Different Solving Strategies on PBO Problems from Automotive Industry

Thore Kübart [1] and **Rouven Walter** [2] and **Wolfgang Küchlin**[2]

Abstract. SAT solvers have proved to be very efficient in verifying the correctness of automotive product documentations. However, in many applications a car configuration has to be optimized with respect to a given objective function prioritizing the selectable product components. Typical applications include the generation of predictive configurations for production planning and the reconfiguration of non-constructible customer orders. So far, the successful application of core guided MaxSAT solvers and ILP-based solvers like CPLEX have been described in literature. In this paper, we consider the linear search performed by DPLL-based PBO solvers as a third solution approach. The aim is to understand the capabilities of each of the three approaches and to identify the most suitable approach for different application cases. Therefore we investigate real-world benchmarks which we derived from the product description of a major German premium car manufacturer. Results show that under certain circumstances DPLL-based PBO solvers are clearly the better alternative to the two other approaches.

1 Introduction

An already well-established approach in the automotive industry is to describe the set M of technically feasible vehicle configurations by a propositional formula φ such that $M = \{\tau \mid \tau(\varphi) = 1\}$ holds, where τ is a satisfying assignment of formula φ [11, 19], i.e. every model of φ is a feasible configuration. SAT solvers are the method of choice for calculating configurations that comprise certain options $o_1, \ldots, o_m$: A model has to be determined that satisfies the formula $\varphi \wedge \bigwedge_{i=1}^{m} o_i$. If there is no such model of the desired configuration, the user is often interested in an alternative model of optimal configured options o_i with respect to given priorities w_i. To reach a best possible configuration, a model of the formula φ has to be calculated that optimizes the target function $\sum_{i=1}^{m} w_i o_i$.

In the literature of the last few years different applications of this optimization problem are described as well as several approaches to solve it. Similar optimization problems arise for example from the task to minimize or maximize product properties such as price or weight [21]. Another example is the task of optimal reconfiguration, e.g., the selected options for a car are not feasible with the constraint set [21]. Furthermore valid configurations that optimize linear objective functions play an important role in demand forecasts [18]. We compare the underlying solving approaches by analyzing real-world instances of a major German premium car manufacturer.

This work is organized as follows: Section 2 introduces the basics of *Propositional Logic*, *Maximum Satisfiability* (MaxSAT), *Pseudo-Boolean Optimization* (PBO) and *Integer Linear Programming* (ILP) and their respective algorithmic solving techniques. Section 3 points out related work. Section 4 describes different optimization problems in automotive configuration. Section 5 presents a detailed evaluation of the different introduced optimization approaches for these problems including a discussion of the results. Finally, Section 6 concludes this work.

[1] Steinbeis-Transferzentrum Objekt- und Internet-Technologien, Sand 13, 72076 Tübingen, Germany

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

2 Preliminaries

In this work, we focus on propositional logic with the standard logical operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ over the set of Boolean variables X and with the constants $\bot$ and $\top$, representing false and true, respectively. The set of variables of a formula φ is denoted by $\mathrm{var}(\varphi)$. A formula φ is called *satisfiable*, if and only if there is an *assignment* τ, a mapping from the set of Boolean variables $\mathrm{var}(\varphi)$ to $\{0, 1\}$, under which the formula φ evaluates to 1. The evaluation of a formula under an assignment τ is the standard evaluation procedure for propositional logic, denoted by $\tau(\varphi)$. The values 0 and 1 are also referred to as *false* and *true*. The well-known NP-complete *SAT* problem asks the question whether a formula is satisfiable or not [6].

