, C_m\}$, we may iteratively try to remove individual clauses $C_i$ until the subproblem $\varphi'$ becomes satisfiable. $\varphi'$ will then be maximal in the sense that adding another clause will make it unsatisfiable, but another, larger, subproblem may exist which could be found by removing clauses from $\varphi$ in a different order.

In SAT solving, clause removal can be simulated by augmenting each clause $C_i$ with a fresh *blocking variable* $b_i$. As long as $b_i$ is set to *false*, the solver needs to satisfy $C_i$, but the constraint $C_i$ can effectively be blocked by setting $b_i$ to *true* instead. Now, in order to remove as few clauses as possible, we add $m$ blocking variables to $\varphi$ as above and restrict the use of the $b_i$ by an additional *cardinality constraint* $CC(k)$, which is a formula that prevents more than $k$ of the $b_i$ to be set to *true*. Iterating over $k$ from below until $\varphi(k)$ becomes satisfiable, or from above until $\varphi(k)$ becomes unsatisfiable, gives us the MaxSAT result $m - k$, and the subset of clauses whose $b_i$ are set to false forms one satisfiable subset of maximum cardinality.

Algorithm 1 reflects basic approach. One improvement of this approach is the use of binary search.

**Algorithm 1:** Basic SAT-based approach

```
**Input**: $\varphi = \{C_1, \ldots, C_m\}$
**Output**: Minimal number of unsatisfied clauses
$\varphi \leftarrow \{C_1 \vee b_1, \ldots, C_m \vee b_m\}$
$cost \leftarrow m$
**while** SAT($\varphi \cup$ CNF($\sum_{i=1}^{m} b_i < cost$)) **do**
    $cost \leftarrow cost - 1$
**return** $cost$
```

**Core-guided SAT-based** Modern proof-tracing SAT solvers return an unsatisfiable subset (unsat core) $\mu \subseteq \varphi$ when given an unsatisfiable $\varphi$. It is then clear that at least one clause of $\mu$ has to be blocked before $\varphi$ can become satisfiable, and thus the search can be narrowed compared to the basic approach. An algorithm based on this idea was proposed by Fu and Malik for partial MaxSAT [Fu and Malik, 2006]. In every iteration where the instance is unsatisfiable, we add a new blocking variable to all soft clauses of the unsatisfiable core and a new cardinality constraint to achieve that exactly one of the currently added blocking variables has to be satisfied. We can not just iterate over the unsat cores and count them, because they may not be disjoint.

This idea can also be extended for partial weighted MaxSAT [Ansótegui *et al.*, 2009].

## 6 Experimental Results

For our benchmarks we used product configuration formulas of a current (2013) product line of the German car manufacturer BMW. We added unit clauses to create unsatisfiable customer orders. We defined the following three categories for hard and soft clauses:

- Order: Soft clauses are unit clauses of the customer's order. All other clauses are hard. This asks, wich of the customer's wishes can be maximally satisfied.

- Packages: Soft clauses are clauses which represent packages, e.g. a sports package, which triggers all relevant sports components. The unit clauses of the customer's order and all other clauses are hard. This asks, which of the package restrictions can be maximally satisfied w.r.t. the customer's wishes.
- Packages & more: Soft clauses are package clauses and additional other sales relevant conditions. The unit clauses of the customer's order and all other clauses are hard. This asks, which of the package restrictions and additional restrictions can be maximally satisfied w.r.t. the customer's wishes.

The upper half of Table 8 shows detailed statistics about each category. The second half of the table shows how many instances have an optimum. No optimum means that there is at least one conflict involving only hard clauses. The average optimum is the average of the result of the minimal number of unsatisfiable clauses. For example, the average optimum of 2.127 within the 'Order' category means that on average 2.127 of the customer's choices can not be satisfied.

Table 8: Benchmark details

| | Benchmark categories | | |
|---|---|---|---|
| | Order | Packages | Packages & more |
| #instances | 777 | 777 | 777 |
| Avg. #variables | 896 | 896 | 896 |
| Avg. #hard clauses | 4474 | 3928 | 3592 |
| Avg. #soft clauses | 15 | 561 | 897 |
| #no optimum | 0 | 688 | 0 |
| #with optimum | 777 | 89 | 777 |
| Avg. optimum | 2.127 | 1.348 | 4.067 |

We applied our benchmarks to three different state-of-the-art MaxSAT solvers, namely:

- akmaxsat [Kügel, 2012]: A partial weighted MaxSAT solver based on a branch-and-bound approach. One of the best performing solvers in last year's MaxSAT competition[1].
- Fu & Malik [Fu and Malik, 2006]: A partial MaxSAT solver based on exploiting unsatisfiable cores and adding blocking variables to each soft clause of each found unsatisfiable core.
- PM2 [Ansótegui *et al.*, 2009]: A partial MaxSAT solver based on exploiting unsatisfiable cores. But unlike the Fu & Malik solver this approach only uses exactly one blocking variable to each clause.

