SATViz: Real-Time Visualization of Clausal Proofs

Tim Holzenkamp, Kevin Kuryshev, Thomas Oltmann, Lucas Wäldele, Johann Zuber, Tobias Heuer, and Markus Iser

Algorithm Engineering, Institute of Theoretical Informatics, KIT-Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany

Abstract
Visual layouts of graphs representing SAT instances can highlight the community structure of SAT instances. The community structure of SAT instances has been associated with both instance hardness and known clause quality heuristics. Our tool SATViz visualizes CNF formulas using the variable interaction graph and a force-directed layout algorithm. With SATViz, clause proofs can be animated to continuously highlight variables that occur in a moving window of recently learned clauses. If needed, SATViz can also create new layouts of the variable interaction graph with the adjusted edge weights. In this paper, we describe the structure and feature set of SATViz. We also present some interesting visualizations created with SATViz.

1 Introduction
Visual representations of algorithms can improve our understanding of algorithmic concepts and the nature of the underlying problems. Visualizations of the Conflict-Driven Clause Learning algorithm (CDCL) are of high interest, as this is currently the most successful algorithm for solving the propositional satisfiability problem (SAT problem). Despite the complexity of the SAT problem, implementations of CDCL in so-called SAT solvers are successfully used in industrial practice, e.g., in software verification [4], hardware verification [12], product configuration [21], or planning [18].

For satisfiable instances, SAT solvers output a variable assignment which serves as a certificate of satisfiability. Such a certificate can be checked in polynomial time and hence SAT \( \in \text{NP} \). However, modern SAT solvers also output certificates of unsatisfiability – a proof given in the DRAT format [8]. Such a proof can be checked efficiently by a procedure with runtime polynomial in the proof length. The output of verifiable proofs increases the trust in the correctness of SAT algorithms. Trust is crucial in scenarios where SAT solvers, e.g., verify the absence of bugs in safety-critical software.

Famous proofs of unsatisfiability that solved previously open mathematical problems, e.g., the Boolean Pythagorean triples problem, have been generated by SAT solvers [9, 10]. The explainability of such automatically generated proofs is of great interest. This is mainly due to their sheer size, e.g., the total size of the proof of the Boolean Pythagorean triples problem is 200 TB.

The source of such generated proofs are the clauses learned by a CDCL SAT solver. In CDCL, learned clauses, i.e., small proofs, also help to accelerate search-space navigation. The structure of the resulting reasoning is hard to grasp as millions of clauses are involved. The average instance of the SAT competition 2021 benchmark contains 2.3 millions of clauses. When running Kissat [2] on these instances we observe an average of 8676 (with a median of 4040) learned clauses per second.

*Project Supervision
†Project Definition and Supervision
To this end, visual representations illustrating learned clauses during the execution of SAT solvers can bridge the gap to gain a better understanding of the structure of automated reasoning and resolution-space navigation. In this paper, we present a tool which creates real-time visualizations of proof-sequences as they are created by a given SAT solver or proof file.

### 2 Background

Instances of the SAT problem are given in *Conjunctive Normal Form* (CNF) and are represented by a set of clauses. Each clause is a set of literals and represents the disjunction of those literals. Each literal is a negated or non-negated Boolean variable.

CDCL combines search (cf. DPLL algorithm) and resolution (cf. saturation algorithm) [3]. CDCL solvers maintain and extend a current partial assignment which is created from repeated decision and subsequent Boolean constraint propagation. On conflict, CDCL infers a new clause by resolution of the conflict-reason clauses which are determined by analyzing the implication graph of the current partial assignment [7]. If the empty clauses can be inferred, then the set of learned clauses forms a testable proof of the unsatisfiability of an instance. Clause learning is also an important means of narrowing down the search space of satisfiable instances, e.g., to escape from unsatisfiable regions of the search space early on. As the relevance of a learned clause for the overall search process can not be known in advance, clause learning and forgetting is controlled by heuristics [15].

