

An Empirical Study of the Effect of Learnt Clause on the Structural Measures of SAT problems

Yoichiro Iida^{1,*}, Tomohiro Sonobe² and Mary Inaba¹

¹Graduate School of Information Science and Technology, The University of Tokyo, Hongo 7-3-1, Bunkyo, Tokyo, Japan

²National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda, Tokyo, Japan

Abstract

Owing to their high efficiency, state-of-the-art solvers are applied to solve a wide range of computational problems for industrial purposes. The satisfiability problem (SAT) is a well-known NP-complete problem, and SAT solvers are the applications for solving SAT. The high performance of SAT solvers is attributed to the exploitation of the underlying structure of industrial SAT problems. Industrial problems derived from real-world problems exhibit more biased and prominent structural properties compared to random ones. However, research has shown that clause learning, an essential technique of modern SAT solvers, decreases the degree of structure of industrial problems. This study intends to answer ‘*why learnt clauses are effective though they destroy the structure of problems —the hypothetical source of efficiency?*’. We hypothesize that this can be explained by the difference in the quality of clauses; the quality of the learnt clause correlates to the change in structure, and high-quality (useful) clauses destroy the structure less. To verify this, we investigated the time variance of two structural measures, treewidth and modularity, throughout the search process to observe the impact of learning. Additionally, we analyzed the relationship between the quality of a learnt clause, measured by the literal block distance (LBD), and its effect on the structural changes, measured by the modularity. Our findings include the following: (1) The value of these measures does not change monotonically rather it fluctuates over time. (2) The effect of learnt clauses on the structural change is correlated with these qualities —low-quality clauses characterized by large LBD values tend to decrease modularity more than high-quality clauses with small LBD values, and vice versa. These findings determine the relationship between the quality of learnt clauses and their impact on the structure of a SAT problem. The study suggests potential avenues for developing methods for a more effective evaluation of learnt clauses based on structural measures.

Keywords

CDCL SAT Solver, Clause Learning, Structure of SAT

1. Introduction


The Boolean satisfiability problem (SAT) is a well-known NP-complete problem; thus, problems containing numerous variables are considered computationally intractable. However, in practice, conflict-driven clause learning (CDCL) SAT solvers, which are applications widely used for solving SAT problems, occasionally solve problems with millions of variables in a reasonable time (e.g., a few minutes or hours). This gap between theoretical understanding and practical


14th International Workshop on Pragmatics of SAT (PoS 2023)

*Corresponding author.

✉ yoichiro-iida@g.ecc.u-tokyo.ac.jp (Y. Iida); tomohiro_sonobe@nii.ac.jp (T. Sonobe); mary@is.s.u-tokyo.ac.jp (M. Inaba)

ORCID 0009-0008-4937-1183 (Y. Iida); 0000-0002-0995-7234 (T. Sonobe)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

results may originate from the structure of SAT problems. Unlike random SAT problems used in theoretical research, industrial problems derived from real-world problems exhibit biased and prominent structural properties. These structural properties are quantified by measures such as backdoors [1], treewidth [2], and modularity [3]. For instance, random SAT problems do not usually have impactful backdoors that can drastically simplify the problem upon correct assignment, and industrial SAT problems, such as planning tasks, may have low treewidth owing to their hierarchical structure. Distinct differences have been observed in these measures through multiple pieces of research between random and industrial problems [4]. A correlation was also observed between the values of the measures and the search time of the SAT solvers [3].

Although the original industrial problems exhibit properties distinct from those of random problems, some studies [5, 6] have revealed that clause learning during search reduces the difference between the values of the measures of industrial problems and those of random ones. The authors of [7] even stated that clause learning “destroys the original structure.” Indeed, this has been observed across numerous measures [4]; the initial structure of SAT problems tends to approach a structure-less, random-like structure as a result of learning. This raises an intriguing question. Learning is one of the pivotal techniques of CDCL and is well-known to markedly improve search efficiency by retaining the learnt clauses. Furthermore, the structural properties of SAT problems have been used to explain the efficiency of SAT solvers on industrial problems. However, learning decreases the structural property. *Why learnt clauses are effective even though they destroy the structure of problems —the hypothetical source of efficiency?*

To address this question, we focus on the clause quality and clause deletion procedure based on the quality measures. Solvers obtain learnt clauses at every conflict during the search process, which can result in the acquisition of millions of clauses. Retaining all these clauses is neither practical nor efficient, considering the memory requirements and the impact on the search efficiency of propagation. Consequently, modern SAT solvers delete learnt clauses based on the quality of these clauses. Based on the learning procedure, our hypothetical story to answer the above question is as follows:

- Through the deletion process, high-quality clauses are maintained in the solver, whereas low-quality ones are deleted. Furthermore, high-quality clauses are more frequently used (i.e., propagated or analyzed) than low-quality ones.
- Although the structure of the problem may explain the efficiency of the solver, learnt clauses are known to decrease the structural properties.
- High-quality clauses decrease less or even increase the structural properties of SAT problems; in contrast, low-quality clauses decrease them.
- This results in maintaining or even increasing the actual structural properties because of the deletion and less usage of lower-quality clauses.
- This maintained structure contributes to finding solutions even in large-size problems.

