On the Configuration of SAT Formulae

Mauro Vallati[0000−0002−8429−3570] Marco Maratea2[0000−0002−9034−2527]

[1] School of Computing and Engineering,
University of Huddersfield,
Huddersfield, United Kingdom
m.vallati@hud.ac.uk
[2] DIBRIS,
University of Genova, Genova, Italy
marco.maratea@dibris.unige.it

Abstract. It is well-known that the order in which clauses and literals are listed in a SAT formulae can have a strong impact on solvers' performance.
In this work we investigate how the performance of SAT solvers can be *improved* by a specifically-designed 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 Conjunctive Normal Form.
Our experimental analysis, involving three state-of-the-art SAT solvers and six different benchmark sets, shows that the configurations identified by the proposed approach can have a significant positive impact on solvers' performance.

Keywords: SATisfiability · Knowledge Configuration · Performance Improvement.

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 [19], test-case generation [4], automated planning [20], and scheduling [6]. Nowadays, thanks also to the SAT competitions and SAT challenges,[3] there is a large-yet-growing 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 [21, 7, 14], allowing them to be optimised for the actual problems at hand. To support this type of customisation, most state-of-the-art solvers expose a large number of parameters whose settings can significantly modify many parts of the solver, like the heuristic

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

and search techniques. Furthermore, in areas of AI such as automated planning [22, 23] 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 is provided as input to automated reasoners, can lead to significant performance improvements of general domain-independent solvers. Intuitively, such results can be due to the fact that the way in which a model is represented: (i) implicitly carries some knowledge about the problem, and such knowledge can positively impact the behaviour of solvers; and (ii) can early guide the search approach towards promising areas of the search space, by ordering the way in which options are considered. It may be argued that the first aspect can be more relevant in "complex" models, such as those exploited in automated planning, while the second aspect can be prominent in the case of less-structured models.

In the SAT field, it is well-known that the ordering of clauses and literals in SAT formulae can have a strong impact on the performance of solvers, and random shuffling has been routinely exploited in international competitions as a technique for avoiding some potential biases. In this context, we propose instead to exploit the impact that orderings can have on SAT solvers in order to improve performance. Here we introduce 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 instances that allows to *improve* the performance of a given SAT solver. In this sense, the proposed approach exploits the fact that the ordering of elements in SAT formulae has an impact on solvers, to improve performance and reduce –at least in part– the *accidental complexity* of formulae.

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 state-of-the-art SAT solvers and six different benchmark sets, we demonstrate that the proposed approach for performing CNF configuration can lead to significant benefits in terms of SAT solvers' performance, and can provide valuable information for the encoding of CNFs and for further improvements of SAT solvers.

This paper is organised as follows. Firstly, we provide the relevant background on the DIMACS format, used for representing CNFs. Then, we describe the proposed approach for the automated configuration of SAT formulae. After that, we show the results of our large experimental analysis. Finally, conclusions are given.

2 SAT Formulae

In this work we focus on SAT formulae represented using Conjunctive Normal Form (CNF) and following the DIMACS format. The DIMACS format is the

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

Fig. 1. An example CNF encoded in the DIMACS format.

standard format supported by SAT solvers, and used in SAT competitions and challenges.[4]

A CNF is a conjunction of clauses, where a clause is a disjunction of literals. Literals can be assigned a boolean value.

Figure 1 gives an example of a CNF, presented in the DIMACS format, including five literals and three clauses. The line starting by p gives information about the formula: the instance is a CNF, and the number of literals and clauses, respectively, are provided. In the DIMACS format a literal is uniquely identified by a number. After the initial descriptive line, clauses are listed. Each clause is a sequence of distinct non-null numbers ending with `0` on the same line. Positive numbers denote the corresponding literals. Negative numbers denote the negations of the corresponding literals. In the computation performed by modern SAT solvers, the formula is satisfied when all the literals have been assigned, and all the clauses are true at the same time. For the given example, a valid solution would be `1 2 3 -4 5`, corresponding to an assignment where all the literals are `True`, but literal `4`.

