Parallel Clause Sharing Strategy Based on Graph Structure of SAT Problem

Yoichiro Iida
Graduate School of Information Science and Technology, The University of Tokyo, Japan

Tomohiro Sonobe
National Institute of Informatics, Tokyo, Japan

Mary Inaba
Graduate School of Information Science and Technology, The University of Tokyo, Japan

Abstract

Parallelization of SAT solvers is an important technique for improving solver performance. The selection of the learnt clauses to share among parallel workers is crucial for its efficiency. Literal block distance (LBD) is often used to evaluate the quality of clauses to select. We propose a new method, *Parallel Clause sharing based on graph Structure* (PaCS), to select good clauses for sharing. First, we conducted three preliminary experiments to assess the performance of LBD in parallel clause sharing: a performance comparison between the LBD and clause size, an analysis of the utilization of shared clauses, and a comparison of the LBD values of shared clauses at originating and receiving workers. These experiments indicate that the LBD may not be optimal for learnt clause sharing. We attribute the results to the LBD's inherent dependency on decision trees. Each parallel worker has a unique decision tree; thus, a sharing clause that is good for its originating worker may not be good for others. Therefore, we propose PaCS, a search-independent method that uses the graph structure derived from the input CNF of SAT problems. PaCS evaluates clauses using their edges' weight in the variable incidence graph. Using the input CNF's graph is effective for parallel clause sharing because it is the common input for all parallel workers. Furthermore, using edge weight can select clauses whose variables' Boolean values are more likely to be determined. Performance evaluation experiments demonstrate that our strategy outperforms LBD by 4% in the number of solved instances and by 12% in PAR-2. This study opens avenues for further improvements in parallel-solving strategies using the structure of SAT problems and reinterpretations of the quality of learnt clauses.

2012 ACM Subject Classification Computing methodologies → Heuristic function construction; Mathematics of computing → Solvers

Keywords and phrases SAT Solver, Structure of SAT, Parallel application, Clause Learning

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.17

Acknowledgements We would like to express our sincere gratitude to the reviewers for their valuable comments and constructive feedback, which have greatly improved the quality of this paper. Additionally, we are grateful to the members of the Imai Laboratory for their insightful discussions and support throughout this research.

1 Introduction

Satisfiability (SAT) solvers are tools that determine whether the input Boolean formula is satisfiable or not. Conflict-driven clause learning (CDCL) SAT solvers [28, 29] are widely used because of their high efficacy in many industrial SAT problems. Clause learning [8] is an important component of CDCL solvers. Clause learning generates new clauses (learnt clauses) to prevent the solver from repeating the wrong assignments. Learnt clauses significantly improve search efficiency by pruning the search space. Parallelization of SAT solvers is also an important technique for improving solver performance. The information can be shared

between parallel workers to enhance the overall parallel efficiency. In parallel SAT solvers, workers exchange their acquired learnt clauses. However, because the solver often learns over millions of clauses during its search, it is impractical to share all of them. Therefore, literal block distance (LBD) [4] is often used for clause selection. LBD was originally proposed as an evaluation metric of clause quality. LBD has been used by many successful solvers in SAT competitions [1], in both sequential and parallel solvers.

First, in this study, to assess the performance of the LBD in parallel clause sharing, we conducted three preliminary experiments: (1) a comparison of solver performance using LBD and clause size, (2) an analysis of the utilization ratio of shared clauses, (3) a comparison of LBD values of shared clauses between originating and receiving workers. The results of the experiments indicate that LBD may not be the optimal metric for learnt clause sharing; the size of the clause performs competitively with (actually slightly better than) LBD, and the utilization ratio of the shared clause is low. We attribute the results to the LBD's inherent dependency on the search state. The LBD value indicates the number of variable decision levels in the clause. The decision level corresponds to the depth of the branching decision tree. Several factors influence the tree, including the decision strategy, learned clauses, heuristics status, and solver configurations. We call these varying conditions "search states" and say LBD has an inherent dependency on the search state. However, each parallel worker has a unique search state; particularly in portfolio-type parallel SAT solvers, they should be different. Therefore, a sharing clause good for its originating worker, as measured by the LBD, may not be good for others. According to Audemard and Simon, "*LBD is relative to the current search of each solver, and a good clause for one thread may not be good for another one.*" [5].

Therefore, we propose *Parallel Clause sharing based on graph Structure* (PaCS), a search-independent method that uses the graph structure derived from the input CNF of SAT problems. It evaluates clauses using a metric [21], that measures the strength of the connection between variables in the clause and their neighboring variables. The weight of the edge in the variable incidence graph (VIG) quantifies the strength of variables connection. This metric was originally proposed for the deletion of learnt clauses and demonstrated comparable performance to LBD. We assume that using the graph of input CNF is effective for parallel clause sharing because it is a common input for all parallel workers, enabling it to identify good clauses for all workers. Furthermore, using edge weight can select clauses whose variables' Boolean values are more likely to be determined. A heavy edge implies that the pair of variables is contained in a short clause or many clauses. Thus, variables with heavier edges have a better chance of undergoing propagation. Moreover, various studies have shown a relationship between the quality of clauses and their graph structures [20, 35].

We implemented a parallel solver that uses our proposal clause sharing selection method and conducted performance evaluation experiments to compare it with LBD. The contributions of our research are as follows:

1. Through three preliminary experiments, we have demonstrated that LBD may not be the optimal solution for selecting learnt clauses to share among parallel workers.
2. We propose PaCS, which uses a metric that represents the strength of the connection in the graph structure for parallel clause sharing.
3. We compared the performance of a parallel solver using PaCS with LBD and found that it outperforms solvers using LBD by 4% in the number of solved instances and by 12% in PAR-2.

[1] SAT competition `http://www.satcompetition.org/`

The remainder of this paper is organized as follows: Section 2 introduces SAT solvers and their techniques, including learnt clauses and the concept of the graph structure of SAT problems. Section 3 discusses related work. Section 4 presents our empirical observations of LBD for clause sharing. Section 5 defines the proposed method for parallel clause sharing using PaCS. Section 6 presents the results of the PaCS performance evaluation. Finally, Section 7 summarizes the study and suggests future research directions.