The established input format of a SAT solver nowadays is a *conjunctive normal form* (CNF) of a formula φ, where φ is transformed into a conjunction of *clauses* and each clause is a disjunction of *literals* (variables or negated variables). The variable of a literal l is denoted by $\mathrm{var}(l)$. For a formula $\varphi = \bigwedge_{i=1}^{k} \bigvee_{j=1}^{m_i} l_{i,j}$ in CNF we also make use of the notation of φ as a set of clauses where each clause is a set of literals: $\varphi = \{\{l_{1,1}, \ldots, l_{1,m_1}\}, \ldots, \{l_{k,1}, \ldots, l_{k,m_k}\}\}$. The transformation of an arbitrary formula φ into a CNF is done by a Tseitin- or Plaisted-Greenbaum-Transformation [16, 20] (denoted as $\mathrm{Tseitin}(\varphi)$). The resulting formula is not semantically equivalent, but equisatisfiable. Also, the models of φ and $\mathrm{Tseitin}(\varphi)$ are the same when restricted to the original variables $\mathrm{var}(\varphi)$.

2.1 MaxSAT

For a given clause set $\varphi = \{c_1, \ldots, c_m\}$, $m \in \mathbb{N}$, the *MaxSAT* problem [13] asks for the maximum number of clauses which can be simultaneously satisfied:

$$\mathrm{MaxSAT}(\varphi) = \max \left\{ \sum_{i=1}^{m} \tau(c_i) \;\middle|\; \tau \in \{0,1\}^{|\mathrm{var}(\varphi)|} \right\} \quad (1)$$

The corresponding problem of finding the minimum number of clauses which can be simultaneously unsatisfied is called MinUNSAT.

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

The *partial weighted MaxSAT* problem is an extended version where: (i) An additional clause set φ_{hard} of *hard* clauses is taken into account, which has to be satisfied, and (ii) weights $w_i \in \mathbb{N}$ are assigned to the *soft* clauses of $\varphi = \{c_1, \ldots, c_m\}$. The resulting problem, $\text{PWMaxSAT}(\varphi_{\text{hard}}, \varphi)$, consists of finding the maximal sum of weights of satisfied clauses of φ while satisfying φ_{hard}. To simplify reading we refer to partial weighted MaxSAT just as MaxSAT in the rest of this work.

MaxSAT can be solved by using a SAT solver as a black box, e.g. by linear search or binary search. Firstly, a fresh variable b_i, called *blocking variable*, is added to each soft clause, which serves to enable or disable the clause. Linear search iteratively checks the SAT instance φ_{hard} and φ with an additional constraint

$$\text{CNF}\left(\sum_{i=1}^{m} w_i \cdot \neg b_i > k\right), \tag{2}$$

where initially $k = 0$. With this check we search for a model with a sum of weights of at least 1. The constraint $\sum_{i=1}^{m} w_i \cdot \neg b_i > k$ is a Pseudo-Boolean constraint (see Subsection 2.2 for details), which can be transformed to a CNF, see for example [4, 8]. The degree k is increased to the sum of weights of the last model plus one in order to check if we can find a better model. Binary search, in contrast, follows the same scheme but restricts the search space with a lower and an upper bound simultaneously. Linear search requires m SAT calls in the worst case, whereas binary search requires only $\log_2(m)$ SAT calls in the worst case.

Another approach is the usage of unsatisfiable cores delivered by a SAT solver for the unsatisfiable case, which was introduced in [3, 9]. The idea is to iteratively call the SAT solver and relax the soft clauses contained in the unsatisfiable core by introducing blocking variables until the formula becomes satisfiable. Solvers using an unsatisfiable core approach performed well on industrial instances in recent MaxSAT competitions[3].

The OPEN-WBO framework [15] is based on using MiniSat-like solvers [7] and was one of the best performing MaxSAT algorithms on industrial instances in the recent MaxSAT competition. Both linear search and unsatisfiable-core guided solvers are included in different variations within the OPEN-WBO framework. The default solver, called WBO, is an unsatisfiable-core guided modification of [3] which partitions the soft clauses [2, 14]. Therefore, only a subset of the soft clauses are given to the SAT solver to make the SAT solver focus on relevant clauses. On the other hand, this can lead to additional SAT calls in the case where a model is found but not all soft clauses were considered. We used this solver for our evaluations, see Section 5.

