PL-PRS-BVA-KISSAT in SAT Competition 2024

Mazigh Saoudi[1], Thibault Lejemble[1], Souheib Baarir[2] and Julien Sopena[2]

[1]*EPITA Research Laboratory (LRE), 18 Rue Pasteur, 94270 Le Kremlin-Bicêtre, France*
[2]*LIP6, Sorbonne University and French National Center for Scientific Research, France, 4 Pl. Jussieu, 75005 Paris, France*

Abstract
This paper introduces PL-PRS-BVA-KISSAT, a parallel SAT solver submitted to the Parallel SAT Competition 2024. PL-PRS-BVA-KISSAT is built using the Painless framework [1] and employs a Portfolio parallelization strategy. It utilizes Kissat-MAB CDCL solvers as its core engines [2] and integrates the HordeSat [3] sharing technique. Furthermore, it incorporates state-of-the-art preprocessing methods, specifically Bounded Variable Addition (BVA) [4] and the preprocessing technique implemented by PRS [5].

Keywords
Parallel Sat Solving, Bounded Variable Elimination, Portfolio Strategy, Clause Sharing

1. Introduction

For several years, Painless [1] has established itself as a crucial tool for developing efficient parallel SAT solvers. This is due to its modular, generic, and flexible architecture, along with continuous engineering efforts to maintain and evolve the tool.

In this paper, we present PL-PRS-BVA-KISSAT, a SAT solver submitted to the Parallel SAT Competition 2024, built using Painless. PL-PRS-BVA-KISSAT implements a Portfolio parallelization strategy, utilizing Kissat-MAB CDCL solvers as core engines [2] and the HordeSat [3] sharing technique. Compared to solvers submitted by the authors in previous competitions, the main innovation here is the fine integration of state-of-the-art pre-processing techniques, specifically Bounded Variable Addition (BVA) [4].

While BVA technique has been successfully used in sequential contexts, its application in a parallel setting is challenging due to soundness issues [6]. We propose a method to leverage BVA in a way that maintains soundness while improving the efficiency of a parallel SAT solver.

This paper begins by introducing Painless and BVA in Section 2. In Section 3, we provide a detailed description of our portfolio solver, PL-PRS-BVA-KISSAT. The experimental evaluations are presented in Section 4. Finally, the paper concludes with suggestions for future work in Section 5.

2. Preliminaries

As our proposed solver leverages both Painless and the BVA technique, the following sections provide an in-depth presentation of each concept.

2.1. Painless framework

The Parallel Instantiable SAT Solver, known as Painless [1], is a framework designed for creating parallel SAT solvers in multi-core environments. The adaptability and effectiveness of Painless are due to its straightforward architecture, which allows for the seamless implementation and execution of various parallel-solving strategies, including the **Portfolio** parallelization strategy. This makes Painless a significant tool in the field of parallel SAT solving.

15th Pragmatics of SAT international workshop, August 20, 2024, Pune, India
mazigh.saoudi@epita.fr (M. Saoudi); thibault.lejemble@epita.fr (T. Lejemble); souheib.baarir@lip6.fr (S. Baarir); julien.sopena@lip6.fr (J. Sopena)

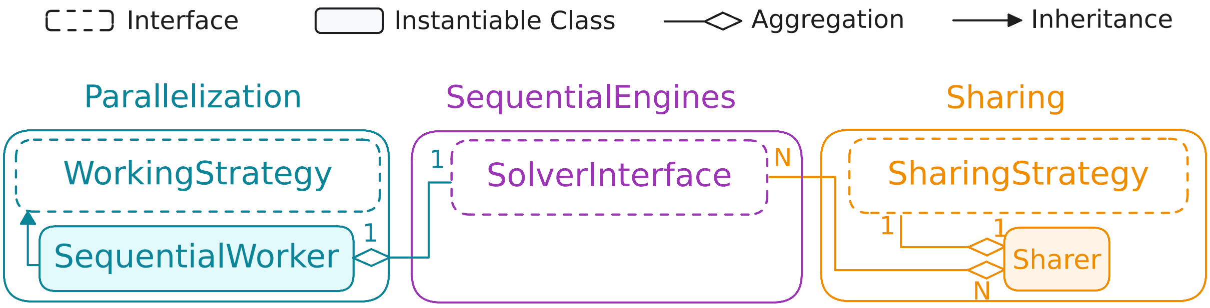

Figure 1: Original Architecture of Painless

The architecture of Painless is illustrated in Figure 1. The base classes that need to be implemented are shown in dotted boxes, while the instantiable ones are in background-colored boxes. Two main relationships can link these different classes: inheritance and aggregation.