For akmaxsat we used the implementation of Adrian Kügel[2]. We implemented the Fu & Malik and PM2 algorithms on top of our own Java SAT solver, which is optimized for our industrial collaborations. The cardinality constraints in the Fu & Malik approach are only of the form $\sum_{i=1}^{n} x_i = 1$ for given variables $x_1, \ldots, x_n$. We encode this constraint through the constraints $(\bigvee_{i=1}^{n} x_i)$ and $\left(\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)\right)$. The cardinality constraints in the PM2 approach uses general limitations, which we implemented with the encoding proposed in [Bailleux *et al.*, 2009].

[1] `http://maxsat.ia.udl.cat:81/12`

[2] `http://www.uni-ulm.de/in/theo/m/alumni/kuegel.html`

All our benchmarks were run on the same environment: Operating System: Ubuntu 12.04 64 Bit; Processor: Intel Core i7-3520M, 2,90 GHz; Main memory: 8 GB; JVM 1.7.0 (for Fu & Malik and PM2).

Table 9 shows the results of our time measurements of each solver in each category. The listed times are the average times a solver needed to solve an instance of a category. We listed the average time in each category Solver akmaxsat has an average time of remarkable less than 0.6 seconds in each category. Our implementation of Fu & Malik has a reasonable average time of less than 6 seconds in each category. Our implementation of PM2 has a reasonable average time for the first category 'Order', but exceeded our time limit of $3,600$ seconds per instance on too many instances of categories 'Packages' and 'Packages & more' to get a reasonable average time.

Table 9: Benchmark results with a time limit of 3,600 sec. per instance

| Avg. time (sec) | akmaxsat | Fu & Malik | PM2 |
|---|---|---|---|
| Order | **0.165** | 4.367 | 4.180 |
| Packages | **0.025** | 1.664 | exceeded limit |
| Packages & more | **0.535** | 5.387 | exceeded limit |

Figures 1, 2 and 3 show the performance of each solver in the first category 'Order'. These figures show the relation between the optimum and the response time of the instances. Especially for Fu & Malik and PM2 the response time seems to grow linearly with increasing optimum.

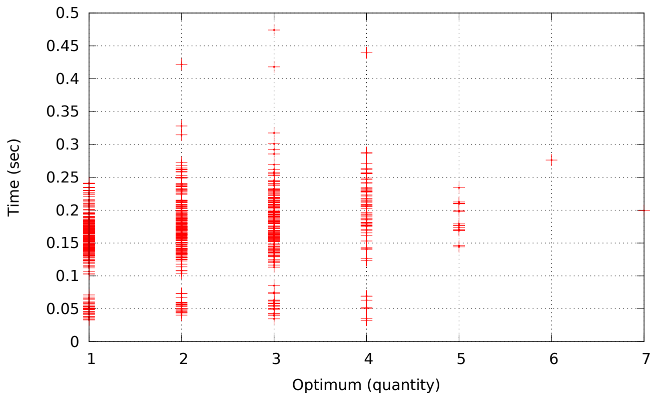

Figure 1: Benchmark 'Order' with akmaxsat

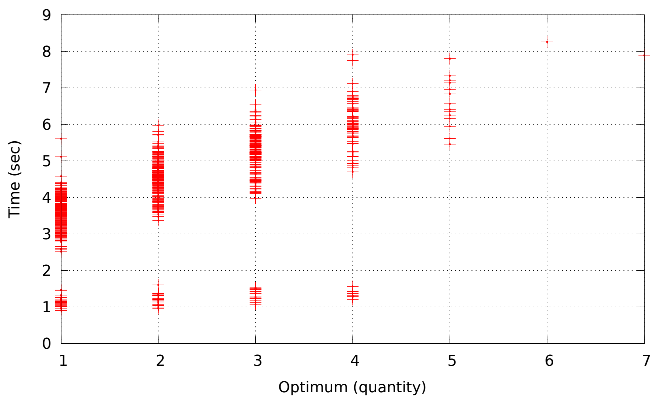

Figure 2: Benchmark 'Order' with Fu & Malik

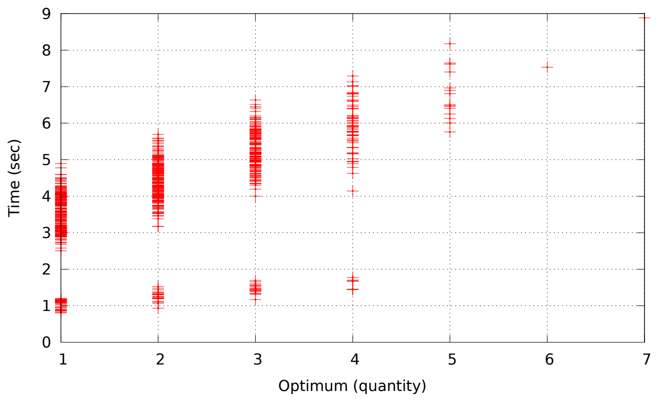

Figure 3: Benchmark 'Order' with PM2

In Figure 4 we can also recognize the linear growing response time with increasing optimum. Also note the lower line of quickly solvable instances.

