Towards Better Heuristics for Solving Bounded Model Checking Problems

Anissa Kheireddine
EPITA, LRE, Kremlin-Bicêtre, France
Sorbonne Université, UMR 7606 LIP6, Paris, France
Contact: anissa.kheireddine@lrde.epita.fr

Etienne Renault
EPITA, LRE, Kremlin-Bicêtre, France
Contact: renault@lrde.epita.fr

Souheib Baarir
Sorbonne Université, CNRS UMR 7606 LIP6, France
Université Paris Nanterre, France
(now at EPITA, LRE)
Contact: souheib.baarir@lip6.fr

Abstract

This paper presents a new way to improve the performance of the SAT-based bounded model checking problem on sequential and parallel procedures 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 approaches with two new heuristics for sequential procedures: Structure-based and Linear Programming heuristics. We extend these study and applied the above methodology on parallel approaches, especially to refine the sharing measure which shows promising results.

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

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

Digital Object Identifier 10.4230/LIPIcs.CONSTRAINTS.2022.

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 [14] 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 [12, 9, 30]. Among all, those called Bounded Model Checking (BMC) [8], 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

[1] `http://fmv.jku.at/hwmcc20/index.html`

problem [15] 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 [44].

These last decades, many improvements have been developed in the context of sequential SAT solving[2] [47, 3, 40, 34, 33], 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 algorithm itself (e.g., LBD [47]), or static information, derived from the underlying structure of the SAT problem (e.g., community [3]). 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. This paper is the one of the first to exploit such insights: existing approaches working on improving SAT-based BMC [21, 24, 50, 22, 48] either focus on improving existing (generic) heuristics or on dividing efficiently the SAT problem.

This paper is an extension of our work published at CP'21 [36] where we proposed a new methodology to build new SAT heuristics oriented for the BMC problem. The methodology is generic and can improve SAT solvers for any problem with its specific characterization. In the CP'21 paper, we applied 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 primary problem.

In addition to the above, we augment the benchmark and integrate our heuristics on a newly sequential Kissat-MAB [43] SAT solver[3] and applied the idea of identifying relevant information in the context of parallel SAT procedures. We implemented our ideas on top of the winner of the parallel SAT competition 2022[4], ParKissat-RS. In order to identify this information, Section 8 first describes the classical architecture of a parallel SAT solver. Based on this architecture, we propose in Section 9, a new heuristic tailored for solving the problem in parallel. More precisely, this new heuristic specifies the information to communicate between the various threads. Further, we combined these parallel heuristics to the sequential ones presented in CP'21 paper [36].

The paper is organized as follow. Section 4 aims to propose a methodology to build new sequential heuristics detailed in Section 5. The result of these heuristics are presented in Section 6. Section 8 recalls parallel state-of-the-art approaches and sets our motivations. Section 9 exposes our sharing heuristic dedicated for the BMC problem. Section 11 presents the evaluation of the parallel heuristic as well as the evaluation of their combination with sequential ones.

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

[2] Our focus here is on CDCL-like complete algorithms [52].
[3] Winner of the SAT competition 2021
[4] https://satcompetition.github.io/2022/

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 [52]. 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 [17]. 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 [46] (a procedure called *conflict-analysis*).

2.2 SAT-based Bounded Model Checking

Model checking [14] 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 [42]). 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 [6]. 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) [8, 13] 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 [50, 23].

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, Figure 1 illustrates these steps using counter-bit alike problem.

In Figure 1, the model's automaton **(1)** and the LTL specification are expressed through an SMV language **(2)**. The encoding starts by translating the model into a Boolean formula: the initial states **(3.1)** and the transition relation **(3.2)**. In the example, four Boolean variables x, y, z and w are initialized with $\bot$ for x and y and $\top$ for the others. The transition relation is encoded as constraints expressing the next value each of the variables will take up to bound **k=2**. The set of variables of this SAT formula can be decomposed in two disjoint subsets: $\mathcal{M}'$ and $\mathcal{T}'$, where $\mathcal{M}'$ is a Boolean representation of the original variables