Inheritance is a straightforward extension of a class, allowing derived classes to inherit attributes and methods from a base class, thus promoting code reuse and hierarchical organization. Aggregation represents the use of another class within a given one that needs it to be meaningful, indicating a "has-a" relationship. This means the aggregated class is crucial for the operation of the whole but can still exist independently.

For example, implementations of the **SharingStrategy** and **SolverInterface** interfaces can be instantiated independently of the **Sharer**. However, the **Sharer** cannot achieve any functional purpose without its components, illustrating a dependency. Conversely, a **SequentialWorker** gains meaning through its aggregation with a **SolverInterface**.

Starting from the left, the **WorkingStrategy** interface encapsulates the behavior of parallelization strategies. This strategy can designate a master that oversees several **SequentialWorker** threads. These threads collaborate according to a specific strategy set by the master, allowing for efficient parallel processing.

The **SequentialWorkers** interact with sequential solvers through the **SolverInterface**, which provides all necessary methods for the **SequentialWorker** to control the solving process. This interface also enables the **SharingStrategy** to manage clause sharing among sets of producer and consumer workers, optimizing resource utilization and performance. The **Sharer** handles the thread responsible for executing the chosen **SharingStrategy**, ensuring that the sharing process is carried out effectively. Thus, the **Sharer** acts as a crucial component in the system, facilitating the coordination and sharing of data between different threads and strategies.

In summary, the architecture of Painless is designed to leverage inheritance for extending functionality and aggregation for integrating essential components. The interplay between these relationships allows for a flexible and efficient system capable of parallel processing and dynamic resource management.

2.2. Bounded Variable Addition

Bounded Variable Addition (BVA) is a preprocessing technique that reduces the number of clauses in a formula by adding new variables [7, 4]. The algorithm identifies groups of resolvents that can be factorized by introducing a new variable. This new variable is added only if the number of resulting clauses is fewer than the number of identified resolvents, hence the term **Bounded** in the technique's name.

Consider the set of clauses presented hereafter. We can apply the BVA algorithm to introduce a new variable z and reduce the number of clauses from 6 to 5.

$$
\begin{array}{r}
v \lor a \lor b \\
v \lor h \lor t \\
v \lor e \\
w \lor a \lor b \\
w \lor h \lor t \\
w \lor e
\end{array}
\quad \overset{BVA}{\Longrightarrow} \quad
\begin{array}{r}
z \lor a \lor b \\
z \lor h \lor t \\
z \lor e \\
\neg z \lor v \\
\neg z \lor w
\end{array}
$$

3. PL-PRS-BVA-KISSAT

The version of Painless used to build PL-PRS-BVA-KISSAT is available on GitHub[1]. During the development of our solver, we contributed to the Painless framework by updating its architecture before integrating the different components we needed to derive our solver. In the following sections, we discuss these improvements and the newly implemented components in detail.

3.1. Architecture Improvements In Painless

While the original BVA algorithm uses a queue of variables ordered by occurrence frequency, with simple tie-breaking, the SBVA technique introduced a more sophisticated 3-Hop tie-breaking heuristic. Building on these advancements, our submission to the 2024 International SAT Competition proposes a novel approach: a portfolio of BVA pre-processors, each employing different variable queue orderings and tie-break heuristics. This diverse pre-processing strategy aims to further enhance the performance of parallel SAT solvers.