3 Configuration of SAT Formulae

In a SAT formula, clauses are usually not ordered following a principled approach, but they are ordered according to the way in which the randomised generator has been coded, following the way in which information from the application domain has been collected, or deliberately shuffled to prevent potential biases. 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 can be tuned to highlights elements that are important for satisfying, or demonstrating the unsatisfability, of the considered SAT instance by the considered SAT solver.

3.1 Automated Configuration of SAT Formulae

In this work we use the state-of-the-art SMAC [10] configuration approach for identifying a configuration of CNFs, encoded using the DIMACS format, that

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

aims at improving the PAR10 performance of a given SAT solver. PAR10 is the average runtime where unsolved instances count as 10× 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 [15] to guide its search for good configurations. More precisely, it uses previously observed ⟨configuration, performance⟩ pairs $\langle c, f(c)\rangle$ and supervised machine learning (random forests [3]) to learn a function

$$\hat{f} : \mathcal{C} \rightarrow \mathbb{R} \tag{1}$$

that predicts the performance of arbitrary parameter configurations (including those not yet evaluated). The performance data to fit these models is collected sequentially. In a nutshell, after an initialisation phase, SMAC iterates the following three steps: (1) use the performance measurements observed so far to fit a random forest model $\hat{f}$; (2) use $\hat{f}$ to select a promising configuration $c \in \mathcal{C}$ to evaluate next, trading off exploration in new parts of the configuration space and exploitation in parts of the space known to perform well; and (3) run c on one or more benchmark instances and compare its performance to the best configuration observed so far.

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. In a nutshell, given a set of parameters that can be used to modify the ordering of some aspect of the CNF, and given the value assigned to each parameter, the online configuration is performed by re-ordering clauses and literals accordingly. Notably, the value of each parameter has to be provided, and can be identified via an appropriate off-line learning step.

Given the depicted online scenario, we are restricted to information about the CNF that can be quickly gathered and that are computationally cheap to extract. Furthermore, the configuration must consider only general aspects that 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 to find a solution (or to demonstrate unsatisfiability).

In this work, we consider the possibility to order *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;
- (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;
- ($viii$) the number of times the literal is involved in clauses where all literals are negative.

The set of proposed ordering criteria is aimed at being as inclusive as possible, so that different characterising aspects of clauses and literals can be taken into account, at the same time, for the configuration process.

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 identified degrees of freedom in CNFs as parameters. This is due to the fact that orders are not natively supported by general configuration techniques [10, 16]. Results presented by Vallati et al. [22] suggest that purely categorical parametrisations are not indicated for the configuration of models, as they tend to fragment the configuration space and to introduce discontinuities. Those combined aspects make the exploration of the configuration space particularly challenging for learning approaches. For this reason, here we generate 7 continuous parameters for configuring the order of clauses, and 8 continuous parameters for configuring the order of literals in clauses. Each parameter corresponds to one of the aforementioned criteria, and they 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 the corresponding 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:

$$o_l(l) = \sum_{c \in C} (value(p, c) \times weight(c)) \tag{2}$$

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

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

Fig. 2. The example CNF non configured (top), and the configured version (bottom). 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.

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, for every clause, the involved literals are ordered following the score $o_l(v)$. Ties are broken following the order in the original CNF configuration. As it is apparent from Equation (2), a positive (negative) value of $weight(c)$ can be used to indicate that the aspect corresponding to the parameter c is important for the SAT solver, and that literals with that aspect should be listed early (late) in the clause to improve performance.

Similarly to what is presented in Equation (2) 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 (3).

$$o_C(d) = \sum_{v \in L(d)} o_l(v) \tag{3}$$

Example 1. Let us consider the CNF presented, using the DIMACS format, in Figure 2 (top). 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 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. Of course, the first line of the considered CNF is unmodified, as the DIMACS format require it to be the first, and to present information in a given order. □

The way in which the considered ordering criteria are combined, via Equations (2) and (3), gives a high degree of freedom for encoding and testing different configurations. Very specific aspects can be prioritised: for instance, it would be possible to present first clauses that are binary, and where both literals are positive, by penalising criterion 5 and giving a high positive weight to criterion 2. Furthermore, additional criteria can be added, with no need to modify or update the overall configuration framework.

