On the Configuration of SAT Formulae

Mauro Vallati

School of Computing and Engineering, University of Huddersfield,
Huddersfield, United Kingdom.

Abstract. In this work we investigate how the performance of SAT solvers can be improved by SAT formulae configuration. We introduce a fully automated approach for this configuration task, that considers a number of criteria for optimising the order in which clauses and, within clauses, literals, are listed in a formula expressed using the CNF.
Our experimental analysis, involving three state-of-the-art SAT solvers and six different benchmark sets, shows that this configuration can have a significant impact on solvers' performance.

1 Introduction

The propositional satisfiability problem (SAT) is one of the most prominent problems in Artificial Intelligence (AI), and it is exploited in a wide range of real-world applications. Well-known examples include hardware and software verification [12], test-case generation [4], and scheduling [6]. Nowadays, thanks also the SAT competition,[1] there is a large number of ready-to-use SAT solvers that can be exploited in applications.

By exploiting algorithm configuration techniques, SAT solvers' behaviour can be adjusted to perform well for a specific type of instances [13, 7, 10]. To support this type of customisation, most state-of-the-art solvers expose a large number of parameters whose settings affect most parts of the solver. Furthermore, in areas of AI such as automated planning [14] or abstract argumentation [5], it has been demonstrated that also the configuration of the knowledge models, i.e. the symbolic representation of the problem that has to be analysed by automated reasoners, can lead to significant performance improvements of general solvers. Intuitively, such results indicate that the way in which a model is represented strongly affect the behaviour of automated reasoners.

In this context, we propose an approach for performing the automated configuration of SAT formulae expressed using the conjunctive normal form (CNF). The configuration aims at identifying an ordering of clauses and, within clauses, the involved literals, of a CNF from a specific type of

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

instances that allows to improve the performance of a given state-of-the-art SAT solver. Notably, due to the fact that the configuration has to be performed online when a new CNF is provided to the solver, it has to rely on characteristics of the CNF that are computationally cheap to extract. Through comprehensive experiments, using three SAT solvers and six different benchmark sets, we demonstrate that the CNF configuration as a remarkable impact on performance, and can provide valuable information for the encoding of CNFs and for further improvements of SAT solvers.

2 SAT Formulae Configuration

In this work we focus on SAT formulae represented using conjunctive normal form. A CNF is a conjunction of clauses, where a clause is a disjunction of literals. In a formula, clauses are usually not ordered following a principled approach, but are ordered according to the way in which the randomised generator has been coded, or following the way in which information from the application domain has been collected. This is also generally true for the order in which literals of a given clause are presented in the formula.

Here we focus on the following question: *given the set of clauses and, for each clause, the set of corresponding literals, in which order should they be listed to maximise the performance of a given solver?* The underlying hypothesis is that the order in which clauses and literals are listed carries some knowledge about their importance for satisfying, or demonstrating the unsatisfability, of the considered SAT instance. Such implicit knowledge can be exposed via a smart ordering of the clauses, that will affect the behaviour of the considered SAT solver.

2.1 Automated Configuration of SAT Formulae

In this work we use the state-of-the-art SMAC [8] configuration approach for identifying a configuration of CNFs, encoded using the DIMACS format, that improves the PAR10 performance of a given SAT solver. PAR10 is the average runtime where unsolved instances count as $10\times$ cutoff time. PAR10 is a metric commonly exploited in machine learning and algorithm configuration techniques, as it allows to consider coverage and runtime at the same time.

SMAC uses predictive models of performance [11] to guide its search for good configurations. More precisely, it uses previously observed $\langle$configuration, performance$\rangle$ pairs $\langle c, f(c)\rangle$ and supervised machine learning (random

forests [3]) to learn a function $\hat{f} : \mathcal{C} \rightarrow \mathbb{R}$ that predicts the performance of arbitrary parameter configurations.

The CNF configuration has to be performed online: as soon as a new formula is provided as input, the formula has to be configured before being presented to the solver. Given this scenario, we are restricted only to information about the CNF that can be quickly gathered. Furthermore, the configuration should consider only general aspects which are common to any CNF. As it is apparent, the use of a computationally expensive configuration of a single CNF, that considers elements that are specific to the given CNF, would nullify the potential performance improvement, by drastically reducing the time available for the solver.