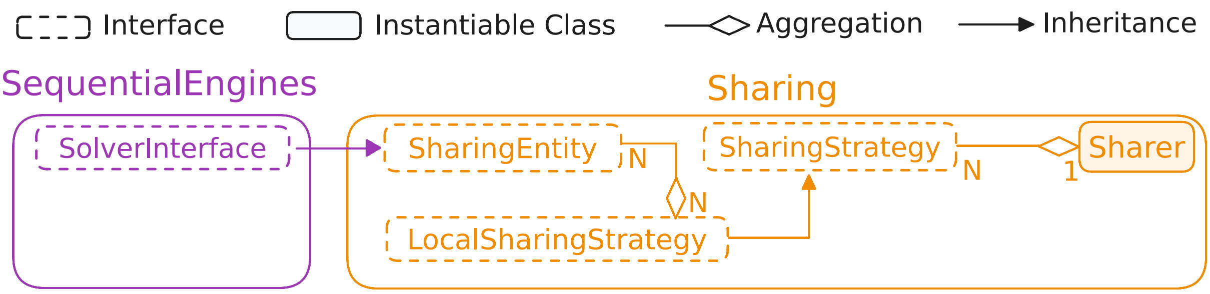

Figure 2: The Updated Modules of Painless

3.2. The Sequential Engines

Our parallel solver employs Kissat-MAB as the CDCL sequential engine, supported by PRS's and BVA as preprocessors.

3.2.1. CDCL engine: Kissat-MAB

Kissat-MAB is the core engine of PL-PRS-BVA-KISSAT. The sequential engine managing this solver is the C++ class KissatMABSolver, part of the 2023 Painless parallel competitor, Pkissat [8]. This class has been updated with a cleaner integration with Kissat-MAB and a new diversification process.

Our new diversification makes each Kissat-MAB solver identify itself as a member of one of these three families: UNSAT_FOCUSED, SAT_STABLE, and MIXED_SWITCH. Each family represents a different configuration of Kissat-MAB, modifying the frequency of restarts, the use of the target phase, and the degree of local search for rephasing or initial phasing [9]:

[1] https://www.github.com/S-Mazigh/painless/tree/satcomp-24

- The UNSAT_FOCUSED solvers primarily use default options, except that `stable` is set to 0, and the frequency of restarts and the use of chronological backtracking are slightly randomized. Approximately a quarter of these solvers employ variable shuffling at initialization, as seen in `P-Kissat` [8] and PRS [5].
- The SAT_STABLE family consistently uses target phasing (`target=2`) with less frequent restarts and uses local search more at initialization and rephasing. About half of these solvers use CCAnr [10] as in `P-Kissat` [8] and have their `tier2` value reduced to 3 [11].
- The MIXED_SWITCH family employs the default `Kissat-MAB` configuration with an initial shuffling of the variables.

3.2.2. Pre-processors

The effectiveness of `Kissat-MAB`, our main engine, is enhanced by integrating pre-processing techniques that simplify the formula.

PRS-PRE : PRS [5], the winner of the parallel track of the SAT Competition 2023, has integrated various preprocessing techniques such as Unit Propagation, Equivalent-literal Substitution and Resolution Checking [12]. These techniques have undeniably contributed to its significant success. Consequently, we have decided to incorporate these approaches into our framework with minimal modifications, we will refer to them as PRS-PRE.

BVA-based preprocessing : As previously mentioned, the BVA technique aims to simplify the formula by reducing the number of clauses. It starts with a queue of variables arranged in a specific order. According to the original algorithm [4], the most frequently occurring variable in the queue is selected first. Each variable is then tested to see if there are matching variables that could reduce the number of clauses by introducing a new variable into the formula. The queue is updated after each new variable is introduced, which involves deleting clauses. This update includes all variables from the removed clauses as well as the newly introduced variable.

During its execution, the BVA algorithm may encounter ties, where multiple variables lead to the same amount of reduction when seeking a matching candidate. The heuristic used to resolve these ties plays a crucial role in the resulting reduced formula. For instance, the success of *StructuredBVA* [7] is largely due to its effective tie-breaking heuristic, which utilizes the variable incidence graph. This heuristic counts the number of paths connecting a variable v with one of its matches v_m via an intermediate variable v_i.

It is clear that the BVA technique is sensitive to the order in which variables are matched. Changing the variable queue or using a different tie-breaking heuristic can lead the algorithm to different results. Therefore, we decided to improve the BVA implementation [13], by equipping it with parameters that enable the choice of variable queue sorting and tie-breaking heuristics:

- *Variable queue sorting.* In addition to the original queue ordering from the most frequently occurring variable to the least frequently occurring one (denoted as O_D) [7, 4], we propose two alternative orderings:
  - From the least frequently occurring variable to the most frequently occurring one (denoted as O_I).
  - Randomly sorted (denoted as O_R).
- *Tie-breaking heuristics.* For tie-breaking heuristics, we retained the `3-hop heuristic` (denoted as T_{3H}) from *StructuredBVA* [7] as an option. This heuristic has demonstrated its usefulness in certain categories of SAT problems, such as the packing k-coloring problem, pigeonhole problem, and Petri Net concurrency. It improves solving time by reducing the formula size while preserving its original structure, even when pre-randomized. However, according to the detailed results of [7], classical BVA [4] offers better performance in some types of SAT formulas.
  As additional tie-breaking heuristics, we implemented the following:

- Choose the most frequently occurring variable (denoted as T_M).
- Choose the least frequently occurring variable (denoted as T_L).
- Choose a random variable (denoted as T_R).