2.2 DPLL-based PBO

In addition to clauses we consider linear pseudo-Boolean (LPB) constraints, which are linear inequalities of the form

$$\sum_{i=1}^{k} a_i l_i \triangleright b, \qquad \triangleright \in \{<, \leq, >, \geq, =\}, \tag{3}$$

where $a_i \in \mathbb{Z}$ and l_i are literals of Boolean variables. Under some assignment τ, the left side is the sum over the coefficients of the satisfied literals and the LPB constraint is satisfied iff the respective inequality holds. For example, clauses $\bigvee_{i=1}^{m} l_i$ can be generalized as LPB constraints $\sum_{i=1}^{m} l_i \geq 1$.

[3] http://www.maxsat.udl.cat/

Pseudo-Boolean solving (PBS) is the decision problem whether a set of LPB constraints can be satisfied by an assignment τ. Hence, PBS is a generalization of SAT. Like SAT solvers, most PBS solvers which prove satisfiability are able to provide a satisfying assignment τ to the user.

Given a satisfiable set of LPB constraints another problem is to identify a best possible assignment τ with respect to a linear objective function:

$$\begin{array}{ll} \min & \sum_i c_i l_i \\ \text{s.t.} & \bigwedge_j \left[\sum_i a_{j,i} l_{j,i} \triangleright b_j\right] \end{array} \tag{4}$$

This problem is called pseudo-Boolean optimization (PBO) [17].

In order to solve the satisifiability problem PBS, DPLL-style algorithms can be used to benefit from recent progress of modern SAT solvers. One approach is to transform the LPB-constraints into CNF and to apply SAT solvers to the resulting formula. Another approach is based on generalized constraint propagation and conflict-based learning, i. e. DPLL-based SAT solvers are enabled to handle LPB constraints directly. Generally learning methods analyze the conflict and learn a new constraint which is falsified on the conflict level and which propagates a new assignment on a higher decision level.

Given a PBS solver, the PBO problem of Formula (4) itself can be solved by iteratively applying the solver to perform a linear search or a binary search. Both approaches proceed analogously to the linear and binary search approaches for MaxSAT using a SAT solver.

In the linear search approach, models of the formula φ are calculated in order to gradually approach the optimal objective value. Through an extra LPB constraint, a model providing a better objective value is enforced. If the extra LPB constraint leads to an unsatisfiable PBS instance, the model last calculated is the optimal one.

In the binary search approach the search space is bisected by every single PBS-instance. If the optimal objective value lies inside the interval $[L, U]$, a model with the objective value $\leq M = (L+U)/2$ is searched for. If such a model exists, the minimal objective value lies inside $[L, M]$. If no such model exists, the minimal objective value lies inside $[M, U]$.

For the calculations in Section 5 we used the Sat4j library. The library is based on a Java version of MiniSat, which was expanded by generalized constraint propagation and conflict-based learning. The PBO solver contained therein realizes a simple linear search. The PBS solver called for this linear search is able to perform the following two learning methods.

The *clause-based learning method* calculates so-called UIPs (unique implications points) by means of propositional resolution just like in modern SAT solvers. SAT solvers such as MiniSat derive a UIP on the basis of the conflict clause and the reason clauses which were propagating the assignments of the conflict clause. If a conflict constraint occurs in the form of a LPB constraint, Sat4j first reduces this constraint to a conflict clause

$$K = \bigvee_{l \in \omega_C,\ \beta(l)=0} l, \tag{5}$$

where β is the partial assignment, that leads to the conflict in the LPB constraint ω_C. Analogously, reasons given by LPB constraints are reduced to reason clauses: If ω is a LPB constraint, that propagates $\tilde{l}$ under the partial assignment β, the clause

$$R(\tilde{l}) = \bigvee_{l \in \omega,\ \beta(l)=0} l \vee \tilde{l},\ \omega \models R(\tilde{l}), \tag{6}$$

is an implication of ω and also propagates the literal $\tilde{l}$ under β.

