Towards Better Heuristics for Solving Bounded Model Checking Problems

Anissa Kheireddine
EPITA, LRDE, Kremlin-Bicêtre, France
Sorbonne Université, UMR 7606 LIP6, Paris, France

Etienne Renault
EPITA, LRDE, Kremlin-Bicêtre, France

Souheib Baarir
Sorbonne Université, CNRS UMR 7606 LIP6, France
Université Paris Nanterre, France

Abstract

This paper presents a new way to improve the performance of the SAT-based bounded model checking problem by exploiting relevant information identified through the characteristics of the original problem. This led us to design a new way of building interesting heuristics based on the structure of the underlying problem. The proposed methodology is generic and can be applied for any SAT problem. This paper compares the state-of-the-art approach with two new heuristics: Structure-based and Linear Programming heuristics and show promising results.

2012 ACM Subject Classification Hardware → Theorem proving and SAT solving; Hardware → Model checking

Keywords and phrases Bounded model checking, SAT, Structural information, Linear Programming

Digital Object Identifier 10.4230/LIPIcs.CP.2021.12

Category Short paper

1 Introduction

Computer systems are omnipresent in our daily life. These range from the simple program that runs a microwave to the very complex software driving a nuclear power plant, passing by our smartphones and cars. Ensuring the reliability and robustness of these systems is an absolute necessity. Model-Checking [10] is one of the approaches devoted to this purpose. Its goal is to prove the absence of failure, or to show a possible one.

Model-Checking is declined into several techniques [8, 6, 19]. Among all, those called Bounded Model Checking (BMC) [5], based on Boolean satisfiability (SAT). BMC is very used for hardware formal verification in the context of electronic design automation[1], but is also applied to many other domains. The idea is to verify that a model, restricted to executions bounded by some integer k, satisfies its specification, given as a set of terms in a temporal logic. In this approach, behaviors are described as a SAT problem. The memory usage in SAT solving does not usually suffer from the well-known space explosion problem and can handle problems with thousands of variables and constraints. The complexity here is shifted to the solving time: SAT problems are NP-complete problems in general [28].

These last decades, many improvements have been developed in the context of sequential SAT solving[2] [31, 2, 25, 22, 21], to name but a few. These approaches are quite generic and are based on exploiting either dynamic information, obtained from the progress of the solving

[1] `http://fmv.jku.at/hwmcc20/index.html`
[2] Our focus here is on CDCL-like complete algorithms [34].

algorithm itself (e.g., lbd [31]), or static information, derived from the underlying structure of the SAT problem (e.g., community [2]). Little attention has been given to structural information that can be extracted and exploited from the original problem (e.g., planning, scheduling, cryptography, BMC, etc.).

Indeed, when reducing a BMC problem to SAT, crucial information is lost. As we will highlight in this work, when reintegrated, this information can be a booster for the solving process. To the best of our knowledge, this paper is the first one to exploit such insights: existing approaches working on improving SAT-based BMC [14, 17, 33, 15, 32] either focus on improving existing (generic) heuristics or on dividing efficiently the SAT problem.

This paper aims to propose a methodology (Section 4) to build new heuristics (Section 5). This methodology is generic and can improve SAT-solvers for any problem with its specific characterization. Here, we apply the proposed techniques to build efficient SAT-solvers dedicated for BMC problems. Our results (Section 6) are promising and demonstrate the interest of exploiting the information provided by the underlying problem.