2 Preliminaries

2.1 Satisfiability problem and SAT solver

The SAT problem determines whether at least one Boolean variable assignment can satisfy a given logical formula. If an assignment can satisfy all clauses, the formula is satisfiable; otherwise, it is unsatisfiable (UNSAT). The formula is generally provided in conjunctive normal form (CNF), wherein variables are combined into clauses with disjunctions, and the clauses are combined with conjunctions. A formula is in CNF if it possesses the form $(C_1 \wedge C_2 \wedge \cdots \wedge C_m)$, where C_i represents a clause. Each clause is a disjunction of literals $(L_{i,1} \vee L_{i,2} \vee \cdots \vee L_{i,n})$, where $L_{i,j}$ represents a literal in clause C_i. A literal is a variable or its negation. An example of a CNF formula is $(x \vee y) \wedge (\neg y \vee z)$. Here, $(x \vee y)$ and $(\neg y \vee z)$ are clauses, and x, y, $\neg y$, and z are literals where y and $\neg y$ represent the positive and negative forms of variable y, respectively.

The SAT solvers are programs for solving SAT problems. It is used to solve real-world problems encoded in SAT, such as computer-aided proof [18] and binary neural network verification [11]. Davis–Putnam–Logemann–Loveland (DPLL) algorithm [13] is the basis of most SAT solvers. The DPLL algorithm contains decisions – providing an assumption of Boolean (True or False) value to a variable – and incorporates propagation – determining other variables' Boolean values as the logical consequences of the decision. A conflict occurs when a clause becomes false due to a wrong decision. Then, the previous decisions are canceled (called backtrack), and the solver makes another decision. Modern SAT solvers are conflict-driven clause-learning (CDCL) solvers [28, 29] that are based on the DPLL algorithm. The CDCL solvers incorporate various techniques and heuristics [15, 27, 29], making it possible to solve large and complex SAT problems efficiently. Probabilistic algorithms [34] are another approach that randomly changes the Boolean values of variables until a solution is found. In this paper, we focus on the improvement of CDCL SAT solvers.

2.2 Clause Learning

Clause learning [8] is an essential technique for CDCL solvers, aiming at improving efficiency by pruning the search space. When a solver finds a conflict, it analyzes the root cause and derives the counter-example as a new clause (learnt clause) to avoid revisiting similar unsatisfiable assignments in future searches. Modern solvers often learn over millions of clauses during searches. However, it is difficult to maintain all of the clauses due to the cost of checking these clauses during propagation. Clause evaluation is necessary to selectively retain more valuable clauses for future searches. The clause size, literal block distance (LBD) [4], and activity (or clause version of a variable state independent decaying sum, cVSIDS) [2] are widely adopted metrics. The size of a clause represents the number of literals contained in

[2] MiniSat `http://minisat.se/`

the clause. Shorter clauses are more effective in reducing the search space. Given a clause c and literals l, $size(c) := |\{l \in c\}|$. The LBD [4] is a popular clause evaluation metric and is widely adopted by many state-of-the-art SAT solvers. The LBD value of a clause is defined as the number of different decision levels to which the literals in the clause belong. The decision level indicates the depth of the branching decision tree. The LBD value of a clause c can be calculated as $LBD(c) := |\{d(l) : l \in c\}|$ where $d(l)$ represents the decision level of literal l. A clause with a lower LBD is considered more valuable. In addition to clause evaluation, the LBD is used in other heuristics such as restart strategy [9] and decision branching [12]. Activity (or cVSIDS) measures how frequently a clause is used in the conflict analysis. The more used clauses are more valuable, and a higher score is provided for recently used ones. Among these three, the results of evaluation by LBD and Activity are based on the current search state, such as the decision tree and maintained learnt clauses. Therefore, these values are different at different search states, while size shows constant values.

2.3 Parallel SAT solver

Parallelization of SAT solvers is an important technique for improving solver performance, leveraging plentiful computing resources such as affordable multi-core processors and cloud computing services. A parallel SAT solver simultaneously utilizes multiple cores to solve SAT problems. It shares valuable learnt clauses among workers to enhance parallel efficiency. This may potentially help to solve larger or more complex problems more efficiently. Two main approaches are adopted in parallel solvers: divide and conquer [10] and portfolio [19]. The divide and conquer approach splits the problem into smaller sub-problems, solves them independently, and then combines the solutions. This approach possesses excellent scalability because assigning Boolean values to some variables can easily split the search space. However, load balancing and UNSAT proof are the challenges of the approach. The portfolio approach, on the other hand, does not split the problem; it uses multiple different strategies (i.e., search) simultaneously for the same problem, and it subsequently selects the best solution among these strategies. Although this may help resolve the issue of divide and conquer by design, scalability is a challenge due to the difficulty of ensuring various search strategies. In the recent SAT competition, most successful parallel solvers adopted the portfolio approach.

Sharing information between parallel workers is crucial to enhancing overall efficiency. This is usually done by sharing learnt clauses among parallel workers. As each worker explores different search spaces, they generate different learnt clauses. Sharing these clauses may help other workers leverage collective knowledge and avoid redundant searches. However, sharing many clauses increases communication overhead and propagation costs. Therefore, it is essential to appropriately manage the volume and frequency of sharing. With regard to these factors, the size and LBD values of clauses are often used as the selection criteria; the clauses expected to be valuable for other workers are shared based on the evaluation of these metrics. For example, ParKissat-RS [3] and PRS [4], which are the winner in the parallel track of SAT competition 2022 and 2023, respectively, share the clauses with LBD values of one or two by default. Section 3.1 gives a detailed explanation of the method.

[3] ParKissat-RS `https://github.com/shaowei-cai-group/ParKissat-RS`
[4] PRS `https://github.com/shaowei-cai-group/PRS-sc23`

2.4 Structure of SAT

Industrial SAT problems encoded from real-world problems exhibit unique structures and patterns, which can differentiate them from randomly generated problems. For example, the treewidth of the graph representation of the industrial SAT problem is notably small [14]. The centrality of SAT refers to how central or critical a variable or clause is within the problem's structure [24]. Industrial problems exhibit a clear community structure [1, 31], and some correlation between the degree of the community (modularity) and runtime of the solver is known [30]. Additionally, many structural properties, such as mergeability [37] and the role of backdoor variables [36], have been studied to understand the efficiency of SAT solvers.

The graph of the SAT problem is often represented using a variable incidence graph (VIG). In this graph, nodes represent variables, and edges indicate the existence of clauses containing these variables. Each edge has a weight, and it is determined based on the strength of the variable connection. Heavier weight is given to edges whose variables coexist in more or shorter clauses. The definition of weight $w(e_{v_i,v_j})$ of the edge between v_i and v_j is $\sum_{c\in C, v_i,v_j\in c} 1/\binom{|c|}{2}$ following the existing research [1]. A clause variable incidence graph (CVIG) is a bipartite graph where variables and clauses are represented as nodes on each side, respectively, and an edge exists between a clause and a variable if the variable exists in the clause. In this study, we use VIG as the graph representation of the SAT problem.