4 Experimental Analysis

Our experimental analysis aims to evaluate the impact of the proposed automated approach for performing CNF configuration, on state-of-the-art SAT solvers' performance.

We selected 3 SAT solvers, based on their performance in recent SAT competitions and in their widespread use: 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 as a wrapper for extracting information from a given CNFs and, according to the parameters' value, reconfigure it and provide it as input for the SAT solver.

In designing our experimental analysis, we followed the Configurable SAT Solver Challenge (CSSC) [14]. The competition aimed at evaluating to which extent SAT solvers' performance can be improved by algorithm configuration for solving instances from a given class of benchmarks. In that, the CSSC goals are similar to the goals of this experimental analysis –i.e., assessing how performance can be improved via configuration–, thus their experimental settings are deemed to be appropriate for our analysis. However, CSSC focused on the configuration of SAT solvers' behaviour by modifying exposed parameters of solvers. In this work we do not directly manipulate the behaviour of SAT solvers via exposed parameters, but we focus on the impact that the CNF configuration can have on solvers.

Following CSSC settings, a cutoff of 5 CPU-time minutes, and a memory limit of 4 GB of RAM, has been set for each solver run on both training and testing instances. This is due to the fact that many solvers have runtime distributions with long tails [9], and that practitioners often use many instances and relatively short runtimes to benchmark solvers for a new application domain [14]. There is also evidence that rankings of solvers in SAT competitions would remain similar if shorter runtimes are enforced [11].

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**

We chose benchmark sets from the CSSC 2014 edition [14], and the benchmarks used in the Agile track of the 2016 SAT competition.[5] These two competitions provide benchmarks that can highlight the importance of configuration (CSSC) –even though a different type of configuration than the one considered in this paper–, and that include instances that have to be solved quickly (Agile). In particular, CSSC benchmarks can allow us to compare the impact of the proposed CNF configuration with regards to the solvers' configuration.

Selected CSSC 2014 benchmark sets include: Circuit Fuzz (Industrial track), 3cnf, K3 (Random SAT+UNSAT Track), and Queens and Low Autocorrelation Binary Sequence (Crafted track).[6] 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, and a similar amount of benchmarks for training.

Experiments were run on a machine equipped with Intel Xeon 2.50 Ghz processors. Each configuration process, i.e. for each pair SAT solver - benchmark set, has been given a budget of 5 sequential CPU-time days, and run on a dedicated processor.

Table 1 summarises the results of the selected SAT solvers on the considered benchmark sets. Results are presented in terms of the number of timeouts on testing instances, achieved by solvers run using either 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

[5] https://baldur.iti.kit.edu/sat-competition-2016/
[6] http://aclib.net/cssc2014/benchmarks.html

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**

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, that were achieved by configuring the solvers' behaviour [14]. In fact, these results may confirm our intuition 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. In that, it may be argued that a solver's behaviour can be controlled internally, by modifying its exposed parameters, and externally by ordering the CNF in a suitable way.

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 benefit by exploiting a solver-specific configuration, in order to mitigate any implicit bias due to the particular CNF configuration exploited. Randomly listing clauses and variables may of course remove some bias, but it can also be the case that different biases are introduced. In that sense, allowing solvers to be provided with a specifically-configured CNF may lead to a better comparison of performance. 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. In other words, there is not a single ordering that allows to maximise the performance of all the SAT solvers at once.

To better understand the impact of configuring CNFs on the runtime of solvers, Table 2 compares performance in terms of IPC score variations. The

IPC score provides a trade-off between runtime and coverage, and is used in the International Planning Competition for comparing planners' performance.

For a solver $\mathcal{C}$ and a SAT instance p, $Score(\mathcal{C}, p)$ is defined as:

$$Score(\mathcal{C}, p) = \begin{cases} 0 & \textit{if } p \textit{ is unsolved} \\ \frac{1}{1+\log_{10}(\frac{T_p(\mathcal{C})}{T_p^*})} & \textit{otherwise} \end{cases}$$

where T_p^* is the minimum amount of time required by any compared system to solve the instance, and $T_p(\mathcal{C})$ denotes the CPU time required by $\mathcal{C}$ to solve the instance p. Higher values of the score indicate better performance.