In a second learning method that is implemented in Sat4j more expressive LPB constraints instead of clauses are learned. For this purpose the principle of propositional resolutions in forms of Hookers cutting planes [10] is directly applied to the LBP constraints [5].

2.3 Integer Linear Programming

Like linear programming, integer linear programming deals with the optimization of linear objective functions over a set which is limited by linear equations and inequations. The difference is that while in linear optimizations any real values can be taken on, in integer optimization some or all variables are restricted to whole-number values.

$$\begin{array}{ll} \min & \sum_i c_i x_i \\ \text{s.t.} & \\ & \bigwedge_j \left[\sum_i a_{j,i} x_{j,i} \triangleright b_j\right] \\ & x_{j,i} \in \mathbb{Z} \end{array} \tag{7}$$

Pseudo-Boolean Optimization, see Formula (4), can be easily transformed into 0-1 integer linear programming (ILP): Negative literals $\neg x$ are replaced by $(1 - x)$ and Boolean variables become decision variables $x \in \{0, 1\}$. Consequently, commercially available state-of-the-art ILP solvers such as CPLEX [1] can be used to solve PBO. Usually, they are based on the branch and cut strategy.

3 Related Work

Sinz et al. suggest a SAT-based procedure to check consistency of the product documentation of a German car manufacturer [19]. Therefore, they consider vehicle configurations as assignments to propositional variables and define a propositional formula, called product overview formula (POF), which evaluates to true iff the vehicle configuration is technically feasible. Their work also lays the foundation for formulating the restrictions of the product documentation as a constraint in mathematical models.

Tilak Singh et al. make use of such a product overview formula in [18]. They describe a mathematical model to calculate car configurations aimed at providing the production planning with test variants before actual sales orders are received. Their configurations are calculated on the basis of PBO problems that are solved by CPLEX. This approach is described in greater detail in Section 4.

Walter et al. [21] describe the possible usage of MaxSAT in automotive configuration by pointing out various use cases. Whenever faced with an over-constrained configuration, MaxSAT can be used to reconfigure the invalid configuration by providing an optimal solution, e.g. giving a repair suggestion of the (possibly prioritized) over-constrained selections of a user by a minimal number of changes.

4 Optimization Problems from Automotive Configuration

The product overview defines the set of valid product configurations and is usually describable as a set of propositional constraints. This means a configuration is only valid if it satisfies the conjunction of all constraints, which is also called the product overview formula (POF). The physical demand of building components for a valid configuration is determined by the bill of materials (BOM). The combination of the product overview and the bill of material is referred to as the product documentation.

Reconfiguration of invalid configurations, as described in [21], is an important issue in automotive configuration. Several practical relevant use cases exist, such as the reconfiguration of customer orders, the reconfiguration of constraints for a given fixed order or the computation of a maximal/minimal car w.r.t. to an assigned value of the options like weights (kg).

Another major task is the prognosis of future part demands in production planning. However, historical demands cannot easily be extrapolated because planning situations in the automotive industry are constantly changing. A typical planning state used for making a forecast comprises

- the product documentation (option A is only available, if option B is selected; part x is needed exactly iff the order satisfies the part rule φ_x; etc.),
- estimated customer buying behavior (option C is selected by 30% of the customers; etc.),
- capacity restrictions (only 5000 units of part X are available; etc.),
- production plans that fix the total number of planned vehicles.

From a mathematical point of view a planning state consists of two parts:

1. The Boolean formula POF, whose models describe the technically feasible configurations,
2. the statistical frequency of certain atoms of the product formula.

A common approach to evaluate the planning state is to calculate an amount of N technically feasible variations that approximates the statistical guidelines as good as possible. Eventually, for indicating the future unit demand, the calculated variations are analyzed by means of the BOM.

Singh et al. [18] propose a linear optimization model for finding such a set of N technically feasible variations. A solution of their model is calculated by *column generation*: In every iteration a technically feasible variation is determined which further improves the solution set. For this, a PBO problem is solved. Its objective function is given by the dual variables of the current approximation and its constraints are given by the restrictions of the product overview.