To validate this hypothesis, this study aims to understand the relationship between the quality and structural measures of learnt clauses through experimental analysis. We investigated the changes of two measures, treewidth and modularity, throughout the search process. Additionally, we analyzed the relationship between the quality of a learnt clause and its effect on the change in the modularity value. The objective of the analysis is to understand: how much and when

the value of structural measures changes by the learnt clauses; and which quality of the learnt clauses changes these values more. The findings of this study provide a deep understanding of clause learning and deletion, as well as the relationship between the problem structure and clause learning.

The remainder of this paper is organized as follows: Section 2 introduces SAT solvers and structural measures. Section 3 presents the results of experimental observations. Section 4 summarizes the paper and suggests future research directions.

2. Preliminaries

2.1. SAT solver and clause learning

The SAT problem is a decision problem of the Boolean formula. A Boolean formula is an expression that uses the logical operators AND, OR, and NOT to combine Boolean variables, which can only hold True or False values. The SAT problem asks if a true/false assignment exists to variables in a Boolean formula that makes the whole formula true, and it is a famous NP-complete problem. SAT solvers are algorithmic applications designed to solve SAT problems, using various techniques to find a satisfying assignment of variables (satisfiable, SAT) or determine that no such assignment exists (unsatisfiable, UNSAT). CDCL is a major algorithm adopted by many solvers. State-of-the-art CDCL solvers occasionally solve problems with even more than one million variables and clauses. Owing to their efficiency, they are used to solve real-world problems such as computer-aided proofing [8] and binary neural network verification [9] encoded as SAT problems.

A key function of CDCL solvers is clause learning [10]. Learnt clauses, the output of learning, is an inference derived from the original SAT instance represented as a clause. The technique has significantly improved the performance of solvers owing to its power to prune the search space. The procedure to obtain learnt clauses is as follows. The SAT solver performs its search by making an assumption that assigns a Boolean (True or False) value to a variable (decision). Subsequently, as the logical consequence of the decision, the solver assigns the Boolean values of other deterministic variables (propagation). If all the literals in any clause are assigned to be false as the result of propagation, the decision is considered to be incorrect (conflict). The CDCL solver analyzes the root cause of the conflict, and a counter-example of the cause is learned as a clause (learnt clause) to avoid the same incorrect assumptions and conflicts in subsequent searches. To select the best counter-example, the first unique implication point [11] is used. Finally, the solver cancels the incorrect decisions and resumes its search for another decision (backtrack). The decision level represents the depth of (number of) the decision at the current search, starting at level 0 and increasing with each decision made or decreasing during backtracking.

The size and literal block distance (LBD) [12] are two popular measures for evaluating the quality of learnt clauses. The size of the clause is defined as the number of literals contained in the learnt clause. A literal is a variable or the negation of a variable. For a clause c comprised of literals $\{l_1, l_2, \dots, l_n\}$, the clause size is calculated as: $|C| = n$. The smaller the clause size is, the more useful the clause is evaluated to be. LBD is a more effective measure, defined as the count of distinct decision levels to which the literals of a clause belong. For a clause c with n

literals from decision levels $\{d_1, d_2, \dots, d_n\}$, LBD is given as:

$$\text{LBD}(c) = |\{d_1, d_2, \dots, d_n\}|$$

A lower LBD value suggests a high degree of interaction among the literals in the clause in the implication graph, enhancing the potential of the clause for effective search space pruning. Therefore, CDCL solvers prefer learnt clauses with lower LBD values. Owing to the effectiveness of LBD, the LBD score has been used not only for learnt clause management but also for other heuristic techniques (e.g., restart strategy [13] and decision branching heuristic [14]).

2.2. Structure of the SAT problem

The input for the SAT solver is usually given in the conjunctive normal form (CNF). A formula is said to be in CNF if it is a conjunction of one or more clauses, where a clause is a disjunction of literals. An example of a CNF formula is $(x \vee y) \wedge (\neg y \vee z)$, where $(x \vee y)$ and $(\neg y \vee z)$ are clauses, and x , y , and $\neg y$, and z are literals. Industrial SAT problems indicate prominent structural properties. Examples of structural measures are treewidth and modularity. For the graph representation of a SAT problem to calculate these measures, a variable incidence graph (VIG) is widely used.

Definition 1. Variable incidence graph (VIG):

A VIG is a graph representation of logical expressions. Let ψ be a SAT problem for a set of variables V and clauses C . The nodes of the VIG are denoted by V , and the edge $e_{v_i, v_j} \in E$ between v_i and $v_j \in V$ denotes the existence of a clause $c \in C$ containing v_i, v_j in c . The edge weight w is defined as $w(e_{v_i, v_j}) = \sum_{c \in C, v_i, v_j \in c} 1/\binom{|c|}{2}$.