3 Related work

3.1 Clause sharing between parallel workers

ManySAT [17], a portfolio-type parallel SAT solver, uses size as a metric for sharing learnt clauses, where all clauses of size eight or less are shared. Then, they suggested dynamically varying the criterion of the shared clauses [16]. Painless [26] is a parallelization framework that has been recently used by many solvers. This involves running a process called Sharer for clause sharing. The winners of the parallel track SAT competition 2022 and 2023, ParKissat-RS and PRS, adopted the painless framework. They used LBD as the criteria to determine the clauses to be shared. P-KISSAT [6], also based on the painless framework, dynamically varies the criterion for sharing learnt clauses depending on the number of clauses generated by the producer of clauses. Hordesat [7], and its derivative Mallob [33] are popular in large-scale parallel environments (such as high-performance computing) In particular, Mallob achieved excellent results in both the parallel track of the SAT competition 2023 and the cloud track. Hordesat is a large-scale parallel search solver designed for distributed memory and compute node environments. In Hordesat's learnt clause sharing, size is used as a metric for all-worker-to-all clause sharing. Workers add their learnt clauses to a shared buffer. The added clauses are sorted in ascending order of size, and the smaller ones are shared with other parallel workers up to a certain number of clauses. Mallob merged buffers according to the job tree of parallel workers to address problems related to this buffer-sharing system in Hordesat, such as duplicate clauses and the sharing of blank spaces within the buffer.

Several studies focused on the mechanism of sharing learnt clauses rather than just the criterion to share. The ppfolio [5] used an extreme strategy in which no learnt clauses were shared and achieved excellent results in the 2011 SAT competition. Lazaar et al. [25]

[5] ppfolio `https://www.cril.univ-artois.fr/~roussel/ppfolio/`

proposed a sharing strategy that focuses on determining the workers to receive them rather than on the selection of clauses. Audemard et al. [3] proposed psm, which measures the usefulness of a clause in the current search context using variable assignments. Audemard et al. [2] also proposed the psm-based "freeze and activate" strategy, which shares clauses but freezes some that are deemed unnecessary in the current search. While most SAT solvers operate on CPU, there have been efforts to use GPU for rapid and parallel evaluation of clauses [32].

3.2 Clause quality evaluation using graph structure of SAT

Although LBD is more popular than size, size is also used as a secondary metric when two clauses have identical LBD values. Size is a structural property of the SAT graph, which is static in any search state; however, it is considered a poorer metric than LBD. However, Jabbour et al. [22] refocused on the effectiveness of the size metric against LBD. They showed that size-based evaluation with some randomness can improve the solver's performance. Vallade et al. [35] demonstrated the relationship in a clause between the LBD value and the number of communities in the graph representation of SAT. They proposed a novel clause evaluation method that combines the number of communities and the clause's LBD value. Jamali et al. [23] proposed using the structural properties of the SAT problem for heuristics, such as decision and clause evaluation. They selected the betweenness centrality of the variables in a learnt clause as an evaluation metric and demonstrated that the method could improve the solver's performance. We proposed a clause evaluation method using the graph structure derived from the input CNF of the SAT problem, called WANCE [21]. It favorably evaluated clauses with variables that had strong (heavy) edges to their neighboring variables in VIG. Details of the definition are presented in Section 5.

4 Observations

This chapter shows the results of preliminary experiments to assess the performance of the popular method, LBD, in parallel clause sharing. Remember that the LBD value indicates the number of literal blocks in the clause; thus, it depends on the search states. This allows evaluation of the clause's quality optimized according to the current search state. However, in the parallel clause-sharing situation, this can be a disadvantage because the search state of each parallel worker could be different; particularly in the portfolio approach, it should be different in terms of efficiency. To test this hypothesis, we conducted the following experiments.

4.1 Performance comparison of criteria for sharing clause selection

First, we compared the performance difference due to the clause-sharing criteria in a parallel SAT solver. We selected ParKissat-RS, the state-of-the-art and champion parallel solver, in the SAT competition 2022 parallel track. ParKissat-RS decides the clauses to share among parallel workers, as follows. It adopts worker-sharer architecture. Worker generates learnt clauses through search. Only learnt clauses with LBD values of either 1 or 2 are submitted to the buffer. Next, the sharer broadcasts these clauses in the buffer to the other workers up to 1500 literal lengths per sharing cycle. In this experiment, we modified the submission criteria. Size-based criteria submit all clauses with their size of or less than x, where we set x as $1, 2, 3, 5, 8, 10$. Also, no sharing policy ($x = 0$) is compared with LBD. The experimental setup is as follows: We configured the number of parallel workers at 16, set the time limit to 5000

s, and established a memory limit of 128 GB, all by specifying the options in ParKissat-RS. For other settings, we adhered to the default configurations and implementations, including the constraint of sharing literal lengths up to 1500. The benchmark is 400 instances from the SAT Competition 2023. The experiments were conducted on a computer equipped with an AMD Threadripper Pro 3995WX processor with 64 cores and 512 GB RAM (four 128 GB DDR4-3200 MHz slots).

Table 1 Performance comparison among clause sharing criteria using LBD, no sharing, and size. Numbers in the table represent the count of instances in each satisfiability, where SAT denotes instances identified as satisfiable and UNSAT as unsatisfiable within the time limit, respectively.

Criterion	SAT	UNSAT	SAT+UNSAT	PAR-2
LBD (≤ 2)	124	141	265	3715
No sharing	126	131	257	3973
Size (≤ 1)	126	**147**	**273**	3631
Size (≤ 2)	126	141	267	3650
Size (≤ 3)	125	143	268	3619
Size (≤ 5)	125	144	269	3601
Size (≤ 8)	**129**	141	270	**3528**
Size (≤ 10)	121	124	245	4300

The experimental results are shown in Table 1. The first column shows the solvers with each sharing criteria. The numbers in the table represent the number of instances in each satisfiability, where SAT denotes instances identified as satisfiable and UNSAT as unsatisfiable. This experiment showed no significant difference in the performance between LBD and Size-based criterion from 1 to 8 (actually, the performance of size is better than that of LBD). The largest number of solved problems is by the size ≤ 1 criterion, and the lowest PAR-2 score is by the size ≤ 8 criterion. LBD is generally an efficient metric for evaluating learnt clauses compared to their size. However, the results of this experiment suggest that in the task of sharing learnt clauses among parallel workers, LBD performs similarly to size.