In this work, we considered the possibility to list *clauses* according to the following criteria: (1) the number of literals of the clause; (2) the fact that the clause is binary; (3) the fact that the clause is ternary; (4) the number of positive literals involved; (5) the number of negative literals of the clause; (6) the fact that the clause is binary, and both literals are negative; and (7) the fact that the clause has only one negative literal.

Literals can be listed in clauses according to: (*i*) the number of clauses in which the literal is involved; (*ii*) the average size of the clauses in which the literal is involved; (*iii*) the number of binary clauses in which the literal in involved; (*iv*) the number of ternary clauses in which the literal is involved; (*v*) the number of times the literal appears in clauses as positive; (*vi*) the number of times the literal appears in clauses as negative; (*vii*) the number of times the literal is involved in clauses where all literals are positive; and (*viii*) the number of times the literal is involved in clauses where all literals are negative.

It is easy to notice that many of the introduced criteria focus on aspects of binary and ternary clauses. This is due to their importance in the search process. For instance, binary clauses are responsible, to a great degree, of unit propagation. There are also criteria that aims at identifying potentially relevant aspects. For instance, criterion 7 aims at identifying clauses that may be representing implication relations between literals.

There are different ways for encoding the degrees of freedom in CNFs as parameters. This is due to the fact that orders are not natively supported by general configuration techniques. Following [14], we generate 7 continuous parameters for configuring the order of clauses, and 8 continuous parameters for configuring the order of literals in clauses. Such parameters correspond to the aforementioned criteria, that have to be combined to generate different possible orderings of clauses and literals

in CNFs. Each continuous parameter has associated a real value in the interval $[-10.0, +10.0]$ which represents (in absolute value) the *weight* given to an ordering criterion. Two additional categorical selectors are also included. One which allows to activate or de-active the ordering of literals in the clauses, and the second that allows to order clauses according to the ordering (direct or inverse) followed by the involved literals. Thus, the configuration space is $\mathcal{C} = [-10.0, +10.0]^{15} \times 2 \times 3$, where 2 are the possible values of the parameter on ordering of literals in clauses, and 3 are the possible values of the categorical parameter describing whether the order of clauses should follow the order of involved literals. An ordering σ instantiates each of the 17 parameters, and can be used on any CNF. Given a CNF and an ordering σ, the corresponding configuration of the formula is obtained as follows. For each literal, an ordering score $o_l(v)$ is defined as:

$$o_l(l) = \sum_{c \in C} (value(p, c) \times weight(c)) \tag{1}$$

where c is a continuous ordering criterion in the set C of the 8 available continuous parameters for configuring literals' order, $value(p, c)$ is the numerical value of the corresponding aspect for the literal v, and $weight(c)$ is the weight assigned to the corresponding continuous parameter by the configuration technique. If the 16th parameter is set to ignore the order of literals in clauses, then literals are ordered as in the provided initial CNF. Otherwise, the literals in a clauses are ordered following the score $o_l(v)$. Ties are broken following the order in the original CNF configuration.

Similarly to what is presented in Equation 1 for literals, clauses are ordered according to a corresponding score $o_c(d)$ –where C is the set of clauses ordering criteria–, unless clauses are forced to follow the order of literals via the appropriate parameter. In that case, clauses are ordered according to the sum of the $o_l(v)$ scores of the involved literals $L(d)$, as shown in Equation 2.

$$o_c(d) = \sum_{v \in L(d)} o_l(v) \tag{2}$$

Example 1. Let us consider the CNF presented, using the DIMACS format, on the left side of Figure 1. Suppose that we are interested in listing clauses according to their length (criterion 1) and to the number of involved negative literals (criterion 5). Similarly, we are interested in listing the literals of a clause according to the number of clauses in which they appear (criterion i). This can be done by leaving all the parameters to

```
1 -3 0
2 3 -1 -4 0
-5 -4 0
```

```
3 -1 -4 2 0
-4 -5 0
1 -3 0
```

Fig. 1. The example CNF non configured (on the left), and the configured version (right). Configuration has been done by listing clauses according to their length and the number of negative literals. Literals are listed following the number of clauses they are involved.

the default value 0.0, but the ones controlling the mentioned criteria to 10.0. Considering only criteria 1 and 5, the clause `2 3 -1 -4 0` has a $o_c(d)$ score of $(4 + 2) \times 10.0 = 60.0$: it involves 4 literals, and 2 literals are negative. According to the same criteria, clause `1 -3 0` has a score of $(2 + 1) \times 10.0 = 30.0$. In a similar way, but considering the corresponding criterion, score of literals can be calculated, and literals are then ordered accordingly in each clause.