2 Preliminaries

2.1 SAT problem

A propositional variable can have two possible values $\top$ (True) or $\bot$ (False). A literal is a propositional variable (x) or its negation ($\neg x$). A clause ω is a finite disjunction of literals. For a given clause ω, $V(\omega)$ denotes the set of variables composing ω. A clause with a single literal is called unit clause. A conjunctive normal form (CNF) formula F is a finite conjunction of clauses (by abuse of notation, F$= \{\omega_1, \omega_2, \dots\}$). For a given F, the set of its variables is noted V. An assignment $\mathbf{A}$ of variables of F is a function $\mathbf{A} : V \to \{\top, \bot\}$. $\mathbf{A}$ is total (complete) when all elements of V have an image by $\mathbf{A}$, otherwise it is partial. For a given formula F and an assignment $\mathbf{A}$, a clause of F is satisfied when it contains at least one literal evaluating to true regarding $\mathbf{A}$. The formula F is satisfied by $\mathbf{A}$ iff $\forall \omega \in F$, ω is satisfied. F is said to be SAT if there is at least one assignment that makes it satisfiable. It is defined as UNSAT otherwise.

Conflict Driven Clause Learning [34]. Conflict-Driven Clause Learning algorithm (CDCL) is one of the main methods used to solve Satisfiability problems and is an enhancement of the DPLL algorithm [12]. CDCL algorithm performs a backtrack search; selecting at each node of the search tree, a decision literal which is set to a Boolean value. This assignment is followed by an inference step that deduces and propagates some forced unit literal assignments (procedure called *unit propagation*). This branching process is repeated until finding a model or reaching a conflict. In the first case, the formula is answered to be satisfiable, and the model is reported, whereas in the second case, a **learnt clause** is generated (by resolution), following a bottom-up traversal of the implication graph [30] (it is called *conflict analysis*).

2.2 SAT-based Bounded Model Checking

Model checking [10] aims at checking whether a model satisfies a property. The model is usually given as a program, defined in a formal language, while the property is given as formula expressed in temporal logic (e.g., LTL [27]). A property is said to be verified if no execution in the model can invalidate it, otherwise it is violated. To achieve this verification a full traversal of the state-space, representing the behaviours of the model, is required.

An LTL property refers to atomic propositions that express a relation between some variables of the model. The model checking approach usually represents the model as a

finite-state automaton called a Kripke structure [4]. Such a structure is defined by a 4-uple $K = \langle S, s_0, T, L\rangle$ with: S a finite set of states, $s_0 \in S$ an initial state, $T \subseteq S \times S$ a transition relation, and L a labelling function that provides, for each state $s \in S$, an interpretation of an atomic proposition a denoted by $L(a)$. $L(a)$ is true iff a is satisfied in s.

Bounded Model Checking (BMC) [5, 9] refers to a model checking approach where the verification of the property is performed using a bounded traversal, i.e., a traversal of symbolic representation of the state-space that is bounded by some integer k. Such an approach does not require storing state space and hence, is found to be more scalable and useful [33, 16].

In SAT-based BMC, the BMC approach is reduced to solving a SAT problem. Given a model M, an LTL property p, and a bound k, it builds a propositional formula such that the formula is said to be satisfiable iff there exists a violation of the property (counterexample) of maximum length k. Otherwise, it is unsatisfiable and the property is verified up to length k. The encoding of this formula requires multiple steps.

First, it translates the model into a Boolean formula. The set of variables of this SAT formula can be decomposed in two disjoint subsets: $\mathcal{M}$ and $\mathcal{J}$, where $\mathcal{M}$ is a Boolean representation of the original variables of the model, while $\mathcal{J}$ is a set of fresh variables (junction variables) used to finalize the conversion into a Boolean formula[3]. Second, the property p is also translated into a SAT formula. This conversion involves $\mathcal{M}$ and $\mathcal{J}$ and introduces new fresh variables $\mathcal{F}$. Let us denote by $\mathcal{M}_p$ the set of variables of $\mathcal{M}$ involved in p, $\mathcal{J}_p$ the set of variables of $\mathcal{J}$ involved in p. With these definitions we can build $\mathcal{P} = \mathcal{J}_p \cup \mathcal{M}_p \cup \mathcal{F}$, the set of the variables used to encode the property. Finally, the two previous steps are combined in the following formula:

$$\underbrace{I(s_0) \bigwedge T(s_0, s_1) \bigwedge \cdots \bigwedge T(s_{k-1}, s_k)}_{Model} \bigwedge \underbrace{P_k}_{Property} \tag{1}$$

It can be observed that both the transition relation of the model and the property have been unrolled up to the bound k. The left-side denotes the model constraints while the right-side is related to the property constraints. $I(s_0)$ are the initialization constraints that verify if s_0 is the the initial state of K, s_i represents the reachable states (in K) in i steps using the transition relation T.