A SAT formula in conjunctive normal form (CNF) can be represented as a hypergraph. Hypergraphs are generalizations of regular graphs in which each hyperedge can connect more than two nodes. In this model, each variable corresponds to a node and each clause is modeled as a hyperedge. While there is little work on visualizing hypergraphs, we transform the hypergraph to a regular graph. The variant in which each hyperedge is replaced by a clique is known as the *Variable Interaction Graph* (VIG) in the SAT domain (cf. Section 4.2).

### 3 Related Work

In the area of SAT solving, the tools *DPvis* and *3Dvis* by Sinz have been used to visualize the structure of SAT instances based on the force-directed layout of their graph representations [20]. *DPvis* can also visualize formula evolution by simplifying according to the branching and propagation steps in runs of the integrated SAT solver Minisat [6]. Visualizations of *3Dvis* appear in Knuth’s “The Art of Computer Programming” Vol. 4.6 on “Satisfiability” (the author’s favorite

| Interactive       | DPvis | 3Dvis | iSAT | SATGraf | SATViz |
|-------------------|-------|-------|------|---------|--------|
| Multiple Reduction Types | x     | x     | ✓    | ✓       | ✓      |
| 3D Layout         | x     | ✓     | x    | x       | x      |
| Real-time Animations| ✓     | x     | x    | ✓       | ✓      |
| Graph Contractions | x     | x     | ✗    | ✗       | ✓      |
| IPASIR Interface  | x     | x     | ✗    | ✗       | ✓      |
| DRAT Interface    | x     | x     | ✗    | x       | ✓      |

Table 1: Feature Coverage of CNF Formula Visualizers
iSAT is a tool for instrumentation and interactive control of SAT solvers. iSAT facilitates the analysis of intermediate solver states. For external visualization, iSAT can export several types of graph representations of formulas to files [16]. The tool SATGraf by Newsham et al. visualizes community structure of SAT formulas. SATGraf can also display statistics and solving progress of specially instrumented SAT solvers running on small SAT instances [14].

Table 1 shows a comparison of the features covered by SATViz and related tools. Except for the more performance-critical 3Dvis, all tools considered offer some form of interactive control to reinitialize the visualization with the current solver state. While DPvis and 3Dvis are limited to the use of the VIG, the other tools offer additional options and some form of extensibility. 3Dvis is the only tool that can generate three-dimensional layouts of SAT instances.

Real-time animations are supported by DPvis, SATGraf and SATViz, but each of them highlights a different aspect. DPvis focuses on search progress via unit resolution, SATGraf focuses on the survival of the pre-computed communities, and SATViz highlights recently learned clauses. Both SATGraf and SATViz can relayout an instance which has been modified by learned and forgotten clauses.

However, SATViz is the only tool that facilitates the handling of large industrial SAT instances in real-time. This is possible since SATViz offers graph contractions for reducing the number of edges while preserving the graph structure. Like this, even large SAT instances can be visualized efficiently and also become less cluttered.

SATViz instruments SAT solvers with standard methods of the IPASIR [1] or can moreover replay proofs given in the DRAT format [8]. In contrast to other tools, SATViz can visualize the course of any SAT solver which implements the IPASIR or exports DRAT proofs.

### 4 The Tool SATViz

With our tool SATViz, we can transform CNF formulas to graphs and visualize them using a graph drawing algorithm. A SAT solver running on the local or a remote machine can then connect and send learned clauses to the application, which are then highlighted in the visual graph representation. Thereby, the color of a node represents how frequently the corresponding variable appeared in learned clauses (similar to a heat-map). Through this visualization we can better analyze the working of CDCL algorithms and better understand the structure of different problem instances. Code and instructions for using SATViz can be found on GitHub.²

### 4.1 Architecture

Figure 1 illustrates the core components of SATViz' architecture. It can be divided into two central components: the clause consumer and the clause producer. The clause consumer is responsible for visualizing SAT instances and learned clauses, while the clause producer sends a stream of learned clauses to the clause consumer over the network. This decoupling adds some flexibility to the architecture, as learned clauses can be sent from different machines from any software implementing the clause producer protocol.