Both the v and $\neg v$ literals belong to node v in VIG. When the same pair of variables (v_i, v_j) appears more than once in C , $w(e)$ sums all the weights of pairs. This means that the shorter the clause becomes, the higher the edge weight is, and vice versa. In this study, we used VIG as a graph representation of the SAT problem. The two structural measures, treewidth and modularity, are calculated on the VIG.

Definition 2. Treewidth:

Treewidth characterizes the degree of resemblance of a given graph to a tree. The smaller the treewidth, the more the graph resembles a tree. For a graph $G = (V, E)$, the treewidth is defined as the size of the largest bag (subgraph) in the optimal tree decomposition of the graph minus one.

$$\text{tw}(G) = \min_T \max_{i \in V(T)} |B_i| - 1 \quad (1)$$

Here, $T = (V(T), E(T))$ is a tree of G , and $B_i \subseteq V$ for $i \in V(T)$ represents a bag in the decomposition. The following conditions must hold: $\bigcup_{i \in V(T)} B_i = V$: Every vertex in G is included in at least one bag; for every edge $(u, v) \in E$, there exists a bag B_i such that $u, v \in B_i$. For all $B_i, B_j, B_k \in B_i | i \in V(T)$, if B_j is on the path from B_i to B_k in T , then $B_i \cap B_k \subseteq B_j$; the bags that contain a particular vertex in G must form a connected subtree in T .

This concept involves decomposing a graph into a tree-structured collection of subgraphs. Each subgraph contains a vertex subset. Treewidth is a useful parameter for various computational problems, as many NP-hard graph problems can be solved in polynomial time for graphs of bounded treewidth [15]. For this reason, the relationship between the hardness of SAT problems and the value of treewidth has been widely studied [4].

Definition 3. Modularity:

Modularity evaluates the quality of a given partition of the input graph. For a weighted edge graph $G = (V, E)$ and a partition $P = \{p_1, p_2, \dots, p_k\}$ of its vertices V , where P is pairwise disjoint and $\bigcup_{i=1}^k p_i = V$, the modularity Q is computed as follows:

$$Q(G, P) = \frac{1}{2m} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \left[A_{ij} - \frac{w(i) \times w(j)}{2m} \right] \delta(c_i, c_j) \quad (2)$$

Here, A_{ij} is the element at the i -th row and j -th column of the adjacency matrix A of the graph G , $w(i)$ denotes the sum of the weights of the edges connected to vertex i , m equals the sum of the weights of all edges in the graph, $c_i \in P$ is the community to which vertex i belongs, and $\delta(c_i, c_j)$ equals 1 if vertices i and j belong to the same community (i.e., $c_i = c_j$), and 0 otherwise.

A community in a graph is defined as a group of vertices more densely connected internally than the rest of the graph. The division of a graph into such communities is often denoted as a partition. Modularity quantifies the degree of deviation of the number of edges within communities from a random distribution. A high modularity score suggests that the graph exhibits a robust community structure. The following subsection introduces various research regarding the modularity and structures of SAT.

2.3. Related work

Ferrara et al. [2] revealed that the graphs of the industrial SAT problems exhibit relatively smaller treewidth than random problems. They attempted a theoretical explanation for the ability of CDCL solvers to efficiently solve industrial problems by demonstrating the characteristics of these industrial problems. Newsham et al. [3] investigated the relationship between the number of communities of a clause and its LBD values. They found that the degree of community structure is associated with the runtime of the instance. Furthermore, Newsham et al. [6] visualized the graph structure of SAT problems and demonstrated that the acquisition of learnt clauses (i.e., the search progress) diminishes the clarity of the structure. Ansótegui et al. [7] demonstrated that industrial SAT problems exhibit considerably stronger community structures, as measured via the modularity Q . Ansótegui et al. [16] analyzed the change in modularity (ΔQ) resulting from the acquisition of learnt clauses. Their findings indicate that although most learnt clauses exhibit a negative ΔQ , some exhibit a positive ΔQ . These studies served as an inspiration for our research to delve deeper into understanding the relationship between learned clauses and the structure of SAT.

3. Experiments

3.1. Change in the structural measures by learning

First, we assessed how much and when learning changes the structural measures. Related works [16, 6] showed that the structure of the problem becomes unclear (to be more random-like structure) as the overall trend after learning. Our objective is to study the change in more detail to understand the impact of learning and its deletion. For the measurement of the degree of the structure, we selected two measures: modularity and treewidth. Modularity is widely used in the context of SAT, and both of the abovementioned works use this concept. Although treewidth is relatively less popular in the context of SAT, it is a well-studied and popular measure in graph theory. We calculated the treewidth and modularity values using approximations as the calculations of exact values of them are NP-hard. FlowCutter [17] calculated the upper-bound value of treewidth, an application submitted at the PACE 2017 competition. The community decomposition for modularity was conducted based on the Louvain community detection algorithm using networkx, a Python library [18]. Our benchmark consisted of 400 problems from the main tracks of the SAT competition in 2021. In this study, we excluded problems from the result that provided no treewidth value within 1200 s or no modularity value within 24 h. We performed experiments on a computer equipped with an AMD Threadripper Pro 3995WX processor (64 cores) and 512 GB (128 GB four slots, DDR4-3200 MHz) of RAM. We selected Glucose 4.2.1 [19] as the base solver for experiments because it is the first solver with the implementation of LBD.