3 Related work

Most of the works on improving SAT solving focus on building heuristics to detect and exploit relevant information during the solving process. Usually CDCL-like solvers maintain a database of interesting learnt clauses in order to speedup the solving. Good performances of these solvers are associated to their ability to preserve interesting clauses while maintaining a reasonable size for the database. So, the issue here is to find the best trade-off between what is considered to be a relevant information and how much of this information must be kept. Some of the state-of-the-art heuristics that are used in the best solvers of the world[4] are described below:

Size bounded learning [13]. This approach protects learnt clauses that are sized less than a certain threshold.

[3] For instance, a 32 bits variable will be represented as 32 Boolean variables, and the logical operators ($\wedge$, $\vee$, $\Longrightarrow$,...) will rely on fresh variables for their representation.

[4] According to the results of the SAT competitions (`https://satcompetition.github.io/2020/results.html`)

Relevant bounded learning [18]. This approach discards learnt clauses when they are no longer relevant according to some metric. For instance, a learnt clause is considered as not relevant if the number of its literals that are assigned (w.r.t the current global assignment) exceeds predefined threshold.

Literal block distance (LBD) [31]. LBD is a positive integer, that is used as a learnt clause quality metric in almost all competitive sequential CDCL-like SAT-solvers. The LBD of a clause is the number of different decision levels on which variables of the clause have been assigned. Hence, the LBD of a clause can change overtime and it can be (re)computed each time the clause is fully assigned. If LBD(ω)=n, then the clause ω spans on n propagation blocks, where each block has been propagated within the same decision level. Intuitively, variables in a block are closely related. Learnt clauses with lower LBD score tend to have higher quality: Glue Clauses [31] have LBD score of 2 and are the most important type of learnt clauses.

Community structure [2]. In this approach, the formula at hand is represented as a graph. The shape of this graph is then analyzed to extract community structure: roughly speaking, variables belonging to the same community are more densely interconnected than variables in different communities. Existing studies [2, 3] showed that using the community structure to detect new learnt clauses results in an improvement of the performance of the solver.

Symmetries [11]. SAT problems often exhibit symmetries, and not taking them into account forces solvers to needlessly explore isomorphic parts of the search space. Symmetries can help learning interesting clauses that the classical learning approaches fail to capture [25, 1].

translates Despite the generic character of these heuristics, they have been tuned by some research works in the case of the BMC problem. We can cite [29, 33, 17, 32] that present a variety of optimizations such as: variable ordering heuristics, branching heuristics, studying the symmetry structure of the BMC formula (1). Other works went for a decomposition of the BMC formula into simpler and independent subproblems showing promising results [14, 15].

4 Studying the characteristics of BMC problem

4.1 Intuition

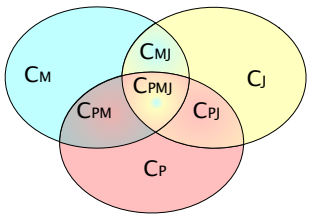

Figure 1 The seven disjunctive classes of clauses according to the combination of variables they handle: model variables (blue), fresh/junction variables (yellow) and property variables (red).

The notion of what is a relevant information is quite unclear for SAT procedures. Most of the existing techniques are generic and try to perform well on any studied formula, without taking a real care of its origin (see Section 3). However, taking the structural information of the original problem into account will eventually lead to an improvement of the solving process. In this paper we explore this idea in the particular case of the BMC problem.