3 Experimental Analysis

Our experimental analysis aims to evaluate the impact of CNFs configuration on state-of-the-art SAT solvers.

We selected 3 SAT solvers, based on their performance in recent SAT competitions: Cadical [2], Glucose [1], and Lingeling [2]. The latest available version of each solver has been considered. For each solver, a benchmark-set specific configuration was generated using SMAC 2.08. A Python 2.7 script is used for extracting information from a given CNFs and, according to the parameters' value, reconfigure it and provide it as input for the SAT solver.

We chose benchmark sets from the Configurable SAT Solver Challenge (CSSC) 2014 edition [10], and the benchmarks used in the Agile track of the 2016 SAT competition.[2] CSSC 2014 sets include: Circuit Fuzz (Industrial track), 3cnf, K3 (Random SAT+UNSAT Track), and Queens and Low Autocorrelation Binary Sequence (Crafted track). Benchmark sets were selected in order to cover most of the tracks considered in CSSC, and by checking that at least 20% of the instances were solvable by considered solvers, when run on the default CNFs. Benchmarks were randomly divided into training and testing instances, aiming at having 150-250 instances for testing purposes. Following the design of the CSSC, a cutoff of 5 CPU-time minutes, and a memory limit of 4 GB of RAM, has been

[2] https://baldur.iti.kit.edu/sat-competition-2016/

Table 1. Results of the selected solvers on the considered benchmark sets. For each solver and benchmark, we show the number of test set timeouts achieved when running on the default and on the configured CNFs. Bold indicates the best result.

	Cadical	**Glucose**	**Lingeling**
	# timeouts: default → configured		
K3	89 → **84**	72 → **69**	76 → **75**
3cnf	219 → **216**	134 → **131**	213 → **210**
Queens	10 → **9**	26 → **25**	24 → **23**
Low Autocorrelation	118 → **116**	115 → **109**	123 → **120**
Circuit Fuzz	19 → **17**	9 → 9	12 → **10**
Agile16	31 → **29**	24 → **19**	55 → **48**
Total	486 → **471**	380 → **362**	503 → **486**

set for each solver run on both training and testing instances. Experiments were run on a dedicated machine equipped with Intel Xeon 2.50 Ghz processors. Each configuration process has been given a budget of 5 CPU-time days on a single processor.

Table 1 summarises the results of the selected solvers on the considered benchmark sets. Results are presented in terms of the number of timeouts on testing instances, achieved by solvers run using the default or the configured CNFs. Indeed, all of the considered solvers benefited from the configuration of the CNFs. Improvements vary according to the benchmark sets: the Agile16 set is, in general, the set where the solvers gained more by the use of configured CNFs. Remarkably, the improvements observed in Table 1 are comparable to those achieved in CSSC 2013 and 2014, by configuring the solvers' behaviour [10]. In fact, our intuition is that the way in which clauses and literals are ordered has an impact on the way in which solvers explore the search space. Listing "important" clauses earlier, may lead the solver to tackle complex situations early in the search process, making it then easier to find a solution.

Interestingly, the overall results (last row of Table 1) indicate that the CNF configuration does not affect all the solvers in a similar way, and that can potentially lead to rank inversions in competitions or comparisons. This is the case of Lingeling (on configured) and Cadical on default. This may suggest that current competitions could have an implicit bias, in the fact that the selected CNF configuration may favour a solver more than others. Finally, It is worth noting that the way in which the CNFs are configured varies significantly between solvers, as well as according to the benchmark set.

Table 2. Results of the selected solvers on the considered benchmark sets. For each solver and benchmark, we show the IPC score achieved when running on the default and on the configured CNFs. Bold indicates the best result. Results of different solvers can not be directly compared.