The scatterplot in Figure 1a reveals the variation in the treewidth value between ψ_{org} and ψ_{last} . Let ψ_{org} be the original problem and C_t be the learnt clauses maintained by the solver at time t . C_t is the result of clauses learnt and deleted up to point t . The time when the search is finished is t_{last} , which is either at the 3600 s timeout or when the solver finds a SAT or UNSAT solution. The problem ψ_t at time t is defined by $\psi_{org} + C_t$; thus $\psi_{last} = \psi_{org} + C_{t_{last}}$. Hereafter, ψ refers to the graph of the problem. A dot in the scatter plot figure represents a problem, and its position (original, last) represents the values of treewidth $tw(\psi_{org})$ and $tw(\psi_{last})$, respectively. In 78% of problems, $tw(\psi_{org}) < tw(\psi_{last})$ were observed, and in particular, 11% of problems have the value changed by a factor of 10 or more. Note that the figure is on a double-logarithmic graph. Similarly, Figure 1b details the modularity value for each problem. This result also reveals $mod(\psi_{org}) > mod(\psi_{last})$ for 84% of the problems. In particular, there were 64% of problems that showed almost zero values below 0.05. These results are consistent with those of existing studies; The learning changes the structure toward the random-like one –in other words, the learning destroys the structure.

Next, Figure 2 details the temporal change of the values of the measures, showing four problems as typical examples. Similar trends are observed in most problems, as seen in [20]. The left and right vertical axes indicate the modularity value and treewidth, respectively. The horizontal axis indicates the time series: ‘original’ indicates the values of ψ_{org} , original CNF graph before preprocessing; ‘0’ is the value of the graph of the preprocessed formula. We observed these measures approximately every 300 s until t_{last} . The figures claim two arguments. First, the values of measures change immediately after starting the search, particularly between 0 and 300 s, and do not vary monotonically. Second, the measures oscillate back in a zigzag

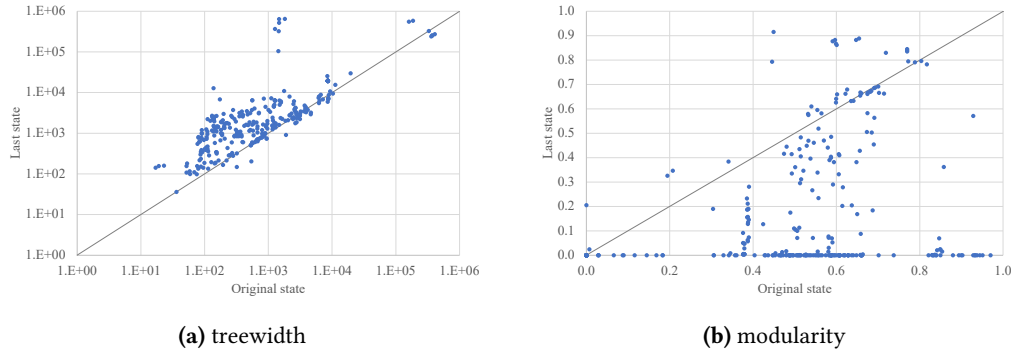


Figure 1: Scatter plot between the original formula ψ_{org} and last formula ψ_{last} including learnt clauses

manner as time progresses. This phenomenon is assumed to occur as the result of the deletion of learnt clauses. These results suggest a hypothetical explanation: the deletion of badly evaluated clauses with large LBD values restores the changed structure; and thus, the deletion of learnt clauses is effective for solver performance because of the power of structure restoration.