The starting point is to study the characteristics of the BMC problem. As a first insight, one can observe that the BMC problem can be trivially divided in two parts: the model and the property. However, when studying the learnt clauses w.r.t. this partitioning no relevant information could be inferred. Indeed, a learnt clause usually spans on variables belonging to both the model and the LTL property at the same time. So, we suggest here a sharper classification based on the clause variables.

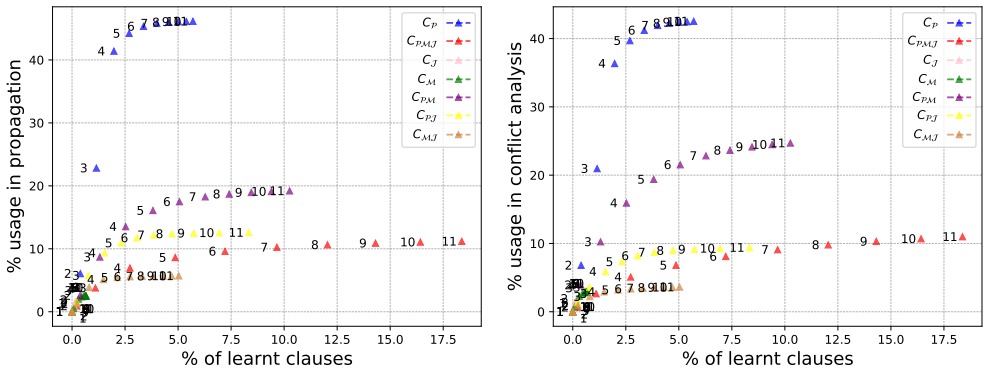

Figure 2 Measures on the *training benchmark* showing learnt clauses usage in propagation (left) and conflict analysis (right) phases. Each class of clauses is colored and annotated by its LBD value.

A clause can be composed of variables belonging to $\mathcal{M}$, $\mathcal{P}$ or $\mathcal{J}$. Let us denote by $C_X = \{\omega \in L \,|\, \forall v \in V(\omega), v \in X\}$ the classes highlighted in Figure 1, where X is either $\mathcal{P}$ (the property), $\mathcal{M}$ (the model), $\mathcal{J}$ (the fresh variables for the model), $\mathcal{PJ}$ (property and fresh variables), $\mathcal{PM}$(property and model variables), $\mathcal{MJ}$ (model and fresh variables) or $\mathcal{PMJ}$ (property, model and fresh variables). We can now study the usefulness of each of the above classes of clauses in the solving process.

4.2 Measures

Let us first precise our setup: all experiments of the paper were conducted on a benchmark of 400 SMV instances. The instances came from the SMV hardware verification problems [7], the BEEM [26] and the RERS Challenge benchmakrs[5]. The SMV instances were translated into DIMACS format for various bound values k={20, 40, 60,. . . , 4000, 6000} [20]. Each instance includes an LTL property provided with the model (46% of Safety property, 30% Guarantee, 14% Persistence, and 10% Recurrence according to the hierarchy of Manna&Pnueli [24]).

To perform our analysis, we developed a tool called BMC-tool[6] that integrates NuSMV tool [7] as a front-end and MapleCOMSPS [23] SAT-solver as a back-end. This solver is the winner of the main track of the SAT competition 2016 and was used as core engine for the best solvers in the last 5 years. The success of this solver relies on the management of the learnt clauses with three different databases according to the LBD value of the clauses: **core** (LBD ≤ 3) for the really important ones (never deleted), **tier-2** (LBD ≤ 6) for not-yet-decided clauses, and **local** database for the remaining clauses. Clauses in **tier-2** can be promoted to the core database or downgraded to local database while those of **local** database can either be promoted to **tier-2** or permanently deleted.

We run our tool on 25% of the whole benchmark (100 instances), the so-called *training benchmark*. For all instances, we logged the information related to each learnt clauses when used in the *unit propagation* and *conflict analysis*. These information are the LBD of the clause and its class (C_X). The results are depicted in Figure 2.