4.2 Analysis of utilization differences between imported and learned clauses with same LBD value

Next, we compared the usefulness of shared clauses between "*learned*" and "*imported.*" A clause is termed "*learned*" when a worker derives it during their search. When this clause is shared with other workers (i.e., exported), it becomes "*imported*" for those other workers. Therefore, the same clause will have a different term depending on the worker by which it was acquired. Additionally, the worker that originally learned the clause is referred to as the "*learned*" worker, while the worker that received the shared clause is called the "*imported*" worker. In general, each worker shares the learnt clauses that are expected to be useful to other workers. However, the LBD value can be different in each parallel worker because each worker has a unique search state. In the ParKissat-RS implementation, the LBD value is converted to the clause's size when sharing. Then, it is updated (re-evaluated) according to the search state of the *imported* worker when necessary. This subsection's experiments aim to analyze the difference in the degree of usefulness between *learned* and *imported* clauses. This experiment used the notion of *used%* as below.

4.2.1 Definition of *used%*

The utilization of the clause means that they were used in conflict analysis. We defined *used* as a Boolean value of the clause utilization, which is 1 if the clause is used at least once in conflict analysis and 0 otherwise.

$$\text{used}(c) := \begin{cases} 1 & \text{if } c \text{ is referred in conflict analysis} \\ 0 & \text{otherwise} \end{cases}$$

In the context of this study, the following terms are defined:

- C: the set of all learnt clauses in all parallel workers.
- i: a worker $i \in W$ where W represents all parallel workers.
- ${}_iC$: the set of learned clauses acquired at worker i, $\forall_i C \subset C$.
- ${}_iC^{\text{learn}}$: the set of clauses *learned* by the worker i.
- ${}_iC^{\text{import}}$: the set of clauses *imported* from another worker to i.
- ${}_iC_x$: the set of learned clauses for which LBD value is x at worker i.
- ${}_iC_{(x,y)}$: the set of learned clauses for which LBD value is x and the clause size is y.

For example, ${}_iC_2^{\text{learn}}$ refers to the set of all clauses with LBD of 2 in worker i that are learned at worker i. The LBD value of *imported* clauses C^{import} refers to the value at the *learned* worker before sharing, not the re-evaluated value at the *imported* worker after sharing. Furthermore, the percentage of used clauses within a set C is defined as:

$$\text{used\%}(C) := \frac{\sum_{i \in W, c \in {}_iC} \text{used}(c)}{|\bigcup_{i=1}^{N} {}_iC|}$$

where $|C|$ represents the number of clauses in C.

We used ParKissat-RS as the base solver, with two parallel workers, a time limit of 1000 s, and a benchmark of 400 instances on the SAT competition 2023 benchmark. All learnt clauses and their information were written to an external file, then *used%* was calculated later. The clause was written when the clause was learned, used, deleted, its LBD value was updated, and the search ended. We exclude clauses with a size equal to or less than 2 because they are always watched for propagation and don't have the concept of being used in ParKissat-RS. In its default setting, only the clauses of LBD value of less than or equal to two are shared up to 1500 literal lengths. To analyze the utilization trends of *imported* clauses using broader sets of clauses, we increased the LBD value criterion for clause sharing from the default of 2 to 5, and we expanded the maximum shared literal size from the default of 1,500 to 30,000. We performed experiments on the same computer as the previous experiment.

4.2.2 Analysis of used% according to LBD values

We analyzed the differences of *used%* between *imported* and *learned* clauses for the same LBD value. We specifically compared the $\text{used\%}(C_x^{\text{import}})$ and $\text{used\%}(C_x^{\text{learn}})$, respectively, for LBD values x ranging from 1 to 10. Remember that the LBD value of the *imported* clause represents the LBD value being evaluated by the *learned* worker, not by *imported* worker that was reevaluated after sharing. This setup is to observe the gap of use of the same evaluation clauses – given a clause, is there any difference of use % at *learned* worker and *imported* worker? There is no difference if the evaluation shows a common usefulness for all parallel workers. Figure 1 shows the result of randomly selected four instances from all

benchmark instances for further analysis. The horizontal and vertical axes present the value of the clause's LBD and the *used%* of these clauses, respectively. The yellow and blue lines represent *learned* and *imported* clauses, respectively. The numbers in the line chart are the number of clauses for each LBD value.

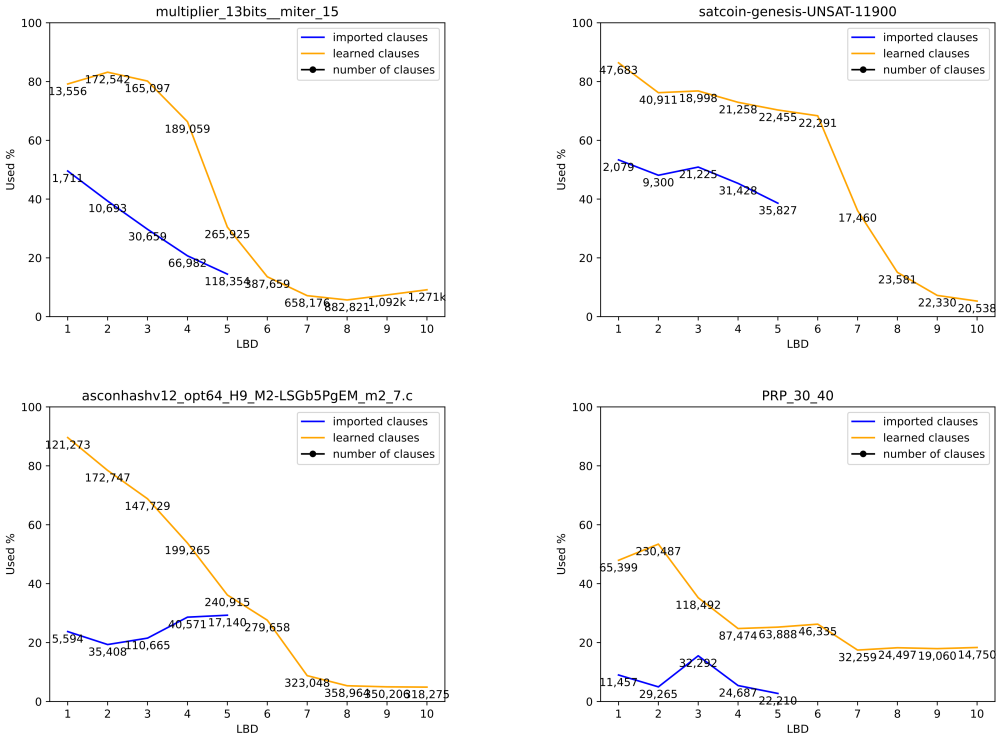

Figure 1 Differences of *used%* at each LBD value between learned and imported clauses.

