Distinguished Dissertations

Dominik Schreiber*

On the dissertation “Scalable SAT Solving and its Application”

https://doi.org/10.1515/itit-2024-0096
Received November 4, 2024; accepted February 25, 2025;
published online May 7, 2025

Abstract: The author’s dissertation, entitled “Scalable SAT Solving and its Application”, advances the efficient resolution of instances of the propositional satisfiability (SAT) problem, one of the prototypical “hard problems” of computer science with many scientific and industrial real-world applications. A particular focus is put on exploiting massively parallel computational environments, such as high-performance computing (HPC) systems or cloud computing. The dissertation has resulted in world-leading solutions for scalable automated reasoning and in a number of awards from the SAT community, and has most recently been acknowledged with a GI Dissertation Award. The article at hand summarizes the topic, approaches, and central results of the dissertation, estimates the work’s long-term impact and its role for future research, and closes with some personal notes.

Keywords: automated reasoning; satisfiability; symbolic AI; high-performance computing; distributed algorithms; automated planning

1 Introduction

The dissertation entitled “*Scalable SAT Solving and its Application*” [1] advances crucial generic tools at the core of automated reasoning and Symbolic AI. On the occasion of receiving a GI Dissertation Award, this article provides a summary and reflection of the work, including its background, central results, and impact.

To briefly put the work into context, I conducted my doctoral studies from 2018 to 2023 at the *Algorithm Engineering* group at the Karlsruhe Institute of Technology (KIT), Germany, and was supervised by Prof. Peter Sanders. Several works featured in my dissertation [2], [3] have been joint work with Peter Sanders, whose tremendous support in steering and advising my research I would like to emphasize. Furthermore, one of the works featured in the dissertation [4] has been a cooperation with researchers affiliated with Amazon Web Services, University of Minnesota, and Carnegie Mellon University – namely, Dawn Michaelson, Marijn J. H. Heule, Michael W. Whalen, and Benjamin Kiesl-Reiter.

2 Background

The center of our attention is the problem of *propositional satisfiability* (SAT). An instance of this problem is a logical expression that only consists of *atomic propositions*, i.e., variables that can only be `true` or `false`, and natural logical connectives like “and”, “or”, and “not”. Formally, a canonical instance of the SAT problem is an expression $F = \bigwedge_{c \in C} \bigvee_{\ell \in c} \ell$. Each *literal* ℓ is a propositional variable or its negation; each *clause* c is a *disjunction* (“or”) of literals; and F is a *conjunction* (“and”) of its clauses C. The task is to find a *satisfying assignment*, which assigns each variable in F to `true` or `false` in such a way that F evaluates to `true`. In some cases, no such assignment exists, which renders F *unsatisfiable*. *SAT solving* is the process of analyzing a SAT instance and either returning a satisfying assignment or reporting unsatisfiability.

The SAT problem is one of the most important problems of computer science. On a theoretical level, SAT is the prototypical *NP-complete problem* [5]; informally speaking, any SAT solving algorithm we can realistically devise will require an exponential amount of time for certain inputs. That being said, NP-completeness also means that a great variety of similarly challenging problems can be *reduced to SAT*, i.e., encoded as a SAT instance and then solved with a SAT solving algorithm. Between a seemingly overwhelming worst-case complexity on the one hand and a remarkable degree of expressivity and genericity on the other hand, SAT solving has assumed a fundamental role in practical and applied disciplines of modern computer science [6]. SAT solving algorithms, so-called *SAT solvers*,

***Corresponding author: Dominik Schreiber**, Karlsruhe Institute of Technology, Institute of Information Security and Dependability, Karlsruhe, Germany, E-mail: dominik.schreiber@kit.edu.
https://orcid.org/0000-0002-4185-1851

have become crucial tools for a surprising variety of scientific and industrial applications, such as formal hardware and software verification [7], security and cryptography [8], operations research and combinatorial optimization [9], knowledge representation and explainable AI [10], electronic design automation [11], bioinformatics [12], and automated theorem proving [13]. As such, advancing SAT solving bears immense potential for simultaneously advancing a plethora of important disciplines from computer science and real-world applications.

Our main focus is to exploit *parallel and distributed computing* for SAT solving. Most of today's computers feature parallel computing, from mobile phones (4–8 parallel cores) to high-performance servers (up to a few hundred cores). In distributed computing, we consider several such machines that perform a computation together and (in general) do not share any kind of memory. Communication between the machines commonly takes place by passing messages over some kind of network. This can enable computations involving thousands of cores (and more) at once to tackle especially large or challenging problems.