In column generation, calculated configurations of previous iterations are partially replaced by new configurations. Thus, solved PBO instances do not necessarily result in a configuration also contained in the final solution. Therefore, it is particularly important to ensure an efficient calculation of the individual PBO instances.

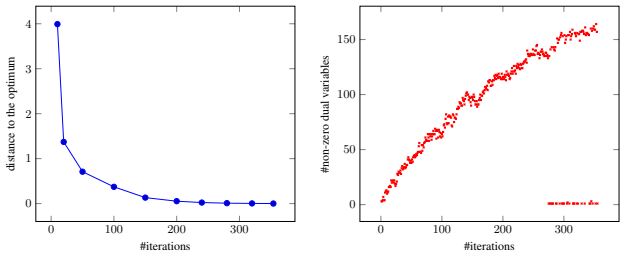

Figure 1. Typical approximation to the optimum using column generation

The typical approximation to the planning state when using column generation is shown in Figure 1. As one can see on the left side, the majority of iterations is used to overcome the last small step towards the optimum. Yet, illustrated by the graph on the right side,

approximation is accompanied by increasingly complex target functions of the PBO instances: the number of non-zero dual variables is increasing. It must be noted that the maximum number of non-zero dual variables is usually proportional to the number of statistical requirements of the planning state.

Concerning implementations of column generation described in the literature, integer linear programming is the method of choice for producing columns. This approach is also taken in [18] using CPLEX. An important objective of this paper is to investigate whether DPLL-based methods are suitable alternatives to CPLEX for calculating predictive configurations.

5 Experimental Evaluation

In this section we present our main contribution by evaluating the different previously described methods on optimization problems from automotive configuration.

As test environment for the experimental evaluations we used Windows 7 Professional 64 Bit on an Intel(R) Core(TM) i7-4800MQ CPU with 2.70 GHz and 2 GB main memory.

5.1 Benchmark Statistics

We evaluate several test series, each of which is composed of a real-world product overview and of randomly generated linear objective functions. We looked at product overviews of two different types, see A and B in Table 1. Both types of product overviews differ in the extent to which attributes are fixed. Additionally, Table 1 shows the minimum and maximum number of clauses for the product overviews of one type. For generating objective functions, n variables were randomly selected (by using Java's Random class with the default constructor) and each was assigned to a random integer coefficient from a range between $-10,000,000$ and $10,000,000$. That way, 10 objective functions were generated for different numbers of variables ($n = 10, 20, \ldots, 200$) so that one test series contains a total of 200 PBO instances. Corresponding test series were generated for 13 different product overviews of type A and for 6 different product overviews of type B.

Type	Fixed Attributes	#Clauses
A	market, model, body type, engine, steering type	1200-5900
B	market, model, body type	19100-137700

Table 1. Characteristics of type A and B

5.2 Results

The test series from the previous section were solved by 3 solvers:

1. OpenWBO as a core guided MaxSat solver [15].
2. Sat4j as an implementation of the DPLL-based linear search [12].
3. CPLEX as an ILP solver [1].

For solving the PBO instances a timeout of 60 seconds was set. In order to set up the linear search correctly, two different learning methods of the underlying PBS solver of Sat4j were tested. Concerning the test series of type A, Figure 2 shows the average running time for learning of