The yellow line consistently showed that the *used%* decreased as the LBD value increased (i.e., worse clauses). In contrast, for the blue lines, the variation in *used%* with LBD value was less. Furthermore, *used%* were much lower than those of *learned* clauses for the same LBD value. This result indicates that a clause considered useful at an *learned* worker is not necessarily useful for *imported* worker.

4.2.3 Analysis of used% according to LBD values and size

We plotted the data into a heatmap to better understand the relationship between LBD, size, and *used%*. The $used\%(C^{imported}_{(x,y)})$ and $used\%(C^{learned}_{(x,y)})$ of LBD of x and size of y clauses are shown in Figure 2. Two instances out of the previous four are shown as examples owing to the space limitation. The vertical and horizontal axes indicate the LBD value and size, respectively. The cell color indicates the *used%*, where the closer the color is to dark blue, the higher the *used%*, whereas the lighter the color, the lower the *used%*. Same to the previous experiment, the LBD value of the *imported* clause is that at the *learned* worker, not after sharing and at *imported* worker. In the *learned* heatmap, higher *used%* was observed for lower LBD values, and vice versa. The result indicates that the LBD value determines how high or low the *used%* is, regardless of the size of the clause. In contrast, in the *imported* heatmap, the *used%* seems to depend on the size rather than the LBD value. This suggests

that the LBD value does not primarily indicate the usefulness of *imported* clauses, as deduced from the *used%* metric. Instead, clause size seems to be associated with *used%*. This result is consistent with that of the previous experiment. The dark blue cells whose LBD and size are equal in the *imported* figure showed considerably high *used%*. However, the number of these clauses was limited.

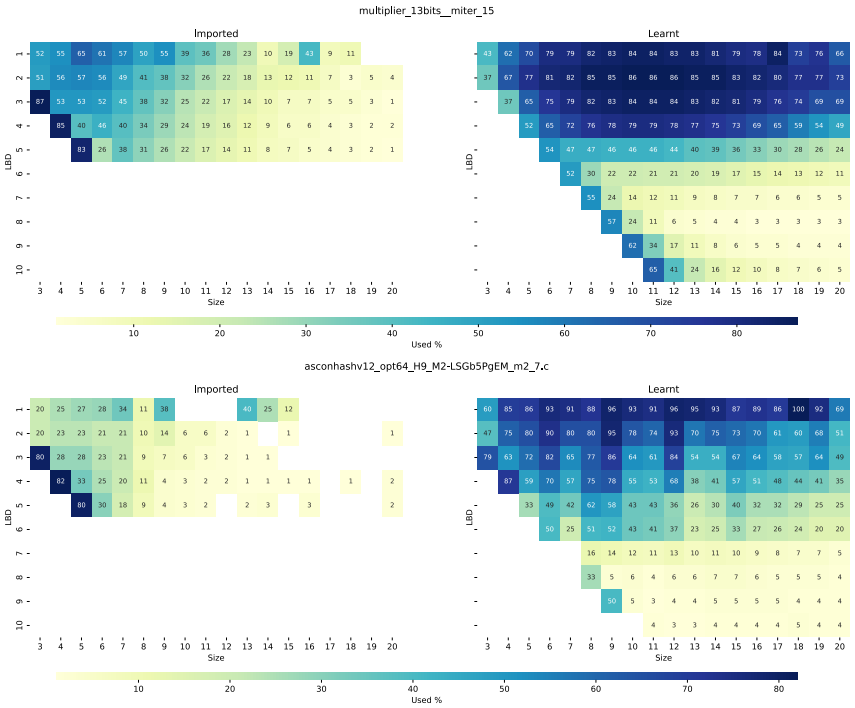

Figure 2 *Used%* heatmap of LBD and size, comparison of clauses learned and imported.

4.2.4 Distribution of used% on LBD among all instances

Figure 3 shows the statistical summary of all the benchmark instances as the box plot. The box plot illustrates the distribution of *used%* among instances. The mean *used%* for each LBD value was used as the value for an instance. The center line of the box represents the median instance, and the ends of the box represent the first and third quartile points, respectively. The edge lines represent the maximum and minimum values, respectively. We excluded outliers from the figure for readability; instances that are 1.5 times smaller or larger than the first and third quartile values. In addition, instances that generate no imported clauses (e.g., the solver finds a solution before no sharing is conducted) are excluded from the result. This figure substantiates that the trend observed in the example instances represents the general trend across many instances.

4.3 Comparison of LBD values of shared clauses at learned and imported workers

In the previous experiments, we compared the *used%* in *learned* worker and *imported* worker, using the LBD value that is calculated at *learned* worker. Next, we quantify the changes in LBD values after sharing; between the values at originating (*learned*) worker and at

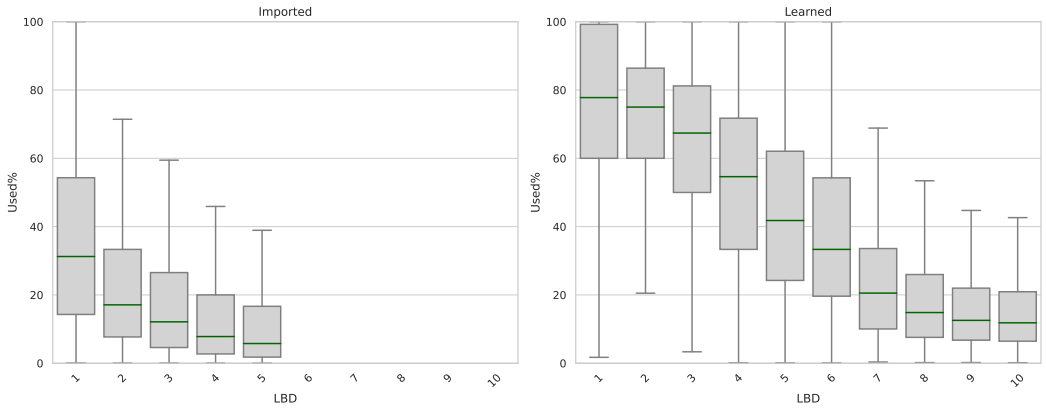

Figure 3 Distribution of *used%* for 400 benchmark instances at each LBD value.

receiving (*imported*) workers after reevaluation. We refer to the LBD value at *learned* worker as *learned* LBD, and that at *imported* worker as *imported* LBD, respectively. The *learned* LBD and *imported* LBD can be different because each worker uses a different decision tree. We compared the change from *learned* LBD to *imported* LBD. This experiment aims at investigating the extent to which the originally highly evaluated (and thus shared) clause is evaluated in the *imported* worker after sharing. This experiment used the same setup as those in the previous experiment. Table 2 shows the result. Each row represents the LBD at the *learned* worker, and each column represents the *imported* LBD. The number in each cell refers to the percentage of clauses defined by

$$\frac{|C_{\text{learned}=x,\text{imported}=y}|}{|C_{\text{learned}=x}|}$$

where x is the *learned* LBD, and y is the *imported* LBD. This indicates the ratio of clauses C_x (LBD $= x$) at *learned* worker that are re-evaluated to y at the *imported* worker. The value is averaged for all instances. Remember that we excluded clauses with a size of one or two following the previous experimental conditions.