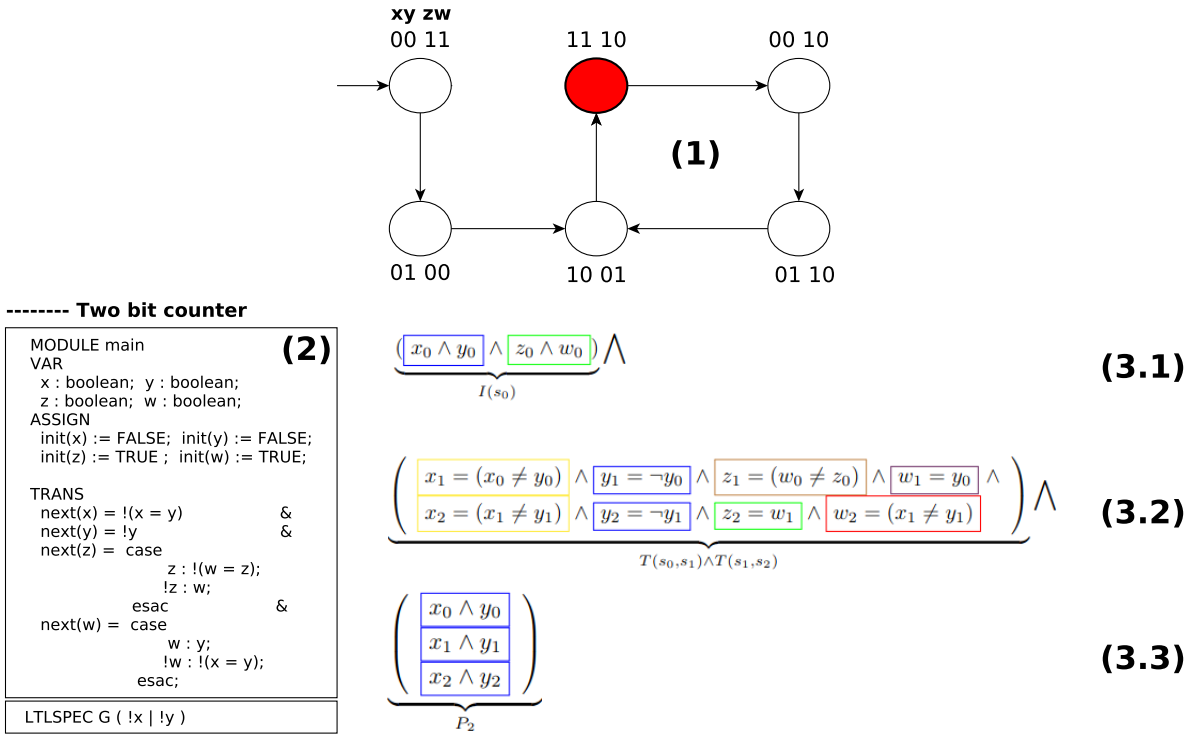

Figure 1 Two bit counter example unrolled up to bound k=2

of the model (x, y, z and w), while $\mathcal{J}'$ is a set of auxiliary variables (junction variables) used to finalize the conversion into a Boolean formula[5]. Second, the property **(3.3)** p is translated into a SAT formula. This conversion involves $\mathcal{M}'$ and $\mathcal{J}'$ and introduces new auxiliary 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 three disjoint sets:

- $\mathcal{P} = {\mathcal{J}'}_p \cup {\mathcal{M}'}_p \cup \mathcal{F}$, the set of the variables used to encode the property (i.e, x and y),
- $\mathcal{M} = \mathcal{M}' \backslash {\mathcal{M}'}_p$, the set of variables that encode the model and are not involved in the LTL specification, and
- $\mathcal{J} = \mathcal{J}' \backslash {\mathcal{J}'}_p$, the set of auxiliary variables that are not involved in the LTL specification.

The encoding of a BMC problem is 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 (k=2 in the example 1). 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.

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

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[6] are described below:

Size bounded learning [20]. This approach protects learnt clauses that are sized less than a certain threshold.

Relevant bounded learning [25]. 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) [47]. It 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 [47] have LBD score of 2 and are the most important type of learnt clauses.

Community structure [3]. 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 [3, 4] showed that using the community structure to detect new learnt clauses results in an improvement of the performance of the solver.

Symmetries [16]. 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 [40, 2].

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 [45, 50, 24, 48] 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 sub-problems showing promising results [21, 22].

4 Studying the characteristics of BMC problem

4.1 Intuition

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

[6] According to the results of the SAT competitions (`https://satcompetition.github.io/2022/`)

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 BMC problems 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.

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 2, where X is either $\mathcal{P}$ (the property), $\mathcal{M}$ (the model), $\mathcal{J}$ (the auxiliary variables for the model), $\mathcal{PJ}$ (property and auxiliary variables), $\mathcal{PM}$ (property and model variables), $\mathcal{MJ}$ (model and auxiliary variables) or $\mathcal{PMJ}$ (property, model and auxiliary variables). The colored boxes of the formula in Figure 1 label a clause according to the variable it handles: yellow boxes define $\mathcal{PJ}$ clauses that join property $\mathcal{P}$ and auxiliary variables $\mathcal{J}$ (used to convert the $\neq$ operator), brown boxes are for $\mathcal{MJ}$ clauses containing model $\mathcal{M}$ and auxiliary variables $\mathcal{J}$,...etc. We can now study the usefulness of each of the above classes of clauses in the solving process.

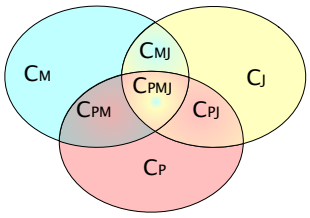

Figure 2 The seven disjunctive classes of clauses according to the combination of variables they handle: model variables (blue), auxiliary/junction variables (yellow) and property variables (red).

Discussion. In this section, we proposed a classification of the variables, and thus a classification of the clauses, based on the underlying BMC problem. The methodology presented here can be applied for any problem provided that the relevant information is captured when converting the original problem into a Boolean SAT problem. For instance, one can consider the graph coloring problem, used for register allocation. Basically this process assigns one variable per register. This is achieved by translating an interference graph into a set of constraints. During this translation, the information about "critical" variables is lost, i.e. which node is in conflict with most of others. One could use this information to split the set of variables into multiple classes. Thus, a clause could be characterized by summing the degree of each of its variables.

4.2 Measures

Let us first precise our setup: all experiments of the paper were conducted on a benchmark of 620 SMV instances. The instances came from the SMV hardware verification problems [11], the BEEM [41], the RERS Challenge benchmarks[7] and the HWMC Competition 2020[8]. The SMV instances were translated into DIMACS format [32] for various bound values k={20, 40, 60,..., 4000, 6000}. Each instance includes an LTL property provided with the model (50% of Safety property, 27% Guarantee, 3% Persistence and 10% Recurrence according to the hierarchy of Manna&Pnueli [39])). Some LTL properties have been generated using Spot [18] (1% Obligation and 9% Reactivity) and we omitted trivial instances that runs less than 1 second on MapleCOMSPS solver.

[7] `https://tinyurl.com/29a4jcme`
[8] http://fmv.jku.at/hwmcc20/

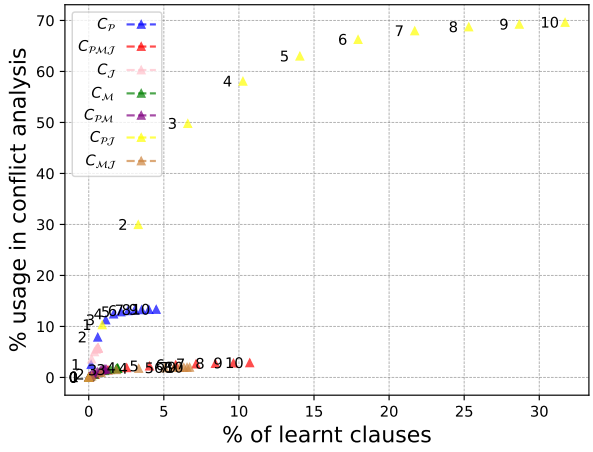

Figure 3 Measures on the *training benchmark* with MapleCOMSPS solver, showing learnt clauses usage in *conflict-analysis* phase. Each class of clauses is colored and annotated by its LBD value.