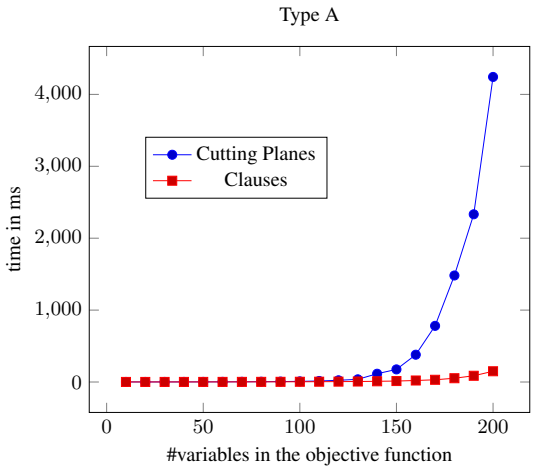

Figure 2. Learning Cutting Planes vs. Clauses in Sat4j

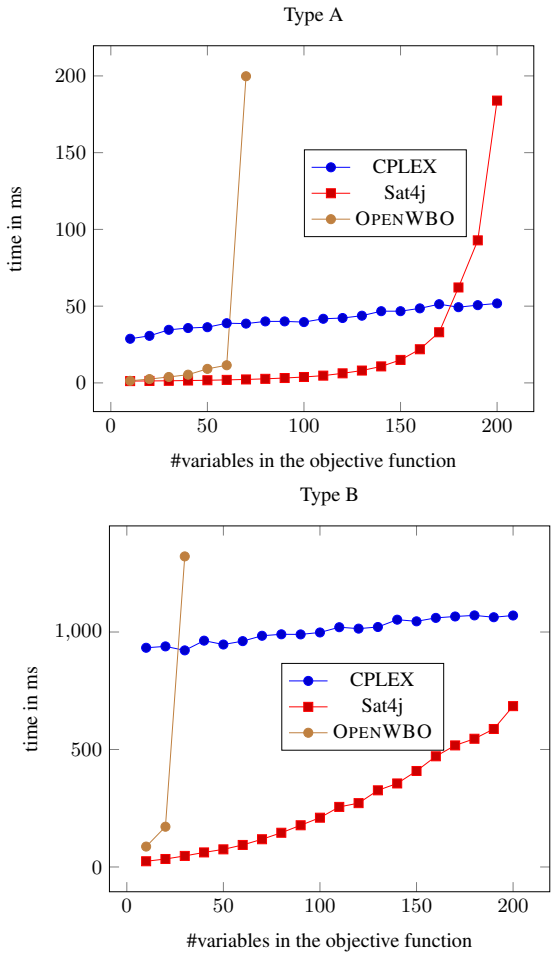

Figure 3. Running times for type A and B

1. clauses
2. cutting planes

depending on the complexity of the objective function. Based on these results, Sat4j was limited to learning clauses when comparing the 3 solvers.