3 Thesis summary

Our research has advanced the scalability and dependability of SAT solving in distributed systems. We have achieved this both by devising novel flexible scheduling algorithms for processing many solving tasks at once and also by advancing the algorithmic approach *within* each parallel solving task. Furthermore, we have devised an approach to let distributed solvers produce *proofs of unsatisfiability*, which are crucial for full confidence in a claimed result. Last but not least, in a major application case study, we have advanced the state of the art in *hierarchical automated planning*, a sub-discipline of symbolic AI, through novel propositional encoding techniques and the utilization of massively parallel hardware.

The following summary is accordingly divided into four sections: *Scheduling*, *Solving*, *Proving*, and *Planning*. Note that these sections feature some translated content from the dissertation's German summary featured in the GI Dissertation Award proceedings [14].

3.1 Scheduling

Our first contributions are motivated by a simple observation: Processing many independent tasks simultaneously can be significantly more efficient than spending the same computational resources on processing a single task. This is the case especially if the used parallelization within a task *does not scale linearly*, i.e., if increasing the computational resources by a factor of n does not accelerate the computation by a factor of $\approx n$. SAT solving and other combinatorial search tasks clearly exhibit such *sub-linear scaling*. Therefore, we investigate the efficient scheduling of SAT instances and similarly irregular tasks in large distributed environments.

Rigid scheduling strategies, which assign a fixed set of computational resources to each parallel task upon its initiation, can only react to a very limited extent to unpredictable events such as tasks finishing early or many tasks arriving at the same time. For SAT instances, where the prediction of running times is a challenging machine learning problem even in sequential settings [15], rigid scheduling can lead to highly suboptimal system utilization and/or high waiting times. In contrast, we call a parallel task *malleable* if it supports a fluctuating number of computational resources *during its execution* [16]. Malleability has long been recognized as a powerful paradigm for fair and flexible job scheduling [17], but has so far been studied mainly for iterative computations with uniform parallelization and (nearly) linear scaling. SAT and other NP-hard computations, on the other hand, feature unpredictable running time behavior and limited scalability, which is why we are interested in a careful management of available resources. Our intuition was that a flexible on-demand processing system with malleable scheduling may provide initial computational resources to incoming tasks virtually instantly, complete trivial tasks in fractions of a second, and offer a fair share of resources to more difficult tasks similarly quickly.

To turn this vision into reality, we present an approach that ensures fast, fair, and bottleneck-free scheduling without any assumptions on the execution time of incoming tasks. We achieve this through a two-part decentralized method. First, the m processes in the system cooperatively compute a fair number $v_j \in \mathbb{N}^+$ of processes for each active task j so that the system utilization is maximized. This assignment takes into account two measures of each task: its *maximum resource demand* $d_j \in \mathbb{N}^+$, which the task can freely choose and adjust based on its application logic; and its *priority* $p_j \in \mathbb{R}^+$, which is provided by the submitting user (and possibly adjusted by the system). The v_j are calculated to be proportional to the tasks' priorities where feasible, subject to the side constraints $1 \leq v_j \leq d_j$. On an algorithmic note, we were able to show that this can be achieved in $\mathcal{O}(m \log m)$ total work and $\mathcal{O}(\log m)$ *span* (which is the maximum depth of the parallel computation's data dependencies).

Secondly, each task j is assigned v_j specific processes. This process assignment is kept as stable as possible over

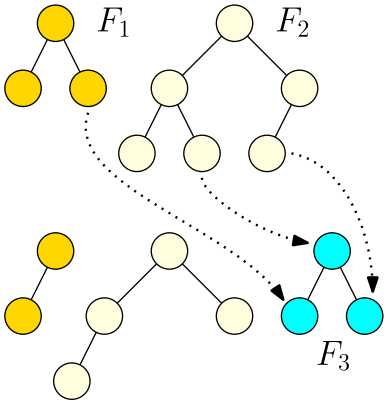

Figure 1: Top: Two tasks, for formulas F_1 and F_2, are using three and six processes respectively. Bottom: A third task for formula F_3 enters, which prompts the two former tasks to release some of their processes to join the solving of F_3.

time: Each task in the system is represented by a complete binary tree of v_j processes, which expands or shrinks at the leaf level as needed (see Figure 1). New processes for a growing tree are found by the current leaf nodes sending special request messages. These requests are routed through the system following a scalable distributed protocol until they are accepted by an idle process.

In massively parallel experiments with up to 6,144 cores of a high-performance computer (≤128 nodes of SuperMUC-NG, Leibniz Supercomputing Center) and hundreds of concurrent tasks, we observed very short waiting and response times – on average, 10 ms for providing an initial process for an incoming task, 1 ms for calculating a fair volume assignment for all active tasks, and 6 ms for providing the desired additional resources for a task with increased resource demands. Thanks to these fast responses, we were able to achieve near-optimal system utilization (>99 %) virtually always. We have conducted several case studies for possible applications of our system, including *k*-means clustering (bachelor thesis of Michael Dörr [18]) and hierarchical planning (master thesis of Niko Wilhelm [19]). The main application considered in our work is of course SAT solving, as we will discuss in the following sections.