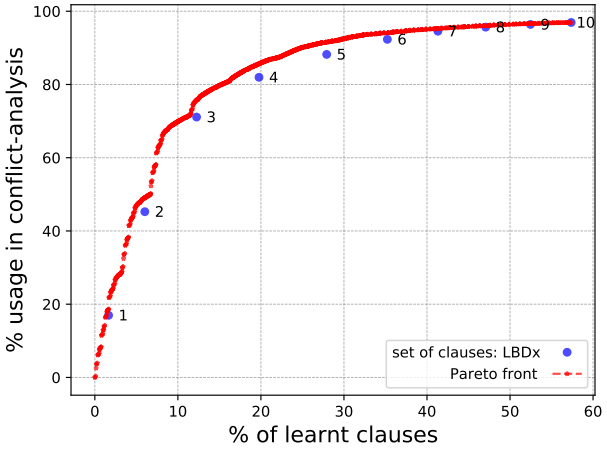

Figure 4 Measures of learnt clauses usage with MapleCOMSPS solver, during *conflict-analysis* phase. Blue dots denote LBD while red points depict the Pareto front of H_{LP} strategy.

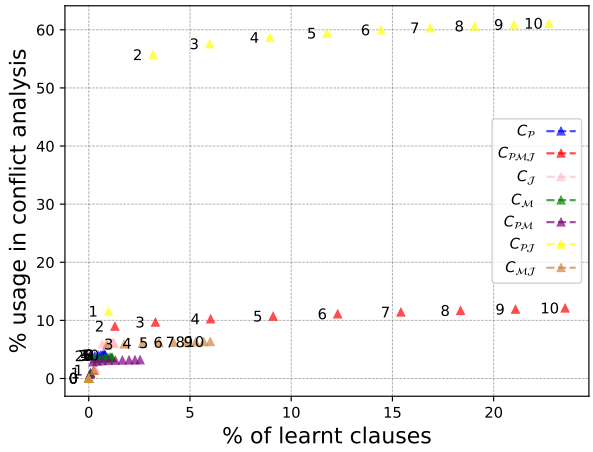

Figure 5 Measures on the *training benchmark* with Kissat-MAB solver, showing learnt clauses usage in *conflict-analysis* phase. Each class of clauses is colored and annotated by its LBD value.

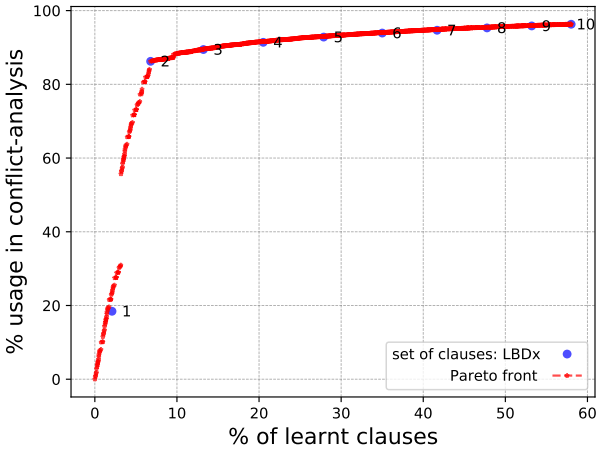

Figure 6 Measures of learnt clauses usage with Kissat-MAB solver, during *conflict-analysis* phase. Blue dots denote LBD while red points depict the Pareto front of H_{LP} strategy.

To perform our analysis, we developed a tool called BSaLTic[9] that integrates NuSMV tool [11] as a front-end and a CDCL SAT solver (MapleCOMSPS [38] and Kissat-MAB [43]) as a back-end. These solvers are the winner of the main track of the SAT competitions (2016 for MapleCOMSPS and 2021 for Kissat-MAB). The success of these solvers relies on the management of the learnt clauses with three different databases according to the LBD value of the clauses: **core** (LBD≤2 in Kissat-MAB and LBD≤3 in MapleCOMSPS) 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 23% of the whole benchmark (140 instances), the so-called *training benchmark*. For all instances, we logged the information related to each learnt clauses when used in the *conflict-analysis*. These information are the LBD of the clause and its class (C_X). The results are depicted in Figure 3 for *conflict-analysis* step using MapleCOMSPS solver and Figure 5 when using Kissat-MAB.

The x-axis of Figure 3 and Figure 5 reports the cumulative mean percentage of learnt clauses for the *training benchmark* and the y-axis corresponds to the cumulative mean usage percentage 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, in Figure 3, the yellow triangle with left annotation 4 shows that 10% of learnt clauses of class $\mathcal{PJ}$ have an LBD≤4 and are used in 58% of the time. We observe in this figure that $C_{\mathcal{PJ}}$ have a significant usage (around 60%) with a total coverage of around 20%. Therefore, these clauses seem to be good candidates for being considered as a relevant information. Figure 5 depicts similar information in *conflict-analysis* phase when using Kissat-MAB. The same observation can be made on the importance of $C_{\mathcal{PJ}}$ clauses where their usage is around 60% for 15% of learnt clauses.

Consider now Figure 4 while ignoring momentarily the red points. This figure depicts the same information as Figure 3 but without clauses classification. Here, we observe that the default strategy for characterizing relevant information in MapleCOMSPS, i.e., LBD≤3 (identified by the blue point) covers 70% of utilization for a total of 12% of the learnt clauses. It appears then that more than half of what is considered as a relevant information came from $C_{\mathcal{PJ}}$. In the same way, Figure 6 of Kissat-MAB (default strategy for characterizing relevant information are those of LBD≤2), more than half (55%) of what is considered as relevant information came from $C_{\mathcal{PJ}}$ (in Figure 6 the blue point tagged with 2 covers 86% of usage for a total of 7% of involved learnt clauses).