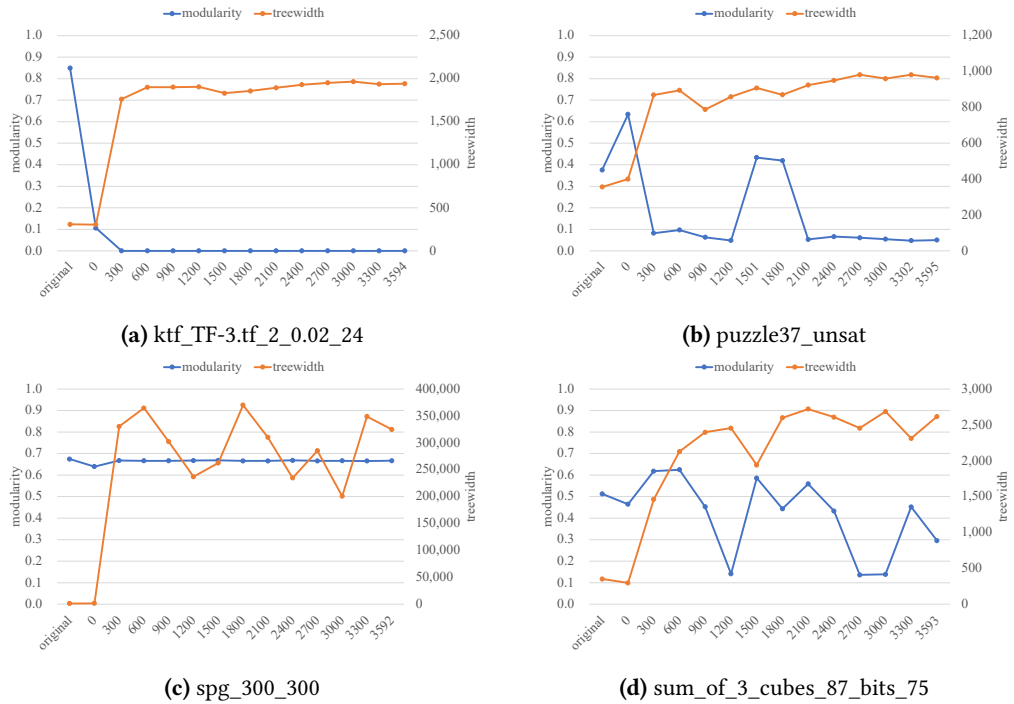


Figure 2: Values of Treewidth and modularity change over time

3.2. Analysis of the relationship between the quality and structural measure

We hypothesize that high-quality clauses decrease less the structural properties of SAT problems less than low-quality clauses, or even increase them. If this is true, it will resolve the contradiction between the explanation that the structure of the problem is a factor in the efficiency of the SAT solver and the observation that learnt clauses increase the efficiency of the SAT solver while destroying the structure of the problem. We conducted an experiment to identify which learnt clause changed the structure more. We analyzed the relationship between LBD and the degree of change by comparing the delta of the modularity Q between ψ_{org} and $\psi_{org} + c$ a learnt clause c , namely, delta with the addition of a learnt clause.

Definition 4. For a learnt clause c , $\Delta Q(c) = Q(\psi_{org} + c, P') - Q(\psi_{org}, P)$.

Modularity Q requires a graph G and its partition P . The calculation of partition (community) detection costs numerous resources for millions of learnt clauses. Therefore, we calculated ΔQ with two assumptions: partition P does not change by adding a clause, that is, $P' = P$; the change in m , i.e., the sum of the weight of all edges, can be ignored by adding a clause with weight 1. These assumptions are reasonable when adding at most one learnt clause to the original problem that has thousands of clauses. We ran Glucose 4.2.1 with a 3600 s time limit to export all the learnt clauses and their information, such as LBD and its literals, to files. The export was conducted when learnt clauses were obtained, updated, or deleted. The clause was determined to be the same as literal identity.

Four typical examples are depicted in Fig 3 as boxplots of the ΔQ distribution for each LBD. Each chart shows the median as the orange line, quartiles as the edges of the box, and outliers as the circles if data points are more than 1.5 times the interquartile box. Clear trends are observed in these figures —the smaller the LBD is, the larger the average ΔQ is. This result indicates that better clauses (smaller LBD clauses) slightly decrease or even increase the modularity value. This trend was observed among most problems, as shown in [20]. We analyzed the differences between all the clauses and the clauses of LBD 2 to determine the overall trend among all problems. The average ΔQ for all clauses across problems was -5.3×10^{-5} , with 76% displaying a negative mean ΔQ . In contrast, clauses with an LBD of 2 had a mean ΔQ of -4.4×10^{-5} , and 68% exhibited negative values. These results show that clauses with small values of LBD (here LBD of 2) are not necessarily positive clauses, while the percentage of clauses with positive ΔQ is high (i.e., enhancing the structure of the problem). This supports our hypothesis that ‘high-quality clauses decrease the structural properties of SAT problems less than low-quality clauses, or even increase them.’

Conversely, to see what characteristics are present in clauses with positive ΔQ values, a comparative analysis was performed between clauses with positive ΔQ and any ΔQ . Mean LBD values and the number of communities per problem for each set of learned clauses were calculated. The aggregate means are shown in Table 1. Similar to the boxplot, positive ΔQ clauses had lower LBD values and fewer community affiliations. Moreover, as anticipated, clauses characterized by a positive ΔQ were affiliated with a smaller number of communities. Ansótegui et al.,[16] revealed that although most of the learnt clauses decreased the modularity value, some clauses increased it. Combining this with our results suggests that learning low-quality clauses decreases modularity, and learning high-quality clauses slightly decreases or