	Cadical	**Glucose**	**Lingeling**
	IPC score: default → configured		
K3	56.7 → **59.9**	71.3 → **76.3**	67.8 → **68.6**
3cnf	27.3 → **31.6**	106.6 → **107.0**	33.6 → **35.9**
Queens	136.5 → **137.6**	119.3 → **121.1**	120.6 → **122.9**
Low Autocorrelation	171.8 → **173.4**	177.2 → **183.7**	171.0 → **175.3**
Circuit Fuzz	156.3 → **160.8**	175.2 → **175.3**	161.3 → **164.3**
Agile16	208.1 → **211.3**	209.1 → **215.9**	188.6 → **196.6**
Total	756.7 → **774.6**	858.7 → **879.3**	742.9 → **763.6**

Table 2 shows the impact of configuring CNFs in terms of IPC score variations. For a solver $\mathcal{C}$ and a SAT instance p, $Score(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1+\log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise (where T_p^* is the minimum amount of time required by any compared system to solve the instance). The IPC score on a set of instances is given by the sum of the score achieved on each considered instance. The IPC score provides a trade-off between runtime and coverage, and is used in the International Planning Competition for comparing planners' performance. In Table 2 the performance of a solver run on the default and configured CNFs are compared, in order to highlight changes due to the configuration process. Results indicate that the configuration provides, for most of the benchmark sets, a noticeable improvement also in terms of IPC score.

To shed some light on the important aspects of CNF configuration, we assessed the importance of parameters in the considered configurations using the fANOVA tool [9]. In terms of clauses, parameters controlling the weight of criteria 4 and 5 are deemed to be the most important. Parameters related to criteria *ii*, *vi*, and *viii* have shown to have a significant impact with regards to the literals' ordering. Generally speaking, the ordering of literals appears as more important than the ordering of clauses: this is also because, in many cases, clauses are ordered according to the (separately-calculated) weight of the involved literals. This behaviour can be due to the way in which data structures are generated by solvers: usually literals are the main element, and clauses are related to them via lists.

In order to test if it is possible to obtain a general configuration that improves the performance of a solver on any CNF, despite of the bench-

mark, for each of the solver we performed a configuration process using a training set composed by an equal proportion of instances from each of the 6 sets. Results on the independent testing set indicate that this sort of configuration has a very limited impact on solvers' performance: it is hence the case that structurally different sets of instances require a different configuration. Intuitively, this seems to point to the fact that, in different structures, the characteristics that identify challenging elements to deal with, vary.

4 Conclusion

In this paper we proposed an approach for performing the automated configuration of CNFs. We considered as configurable the order in which clauses are listed and the order in which literals are listed in the clauses.

The performed analysis, aimed at investigating how the configuration of CNFs affects the performance of state-of-the-art SAT solvers: (i) demonstrates that configuration has a significant impact on solvers' performance; (ii) indicates that the configuration should be performed on specific types of CNFs; and (iii) highlights important aspects of CNFs, that have a potentially strong impact on the performance of solvers.

We see several avenues for future work. We are interested in comparing configurations obtained on the same type of instances, but for only SAT or only UNSAT cases. We plan to evaluate the impact of configuration on weighted max SAT, where the weight of the clauses can provide another important information to the configuration process. Finally, the plan to incorporate the re-ordering of clauses and literals into existing SAT solvers, in order to further improve performance, and to investigate the concurrent configuration of CNFs and solvers.

References

1. Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Theory and Applications of Satisfiability Testing - SAT*, pages 309–317, 2013.
2. Armin Biere. Cadical, lingeling, plingeling, treengeling and yalsat entering the sat competition 2017. In *SAT competition 2017, Solver and Benchmark Descriptions*, 2017.
3. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
4. Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pages 209–224, 2008.

5. Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. On the impact of configuration on abstract argumentation automated reasoning. *Int. J. Approx. Reasoning*, 92:120–138, 2018.
6. J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. *AAAI*, pages 1092–1097, 1994.
7. Stefan Falkner, Marius Thomas Lindauer, and Frank Hutter. Spysmac: Automated configuration and performance analysis of SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2015*, pages 215–222, 2015.
8. F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION)*, pages 507–523, 2011.
9. Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of The 31st International Conference on Machine Learning*, pages 754–762, 2014.
10. Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger H. Hoos, and Kevin Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artif. Intell.*, 243:1–25, 2017.
11. Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.
12. Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7(2):156–173, April 2005.
13. Dave A. D. Tompkins, Adrian Balint, and Holger H. Hoos. Captain jack: New variable selection heuristics in local search for sat. In *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 302–316, 2011.
14. M. Vallati, F. Hutter, L. Chrpa, and T.L. McCluskey. On the effective configuration of planning domain models. In *Proceedings of (IJCAI)*, 2015.