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 clauses implying the property 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

[9] `https://akheireddine.github.io/`

to focus on clauses that involve the property ($\mathcal{P}$ variables) such as $C_{\mathcal{PJ}}$. Indeed, a large model-checking procedures rely on the structure of the LTL property. To integrate this idea on a SAT-based approach, we augment the **core** database of the solver by a subset of $C_{\mathcal{PJ}}$, we take all clauses of $C_{\mathcal{PJ}}$ that have an LBD≤5 (resp. LBD≤4) with MapleCOMSPS engine (resp. Kissat-MAB). Indeed, after this threshold, the curve of Figure 3 (resp. Figure 5) seems 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 Figures 3,6). 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≤i of class j.
- v_i^j: the percentage usage of learnt clauses (y-axis) with LBD≤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≤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[10], 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 of MapleCOMSPS (resp. Kissat-MAB), highlighted with red points in Figure 4 (resp. Figure 6) 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) on both graphics of Figures 4 and 6. 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

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

tagged 3 and 4 for MapleCOMSPS (i.e., those with LBD≤3 and LBD≤4, respectively) and red points located between blue points tagged 2 and 3 for Kissat-MAB are the best candidates. They are located at the inflection on their respective curves. Among these points, we found that in:

MapleCOMSPS: the best promising one covers 81.8% on *conflict-analysis* for a total of 16.5% of learnt clauses. This point characterizes the clauses with properties on first line of Table 1: LBD≤3 for all classes but $C_{\mathcal{P}}$, $C_{\mathcal{J}}$ and $C_{\mathcal{PJ}}$. These latter have the configurations LBD≤4, LBD≤10 and LBD≤5, respectively.

Kissat-MAB: The promising point covers 87.3% on *conflict-analysis* for a total of 9.57% of learnt clauses. It characterizes the clauses with the following configuration resumed in Table 1: LBD≤3 for $C_{\mathcal{M}}$, $C_{\mathcal{J}}$ $C_{\mathcal{PM}}$ and $C_{\mathcal{PJ}}$ and LBD≤2 for remaining classes.

Therefore, this confirms the usefulness of clauses with LBD≤3 (resp. LBD≤2) on MapleCOMSPS solver (resp. Kissat-MAB solver) but also identifies new interesting ones.

Solver	$C_{\mathcal{J}}$	$C_{\mathcal{M}}$	$C_{\mathcal{MJ}}$	$C_{\mathcal{PM}}$	$C_{\mathcal{P}}$	$C_{\mathcal{PMJ}}$	$C_{\mathcal{PJ}}$	Default CORE
MapleCOMSPS	**10**	3	3	3	**4**	3	**5**	3
Kissat-MAB	**3**	**3**	2	**3**	**3**	2	**3**	2

Table 1 Configurations computed using H_{LP} on MapleCOMSPS and Kissat-MAB training information.

6 Sequential Experiments

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. Table 2 details the results of our experiments using MapleCOMSPS and Kissat-MAB with H_S or H_{LP} heuristics. The table displays, the number of UNSAT and SAT solved instances, the total number of solved instances, the cumulated time (PAR-1), the PAR-2 metric[11] used in SAT competitions, the CTI metric[12] (it consists of 357 instances for MapleCOMSPS results and 433 for Kissat-MAB). H_S and H_{LP} do not include pre-processing time (took 68h for MapleCOMSPS and 47h for Kissat-MAB) and the Pareto front computation in H_{LP} does not take more than one second.

We observe that MapleCOMSPS solves 368 instances with a PAR-2 of 943h. Besides, augmenting the **core** database to protect learnt clauses with LBD≤4 (MapleCOMSPS-LBD≤4) seems equivalent to the default MapleCOMSPS version: 368 instances with a PAR-2 of 1 hour faster than MapleCOMSPS. It translates the fact that some problems were not overwhelmed by the additional clauses (of LBD=4) but also 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 additional SAT instance while MapleCOMSPS-H_{LP} solves 4 UNSAT and 2 SAT more. The PAR-2 of MapleCOMSPS-H_{LP} heuristic shows a significant improvement with a gain of 13 hours, where MapleCOMSPS-H_S heuristic gave similar PAR-2 in comparison to the classical MapleCOMSPS even if the PAR-1 and CTI time is slower (1h40 for the PAR-1 and 3 hours for the

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

Solver	UNSAT	SAT	TOTAL	PAR-1	PAR-2	CTI
MapleCOMSPS	255	113	368	523h00	943h00	**91h50**
MapleCOMSPS-LBD≤4	254	114	368	522h00	942h00	**91h50**
MapleCOMSPS-H_S	253	116	369	524h40	943h00	94h50
MapleCOMSPS-H_{LP}	**258**	**117**	**375**	**521h30**	**929h50**	93h30
Kissat-MAB	310	**144**	454	384h20	661h00	86h20
Kissat-MAB-LBD≤3	302	142	444	397h00	690h20	92h30
Kissat-MAB-H_S	311	142	453	386h10	664h30	84h20
Kissat-MAB-H_{LP}	**319**	142	**461**	**379h40**	**644h40**	**80h40**

Table 2 Comparison between state-of-the-art MapleCOMSPS and Kissat-MAB solvers and H_S and H_{LP} heuristics. MapleCOMSPS-LBD≤4 (resp. Kissat-MAB-LBD≤3) uses a strategy where learnt clauses with LBD≤4 (resp. LBD≤3) are considered as relevant.

CTI). These variations are left to the user considering which constraint is to prioritize in terms of number of solved instances or the limited time to solve them.

The remain segment displays the experiments using Kissat-MAB engine. The original solver solves 454 instances with a PAR-2 of 661 hours. Augmenting the core database to protect learnt clauses with LBD≤3 (Kissat-MAB-LBD≤3) seems to deteriorate the performances: 10 instances less with a PAR-2 (resp. CTI) of 29 hours slower (resp. CTI of 6 hours slower) than the original Kissat-MAB solver. The last two lines present the tuned versions of Kissat-MAB using H_S and H_{LP} heuristics. Kissat-MAB-H_S solves 1 UNSAT instance more but 2 instances less than state-of-the-art, whereas Kissat-MAB-H_{LP} solves 9 UNSAT more but 2 SAT instances less and manages to decrease the PAR-2 time to 3 hours.