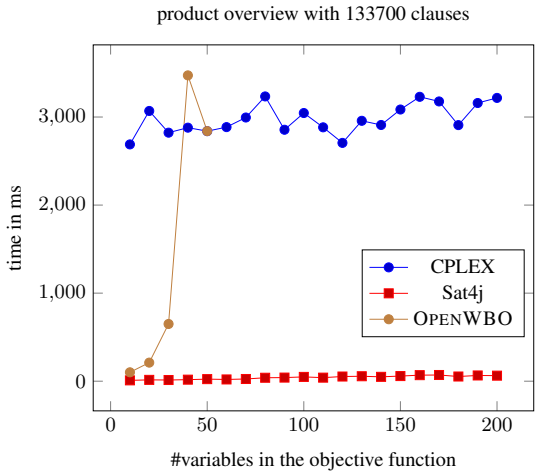

Figure 4. Running times for the largest product overview of type B

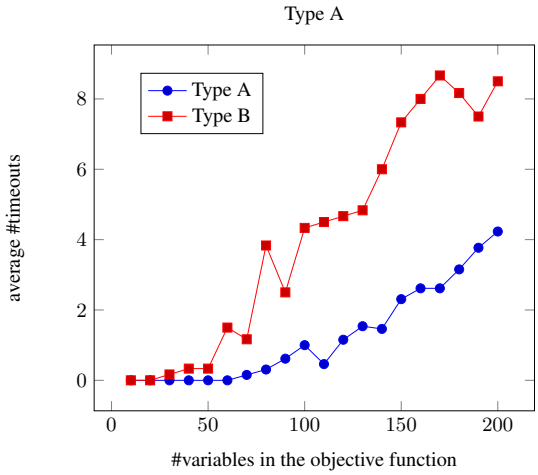

Figure 5. Average number of timeouts for OpenWBO

In Figure 3 the running times of OpenWBO, Sat4j and CPLEX are compared. The upper graph shows the average running times of the test series of type A concerning objective functions of different complexity. Accordingly the bottom graph displays the results of the test series of type B. Time was limited in both subfigures (200ms, 1400ms). In average, for all test series of type A and B, the DPLL-based solvers OpenWBO and Sat4j perform significantly better than CPLEX with regard to objective functions of low complexity. In test series of type A the solver Sat4j performs better than CPLEX concerning objective functions of up to $N = 170$ variables.

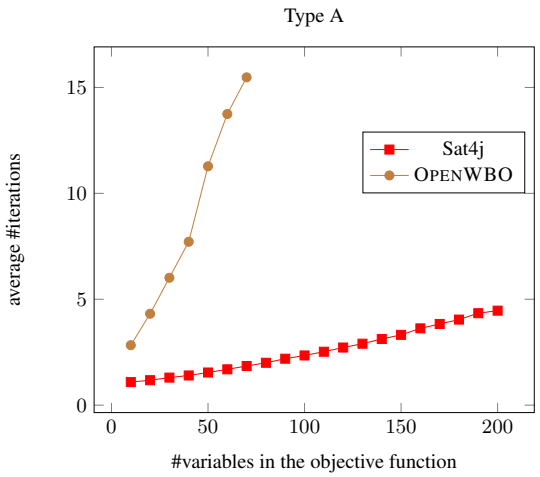

Figure 6. Average number of DPLL-calls for OpenWBO and Sat4j

The MaxSAT solver OpenWBO, however, produces reliable running times for up to $N = 70$ variables.
In contrast to the DPLL-based solvers, there was only a slight increase in the running times for CPLEX with a growing number of variables in the objective function. Hence, once a certain length of the target function has been reached, CPLEX leads to better results.

Across all test series of type B, Sat4j performed on average better than CPLEX (Figure 3, bottom graph). Compared to CPLEX however, OpenWBO leads to slower running times for objective functions of only $N = 30$ variables. This observation is especially illustrated by the most extensive product overview of type B (137700 clauses) as demonstrated in Figure 4, where the time-domain is restricted to 3400 ms.

Timeouts were observed exclusively for the MaxSAT solver OpenWBO. Complementary to the results of running times, Figure 5 shows the average number of timeouts when solving 10 instances.

For a better understanding of the observed difference in running times of both DPLL-based approaches, Figure 6 shows the average number of DPLL-calls. The graphs refer exclusively to the test series of type A and represent the numbers of SAT/UNSAT-calls for OpenWBO or, in case of Sat4j, the numbers of PBS-calls.

5.3 Discussion

The test instances described in Section 5.1 differ with respect to the following aspects:

- the complexity of the objective function (number of variables $N = 10, 20, \ldots, 200$)
- the size of the product overview (number of models and number of clauses; type A and B)

The findings of the previous section lead us to the following conclusions:

- Using DPLL allows the quick calculation of a model of a formula - in that way DPLL-based solvers are superior to ILP solvers.
- Yet, when objective functions become more and more complex, the particular suitability of CPLEX in solving 0-1-optimization problems dominates.

The latter point is evident in the strong increase of running times for the DPLL-based methods once a certain length of objective functions has been reached.

In comparison with OPENWBO, Sat4j leads to significantly better running times. This is also due to the special suitability of the examined instances for a linear search (see Figure 6). The number of iterations for a linear search increases only linearly with the numbers of variables in the objective function - this is a surprising result. Hence, the nonlinear increase of running times can only be explained by the growing complexity of PBS instances in linear search.

Concerning product overviews of type A, the critical length of the objective function for Sat4j is between $N = 160$ and $N = 200$. With increasing scale of the product overview, this critical value shifts upwards. In some cases of the product overviews of type B, the critical value is greater than $N = 200$, such as seen in Figure 4.