During our testing with the original implementation of the T_{3H} heuristic [13], we observed in some instances with high connectivity between variables an overflow in the tie-break heuristic's calculated value. While this overflow diminishes the heuristic's effectiveness, it does not render it incorrect. To address this issue, we reduced the probability of overflows by using `unsigned int` instead of `int`. In our tests, using `unsigned` resolved the overflow problem. However, re-adapting the algorithm to use `long` would be a safer approach, despite the significant increase in memory usage.

	R-R	R-3H	R-L	R-M	I-R	I-3H	I-L	I-M	D-R	D-3H	D-L	D-M	Others
R-R	-	1	0	1	23	23	23	23	70	62	71	76	95
R-3H	5	-	2	2	26	26	26	26	72	64	73	78	99
R-L	3	1	-	1	25	25	25	25	71	63	72	77	98
R-M	5	2	2	-	26	26	26	26	72	64	73	78	99
I-R	2	1	1	1	-	0	0	0	58	50	59	64	74
I-3H	2	1	1	1	0	-	0	0	58	50	59	64	74
I-L	2	1	1	1	0	0	-	0	58	50	59	64	74
I-M	2	1	1	1	0	0	0	-	58	50	59	64	74
D-R	3	1	1	1	12	12	12	12	-	0	6	8	28
D-3H	**3**	**1**	**1**	**1**	**12**	**12**	**12**	**12**	**8**	**-**	**9**	**14**	**36**
D-L	3	1	1	1	12	12	12	12	5	0	-	10	27
D-M	3	1	1	1	12	12	12	12	2	0	5	-	22
Others	1	1	0	1	0	0	0	0	1	0	4	6	

Table 1
A comparison of all configurations was conducted on 181 instances from the SAT Competition 2023. Each cell indicates the number of instances where the configuration in the column achieved a greater reduction than the configuration in the row.

In our evaluation (in Section 4), we tested all 12 possible configurations using all 400 instances from the SAT Competition 2023 benchmarks. A configuration is a couple of a queue ordering (O_X) and tie-breaking heuristics (T_Y), namely O_X-T_Y, and denoted X-Y in Table 1. In this table, each cell represents the number of instances that the column's configuration reduced further than the line's configuration. The table summarizes the results of the 181 instances where the BVA technique successfully introduced new variables for new solvers. The bottom row indicates the number of instances in which each column's configuration achieves the maximum reduction on its own. The rightmost column shows the number of instances where the given row configuration fails to produce the most reduced formula.

From Table 1, we confirm that the queue ordering O_D remains the most effective option. It permits the BVA technique to achieve maximum reduction the most. Applying different tie-breaking heuristics with O_D can result in varying reduction rates. Notably, the O_D-T_M configuration stands out as the most promising in terms of size reduction.

Even though at first glance we would judge the O_I and O_R alternatives as meaningless, we cannot ignore them since they achieve maximum reduction where their O_D counterparts don't. For example, despite the generally negative impact of randomizing the queue on the BVA algorithm, the configurations O_R-T_R, O_R-T_{3H}, and O_R-T_M achieve a maximum reduction that the other combinations cannot match.

Our objective is to seek all the possible maximum reduction rates, and the 11 configurations distinct from the state-of-the-art (O_D-T_{3H}) enable us to achieve further reductions in 36 instances, which constitutes 20% of the total 181 instances. It is evident that no single configuration can achieve the maximum reduction across all instances.

3.3. The Sharing Strategy

The sharing strategy among the underlying solvers (`Kissat-MAB`) is inspired by the approaches used in [3] and [14], namely, `HordeSat`: a set of producers and a set of consumers are registered into the local strategy. Then, every 0.1 seconds, the strategy selects clauses[2] from each producer, filters them by their LBD value [15] and finally provides them for its consumers. Initially, the threshold is set to LBD $= 2$. If our local strategy finds that a specific solver is sharing too few or too many clauses, it adjusts the threshold accordingly.