Table 2 Comparison of LBD values of shared clauses at originating and receiving workers. 10+ denotes the sum of the percentages for clauses whose size is greater than or equal to 10. The sum of the horizontal axis (rows) adds up to 100%.

		imported LBD									
		1	2	3	4	5	6	7	8	9	10+
learned LBD	1	5%	8%	**33**%	16%	11%	7%	5%	3%	3%	2%
	2	3%	6%	**36**%	14%	9%	6%	5%	4%	3%	3%
	3	3%	5%	5%	**25**%	13%	9%	8%	6%	5%	4%
	4	2%	2%	2%	3%	**18**%	10%	10%	8%	7%	5%
	5	2%	2%	2%	2%	2%	**13**%	10%	9%	9%	8%

If LBD can measure the common quality for all parallel workers, *imported* LBD is expected to be similar to *learned* LBD; A valuable clause for a *learned* worker is also valuable for *imported* workers. However, the result reveals that more than 90% of *learned* clauses are re-evaluated worse at *imported* worker. This means a significant change in the LBD values after sharing. This observation can be explained with the previous *used%* analysis; The possibility of the clause being updated is low because the utilization of *imported* clauses is limited.

Through these experiments, we conclude that LBD is not enough for the clause quality evaluation in the parallel environment. We suggest that it is attributed to LBD's search state dependence. Under such conditions, we suggest using another clause evaluation metric that is independent of the search state.

5 Proposal method and implementation

Our proposed method, PaCS, evaluates clauses using a metric originally proposed for the clause deletion task [21]. The metric, namely average external edge weight (AEEW), measures the weight of the edges in the VIG of the input SAT problem. The sequential solver that used AEEW as its primary criterion for clause evaluation demonstrated competitive performance with LBD. The value of AEEW for a clause c is expressed as follows:

$$AEEW(c) := avg(\{w(e)/|c| : e = (v_i, v_j) \in E, v_i \in c, v_j \notin c\})$$

This concept denotes the strength (presented as the weight of edge) of the connection between the nodes v_i and v_j, where v_i belongs to the learnt clause c ($v_i \in c$) and v_j is the neighboring variable of v_i that does not belong to c ($v_j \notin c$). E denotes all edges in G, $\forall e \in E$, and $e = (v_i, v_j)$ indicates that edge e connects to variables v_i and v_j. Function $w(e)$ returns the weight of edge e in G, which is defined in Section 2.4. $|c|$ denotes the size of c, and the function $avg(X)$ calculates the arithmetic mean of set X, defined as $avg(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$, where n denotes the count of elements in X ($|X|$). A higher AEEW value (heavier edge on average or smaller size) indicates a higher quality.

We suggest that AEEW can measure a clause's quality from a new perspective: "the possibility of determining the Boolean values of variables in the evaluating clause". A higher AEEW value indicates either a heavier average edge weight or fewer variables in the clause. A heavy edge implies that the pair of variables belonging to the edge is contained in a short clause and/or many clauses as defined by the weight. When the Boolean value of one variable is determined, the Boolean value of the other variable is more likely to be determined through propagation. Also, the fewer variable clauses (shorter clauses) have a higher possibility of determining all variables in the clause. Thus, higher AEEW clauses have a better chance of determining their variables' Boolean values easily. Consequently, this allows AEEW to positively evaluate clauses that are more likely to be satisfied, initiate unit propagation, or induce conflicts.

We focused on AEEW's search-independent property, which is attributed to its definition using the graph structure of INPUT CNF. Thus, we propose PaCS using AEEW as a metric for selecting sharing clauses in parallel SAT solvers. This has the potential to address the challenges LBD faces in parallel environments. Algorithms 1 and 2 show how PaCS determines which clauses to share. For implementation, we used ParKissat-RS as the base solver. ParKissat-RS adopts a worker-sharer parallel architecture. Workers add acquired learnt clauses that meet the criteria to a buffer for sharing. In the default configuration, the criterion has an LBD value of two or less. The sharer broadcasts all clauses in the buffer to the workers at regular intervals. We modified both procedures in the worker and sharer and adopted a two-step selection in PaCS. Algorithm 1 describes the procedure in a worker, the first-step selection. A clause is stored in *buffer* and forwarded to sharer if its size is less than or equal to a predefined value *limit_size* or its LBD values less than or equal to two. The *limit_size* and LBD ≤ 2 serve as the initial screening criteria to decrease the computational cost of calculating and sorting the AEEW values in the sharer procedure. In the case of LBD and size, screening is not necessary by setting a constant threshold (e.g.

size ≤ 8) among instances. However, it is difficult to set a constant threshold by AEEW since its value varies largely from instance to instance. Therefore, we set initial screening to shortlist millions of obtained clauses by size to a manageable amount and then sort the remaining clauses according to the value of AEEW. Furthermore, we added the LBD ≤ 2 criterion for the following *cnt* implementation. Then, the procedure in sharer serves as the second screening, as described in Algorithm 2. One worker constructed the instance graph in a parallel environment on shared memory, and then, upon completion, other workers or sharers used it. The AEEW values are calculated for all clauses in the buffer and then sorted. The sorting is in descending order of AEEW, ascending order of LBD if the AEEW values are equivalent, and ascending order of size if the LBD values are equivalent. Higher AEEW clauses are shared more preferentially. The solver counts *cnt* as the number of clauses in the buffer with an LBD value less than or equal to two. This *cnt* constrains the number of clauses to be shared at line 6; this helps to exclude the effect of the total number of sharing clauses and observe only the effect of the change in selection criteria.

This implementation allows us to isolate and compare the effects of changing the criteria for selecting shared clauses. Subsequently, the sharer executes learnt clause sharing at predetermined intervals every 0.5 s. The selected clauses are broadcast to parallel workers from the top of the sorted clauses.

Algorithm 1 Clause sharing procedure of PaCS in worker.

Require: obtained learnt clause c, LBD criteria *limit_size*
1: **if** $c.size \leq limit\_size$ or $c.lbd \leq 2$ **then**
2: $\quad buffer \leftarrow c$
3: **end if**
4: submit c to *buffer*

Algorithm 2 Clause sharing procedure of PaCS in sharer.