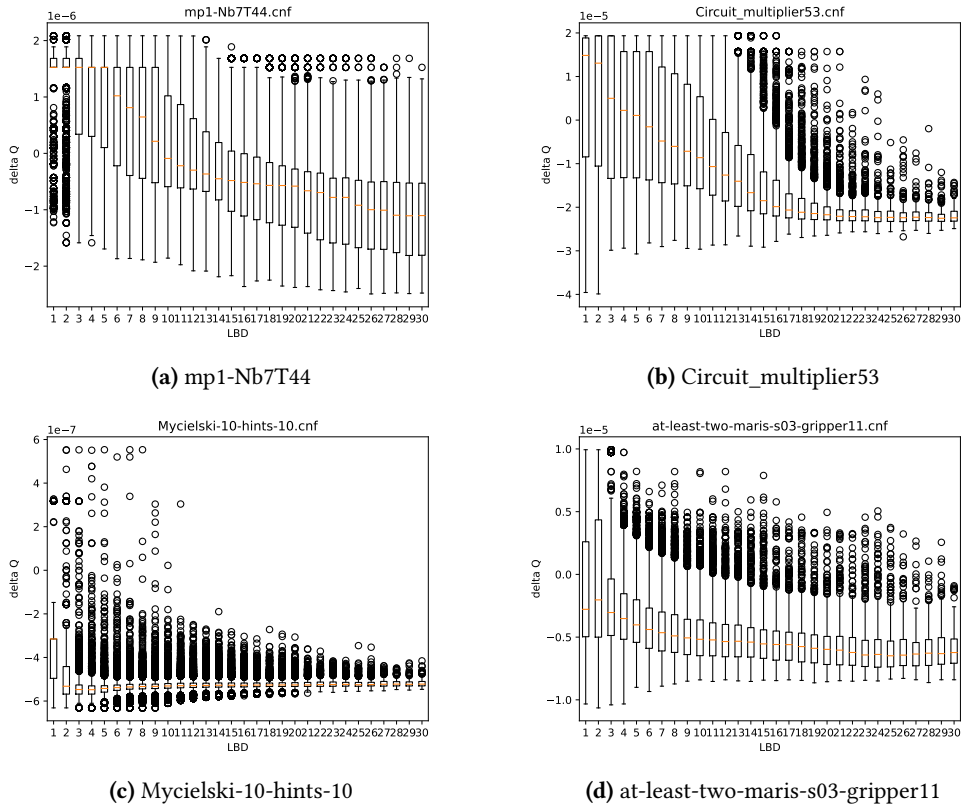


Figure 3: Boxplot between modularity ΔQ and the learnt clause LBD

even increases modularity.

Table 1

Differences between clauses of positive ΔQ and all clauses. The values are the average of all problems.

	ratio	average LBD value	number of community
clauses of positive ΔQ	24%	6.75	3.04
all clauses	100%	11.93	6.90

3.3. Further analysis on the relationship between LBD and modularity

The previous analysis showed that high-quality clauses (measured by LBD values) have a small impact on the decrease of modularity (i.e., ΔQ). However, this is probably because of the pseudo-correlation with size, not between LBD and ΔQ , because modularity is a measure on the graph, LBD is correlated with the size of the clause, and the size strongly influences the VIG weight. To analyze the impact solely based on the LBD values excluding the impact of size, we visualized the correlation as a heatmap among the clause size, LBD, and mean ΔQ in Figure 4. The color of each cell represents the mean of ΔQ , and the vertical and horizontal axes

represent the LBD and clause size, respectively. When examining colors vertically for a given size, a notable difference is observed —smaller LBD values correspond to larger average values of ΔQ . These results support the argument presented in Figure 3. The complete results of all problems are available in [20] where most of the problems show a similar trend.

Furthermore, we analyzed the overall trend of all problems by calculating the partial correlation coefficients of the LBD, size, and ΔQ values to confirm the above implication from the observations. A partial correlation coefficient measures the relationship between two variables when one variable is held constant. This assesses the pure correlation between two variables, excluding the influence of other variables. To define the partial correlation coefficient, we consider three random variables x, y, z . The partial correlation coefficient of x and y is denoted as $r_{xy.z}$, and the following formula can be derived:

$$r_{xy.z} = \frac{r_{xy} - r_{xz}r_{yz}}{\sqrt{(1 - r_{xz}^2)(1 - r_{yz}^2)}}$$

. Here, r_{xy} represents the correlation coefficient of x and y , and z is the control variable of $r_{xy.z}$ calculating the correlation coefficient of x, y without the influence of z . The partial correlation coefficients were found to be $+0.54$ between the size and LBD, -0.09 between the size and ΔQ , and -0.24 between LBD and ΔQ , respectively. These results indicate that LBD and ΔQ weakly correlate, even without any outlier operation, and allow us to conclude that the value of LBD affects the value of ΔQ . This could explain the effectiveness of the three-tier scheme for clause management. The clauses that can enhance the structure remain in tier 1 (measured by LBD), and those that destroy the structure are put in tier 3 and will be removed soon.