The clause producer of SATViz can read clauses from two types of sources: a file containing a clausal proof or a shared library of an SAT solver implementing the IPASIR interface [1]. In the former, SATViz reads a clausal proof, e.g., in DRAT file format [8], and streams it to the clause consumer. In the latter, the clause producer starts a SAT solver initialized with a given

---

¹https://satcompetition.github.io/2022/logo2022-large.png
²https://github.com/satviz/satviz
CNF formula and streams learned clauses to the clause consumer at the time they are exposed over the IPASIR interface (intercepted in an \texttt{ipasir\_set\_delete} callback function).

The clause consumer buffers the incoming stream of learned clauses and sends chunks of incoming clauses with a fixed frame rate to the \textit{graph transformer} component. The graph transformer is responsible for visualizing learned clauses in the graph representation. It adds new edges or updates their weight, and changes the colors of the corresponding nodes contained in the learned clauses. This transformation is configurable and extensible. The graph transformer configuration includes parameters to control the heat-map which highlights learned clauses (cf. Section 4.4), different graph models to represent CNF formulas (cf. Section 4.2), as well as options for reducing the size of larger instances to improve the readability of the visualization and to speedup the performance of the animation (cf. Section 4.3).

Given a graph, the layout algorithm uses force-directed placement to create the visual representation of the graph. We use the implementation provided by the Open Graph Drawing Framework (OGDF) \cite{5}. The heat-map algorithm continuously updates node colors according to the incoming learned clauses. The placement algorithm creates and uses an initial layout based on the original formula. But on demand, it can recalculate a new layout based on the modified graph induced by the clauses which have been learned and forgotten so far. The user interface also allows to pause, stop, rewind and replay the solver run and to step in at any time into the proof. For larger graph the magnification feature using the mouse wheel is very practical.

\textsc{SATViz} also supports to record the visualization including user interactions and exports this to a video file. \textsc{SATViz} can also operate in a headless mode and silently create videos from given proofs or SAT solver runs in the background. Some sample videos can be found in the dedicated YouTube playlist of Kuryshev.\footnote{https://www.youtube.com/playlist?list=PLhFsx92qubLxMsFmJvTbYJS}

\section*{4.2 Hypergraph Reduction}

A CNF formula induces a hypergraph such that the variables correspond to nodes and each clause forms a hyperedge spanning its variables. Hypergraphs are hard to visualize directly, which is why reductions to graph representations are used in practice \cite{20}. The two most common representations are the \textit{clique expansion} and the \textit{bipartite graph representation} \cite{11,19}. The former adds a clique between all nodes contained in a hyperedge, while the latter additionally models the hyperedges as nodes and connects each with their corresponding nodes. However, the clique expansion unnecessarily increases the size of our graph model in presence of large hyperedges, while the bipartite graph representation adds much more nodes than necessary.
Figure 2: Visualization of instance AProVE09-15 (94663 variables) highlighting clauses learned by Kissat with no contraction (left) and 10 iterations of max-weight contractions (right).

We therefore present a third reduction to which we refer to as the ring reduction. Here, we sort the node IDs of each hyperedge in increasing order and connect consecutive nodes with an edge. Additionally, we add an edge between the two nodes with the lowest and highest ID in each hyperedge. This representation combines the advantages of both standard representations as the number of nodes equals to the nodes in clique expansion, while the number of edges equals to the edges in the bipartite graph representation.

In SATViz, we support the ring reduction and clique expansion as graph models. Additionally, each hyperedge contributes an edge weight which is a configurable function inversely proportional to its own size.

4.3 Graph Contraction

The size of SAT instances have a large impact on visualization performance. In initial experiments, the time to layout SAT instances with more than 50k variables reached a for usability critical time limit of one minute. Moreover, visualizations of larger SAT instances can become cluttered and their structure concealed as too many nodes compete for space. For these reasons, SATViz features a pre-processing step in which it runs a graph contraction algorithm which is based on the well-known label propagation algorithm [17]. The algorithm assigns labels to the nodes. Initially, each node has its own label. Then, the algorithm iterates over the nodes in random order – also called a round – and whenever a node is visited, it is assigned the label appearing most frequently in its neighborhood. This is repeated for a fixed number of rounds or until none of the nodes changed its label in a round. Subsequently, we collapse nodes with the same label into a single node and connect them to nodes corresponding to adjacent labels. We repeat the clustering and contraction procedure recursively until the size of the graph becomes manageable for our graph drawing algorithm.
Figure 3: Visualization of instance AProVE09-15 highlighting clauses learned by Kissat – with a heat-map frame width of 1000 clauses – after 1000, 1690, and 3090 conflicts (from left to right).