The x-axis reports the cumulative mean percentage of learnt clauses for the *training benchmark* and the y-axis corresponds to the cumulative mean usage percentage (left-side

[5] `https://tinyurl.com/29a4jcme`
[6] `https://gitlab.XXX`

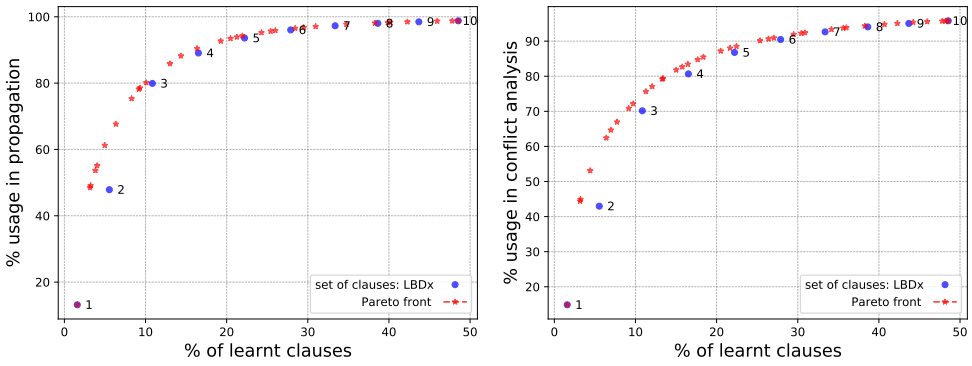

Figure 3 Measures of learnt clauses usage during propagation (left) and conflict analysis (right) phases. Blue dots denote LBD while red points depict the Pareto front of H_{LP} strategy.

for *unit propagation* and right-side for *conflict analysis*). Each point represents the used percentage of learnt clauses of a certain LBD (from 1 to 10) for a certain class. For example, the purple triangle with left annotation 4 shows 2.5% of learnt clauses of class $\mathcal{PM}$ have an LBD$\leq$4 and are used in 13% of the *unit propagation* time (resp. 16% on *conflict analysis*).

We observe that $C_{\mathcal{P}}$ have a significant usage (around 45%) with a total coverage of around 5% in both *propagation* and *conflict analysis*. Therefore, these clauses seem to be good candidates for being considered as a relevant information.

Consider now Figure 3 while ignoring momentarily the red points. This figure depicts the same information as Figure 2 but without clause classification. Here, we observe that the default strategy for characterizing relevant information in MapleCOMSPS, i.e., LBD ≤ 3 (identified by the blue point) covers 75% (mean value between *propagation* and *conflict analysis* curves) of utilization for a total of 11% of the learnt clauses. It appears then that more than half of what is considered as a relevant information came from $C_{\mathcal{P}}$.

This measure comforts our thoughts that the performances of the SAT-solver are conditioned by a certain class of clauses. Our fine grained classification reveals that property clauses seem to be the more pertinent ones.

5 Heuristics for BMC

Based on the previous study, we present our ideas for improving the solving of SAT-based BMC problem. Our proposal is to identify and protect (from deletion) new sets of clauses that are relevant for the solving of a BMC problem. We introduce for this, two new heuristics:

Structural heuristic (H_S). The intuition behind this heuristic is to encourage the solver to focus on $C_{\mathcal{P}}$ clauses (probably these are used to falsify the property). To achieve this, we augment the **core** database of MapleCOMSPS by a subset of $C_{\mathcal{P}}$: we take all clauses of $C_{\mathcal{P}}$ that have an LBD ≤ 5. Indeed, after this threshold, the curves of Figure 2 seem to initiate an inflection that suggests that no more relevant information is captured.