4. Conclusion

This study explored the impact of clause learning in CDCL SAT solvers on the structural properties of SAT problems, focusing specifically on two measures: treewidth and modularity. The investigation was organized around two key experiments: the time variance in these measures during the search process to assess the impact of clause deletion, and the relationship between the quality of a learnt clause and its subsequent effect on modularity.

The first experiment revealed that these measures do not change monotonically but fluctuate periodically. This fluctuating pattern is due to the continuous interplay of clause learning and deletion processes, indicating that clause deletion can restore the changed structure. This implies differences in the impact of the structural change between the deleted and remaining clauses. The second experiment showed that the quality of learnt clauses, measured by LBD values, was a significant factor impacting structural changes; low-quality clauses, characterized by large LBD values, were found to decrease in modularity more than high-quality clauses. This result explains the fluctuation in the first experiment. More importantly, it resolves the question of ‘*why learnt clauses are effective though they destroy the structure of problems?*’. The learnt clauses as a whole (considering all obtained clauses) tend to decrease the structural properties of problems, but the actual structural properties are maintained or even increased owing to high-quality clauses. These insights can potentially inform that the benefits of clause deletion are beyond addressing memory constraints and reducing the propagation time —forgetting

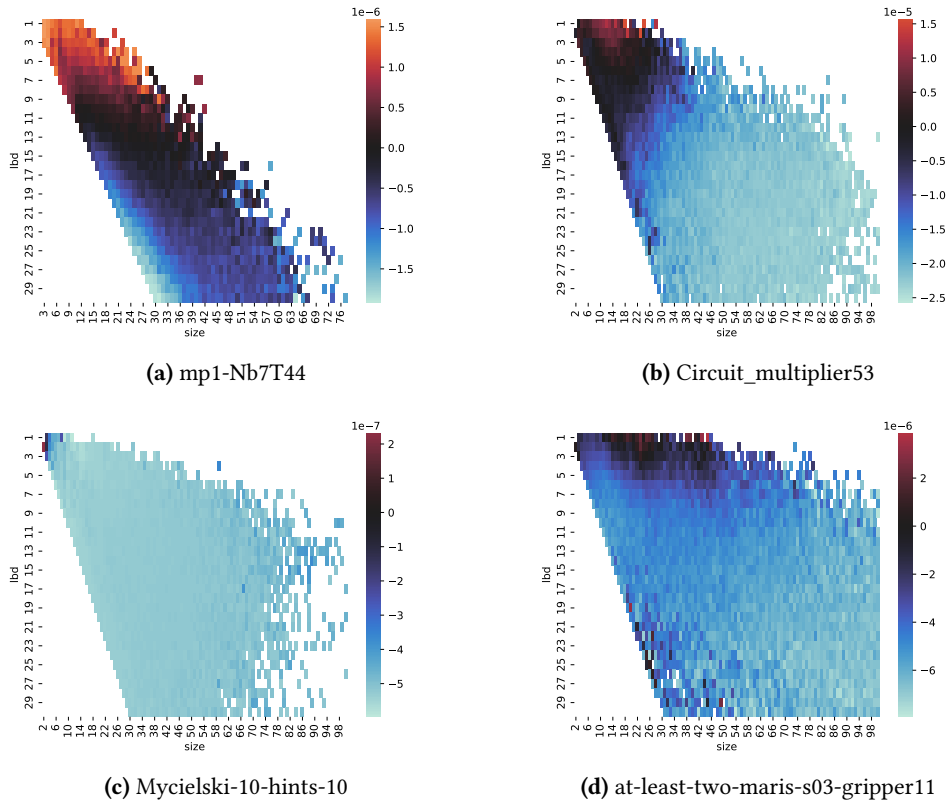


Figure 4: Heatmap among modularity ΔQ , size, and LBD

low-quality learnt clauses helps restore the distinct structural properties of problems to maintain the problem ‘structured’, enabling its efficient solution.

In future research, exploring similar studies employing other measures, such as scale-free and centrality, could offer further insight into our current findings. The interrelation between the degree of modularity changes (ΔQ) and the consequent effect on the solver runtime presents yet another intriguing area of study. If a robust correlation in the runtime is found, ΔQ or other structural measures could provide effective evaluation measures for the quality of learnt clauses. They could even replace traditional measures, such as LBD or size; for instance, a clause that increases the structural measure might be deemed beneficial. Beyond modularity ΔQ , other structural measures also warrant consideration for this purpose and can enhance the performance of CDCL SAT solvers. Moreover, these measures could be invaluable specifically for parallel learnt clause sharing. As LBD is derived from the solver decision tree, it inherently depends on the state of the search. Given that each parallel worker would operate in a distinct state of search, this inevitably results in different decision trees and LBD values. This suggests that basing the decision of learnt clause sharing on LBD might not be the most effective approach. On the other hand, structural measures derived from the original instance, such as ΔQ employed in this study, potentially offer better solutions. These measures are common

information available to all parallel workers and can thus provide a unified measure to judge the usefulness of clauses.