Figure 2 shows two visualizations of the SAT instance AProVE09-15 which has 94663 variables. The left visualization represents the input instance, while the right visualization was obtained with our graph contraction algorithm. As it can be seen, the graph contractions facilitate the analysis of large SAT instances and can unveil hidden structures.

### 4.4 Highlighting Areas of Interest

SATViz uses configurable heat-map colors to highlight recently learned clauses. We use a heat value to determine the color for each node. This value is determined by either (i) normalizing each variable’s occurrence count for the last $k$ learned clauses or (ii) assigning the maximum heat value to the most recently used variables and then reduce the value to zero again within the next $k$ time steps (unless it appears again in a learned clause). The parameter $k$ is configurable and assists in switching between coarse- and fine-grained proof analysis. The color of node contractions is determined by the average heat value of the variables which it represents. SATViz also has a parameter for adjusting the speed of the animation.

Figure 3 shows three snapshots of a run of Kissat [2] on the instance AProVE09-15 within the first 5000 learned clauses. The heat-map highlights variables occurring in the $k = 1000$ most recently learned clauses. The displayed sequence of resolution steps appears in the animation like a windshield wiper.

### 4.5 Evolution of a Proof via Relayouting

SATViz offers to pause the animation and relayout the instance with the weights adjusted by the learned and deleted clauses. Figure 4 shows the structural changes in the AProVE09-15 instance after 6000 clauses learned by Kissat. We observe several variable connections become loose and degenerate to simple structures, which means that many original clauses become redundant and get deleted within the first 6000 conflicts. However, some variable connections get stronger, as the graph forms a densely connected community at the center of the circular structure of the relayouted instance in Figure 4.

---

4https://gbd.it.kit.edu/file/8a244542d09e20e9e9e8813ce7089c4135
Figure 4: Visualization of instance AProVE09-15 solved by Kissat. The right graph shows the instance after a relayout using adapted weights after 6000 learned or forgotten clauses. The left graph shows the magnified center of the circular region of the right graph.

4.6 Observing the Evolution of Formula Structure

Figure 5 shows the evolution of the SAT instance Newton.5.1.i.smt2-cvc4 as it was solved by Kissat. We observe that the original structure decays and a new structure emerges in which several nodes form circular shapes and tentacles. However, the resulting tentacles and circles are arranged around a densely connected core of nodes. This is due to several variables becoming weakly connected or even disconnected from the densely connected instance core. This densely connected core is where further searching and learning takes place, while nothing happens to the nodes that are now loosely connected. As learning progresses, more and more nodes detach from this core, so that it thins out and becomes flatter, while the inner circles around this core become more densely populated.

5 Future Work and Conclusion

Little research has gone into incremental weight updates in a force-directed layout algorithm. We conducted initial experiments with real-time incremental adjustment of force-directed node placement. However, it was problematic to regulate the extend of which node positions are allowed to change in such incremental updates. Developing a stable incremental placement algorithm is a research project on its own.

Our future work will mainly focus on the consolidation and extension of SATViz in the following respects. SATViz should be able to process additional input about named equivalence classes of variables. Such extra information can be displayed, e.g., by using additional node colors or when hovering nodes with the mouse. Equivalence classes of variables can also be used to specify preferences for the contraction algorithm or to increase the node proximity for such

\footnote{https://gbd.iti.kit.edu/file/92b98ab8055b143e0283a215a70cf001}
related variables with an adapted layout algorithm. With the possibility to process and use classes of variables, SATViz can be even more helpful in analyzing large proofs and in identifying patterns and stages in proof generation.