To sum up, both SAT solvers perform better with H_{LP} heuristic. Kissat-MAB seems more suitable when solving BMC problems especially UNSAT ones, known to be difficult. Hence, the two presented heuristics highlight the importance of the information captured by $C_{\mathcal{PJ}}$, 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. Also, it appears that H_{LP} captures other important information such as: $C_{\mathcal{J}}$ (and $C_{\mathcal{M}}$ when using Kissat-MAB) clauses. $C_{\mathcal{J}}$ is composed of auxiliary variables used for the conversion of the problem into a CNF formula [32]. They make the connection between the property and the model. Consequently, they also help to compute information related to the synchronous product.

7 Overview of parallel SAT approaches

The next sections are dedicated to the application of our previous studies in the context of parallelism (see. Section 5). Indeed, parallel procedures have become prominent axis to increase SAT solver efficiency due to the emergence of multi-core machines. Figure 7 describes the classical architecture of parallel SAT solver. Three main components can be observed: **the Parallel strategy** specifies the split of the workload among threads, **the Sharing strategy** broadcasts (relevant) information and **the Solver strategy** determines the SAT algorithm to use for given thread(s). The main purpose of all existing parallel SAT

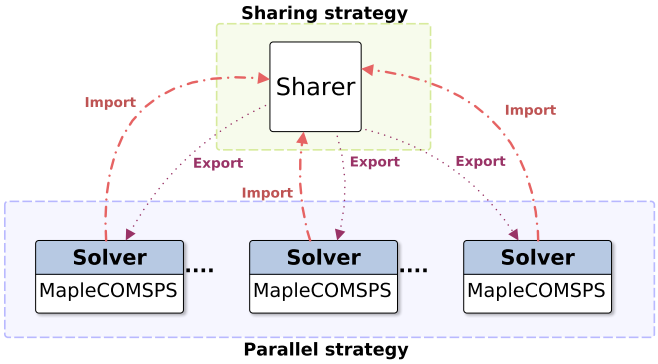

Figure 7 Parallel architecture using MapleCOMSPS as SAT solver and an All-to-All communication

solver is therefore to find the best trade-off between these three components. For instance, using a Sharing strategy that exchanges too much information may degrades the overall performances by flooding the underlying solvers. On the other hand, no sharing at all could not help to solve more UNSAT problems.

7.1 Parallel strategies

Exploiting multi-core architectures is a way to tackle the CPU time consumption when solving SAT problems. Two standards classes of parallel techniques have been developed:

Divide-and-Conquer (cooperation-based) [51]. It aims at partitioning the problem into disjoint sub-parts. It uses the guiding path method to decompose the search space. These parts are solved separately by sequential solvers. Since the parts are disjoint, if one of the partitions is proven to be SAT then the initial problem is SAT. The formula is UNSAT if all the partitions return UNSAT. The challenging points of this method are: dividing the search space and balancing jobs between solvers.

Portfolio (competition-based) [28]. It consists in making several solvers competing, on the same problem and the winner will be the first that answers. This strategy aims at increasing the probability of finding a solution using the diversification principle [26]: several solvers with different heuristics are instantiated. They differ by their decision strategies, learning schemes, the used random seed, etc.

In practice, Portfolio strategies are the most commonly used and showed a huge efficiency in recent SAT competitions[13].

7.2 Sharing strategies

In the aforementioned parallel approaches, the solvers can dynamically share information. This exchange warrants a particular focus: if a solver shares his knowledge (consisting of its learnt clauses), then this information will allow the other solvers to avoid recomputing the same information (i.e., descending into parts of the search tree that have already been proven to be unsatisfiable). Thus, exchanging learnt clauses is helpful in increasing the performance of the global system. However, sharing *"all"* learnt clauses can have a negative impact on

[13] (http://www.satcompetition.org/)

the overall behavior. Indeed, a massive exchange can either flood the solver or redirect it towards *irrelevant* part of the search space. Therefore, existing approaches [5, 27, 35, 28] raise the following question: **what are the relevant clauses to share?**
The literature identifies two main heuristics for an efficient sharing:

Static threshold. Many solvers rely on the standard measures defined for sequential solvers (i.e., activity [25], size [20] or the LBD value [47]). Only clauses less than a given threshold for these measures are shared. One simple way to get the threshold is to define it as a constant (e.g., clauses up to size 8 are shared in ManySat [28], clauses up to LBD$\leq$4 are shared in recent strategies using parallel MapleCOMSPS engines [37] or LBD$\leq$2 for parallel ParKissat-RS portfolio.

Dynamic threshold. Other approaches adapt the above thresholds dynamically in order to control the flow of shared clauses during the solving such as HordeSat strategy [7]. It starts by sharing clauses with a certain LBD threshold. This latter is increased when it appears that more clauses should be exchanged (each solver has its own threshold).

Remark. In almost all parallel SAT solvers, each thread shares information with all the other threads. Nonetheless, there exist, finer and more complex communications schemes that let each solver to choose its emitters [31].

7.3 Solving strategies

Despite the parallel or the sharing strategies, each thread can use its own solving algorithm. This allows to mix the strengths of multiple algorithms. Since implementing a SAT solving algorithm requires a lot of tuning, there exist frameworks, like PaInleSS [37], that allows to easily integrate state-of-the-art SAT solvers: Minisat [19], Glucose [47], MapleCOMSPS [38], Kissat-MAB [43], etc.

We extend this framework by adding the solvers defined in Section 5: MapleCOMSPS-H_{LP} and Kissat-MAB-H_{LP} solvers.

Remark. In the next sections, we left aside the H_S heuristic since it does not provides massive improvement in the sequential experiments (Section 6) compared to H_{LP}.

8 A need for an efficient Parallel BMC

State-of-the-art

Most of the parallel existing solutions remain generic and do not take into account the structure of the original problem. Nonetheless, some works on parallel SAT-based BMC problems exploit the specificity of the BMC [1, 49]. Ábrahám et al. [1] suggest to parallelize the search for counterexample by spawning (in parallel) multiples solvers that solve different depth k of the same problem rather than parallelizing on the same bound k. The work of S. Wieringa [49] shows that the information obtained from previous step k could be used to tune upcoming one $(k+1)$. These approaches can exploit the work of Shtrichman [45] to the parallel setting (constraint sharing and clause replication).

Other works [29, 22, 10] focus on a specific bound k of the problem and decompose it into a Server-Clients scheme where each client is in charge of solving a portion of the original problem. They must agree on a solution and the final decision is left to the master (reconciliator). The master is also responsible of controlling the communication between solvers.