3.4. The Working Strategy

The parallel solver `PL-PRS-BVA-KISSAT` is a portfolio of the previously discussed sequential engines with `HordeSat` as its sharing strategy. The global architecture of `PL-PRS-BVA-KISSAT` is illustrated in Figure 3. Essentially, all `Kissat-MAB` solvers act as both producers and consumers within this sharing strategy.

Our portfolio is configured with 31 `Kissat-MAB` solvers. Nineteen of these solvers are launched directly on the formula obtained after `PRS-PRE`, while the remaining twelve are initiated on the formula reduced by the `BVA` technique. We opted to use all twelve `BVA` configurations, as shown in Table 1, because each configuration can outperform the others on certain problems. This approach is advantageous as it does not rely on the specific type of SAT problem being addressed. However, if the number of clauses in the formula processed by `BVA` reaches 10 million, only the O_D-T_M configuration is instantiated. This choice is based on its overall superior performance as indicated in Table 1.

Clause sharing in the presence of preprocessing techniques must be handled with great care, as some simplifications can compromise the soundness of the entire solving process. According to [6], clause sharing among solvers working on formulas treated with `BVA` preprocessing remains sound if the introduced variables are globally fresh, meaning no two solvers have different definitions for the same variable. Therefore, to maintain clause sharing between the initial `Kissat-MAB` solvers and those launched after the various `BVA` configurations, we must select a single reduced formula. In `PL-PRS-BVA-KISSAT`, we choose the most reduced formula (with the fewest clauses) from the different `BVA` configurations. This approach is both simple and efficient, as it does not require significant CPU time. Figure 4 illustrates this final point.

It is worth noting that each `Kissat-MAB` populates its **SharingEntity** export buffer with clauses derived from conflict analysis, leading to the presence of new variables in the shared clauses. Thus the solvers filter their import buffer by examining the variables in the received clauses. If a variable from a received clause is either eliminated or unknown, the solvers ignore that clause.

[2]The threshold for the cumulative size of the selected clauses is 1500 in terms of literals