We also want to increase the number of available graph reduction methods and parameters to control them. As SATViz decouples clause producers via sockets, SATViz can also be extended to monitor clause exchange of parallel SAT solvers in the cloud.

Visually observing the effects of the methods and heuristics that control clause learning and forgetting, or even clause sharing, can deepen our understanding of these methods and how they interact. The visualizations of SATViz can even leverage our understanding of the coarse structure of clausal proofs. SATViz can help identify and analyze arguments that consist of a large number of clauses. Recognizing patterns in clause proofs can help us better understand, compress, and explain them.

References

[1] Tomás Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT race 2015. *Artif. Intell.*, 241:45–65, 2016.
[2] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 - Solver and Benchmark Descriptions*, volume B-2020-1 of Department of Computer Science Report Series B, pages 51–53. University of Helsinki, 2020.
[3] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.
[4] Marko Kleine Büning, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (ogdf). *Handbook of graph drawing and visualization*, 2011:543–569, 2013.
[5] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2003.
[6] Nick Feng and Fahiem Bacchus. Clause size reduction with all-uip learning. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2020.
[7] Marijn J. H. Heule. The DRAT format and drat-trim checker. *CoRR*, abs/1610.06229, 2016.
[8] Marijn J. H. Heule. Schur number five. In *Proc. AAAI*, pages 6598–6606, 2018.
[9] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proc. SAT*, pages 228–245, 2016.
[10] T. C. Hu and K. Moerder. Multiterminal Flows in a Hypergraph. In T.C. Hu and E.S. Kuh, editors, *VLSI Circuit Layout: Theory and Design*, chapter 3, pages 87–93. 1985.
[12] Daniela Kaufmann and Armin Biere. Amulet 2.0 for verifying multiplier circuits. In Jan Friso Groote and Kim Gulstrand Larsen, editors, Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II, volume 12652 of Lecture Notes in Computer Science, pages 357–364. Springer, 2021.

[13] Donald E Knuth. The art of computer programming, Volume 4, Fascicle 6: Satisfiability. Addison-Wesley Professional, 2015.

[14] 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 A. Weaver, editors, Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, volume 9340 of Lecture Notes in Computer Science, pages 62–70. Springer, 2015.

[15] Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In Marijn Heule and Sean A. Weaver, editors, Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, volume 9340 of Lecture Notes in Computer Science, pages 307–323. Springer, 2015.

[16] Ezequiel Orbe, Carlos Areces, and Gabriel G. Infante López. isat: Structure visualization for SAT problems. In Nikolaj S. Bjørner and Andrei Voronkov, editors, Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings, volume 7180 of Lecture Notes in Computer Science, pages 335–342. Springer, 2012.

[17] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. Physical Review E, 76(3):036106, 2007.

[18] Dominik Schreiber. Lilotane: A lifted sat-based approach to hierarchical planning. J. Artif. Intell. Res., 70:1117–1181, 2021.

[19] Daniel G. Schweikert and Brian W. Kernighan. A Proper Model for the Partitioning of Electrical Circuits. pages 57–62, 6 1972.

[20] Carsten Sinz. Visualizing SAT instances and runs of the DPLL algorithm. J. Autom. Reason., 39(2):219–243, 2007.

[21] Johannes Werner, Tomás Balyo, Markus Iser, and Michael Klein. Fast approximate calculation of valid domains in a satisfiability-based product configurator. In Michel Aldanondo, Andreas A. Falkner, Alexander Felfernig, and Martin Stettinger, editors, Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021, volume 2945 of CEUR Workshop Proceedings, pages 24–32. CEUR-WS.org, 2021.
Figure 5: Visualization of instance Newton.5.1.i.smt2-cvc4 solved by Kissat. The first row shows the instance in its original layout (left) and after 5000 learned or forgotten clauses (right). The second row shows the instance layout after one, two, and three million learned or forgotten clauses (from left to right). The third row shows the evolution of the magnified centers of the updated layouts after one, and three million learned or forgotten clauses and how it looks close to the of search at about 3.5 million learned or forgotten clauses (from left to right).