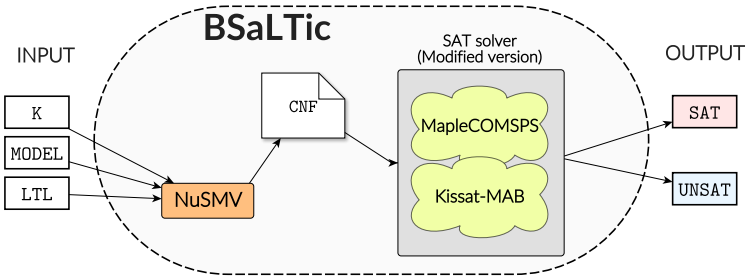

Figure 8 BSaLTic's Framework. The dashed box represents the **BSaLTic** tool

Motivations

To the best of our knowledge, no study has focused on designing a Portfolio-based approach that exploit the characteristics of the BMC problem with a fix bound k. In the next section, we propose the firsts BMC-based Portfolios that uses specialized sharing strategies for exchanging clauses. We experiment these Portfolios with state-of-the-art approaches in Section 11.

9 BMC-based Heuristics for Sharing strategies

In Section 4, we identified seven classes of clauses. These classes have been exploited in the context of sequential SAT solving to design two new heuristics dedicated to BMC (see. Section 5). These heuristics protect from deletion relevant clauses. The idea defended in this section is that a similar method could be applied in the context of parallel SAT solving. Indeed, since we already characterize relevant clauses, we can use this information to tune the sharing strategies.

Current procedures use generic measures to identify clauses to share, the best being the LBD value: only learnt clauses with a fixed threshold of LBD are shared between solvers. With this strategy, all learnt clauses, w.r.t a certain LBD and with no distinction between classes C_X, are considered as relevant.

In Section 5, we observed that some classes of clauses are more interesting for the solving than others. To leverage these pertinent classes to be massively shared, we propose to increase their corresponding LBD threshold.

- SH_{LP}: add clauses computed by H_{LP} heuristic to the sharing w.r.t. to the used solver, see Table 1. For instance, clauses of class $C_{\mathcal{J}}$ that have an LBD$\leq$10 are allowed to be shared between solvers.

10 Using BSaLTic

The Figure 8 shows the architecture of the BSaLTic framework which involves NuSMV [11] and the two SAT solvers (MapleCOMSPS [38] and Kissat-MAB [43]). BSaLTic takes three parameters as input: (1) an SMV program, (2) an LTL property (*not negated*), and (3) a bound k, required for any BMC problem. NuSMV produces a file, representing the SAT encoding of the BMC problem at hand [32]. This file is then processed by (our modified version of) a SAT solver. Note that the SAT solver component is encapsulated inside PaInleSs [37] which enable us to integrate other SAT solvers into the framework but also use parallel mechanisms (sharing strategies and parallel approaches).

SAT strat.	Sharing strat.	UNSAT	SAT	TOTAL	PAR-1	PAR-2	CTI (472)
MapleCOMSPS	HORDESAT	357	120	477	342h50	590h35	107h20
	SH_{LP}	362	120	482	339h20	578h50	106h50
MapleCOMSPS-H_{LP}	HORDESAT	361	119	480	343h05	585h50	107h00
	SH_{LP}	362	120	482	341h20	580h40	108h00

Table 3 Comparison between state-of-the-art MapleCOMSPS Portfolio and our tuned portfolio MapleCOMSPS-H_{LP} for various sharing approaches.

Here, we provide the necessary commands to run the tool.

For Sequential approaches. Executing the $\mathbf{H_{LP}}$ heuristic on a problem unrolled up to bound 20, you can run the following command using MapleCOMSPS solver:

```
$$ bsaltic.sh -s maple -k 20 -heur hlp -ltl file.ltl file.smv
```

or with Kissat-MAB:

```
$$ bsaltic.sh -s kissat -k 20 -heur hlp -ltl file.ltl file.smv
```

For Parallel approaches. Executing a portfolio of 10 MapleCOMSPS-$\mathbf{H_{LP}}$ engines on a problem unrolled up to bound 20 with a HORDESAT sharing strategy, you can run the following command:

```
$$ bsaltic.sh -s maple -c 10 -k 20 -shr hordesat -heur hlp -ltl file.ltl file.smv
```

or running Kissat-MAB-H_{LP} portfolio with the S$\mathbf{H_{LP}}$ sharing approach:

```
$$ bsaltic.sh -s kissat -c 10 -k 20 -shr hlp -heur hlp -ltl file.ltl file.smv
```

11 Experimental results on Parallel

This section aims at evaluating the impact of the sharing heuristics in a parallel portfolio setting. All the experiments have been executed on the full benchmark presented in Section 4 on an Intel Xeon@2.40GHz machine with, 12 processors, a 64 Go of memory, a time limit of 6000 seconds and using Portfolios of 10 solvers (threads).

Table 3 details the results of our parallel experiments. We tested several portfolios: a MapleCOMSPS-based portfolio and MapleCOMSPS-H_{LP}-based portfolio (presented in Section 5). For a given portfolio, threads only differ by their diversification strategies [26].

The Table shows the different used sharing strategies: HORDESAT (a strategy that increases dynamically the LBD threshold w.r.t. some constraints)[14], and SH_{LP}. For each strategy, the number of solved UNSAT and SAT instances, the PAR-1, the PAR-2 and the CTI are displayed.

The first observation goes to the used SAT solver.

MapleCOMSPS. the utilization of a BMC-based sharing strategy (SH_{LP}) on a MapleCOMSPS portfolio performs better than state-of-the-art HORDESAT sharing: it solves 5 UNSAT instances more and decreases the running time up to 12 hours for the PAR-2 metric and 30 min less for the CTI.