Linear programming heuristic (H_{LP}). This approach aims at predicting the usefulness of each learnt clause mathematically. It determines the adequate LBD value for each class of clauses by maximizing the total usage of learnt clauses while minimizing their number. This is achieved by solving a linear programming system, that will provide multiple solutions specifying the suitable value of LBD for each class.

The linear program is written such that the objective is an aggregation function of the two above criteria. The constraints restrict the search-space to select at most one LBD

value per class of clauses (input information is captured from Figure 2). The description of the linear system needs the introduction of the following notations:

- u_i^j: the percentage of learnt clauses (x-axis) with LBD$\leq i$ of class j.
- v_i^j: the percentage usage of learnt clauses (y-axis) with LBD$\leq i$ of class j.
- x_i^j: a Boolean variable representing the decision variable of the linear system. It takes the value 1 if the LBD$\leq i$ is chosen for the class of clauses j, 0 if not.
- C=$\{\mathcal{P}, \mathcal{M}, \mathcal{J}, \mathcal{PM}, \mathcal{PJ}, \mathcal{MJ}, \mathcal{PMJ}\}$ denotes the set of classes.

Hence, our modeling of the optimisation problem is as follows:

$$\textbf{maximize } \mathbf{f}_\mu = -\mu \overbrace{\sum_{i=1}^{10} \sum_{j \in C} u_i^j \; x_i^j}^{\mathbf{O_1}} + (1-\mu) \overbrace{\sum_{i=1}^{10} \sum_{j \in C} v_i^j \; x_i^j}^{\mathbf{O_2}}$$

$$\text{subject to} \begin{cases} \sum_{i=1}^{10} x_i^j \leq 1 \quad \forall j \in C & \textit{//At most one LBD value per class} \\ x_i^j \in \{0,1\} \quad \forall i \in [\![1;10]\!], \; \forall j \in C \end{cases}$$

$\mathbf{f}_\mu$ is the aggregation function, defined as a weighted sum[7], and parameterized with μ ($0 \leq \mu \leq 1$): the term $\mathbf{O_1}$ represents the number of learnt clauses that should minimized and the term $\mathbf{O_2}$ is the used percentage that should be maximized. Then, this bi-objective optimization problem is converted to a single maximization problem using the parameter μ as described above. Solving this system (using data collected on the training benchmark) with various values for μ allows to draw a Pareto front (possibly not optimal).

The Pareto front highlighted in Figure 3 (red points) is obtained by solving this system using an increment of 0.01 for the parameter μ: each of these points corresponds to a configuration of the form: class $C_\mathcal{P}$ with LBD x, class $C_\mathcal{J}$ with LBD y, etc.

Our first observation is that the red points dominate the blue ones (representing the LBD-based approach of MapleCOMSPS) on both graphics of Figure 3. It appears then that we can improve the performance of the standard approach by choosing one of this point as a basis for detecting new relevant information: red points located between blue points tagged 3 and 4 (i.e., those with LBD$\leq$3 and LBD$\leq$4, respectively) are the best candidates. They are located at the inflection on both *propagation* and *conflict analysis* curves. Among these points, we found that the best promising one covers 83% on *unit propagation* (resp. 81% on *conflict analysis*) for a total of 15% of learnt clauses. This point characterizes the clauses with the following properties: LBD$\leq$3 for all classes but $C_\mathcal{P}$ and $C_\mathcal{J}$. These latter have the configurations LBD$\leq$4 and LBD$\leq$9, respectively. Therefore, this confirms the usefulness of clauses with LBD$\leq$3 but also identifies new interesting ones.

6 Experimental results

All the experiments have been executed on the full benchmark presented in Section 4 on an Intel Xeon@2.40GHz machine with 12 processors and 64 Go of memory and a time limit of 6000 seconds[8]. Table 1 details the results of our experiments using MapleCOMSPS with H_S or H_{LP} heuristics . The table displays, the number of UNSAT and SAT solved instances,

[7] Other aggregation functions can be used, for example: Ordered Weighted Average, Choquet integral,...