In order to optimally configure customer orders, an optimal subset of about 20 options has to be determined. For this purpose the linear search implemented in Sat4j and the core guided MaxSAT solver OPENWBO are more than sufficient.

For calculating predictive configurations for the product planning (see Figure 1) Sat4j can be far more effective than the other tested methods, at least for the first iterations of a column generation based process. Yet it is possible that the critical area of linear search is reached depending on the given number of statistical requirements in the planning state. In such a case configurations should be calculated by CPLEX just like in [18].

6 Conclusion

We compared current state-of-the-art solvers to calculate optimal product configurations of a major German car manufacturer. So far, the use of core guided MaxSAT solvers and ILP solvers like CPLEX for PBO-instances of automotive industry was described in the literature. For the purpose of comparison we additionally applied a DPLL-based linear search.

Results were analyzed with respect to the granularity of the product overview and with respect to complexity of the objective functions. The results show that the investigated approaches have different suitability for different application cases. For reconfiguration the linear search performed by a PBS-Solver is a stronger alternative compared to core guided MaxSAT. For calculating predictive configurations – up to a certain amount of given frequency restrictions – the DPLL-based linear search is even more suitable than CPLEX.

An important result is the small number of iterations observed in linear search. The approach of linear search thus appears to be especially suitable for PBO-instances whose constraints are given by a product overview. For reliable usage of the DPLL-based linear search, also for instances of long objective functions, a customized PBS solver needs to be developed. Such a solver must be able to more efficiently solve PBS instances that are characterized by a set of product overview clauses and one extensive LPB constraint.

REFERENCES

[1] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html, June 2015.

[2] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy, 'Improving SAT-based weighted MaxSAT solvers', in *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012*, ed., Michela Milano, volume 7514 of *Lecture Notes in Computer Science*, pp. 86–101. Springer, (2012).

[3] 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).

[4] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel, 'A translation of pseudo boolean constraints to SAT', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**(1–4), 191–200, (2006).

[5] Donald Chai and Andreas Kuehlmann, 'A fast pseudo-boolean constraint solver', *IEEE Transactions on CAD of Integrated Circuits and Systems*, **24**(3), 305–317, (2005).

[6] Stephen A. Cook, 'The complexity of theorem-proving procedures', in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pp. 151–158, New York, NY, USA, (1971). ACM.

[7] 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).

[8] Niklas Eén and Niklas Sörensson, 'Translating pseudo-boolean constraints into SAT', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**, 1–26, (2006).

[9] 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).

[10] John N. Hooker, 'Generalized resolution and cutting planes', *Annals of Operations Research*, **12**(1), 217–239, (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] Daniel Le Berre and Anne Parrain, 'The Sat4j library, release 2.2', *Journal on Satisfiability, Boolean Modeling and Computation*, **7**(2–3), 59–6, (2010).

[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] Ruben Martins, Vasco M. Manquinho, and Inês Lynce, 'On partitioning for maximum satisfiability', in *ECAI 2012 - 20th European Conference on Artificial Intelligence*, eds., Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pp. 913–914. IOS Press, (2012).

[15] Ruben Martins, Vasco M. Manquinho, and Inês Lynce, 'Open-WBO: A modular MaxSAT solver', in *Theory and Applications of Satisfiability Testing – SAT 2014*, eds., Carsten Sinz and Uwe Egly, volume 8561 of *Lecture Notes in Computer Science*, pp. 438–445. Springer International Publishing, (2014).

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

[17] Olivier Roussel and Vasco M. Manquinho, 'Pseudo-boolean and cardinality 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 22, 695–733, IOS Press, (2009).

[18] Tilak Raj Singh and Narayan Rangaraj, 'Generation of predictive configurations for production planning', in *Proceedings of the 15th International Configuration Workshop*, eds., Michel Aldanondo and Andreas Falkner, pp. 79–86, Vienna, Austria, (August 2013).

[19] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (January 2003). Special issue on configuration.

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

[21] 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).