MapleCOMSPS-H_{LP}. similarly, SH_{LP} gives better performances than HORDESAT on both SAT and UNSAT instances: 2 instances more in total with a PAR-2 of 5 hours less

[14] This is the strategy used by the best parallel solver of the parallel track of the SAT competition 2021

SAT strat.	Sharing strat.	UNSAT	SAT	TOTAL	PAR-1	PAR-2	CTI (514)
ParKissat-RS	LBD≤2	368	148	516	248h00	429h20	81h30
	SH_{LP}	372	149	521	246h40	416h20	79h25
ParKissat-RS-H_{LP}	LBD≤2	373	149	522	248h40	415h40	80h10
	SH_{LP}	375	150	525	244h10	405h40	77h20

Table 4 Comparison between state-of-the-art ParKissat-RS Portfolio and our tuned portfolio ParKissat-RS-H_{LP} for various sharing approaches.

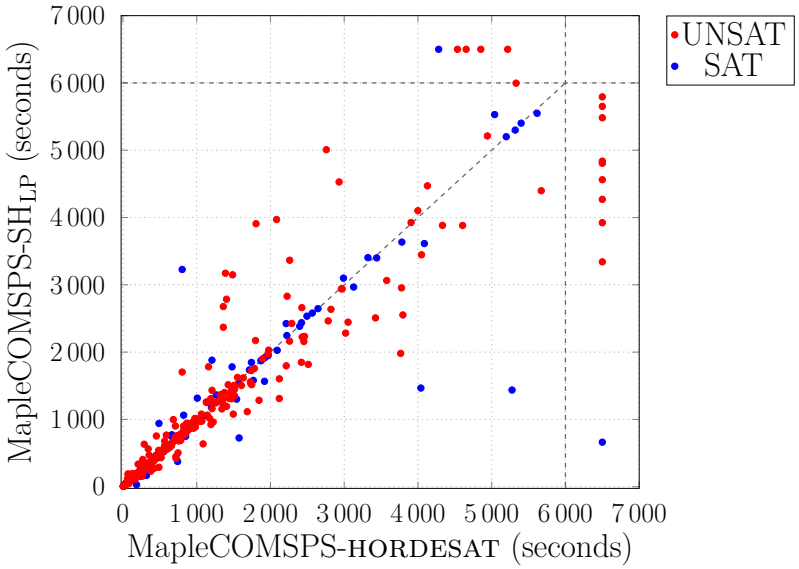

Figure 9 Scatter-plot comparing state-of-the-art portfolio (MapleCOMSPS using HORDESAT-strategy) to our best one (MapleCOMSPS with SH_{LP}-strategy)

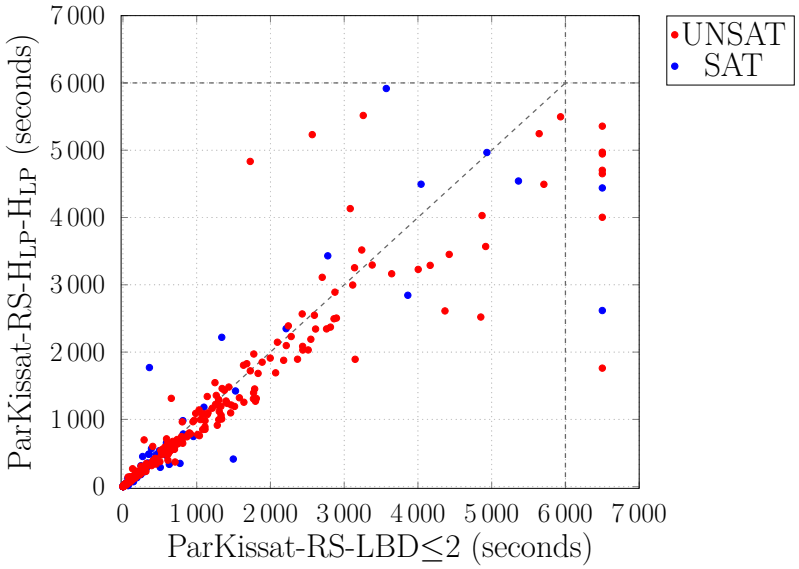

Figure 10 Scatter-plot comparing state-of-the-art portfolio (ParKissat-RS that shares clauses of LBD≤2 only) to our best one (ParKissat-RS-H_{LP} with H_{LP}-strategy)

computation.

The results reflect that using a BMC-based sharing strategy improves and performs better than generic strategies.

A detailed view of this comparison is depicted in Figure 9. The scatter-plot compares the state-of-the-art portfolio to our best one. A dot represents one solved instance (red if UNSAT and blue if SAT). The y-axis gives the solving time (seconds) for our best Portfolio and the x-axis is for the state-of-the-art Portfolio. It clearly appears that a MapleCOMSPS-based portfolio using an SH_{LP} is globally faster than generic portfolio, especially on UNSAT problems.

The second observation goes to the sharing approach:

Hordesat. The suitable solver seems to be MapleCOMSPS-H_{LP} with 480 solved instances. In contrast to a generic MapleCOMSPS portfolio which solves 3 instances less. From here, we can notice a first improvement when using our tuned solver.

SH_{LP}. the two engines give similar results in terms of number of solved instances: 362 UNSAT and 120 SAT solved instances. However, the original MapleCOMSPS-based portfolio stands out with a reduction on the running time (PAR-1, PAR-2 and CTI) with at least 2 hours. This configuration seems to perform better where other configurations failed to reach this running time score. It seems to demonstrate that information captured by the tuned SAT solver doesn't bring better information, so focusing on tuning the sharing maybe sufficient when using a MapleCOMSPS engine.

Based on the above results, we go further and conduct the experiments displayed in Table 4. These are realized using a parallel portfolio ParKissat-RS, the winner of the the Parallel SAT Competition 2022 that uses Kissat-MAB as a back-end engine, and we compared it when using our tuned Kissat-MAB-H_{LP} solver, namely ParKissat-RS-H_{LP}. ParKissat-RS uses the PaInLeSS framework to implement the clause sharing method, and each thread shares clauses with LBD$\leq$2. Diversification is also used.

We observe that state-of-the-art portfolio ParKissat-RS that uses static sharing (LBD$\leq$2) solves 516 instances and has a PAR-2 metric of 429 hours. In contrast, our best portfolio ParKissat-RS-H_{LP} with a SH_{LP} sharing solves 9 instances more with faster PAR-2 of 23 hours. It agrees to say that ParKissat-RS-H_{LP} portfolio with SH_{LP} sharing is best suitable for parallelism when using Kissat-MAB solver.