In Table 2 the performance of a solver run on the default and configured CNFs are compared. 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 most relevant aspects of the CNF configuration, we assessed the importance of parameters in the considered configurations using the fANOVA tool [12]. We observed that in most of the cases, improvements are mainly due to the effect of the correct configuration of a single criterion, rather then to the interaction of two or more criteria together. In terms of clauses, parameters controlling the weight of criteria 4 and 5 are deemed to be the most important: in other words, the number of positive (or negative) literals that are involved in a clause are a very important aspect for the performance of SAT solvers. The solver that can gain the most by ordering the clauses is Lingeling. In particular, this solver shows best performance when clauses with a large number of negative literals are listed early.

Parameters related to criteria *ii*, *vi*, and *viii* have shown to have a significant impact with regards to the literals' ordering in clauses. For Glucose and Cadical, criterion *ii* –i.e. the average size of the clauses in which the literal is involved– is the most important single criterion that has to be correctly configured. However, it is a bit hard to derive some general rules, as their impact on orderings vary significantly with regards to the solver and the benchmark. In a nutshell, they are important, but the best way to present the literals varies.

Generally speaking, also in the light of the criteria that are most important for clauses, the ordering of literals appears to be the most important in a CNF: 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 –that is also the focus of heuristic search exploited by SAT solvers. Instead, clauses from the CNF tend to have a less marked importance during the exploration of the search space, as they are related to literals mostly via lists, and are exploited only for checking satisfiability and performing unit propagation. Clauses learnt during the search process are not included in our analysis, as they are not part of the CNF –but are generated online by the solver.

Finally, we want to test if there is a single general configuration that improves the performance of a solver on any CNF, despite of the benchmark and underly-

Table 3. Results achieved by the selected solvers on the general testing set. For each solver, we show the PAR10, number of test set timeouts, and IPC score achieved when running on the default and on the CNFs configured using the general configuration. Bold indicates the best result.

Solver	Performance: default → configured	
	# timeouts	IPC score
Cadical	101 → **98**	**172.7** → 172.5
Glucose	80 → **78**	207.5 → **211.2**
Lingeling	109 → **108**	190.7 → **191.7**

ing structure. Therefore, we trained each of the considered solvers on a training set composed by an equal proportion of instances from each of the 6 benchmark sets. As for previous configurations, we gave 5 days of sequential CPU-time for each learning process, and obtained configurations have been tested on an independent testing set that includes instances from all the benchmark sets. Results are presented in Table 3.

Results on the independent testing set indicate that this sort of configuration has a very limited impact on solvers' performance. This seems to confirm our previous intuition that solvers require differently configured CNFs according to the underlying structure of the benchmark: it is therefore the case that structurally different sets of instances require a very different configuration. Intuitively, this seems to point to the fact that, in different structures, the characteristics that identify challenging elements to deal with, vary. Solvers, when dealing with different sets of benchmarks, are then sensitive to different aspects of the CNFs, that should be appropriately highlighted and configured. On the one hand, this result may be not fully satisfying, as it suggests that there is not a quick way to improve the performance of SAT solvers. On the other hand, the results of the other experiments indicate that, for real-world applications of SAT where instances share some underlying structure, there is the possibility to furtherly improve the SAT solving process by identifying a specific configuration for the solver at hand.

5 Conclusions

Previous work in the area of algorithm configuration for SAT focused on modifying the exposed parameters of SAT solvers in order to affect their behaviour. Well-known examples include the use of ParamILS for configuring SAPS and SPEAR [13] or of ReACTR for configuring LingeLing [8], as well as the dedicated design and development of highly modular and configurable SAT solvers such as SATenstein [17] that can then be tuned for a specific set of benchmarks. Algorithm configuration has also been used as a technique for selecting and com-

bining different SAT solvers into portfolios [18], and for creating suitable SAT solvers that would complement the performance of a given portfolio [24].