[8] For a description of our setup, detailed results and code, see `https://akheireddine.github.io/`

Solver	UNSAT	SAT	TOTAL	PAR-2	CTI (279)	Cumulated time
MapleCOMSPS	173	116	289	423h58	44h08	238h59
MapleCOMSPS-LBD≤4	169	**118**	287	429h33	**43h12**	241h13
MapleCOMSPS-H_S	174	**118**	292	418h10	43h24	**238h10**
MapleCOMSPS-H_{LP}	**177**	**118**	**295**	**413h53**	45h02	238h53

Table 1 Comparison between state-of-the-art MapleCOMSPS solver and H_S and H_{LP} heuristics. MapleCOMSPS-LBD≤4 uses a strategy where learnt clauses with LBD≤4 are considered as relevant.

the total number of solved instances, the PAR-2 metric[9] used in SAT competitions, the CTI metric[10] and the cumulated time. H_S and H_{LP} don't include pre-processing time (took 44h27) and the Pareto front computation in H_{LP} doesn't take more than one second.

We observe that MapleCOMSPS solves 289 instances with a PAR-2 of 423h58. Besides, augmenting the **core** database to protect learnt clauses with LBD≤4 (MapleCOMSPS-LBD≤4) seems to deteriorate the performances: 2 instances less with a PAR-2 of 429h33 (5 hours slower than the original solver). This result shows that increasing the number of relevant clauses based entirely on the LBD cannot bring better performances.

The two next lines display the results of our heuristics. It appears that both of these strategies perform better than state-of-the-art: MapleCOMSPS-H_S solves 1 UNSAT and 2 SAT more while MapleCOMSPS-H_{LP} solves 4 UNSAT and 2 SAT more. The PAR-2 of these two heuristics shows a significant improvement with a gain of (at least) 6 hours.

Thus, the two presented heuristics demonstrate the importance of the information captured by $C_{\mathcal{P}}$, since it is used by both of them: when performing the model-checking approach, a synchronous product between the Kripke structure and the (automaton of the) property is executed. Forcing the SAT procedure to consider property clauses will eliminate invalid paths in the property automaton, leading to a smaller synchronized product, i.e the state-space size is reduced efficiently. Also, it appears that H_{LP} captures another important information with $C_{\mathcal{J}}$ clauses: it is composed of fresh variables that make the connection between the property and the model. Consequently, they also help to compute information related to the synchronous product.

7 Conclusion and future work

Our journey towards building new heuristics for SAT procedures started with the observation that the relevant information used by SAT-solvers can be refined. We proposed a generic methodology to classify learnt clauses and we applied it to the special case of BMC. These learnt clauses have been classified according to their meaning in the original problem which helped us to suggest two heuristics (H_S and H_{LP}) based on the information carried by the LTL property. The two heuristics improve the state-of-the-art approach, with the particularity of H_S to have a structural reasoning behind. In the other hand, the procedure used to build H_{LP} relies on a mathematical reasoning.

Future work aims to refine the proposed classification by exploiting the specification of the property or the synchronicity of the model. Moreover, we would like to propagate this idea to offer new sharing strategies on parallel SAT-solvers. And finally, building a SAT-solver to exploit exclusively structural information of the original problem is in our perspectives.

[9] PAR-k is the penalised average runtime, counting each timeout as k times the running time cutoff.
[10] Cumulated execution Time of the Intersection for instances solved by all solvers

References

1 F.A. Aloul, K.A. Sakallah, and I.L. Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006. doi:10.1109/TC.2006.75.

2 Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of sat formulas. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 410–423, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

3 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Community structure in industrial sat instances, 2019. arXiv:1606.03329.

4 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

5 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking, 12 2003. doi:10.1016/S0065-2458(03)58003-2.