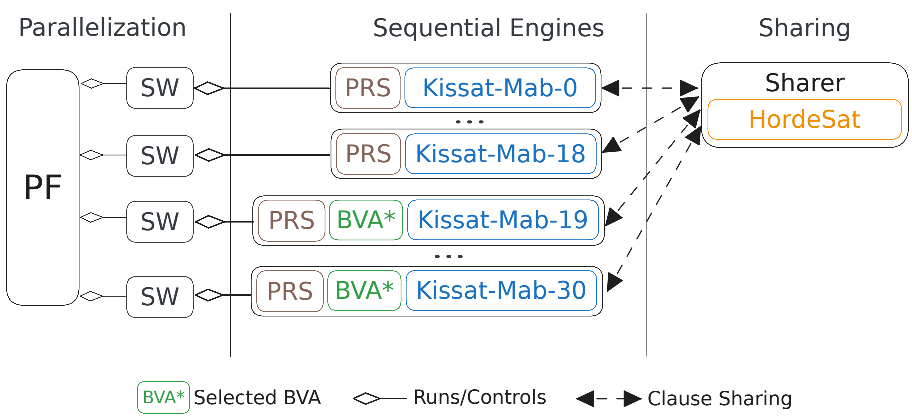

Figure 3: Architecture of `PL-PRS-BVA-KISSAT`

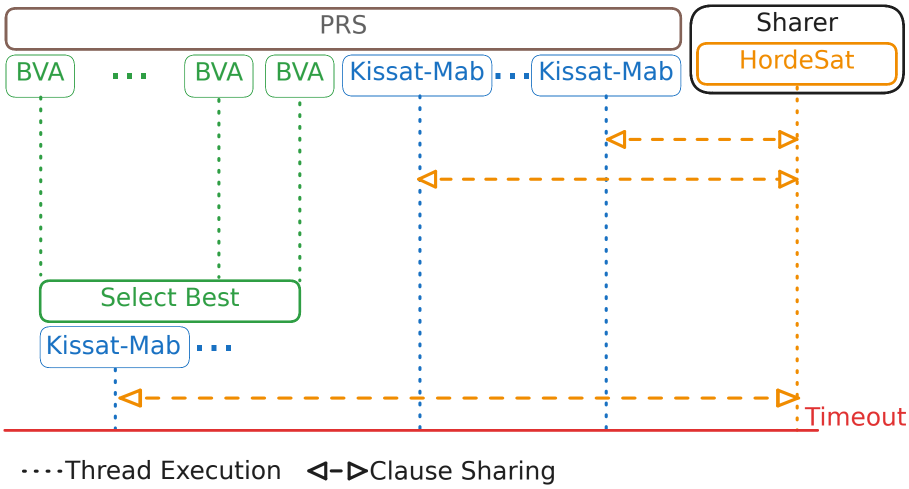

Figure 4: Execution sequence of PL-PRS-BVA-KISSAT

4. Evaluation

For a thorough analysis of the performance of our parallel solver, we ran an experiment using all 400 instances from the 2023 SAT competition with a timeout of 5000 seconds. This experiment was done using the EC2 instances `m6i.16xlarge` from AWS. The instance consists of an Intel Xeon 8375C (Ice Lake) CPU coupled to a 256 GB DDR4 RAM and runs Ubuntu 24.04. The CPU contains 32 cores with hyperthreading (64 logical threads) [16].

We compare the results of PL-PRS-BVA-KISSAT with those of PRS 2023 [5], the winner, and P-Kissat 2023 [8], the third-place solver of the 2023 Parallel SAT Competition. The choice of the winner as a baseline is evident. The selection of the third-place solver is motivated by the fact that our new solver is essentially an upgrade of P-Kissat 2023. Table 2 and **??** highlight the outputs. The top line in Table 2 shows the virtual best solver **(VBS)**.

solver	Average PAR2	SAT	UNSAT	Resolved
VBS	1432.20	153	194	347
PL-PRS-BVA-KISSAT	1556.70	151	189	340
PRS 2023	2272.36	143	177	320
P-Kissat	2824.57	127	170	297

Table 2
PL-PRS-BVA-KISSAT performance in the experiment compared to the performance of PRS 2023.

Table 2 and **??** demonstrate that PL-PRS-BVA-KISSAT outperforms PRS 2023 in both SAT and UNSAT instances. Specifically, in Table 2 we see that our solver is quite close to the **VBS**, especially on the SAT instances.

On one hand, **??** illustrates that numerous SAT instances were solved exclusively by our solver compared to P-Kissat 2023, with PRS-PRE significantly simplifying the process for many of them. When compared to PRS 2023, as shown in **??**, we observe a notable speedup in some instances, we think that it could be thanks to our new diversification process, but further evaluation is required to confirm it. On the other hand, **??** and **??** show a substantial number of UNSAT instances that were efficiently solved solely by PL-PRS-BVA-KISSAT which is likely thanks to the use of BVA preprocessing technique. Additionally, it should be noted that PRS-PRE enables PL-PRS-BVA-KISSAT to solve a considerable number of UNSAT instances on which P-Kissat 2023 timed out.

Running the Kissat-MAB solvers in parallel with the BVA preprocessing allows us to efficiently solve problems without having all solvers start on the reduced formula obtained from the BVA technique. Specifically, out of the 340 instances, 202 instances (which represents 59%) were resolved by the initial nineteen Kissat-MAB solvers. This highlights the effectiveness of our parallel approach in handling a significant portion of the problem instances without the need for additional simplification steps in the initial group of solvers, clause sharing with the new solvers is enough.

5. Conclusion

In this paper, we introduced PL-PRS-BVA-KISSAT, a parallel SAT solver submitted to the SAT Competition 2024. By leveraging our new version of the Painless framework, we effectively integrated advanced preprocessing techniques, including Bounded Variable Addition (BVA) and the PRS-PRE method, with Kissat-MAB CDCL solvers and the HordeSat clause-sharing strategy.

In our implementation of the BVA technique we proposed the use of different queue orderings and tie-breaking heuristics. Aligning with our objective of achieving maximum reductions, this approach led to a higher reduction rate in 36 instances when compared to the state-of-the-art O_D-T_{3H} configuration. The results highlight that the queue ordering strategy O_D, combined with specific tie-breaking heuristics, particularly O_D-T_M, yields the most substantial reductions in formula size. Conversely, configurations based on random ordering (O_R) were less effective overall but still achieved notable maximum reductions in some instances, suggesting that randomness can occasionally benefit the reduction process.