Require: all buffered learnt clauses C, counter *cnt*, instance graph G
1: **for** clause c in C **do**
2: $\quad$ calculate AEEW value for clause c using G
3: $\quad$ **if** $c.lbd \leq 2$ **then** $cnt{+}{+}$
4: **end for**
5: sort(*buffer*)
6: broadcast the top *cnt* clauses in the *buffer* according to the sorted order.

6 Performance Evaluation

6.1 Experiment setup

We evaluated the performance of the solver using the PaCS method and compared its performance with that of the base solver, ParKissat-RS, using LBD. We compared four types of solvers in 16 parallel environments (16 workers): Base – the default ParKissat-RS. Size (size ≤ 1 or 8) – share clauses whose size is less than or equal to eight (same as presented in Section 4.1). Random – randomly select clauses from those whose sizes are less than 100, up to the quantity corresponding to the number of clauses with LBD values of one or two; PaCS (size $\leq x$) – select clauses using PaCS from those whose size is less than or equal to x (*limit_size*), up to the quantity corresponding to the number of clauses with LBD values of one or two.

The benchmark comprised 1200 instances from the main tracks of SAT competitions held between 2021 and 2023 (400 instances per year). We conducted the experiments on a computer with an AMD Threadripper Pro 3995WX processor (64 core) and 512 GB (128 GB 4 slots, DDR4-3200 MHz) RAM. We used the default ParKissat-RS implementation, adding the necessary functions for PaCS, and only changed the timeout option. No other options for the running solvers. We evaluated the solver's performance based on the number of instances solved within a time limit of 5000 s on the CPU clock and the PAR-2 score, which represents the mean time required to solve an instance with an additional penalty of 5000 s for each unsolved instance.

6.2 Evaluation result

Table 3 summarizes the results of our experiments, in which we compared base, random, size ≤ 1, 8, and PaCS, whose initial screening size is between 5 to 9. SC21, SC22, and SC23 present the benchmark instance set from the SAT competition 2021-2023, respectively. SAT and UNSAT indicate the number of instances identified as satisfiable and unsatisfiable, respectively. Therefore, a higher number indicates a better result. The cactus plot in Figure 4 demonstrates the same results; however, the PaCS results are limited to size $\leq 5, 7$ for readability.

Table 3 Performance evaluation results corresponding to each solver.

	SC21		SC22		SC23		Total	PAR-2
Criterion	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT		
Base	153	162	158	153	124	141	891	2934
Random	154	156	159	148	125	137	879	3170
Size (size ≤ 1)	155	160	160	151	126	147	899	2931
Size (size ≤ 8)	154	167	159	160	**129**	141	910	2754
PaCS (size ≤ 5)	155	**170**	162	164	128	148	927	2612
PaCS (size ≤ 6)	155	169	**163**	164	124	**150**	925	2634
PaCS (size ≤ 7)	**157**	169	160	**166**	127	**150**	**929**	**2595**
PaCS (size ≤ 8)	154	168	161	164	127	148	922	2659
PaCS (size ≤ 9)	156	168	**163**	164	124	**150**	925	2642

The size-based selection solver performed marginally better than the LBD, which is consistent with the results of Section 4.1. However, the random one worsened the performance. The solver's performance using PaCS is overall better than that of the base solver using LBD, particularly with the initial size seven screenings. It solved 38 more instances (+4.2%) and achieved an average PAR-2 score improvement of 339 (−11.6%). More improvement was observed in the UNSAT instances. At a high level, UNSAT instances require good learning for their proof, and we assume that PaCS contributed to the identification of these valuable clauses for all workers. This improvement can be attributed to two properties of PaCS. PaCS can share valuable clauses for all parallel workers irrespective of their search states because it depends only on the graph structure converted from the CNF of the input problem. This implies that PaCS can assess the general usefulness of clauses across parallel workers, whereas LBD indicates the value of a worker's search state. Furthermore, PaCS can favorably select clauses whose variables are more likely to be propagated because it highly values heavier edges in VIG. After determining the Boolean values of variables in the clauses, it can identify the clauses that cause conflict more frequently.

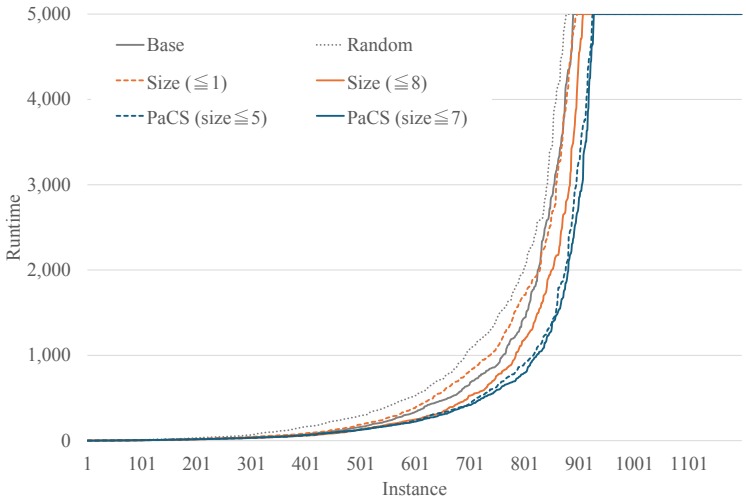

Figure 4 Performance evaluation results in cactus-plot. PaCS results are limited to size $\leq 5, 7$ for readability.

7 Conclusion

This study focuses on the clause-sharing strategy for parallel SAT solvers to improve them. First, we investigated the performance of LBD, the current popular metric, in parallel clause sharing. Preliminary experiments showed that LBD is not optimal for clause evaluation in a parallel environment. Therefore, we propose a novel clause-sharing method, Parallel Clause sharing based on graph Structure, PaCS. It can evaluate the common quality for all parallel workers using the graph structure derived from the input CNF of the SAT problems. The performance evaluation experiments demonstrated that PaCS outperforms the state-of-the-art parallel solver using LBD. These results showed the potential for enhancing the clause-sharing strategy of parallel solvers by leveraging the graph structure inherent in SAT problems. Furthermore, we believe that this study opens avenues to reinterpret and understand the quality of learnt clauses more deeply. The quality of clauses has often been evaluated by their size (including substantial size measured by LBD). We argue that AEEW is the extended concept of size, which encompasses what size implies. Short clauses have a higher possibility of determining Boolean values, which are more likely to induce unit propagation and conflict, contributing to the search. The AEEW can assess the possibility directly using the weight of edges in the VIG.