In this paper we proposed an approach for exploiting the fact that the order in which literals and clauses are listed in CNFs can strongly affect the performance of SAT solvers. The proposed approach allows to perform 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. In our experimental analysis we configured CNFs for improving the PAR10 performance of solvers, i.e. a tradeoff between runtime and coverage. The performed analysis, aimed at investigating how the configuration of CNFs affects the performance of state-of-the-art SAT solvers: (i) demonstrates that the automated configuration has a significant impact on solvers' performance; (ii) indicates that the configuration should be performed on specific set of benchmarks for a given solver; and (iii) highlights important aspects of CNFs, that have a potentially strong impact on the performance of solvers.

It should be noted that different metrics can be used to configure CNFs. In this work we focused on the PAR10 value, but metrics with a stronger focus on runtime, coverage, or even "quality" of generated solutions can be straightforwardly exploited in the introduced framework. Similarly, additional criteria to control the ordering of clauses and literals can be included.

We see several avenues for future work. 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. We are also interested in evaluating if ordering clauses (and literals) that are learnt during the search process of a SAT solver can be beneficial for improving performance. Finally, we plan to incorporate the re-ordering of clauses and literals into existing approaches for configuring portfolios of SAT solvers, such as SATenstein, in order to further improve performance, and to investigate the concurrent configuration of CNFs and solvers.

References

1. Audemard, G., Lagniez, J., Simon, L.: Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: Theory and Applications of Satisfiability Testing - SAT. pp. 309–317 (2013)
2. Biere, A.: Cadical, lingeling, plingeling, treengeling and yalsat entering the sat competition 2017. In: SAT competition 2017, Solver and Benchmark Descriptions (2017)
3. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
4. Cadar, C., Dunbar, D., Engler, D.: 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. pp. 209–224 (2008)
5. Cerutti, F., Vallati, M., Giacomin, M.: On the impact of configuration on abstract argumentation automated reasoning. Int. J. Approx. Reasoning **92**, 120–138 (2018)
6. Crawford, J., Baker, A.: Experimental results on the application of satisfiability algorithms to scheduling problems. AAAI pp. 1092–1097 (1994)

7. Falkner, S., Lindauer, M.T., Hutter, F.: Spysmac: Automated configuration and performance analysis of SAT solvers. In: Theory and Applications of Satisfiability Testing - SAT 2015. pp. 215–222 (2015)
8. Fitzgerald, T., Malitsky, Y., O'Sullivan, B.: Reactr: Realtime algorithm configuration through tournament rankings. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI. pp. 304–310 (2015)
9. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning **24**(1), 67–100 (2000)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION). pp. 507–523 (2011)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Tradeoffs in the empirical evaluation of competing algorithm designs. Annals of Mathematics and Artificial Intelligence **60**(1), 65–89 (2010)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of The 31st International Conference on Machine Learning. pp. 754–762 (2014)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
14. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H.H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artif. Intell. **243**, 1–25 (2017)
15. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence **206**, 79–111 (2014)
16. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: Isac–instance-specific algorithm configuration. In: Proceedings of the 9th European Conference on Artificial Intelligence ECAI. pp. 751–756 (2010)
17. KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: Satenstein: Automatically building local search SAT solvers from components. Artificial Intelligence **232**, 20–42 (2016)
18. Lindauer, M., Hoos, H., Hutter, F., Leyton-Brown, K.: Selection and Configuration of Parallel Portfolios, pp. 583–615. Springer (2018)
19. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in sat-based formal verification. Int. J. Softw. Tools Technol. Transf. **7**(2), 156–173 (Apr 2005)
20. Rintanen, J.: Engineering efficient planners with SAT. In: European Conference on Artificial Intelligence ECAI. pp. 684–689 (2012)
21. Tompkins, D.A.D., Balint, A., Hoos, H.H.: Captain jack: New variable selection heuristics in local search for sat. In: Theory and Applications of Satisfiability Testing - SAT 2011. pp. 302–316 (2011)
22. Vallati, M., Hutter, F., Chrpa, L., McCluskey, T.: On the effective configuration of planning domain models. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1704–1711 (2015)
23. Vallati, M., Serina, I.: A general approach for configuring pddl problem models. In: Proceedings of the International Conference on Automated Planning & Scheduling (ICAPS) (2018)
24. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI (2010)