6 J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142–170, 1992. doi:https://doi.org/10.1016/0890-5401(92)90017-A.

7 A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

8 E. Clarke, K. McMillan, S. Campos, and V. Hartonas-Garmhausen. Symbolic model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 419–422, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

9 Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001. doi:10.1023/A:1011276507260.

10 Edmund Clarke, E. Emerson, and Joseph Sifakis. Model checking. *Communications of the ACM*, 52, 11 2009. doi:10.1145/1592761.1592781.

11 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, page 148–159, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

12 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. doi:10.1145/368273.368557.

13 Daniel Frost and Rina Dechter. Dead-end driven learning. *Proceedings of the National Conference on Artificial Intelligence*, 1, 08 2000.

14 Malay Ganai and Aarti Gupta. Tunneling and slicing: Towards scalable bmc. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, page 137–142, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1391469.1391507.

15 Malay Ganai, Aarti Gupta, Zijiang Yang, and Pranav Ashar. Efficient distributed sat and sat-based distributed bounded model checking. *International Journal on Software Tools for Technology Transfer*, 8:387–396, 08 2006. doi:10.1007/s10009-005-0203-z.

16 Malay K. Ganai. Sat-based scalable formal verification solutions. In *Series on Integrated Circuits and Systems, Springer-Verlag New York*, 2007.

17 Malay K. Ganai. Propelling SAT and sat-based BMC using careset. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 231–238. IEEE, 2010. URL: http://ieeexplore.ieee.org/document/5770954/.

18 Matthew L. Ginsberg and David A. McAllester. Gsat and dynamic backtracking. In Alan Borning, editor, *PPCP*, volume 874 of *Lecture Notes in Computer Science*, pages 243–265. Springer, 1994. URL: http://dblp.uni-trier.de/db/conf/ppcp/ppcp94-lncs.html#GinsbergM94.

19 Gerard J. Holzmann. Explicit-state model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 153–171, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-10575-8_5.

20 Paul Jackson and Daniel Sheridan. Clause form conversions for boolean circuits. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing*, pages 183–198, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

21 Sima Jamali and David Mitchell. Centrality-based improvements to cdcl heuristics. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, pages 122–131, Cham, 2018. Springer International Publishing.

22 George Katsirelos and Laurent Simon. Eigenvector centrality in industrial sat instances. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, pages 348–356, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

23 J. Liang, Vijay Ganesh, P. Poupart, and K. Czarnecki. Learning rate based branching heuristic for sat solvers. In *SAT*, 2016.

24 Z. Manna and A. Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *PODC '90*, 1990.

25 Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. Cdclsym: Introducing effective symmetry breaking in sat solving. In *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'18)*, volume 10805 of *Lecture Notes in Computer Science*, pages 99–114, Thessaloniki, Greece, April 2018. Springer.

26 Radek Pelánek. Beem: Benchmarks for explicit model checkers. In Dragan Bošnački and Stefan Edelkamp, editors, *Model Checking Software*, pages 263–267, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

27 Kristin Y. Rozier. Survey: Linear temporal logic symbolic model checking. *Comput. Sci. Rev.*, 5(2):163–203, May 2011. doi:10.1016/j.cosrev.2010.06.002.

28 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, page 216–226, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804350.

29 Ofer Shtrichman. Tuning sat checkers for bounded model checking. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 480–494, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

30 João P. Marques Silva and Karem A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, page 220–227, USA, 1997. IEEE Computer Society.

31 Laurent Simon and Gilles Audemard. Predicting Learnt Clauses Quality in Modern SAT Solver. In *Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, Pasadena, United States, July 2009. URL: https://hal.inria.fr/inria-00433805.

32 Chao Wang, HoonSang Jin, Gary D. Hachtel, and Fabio Somenzi. Refining the sat decision ordering for bounded model checking. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, page 535–538, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/996566.996713.

33 Emmanuel Zarpas. Simple yet efficient improvements of sat based bounded model checking. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design*, pages 174–185, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

34 Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '01, page 279–285. IEEE Press, 2001.