Acknowledgments

We express our deep gratitude to the reviewers for their insightful comments and to participants at the Pragmatics of SAT workshop for enriching discussions. We also wish to acknowledge the meaningful discussion with Professor Hiroshi Imai and the members of his laboratory.

References

- [1] R. R. Williams, C. P. Gomes, B. Selman, On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search, *structure* 23 (2003).
- [2] A. Ferrara, G. Pan, M. Y. Vardi, Treewidth in verification: Local vs. global, in: *Logic for Programming, Artificial Intelligence, and Reasoning*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 489–503. doi:10.1007/11591191_34.
- [3] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, L. Simon, Impact of community structure on sat solver performance, in: *Theory and Applications of Satisfiability Testing – SAT 2014*, Springer International Publishing, Cham, 2014, pp. 252–268. doi:10.1007/978-3-319-09284-3_20.
- [4] T. N. Alyahya, M. E. B. Menai, H. Mathkour, On the structure of the boolean satisfiability problem: A survey, *ACM Comput. Surv.* 55 (2022) 1–34. doi:10.1145/3491210.
- [5] P. Gregory, M. Fox, D. Long, A new empirical study of weak backdoors, in: P. J. Stuckey (Ed.), *Principles and Practice of Constraint Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 618–623. doi:10.1007/978-3-540-85958-1_53.
- [6] Z. Newsham, W. Lindsay, V. Ganesh, J. H. Liang, S. Fischmeister, K. Czarnecki, Satgraf: Visualizing the evolution of sat formula structure in solvers, in: *Theory and Applications of Satisfiability Testing – SAT 2015*, Springer International Publishing, Cham, 2015, pp. 62–70. doi:10.1007/978-3-319-24318-4_6.
- [7] C. Ansótegui, M. L. Bonet, J. Giráldez-Cru, J. Levy, L. Simon, Community structure in industrial SAT instances, *The journal of artificial intelligence research* 66 (2019) 443–472. doi:10.1613/jair.1.11741.
- [8] M. J. H. Heule, O. Kullmann, V. W. Marek, Solving and verifying the boolean pythagorean triples problem via Cube-and-Conquer, in: *Theory and Applications of Satisfiability Testing – SAT 2016*, Springer, Cham, 2016, pp. 228–245. doi:10.1007/978-3-319-40970-2_15.
- [9] C. Bright, I. Kotsireas, A. Heinle, V. Ganesh, Complex golay pairs up to length 28: A search via computer algebra and programmatic sat, *Journal of Symbolic Computation* 102 (2021) 153–172. doi:10.1016/j.jsc.2019.10.013.
- [10] R. J. Bayardo, R. 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*, AAAI Press, 1997, p. 203–208. URL: <https://dl.acm.org/doi/10.5555/1867406.1867438>.
- [11] L. Zhang, C. Madigan, M. Moskewicz, S. Malik, Efficient conflict driven learning in a

- boolean satisfiability solver, in: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281), 2001, pp. 279–285. doi:10.1109/ICCAD.2001.968634.
- [12] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: Proceedings of the 21st international joint conference on Artificial intelligence, IJCAI'09, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009, pp. 399–404. URL: <https://dl.acm.org/doi/10.5555/1661445.1661509>.
- [13] A. Biere, A. Frohlich, Evaluating cdcl restart schemes, in: Proceedings of Pragmatics of SAT 2015 and 2018, volume 59 of *EPiC Series in Computing*, EasyChair, 2019, pp. 1–17. doi:10.29007/89dw.
- [14] W. Chang, G. Wu, Y. 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), 2017, pp. 1–6. doi:10.1109/ISKE.2017.8258780.
- [15] E. C. Freuder, Complexity of k-tree structured constraint satisfaction problems, in: Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1, AAAI'90, AAAI Press, 1990, p. 4–9. URL: <https://dl.acm.org/doi/10.5555/1865499.1865500>.
- [16] C. Ansótegui, J. Giráldez-Cru, J. Levy, L. Simon, Using community structure to detect relevant learnt clauses, in: Theory and Applications of Satisfiability Testing – SAT 2015, Springer International Publishing, 2015, pp. 238–254. doi:10.1007/978-3-319-24318-4_18.
- [17] M. Hamann, B. Strasser, Flow-cutter-pace17, 2017. URL: <https://github.com/kit-algo/flow-cutter-pace17>, last accessed: 2023-04-17.
- [18] Networkx: Modularity, <https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.quality.modularity.html>, 2023. Last accessed: 2023-04-17.
- [19] G. Simon, Laurent; Audemard, About the glucose sat solver, 2018. URL: <https://www.labri.fr/perso/lsimon/research/glucose/>, accessed: 2023-3-8.
- [20] Image repository for all instances, <https://github.com/messhiida/PoS2023-images.git>, 2023. Last accessed: 2023-05-03.