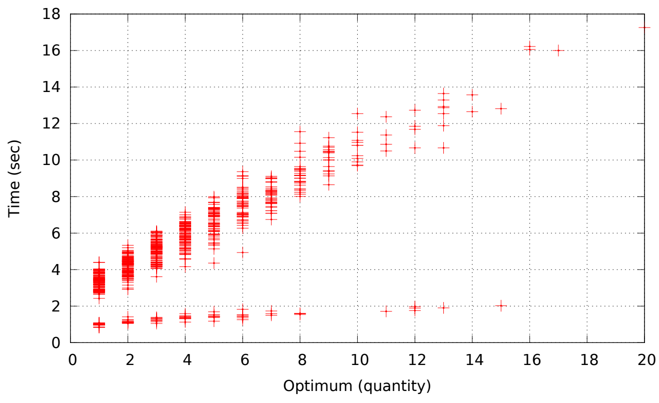

Figure 4: Benchmark 'Packages & more' with Fu & Malik

## 7 Related Work

In [Junker, 2004] general satisfaction problems are considered, where we have a knowlegde base of constraints which have to be satisfied and customer requirements, which we would like to satisfy. In the context of MaxSAT, the knowledge base can be considered as hard clauses, whereas the customer requirements can be considered as soft clauses. In the case of inconsistency, the proposed algorithm QuickXplain delivers preferred explanations, which are based on a given total ordering of the constraints.

The work of [Reiter, 1987] proposes an algorithm for computing minimal diagnoses using a conflict detection algorithm. A diagnosis is a minimal subset $\Delta$ of the customer requirements, such that the constraints without $\Delta$ is consistent. In [Felfernig *et al.*, 2012] another algorithm is proposed, called FastDiag, which computes a preferred minimal diagnosis without calculating the corresponding conflicts.

## 8 Conclusion

In this paper we showed detailed examples of how MaxSAT and its extensions can be applied in automotive configuration. With this approach we are able to repair an unsatisfiable customer order by computing the optimal solution which satisfies as many of the customer's choices as possible. Furthermore, we showed how MaxSAT also can be used in reconfiguration scenarios. From an already configured car we can compute the minimal number of components to change when adding, changing, or removing components.

We created realistic benchmarks for our MaxSAT applications out of the product formulas of our commercial collaboration with BMW. Our time measurements of these benchmarks against the state-of-the-art MaxSAT solvers akmaxsat, Fu & Malik, and PM2, showed that we have a reasonable response time, except for PM2 in two categories. These results suggest that MaxSAT can be applied for industrial automotive (re-)configuration problems.

## References

[Ansótegui *et al.*, 2009] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On solving MaxSAT through SAT. In *Proceedings of the 2009 conference on Artificial Intelligence Research and Development*, pages 284–292. IOS Press Amsterdam, Amsterdam, Netherlands, 2009.

[Asín *et al.*, 2010] Robert Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Practical algorithms for unsatisfiability proof and core generation in SAT solvers. *AI Communications*, 23(2–3):145–157, 2010.

[Astesana *et al.*, 2010] Jean Marc Astesana, Yves Bossu, Laurent Cosserat, and Helene Fargier. Constraint-based modeling and exploitation of a vehicle range at Renault's: Requirement analysis and complexity study. In *Proceedings of the 13th Workshop on Configuration*, pages 33–39, 2010.

[Bailleux *et al.*, 2009] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into CNF. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing—SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer Berlin Heidelberg, 2009.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, volume 185. IOS Press, Amsterdam, Netherlands, 2009.

[Felfernig *et al.*, 2012] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(1):53 – 62, 2012.

[Friedrich *et al.*, 2011] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. (re)configuration using answer set programming. In Kostyantyn Shchekotykhin, Dietmar Jannach, and Markus Zanker, editors, *IJCAI-11 Configuration Workshop Proceedings*, pages 17–24, Barcelona, Spain, July 2011.

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

[Junker, 2004] Ulrich Junker. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 167 – 172. AAAI Press / The MIT Press, 2004.

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

[Kügel, 2012] Adrian Kügel. Improved exact solver for the weighted max-sat problem. In Daniel Le Berre, editor, *POS-10. Pragmatics of SAT*, volume 8 of *EPiC Series*, pages 15–27. EasyChair, 2012.

[Manhart, 2005] Peter Manhart. Reconfiguration – a problem in search of solutions. In Dietmar Jannach and Alexander Felfernig, editors, *IJCAI-05 Configuration Workshop Proceedings*, pages 64–67, Edinburgh, Scotland, July 2005.

[Marques-Silva and Planes, 2008] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe*, pages 408–413. IEEE, 2008.

[Marques-Silva, 2008] João Marques-Silva. Practical applications of boolean satisfiability. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 74–80. IEEE, 2008.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 95, April 1987.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming—CP 2005*, Lecture Notes in Computer Science, pages 827–831. Springer Berlin Heidelberg, 2005.

[Wallace and Freuder, 1993] Richard Wallace and Eugene C. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *Discrete Mathematics and Theoretical Computer Science*, pages 587–615. American Mathematical Society, USA, 1993.

[Zhang and Malik, 2003] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the conference on Design, Automation and Test in Europe*, volume 1, pages 10880–10885. IEEE Computer Society, 2003.