Overall, our experiments demonstrate that PL-PRS-BVA-KISSAT significantly outperforms the previous year's winner, PRS 2023, both in terms of PAR2 scores and the number of instances solved. Notably, our solver excelled in instances where the BVA technique introduced new variables, underscoring the importance of effective preprocessing in enhancing solver performance.

Looking forward, our work opens avenues for further exploration into the integration of diverse preprocessing techniques within parallel solvers. Future efforts will focus on developing more sophisticated heuristics for selecting and combining preprocessing strategies to optimize clause sharing while maintaining soundness. Additionally, we aim to refine the decision-making processes within the solver to better adapt to the structural characteristics of different SAT problems, thereby improving the overall efficiency and robustness of parallel SAT solving methodologies.

Our findings confirm the potential of parallel preprocessing techniques in SAT solving and provide a solid foundation for future advancements in this field.

References

[1] L. Le Frioux, S. Baarir, J. Sopena, F. Kordon, Painless: a framework for parallel sat solving, in: Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT), Springer, 2017, pp. 233–250.

[2] M. Sami Cherif, D. Habet, C. Terrioux, Un bandit manchot pour combiner CHB et VSIDS, in: Actes des 16èmes Journées Francophones de Programmation par Contraintes (JFPC), Nice, France, 2021. URL: https://hal-amu.archives-ouvertes.fr/hal-03270931.

[3] T. Balyo, P. Sanders, C. Sinz, Hordesat: A massively parallel portfolio sat solver, in: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT), Springer, 2015, pp. 156–172.

[4] N. Manthey, M. J. H. Heule, A. Biere, Automated reencoding of boolean formulas, in: Proceedings of the 8th International Conference on Hardware and Software: Verification and Testing, HVC'12, Springer-Verlag, Berlin, Heidelberg, 2012, p. 102–117. URL: https://doi.org/10.1007/978-3-642-39611-3_14. doi:10.1007/978-3-642-39611-3_14.

[5] Z. Chen, X. Zhang, Y. Qian, S. Cai, Prs: A new parallel/distributed framework for sat, in: SAT COMPETITION 2023 Proceedings, 2023, pp. 39,40.

[6] N. Manthey, T. Philipp, C. Wernhard, Soundness of inprocessing in clause sharing sat solvers, in: M. Järvisalo, A. Van Gelder (Eds.), Theory and Applications of Satisfiability Testing – SAT 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 22–39.

[7] A. Haberlandt, H. Green, M. J. H. Heule, Effective auxiliary variables via structured reencoding, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2023.11. doi:10.4230/LIPICS.SAT.2023.11.

[8] V. Vallade, S. Baarir, J. Sopena, New concurrent painless solvers based on kissat-mab: P-kissat and p-kissat-str, in: SAT COMPETITION 2023 Proceedings, 2023, pp. 42,43.

[9] S. Cai, X. Zhang, M. Fleury, A. Biere, Better decision heuristics in cdcl through local search and target phases, J. Artif. Int. Res. 74 (2022). URL: https://doi.org/10.1613/jair.1.13666. doi:10.1613/jair.1.13666.

[10] S. Cai, C. Luo, K. Su, Ccanr: A configuration checking based local search solver for non-random satisfiability, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing – SAT 2015, Springer International Publishing, Cham, 2015, pp. 1–8.

[11] C. Oh, Between sat and unsat: The fundamental difference in cdcl sat, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing – SAT 2015, Springer International Publishing, Cham, 2015, pp. 307–323.

[12] Z. Chen, X. Zhang, S. Cai, P. Lu, Cdcl solvers with improved local search cooperation and pre-processing, in: SAT COMPETITION 2022 Proceedings, 2022, pp. 37,38.

[13] A. Haberlandt, H. Green, Sbva-cadical and sbva-kissat: Structured bounded variable addition, in: SAT COMPETITION 2023 Proceedings, 2023, p. 18.

[14] Y. Hamadi, S. Jabbour, J. Sais, Control-based clause sharing in parallel sat solving, in: Autonomous Search, Springer, 2011, pp. 245–267.

[15] G. Audemard, L. Simon, Predicting learnt clauses quality in modern sat solvers., in: IJCAI, volume 9, 2009, pp. 399–404.

[16] AWS, Cpu cores and threads per cpu core per instance type - amazon elastic compute cloud, 2024. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/cpu-options-supported-instances-values.html.