Moving on to more details, when fixing the used SAT solver, we observe that:

ParKissat-RS. the best configuration uses SH_{LP} sharing which solves 5 instances more, reducing the PAR-2 time (resp. CTI) up to 13 hours (resp. 2 hours).

ParKissat-RS-H_{LP}. tuning both the SAT solver and the sharing mechanism performs better than sharing only clauses of LBD$\leq$2: 3 instances more and 4 hours less computation on the PAR-2 metric.

The experiments strengthen our intuition that BMC-based sharing strategies enhance the solving on the total number of solved instances but also the computation time.

A detailed view of this comparison is depicted in Figure 10. The scatter-plot compares ParKissat-RS with LBD$\leq$2 sharing to our best one (ParKissat-RS-H_{LP} with SH_{LP} sharing). The next analysis goes to the used sharing approach:

LBD$\leq$2. the tuned portfolio is more suitable than the original portfolio: ParKissat-RS-H_{LP} won on 5 UNSAT instances and 1 SAT instance against ParKissat-RS portfolio and lower the computation time with 13 hours for the PAR-2 and 1 hour for the CTI and equivalently the same on the PAR-1.

SH_{LP}. ParKissat-RS-H_{LP} remains the best on this strategy with 4 more instances and 10 hours faster in PAR-2 than state-of-the-art ParKissat-RS.

To conclude, the observations derived from Tables 3,4 and Figures 9,10 support our claim that using of components tuned for BMC (sharing heuristics, relevant clauses heuristics) have a non-negligible impact on the performances.

12 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 having a structural reasoning behind H_S heuristics. In the other hand, the procedure used to build H_{LP} relies on a mathematical reasoning.

We extended this idea to the context of parallel BMC solving by offering a new sharing strategy, namely SH_{LP}. This focuses on tuning two, out of the three, components of a parallel SAT solving, i.e., the sharing strategy and the SAT strategy.

Future work aims to refine the proposed classification by exploiting the specification of the property (using the Hierarchy of Manna&Pnueli) or the synchronicity of the model. And finally, building a SAT solver and extensively, a Portfolio-based approach, to exploit exclusively structural information of the original problem is in our perspectives.

Conflict of Interest - None.

References

1 Erika Ábrahám, Tobias Schubert, Bernd Becker, Martin Fränzle, and Christian Herde. Parallel sat solving in bounded model checking. In Luboš Brim, Boudewijn Haverkort, Martin Leucker, and Jaco van de Pol, editors, *Formal Methods: Applications and Technology*, pages 301–315, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

2 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.

3 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.

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

5 Gilles Audemard, Benoît Hoessen, Said Jabbour, Jean-Marie Lagniez, and Cédric Piette. Revisiting Clause Exchange in Parallel SAT Solving. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, volume 7962 of *Lecture Notes in Computer Science (LNCS)*, pages 200–213, Trento, Italy, 2012. Springer. URL: https://hal.archives-ouvertes.fr/hal-00865596.

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

7 Tomas Balyo, Peter Sanders, and Carsten Sinz. Hordesat: A massively parallel portfolio sat solver, 2015. arXiv:1505.03340.

8 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.

9 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.

10 Xi Cheng, Min Zhou, Xiaoyu Song, Ming Gu, and Jiaguang Sun. Parallelizing smt solving: Lazy decomposition and conciliation. *Artificial Intelligence*, 257:127–157, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0004370218300237, doi:https://doi.org/10.1016/j.artint.2018.01.001.

11 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.

12 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.

13 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.

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

15 Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

16 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.

17 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.

18 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016.

19 Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

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

21 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.

22 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.

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

24 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/.

25 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.

26 Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Diversification and intensification in parallel sat solving. In David Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010*, pages 252–265, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

27 Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Saïs. Diversification and Intensification in Parallel {SAT} Solving. In *16th International Conference on Principles and Practice of Constraint Programming (CP'10)*, pages 252–265, United Kingdom, 2010. URL: https://hal.archives-ouvertes.fr/hal-00865417.

28 Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *J. Satisf. Boolean Model. Comput.*, 6(4):245–262, 2009. URL: http://dblp.uni-trier.de/db/journals/jsat/jsat6.html#HamadiJS09.

29 Youssef Hamadi, Joao Marques-Silva, and Christoph Wintersteiger. Lazy decomposition for distributed decision procedures. In *Proceedings 10th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC'11)*, volume 72, pages 43–54, nov 2011.

30 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.

31 Said Jabbour, Nadjib Lazaar, Youssef Hamadi, and Michèle Sebag. Cooperation control in Parallel SAT Solving: a Multi-armed Bandit Approach. In *Workshop on Bayesian Optimization and Decision Making*, Lake Tahoe, United States, 2012. URL: https://hal.archives-ouvertes.fr/hal-00870946.

32 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.

33 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.

34 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.

35 Michael Kaufmann, Stephan Kottler, Michael Kaufmann, and Stephan Kottler. Sartagnan - a parallel portfolio sat solver with lockless physical clause sharing. In *In Pragmatics of SAT*, 2011.

36 Anissa Kheireddine, Etienne Renault, and Souheib Baarir. Towards Better Heuristics for Solving Bounded Model Checking Problems. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:11, Dagstuhl, Germany, 2021. URL: `https://drops.dagstuhl.de/opus/volltexte/2021/15298`.

37 Ludovic Le Frioux, Souheib Baarir, Julien Sopena, and Fabrice Kordon. PaInleSS: a framework for parallel SAT solving. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 233–250. Springer, Cham, August 2017.

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

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

40 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.

41 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.

42 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`.

43 Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Un bandit manchot pour combiner CHB et VSIDS. In *Actes des 16èmes Journées Francophones de Programmation par Contraintes (JFPC)*, Nice, France, June 2021. URL: `https://hal-amu.archives-ouvertes.fr/hal-03270931`.

44 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`.

45 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.

46 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.

47 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`.

48 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`.

49 Siert Wieringa. On incremental satisfiability and bounded model checking. *CEUR Workshop Proceedings*, 832:13–21, 01 2011.

50 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.

51 Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 12 1996. doi:10.1006/jsco.1996.0030.

52 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.