The following items are for future work. First, there is potential for improving performance by refining the implementation. The construction of graphs entails a certain duration, which may span up to several hundred seconds in one worker, depending on the problem. Further enhancements to the data structures can contribute to performance improvements. The current algorithm includes pre-defined parameters, such as initial screening before submission to a buffer, which can be optimized. The second entails exploring adaptive strategies, such as altering selection criteria or the number of clauses to be shared. Third, large-scale parallel experiments may be studied further. Conducting these experiments may validate the

scalability and practical applicability of our proposal method. The fourth is to replace LBD with PaCS for all use in parallel solvers, for example, clause deletion strategy in each parallel worker and optimization of the number of sharing clauses without LBD. Fifth, we would like to explore the theoretical justification and investigation of the implications of AEEW values, the possibility of propagation. Finally, in-depth investigations of the relationship between clause quality and graph structure from a broad perspective can contribute to understanding the behavior of SAT solvers and the quality of learnt clauses. These future works aim to optimize and expand the application of PaCS and explore new frontiers of clause evaluation.

References

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

2 Gilles Audemard, Benoît Hoessen, Saïd Jabbour, Jean-Marie Lagniez, and Cédric Piette. Revisiting clause exchange in parallel sat solving. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 200–213, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

3 Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs. On freezing and reactivating learnt clauses. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 188–200, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

4 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, July 2009. Morgan Kaufmann Publishers Inc.

5 Gilles Audemard and Laurent Simon. Lazy clause exchange policy for parallel sat solvers. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 197–205, Cham, 2014. Springer International Publishing.

6 Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.

7 Tomáš Balyo, Peter Sanders, and Carsten Sinz. Hordesat: A massively parallel portfolio sat solver. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 156–172, Cham, 2015. Springer International Publishing.

8 Roberto J. Bayardo and Robert C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, pages 203–208. AAAI Press, 1997.

9 Armin Biere and Andreas Frohlich. Evaluating cdcl restart schemes. In *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 1–17. EasyChair, 2019. doi:10.29007/89dw.

10 Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969–994, 2003. doi:10.1016/S0167-8191(03)00068-1.

11 Curtis Bright, Ilias Kotsireas, Albert Heinle, and Vijay Ganesh. Complex golay pairs up to length 28: A search via computer algebra and programmatic sat. *Journal of Symbolic Computation*, 102:153–172, 2021. doi:10.1016/j.jsc.2019.10.013.

12 Wenjing Chang, Guanfeng Wu, and Yang Xu. Adding a lbd-based rewarding mechanism in branching heuristic for sat solvers. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6, 2017. doi:10.1109/ISKE.2017.8258780.

13 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.

14 Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. Treewidth in verification: Local vs. global. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 489–503, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

15 Carla P Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *15th national conference on artificial intelligence and 10th conference on Innovative applications of artificial intelligence*, pages 431–437. AAAI, 1998.

16 Youssef Hamadi, Said Jabbour, and Jabbour Sais. *Control-Based Clause Sharing in Parallel SAT Solving*, pages 245–267. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-21434-9_10.

17 Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6(4):245–262, 2010. doi:10.3233/SAT190070.

18 Marijn J H Heule, Oliver Kullmann, and Victor W Marek. Solving and verifying the boolean pythagorean triples problem via Cube-and-Conquer. In *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245, Cham, 2016. Springer. doi:10.1007/978-3-319-40970-2_15.

19 Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997. doi:10.1126/science.275.5296.51.

20 Yoichiro Iida, Tomohiro Sonobe, and Mary Inaba. Structural impact of learnt clauses in sat solvers: An empirical analysis. In *Pragmatics of SAT*, 2023.

21 Yoichiro Iida, Tomohiro Sonobe, and Inaba Mary. Wance: Learnt clause evaluation method for sat solver using graph structure. In *The 18th Learning and Intelligent Optimization Conference (LION 2024)*, 2024. submitting and under review.

22 Saïd Jabbour, Jerry Lonlac, Lakhdar Saïs, and Yakoub Salhi. Revisiting the learned clauses database reduction strategies. *International journal of artificial intelligence tools: architectures, languages, algorithms*, 27(08):1850033, 2018. doi:10.1142/S0218213018500331.

23 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.

24 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.

25 Nadjib Lazaar, Youssef Hamadi, Said Jabbour, and Michèle Sebag. Cooperation control in parallel sat solving: a multi-armed bandit approach. Technical report, Institut National de Recherche en Informatique et en Automatique, 2012.

26 Ludovic Le Frioux, Souheib Baarir, Julien Sopena, and Fabrice Kordon. Painless: A framework for parallel sat solving. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, pages 233–250, Cham, 2017. Springer International Publishing.

27 Jia Hui Liang, Hari Govind V.K., Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. An empirical study of branching heuristics through the lens of global learning rate. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, pages 119–135, Cham, 2017. Springer International Publishing.

28 J.P. Marques-Silva and K.A. Sakallah. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.

29 M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 530–535, 2001. doi:10.1145/378239.379017.

30 Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, Gilles Audemard, and Laurent Simon. Impact of community structure on sat solver performance. In *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 252–268, Cham, 2014. Springer International Publishing.

31 Zack Newsham, William Lindsay, Vijay Ganesh, Jia Hui Liang, Sebastian Fischmeister, and Krzysztof Czarnecki. Satgraf: Visualizing the evolution of sat formula structure in solvers. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 62–70, Cham, 2015. Springer International Publishing.

32 Nicolas Prevot, Mate Soos, and Kuldeep S. Meel. Leveraging gpus for effective clause sharing in parallel sat solving. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 471–487, Cham, 2021. Springer International Publishing.

33 Dominik Schreiber and Peter Sanders. Scalable sat solving in the cloud. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 518–534, Cham, 2021. Springer International Publishing.

34 Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, 26:521–532, September 1996.

35 Vincent Vallade, Ludovic Le Frioux, Souheib Baarir, Julien Sopena, Vijay Ganesh, and Fabrice Kordon. Community and lbd-based clause sharing policy for parallel sat solving. In *Theory and Applications of Satisfiability Testing – SAT 2020*, pages 11–27, Cham, 2020. Springer International Publishing.

36 Richard Williams, Carla Gomes, and Bart Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Theory and Applications of Satisfiability Testing 2003*, 2003.

37 Edward Zulkoski, Ruben Martins, Christoph M. Wintersteiger, Jia Hui Liang, Krzysztof Czarnecki, and Vijay Ganesh. The effect of structural measures and merges on sat solver performance. In *Principles and Practice of Constraint Programming*, pages 436–452, Cham, 2018. Springer International Publishing.