3.2 Solving

In the following, we address the efficient parallelization of SAT solving itself. The currently best parallelizations for general-purpose SAT solving can be explained with a simple analogy: A number of puzzle experts are gathering to cooperatively find a solution to a difficult puzzle. Since our experts are not particularly social beings, the predominant strategy is to let each expert attempt to solve the puzzle on their own, which means that the group solves the puzzle as quickly as the *single* best expert for the present problem. To achieve scalability beyond this pure *portfolio* [20], [21] of experts, we conduct brief standup meetings from time to time where each expert can share the most useful insights that they gained since the last meeting. As such, a crucial insight gained by a single expert can now benefit *all* experts.

In parallel SAT solving, each expert corresponds to a highly engineered sequential algorithm that efficiently searches the space of variable assignments for a solution and produces a so-called *conflict clause* whenever it encounters a logical conflict under its current assignment [22]. These conflict clauses represent a pruned sub-space of the search space at hand and correspond to the insights exchanged among our experts. The notion of individual mindsets and approaches across the experts is translated to parallel SAT solving by *diversifying* the sequential solvers, which can range from deviating random decisions over different configuration options up to completely different solver implementations [21], [23].

Some earlier parallel SAT solving systems using the described solving paradigm, i.e., a solver portfolio with clause sharing, show decent performance for low to moderate levels of parallelization, e.g., a few dozen cores [24], [25]. In massively parallel computing environments, however, previous approaches fail to scale beyond few hundred cores for average inputs. Specifically, the previous state of the art in distributed SAT solving, HordeSat [23], achieved a median speedup of 13 on 2048 cores on industrial benchmarks, which corresponds to a (resource) efficiency of only $13/2048 = 0.6$ %.

We use HordeSat as a point of departure and rethink its algorithmic components. In particular, we design a compact approach for periodic *all-to-all* clause exchange that is suitable for thousands of solvers. In HordeSat, each process contributes a fixed-size buffer of clauses, and the concatenation of all buffers is then broadcast to all m processes. This can result in gaps featuring no information (if a process does not export sufficiently many clauses) or in sharing m almost identical sets of clauses. By contrast, the motivation of our approach is to identify a fixed amount of the *globally best distinct* learned clauses using a scalable protocol (Figure 2). Since the clause buffer's output size at each layer of the aggregation tree is strictly limited, only the most promising clauses are passed on to the tree's higher layers, hence successively transforming the processes' local notions of quality into a *global* quality criterion. The resulting buffer of shared clauses is then passed on to all solvers. Our approach reliably detects duplicates within an exchange operation and can additionally eliminate temporally shifted duplicates using a distributed filter operation. Furthermore, our

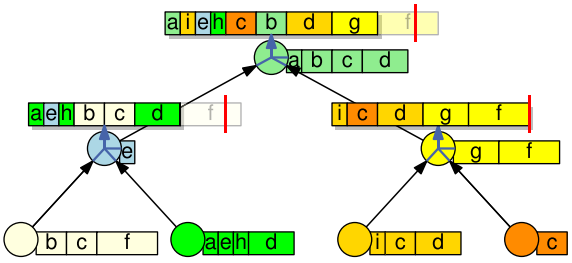

Figure 2: Illustration of our clause exchange operation. Each process (circle) contributes a fixed-size buffer of the best, i.e., shortest clauses it produced. Different colors/shades denote the clauses' respective processes of origin. At each inner node of the process tree, a space-limited merge operation is performed that also detects duplicates. The resulting clause buffer at the tree's root is then broadcast to all processes, followed by a post-hoc filtering operation (not depicted).

distributed solver is *malleable* and can therefore be used within our scheduling platform to simultaneously solve many tasks with flexible scheduling. On a pragmatic level, we have taken some measures to reduce the required main memory and integrated some of the currently best available sequential SAT solvers [26]–[28] into our system.

In experiments with up to 3,072 cores, our approach was able to more than double the best previous speedups of distributed SAT solving. Among the benchmark instances that took the best sequential approach at least an hour each to solve, our system at 3,072 cores resulted in a (geometric) mean speedup of 43.7, corresponding to a resource efficiency of 14.2 %. We were able to demonstrate that our clause exchange between solvers is the main reason for this scalability. In the International SAT Competition [29], our system has dominated the cloud category (1,600 threads on 100 machines) for four years in a row and has also proven to be highly competitive on moderately parallel hardware (64 threads). Moreover, we highlight the advantages of a combination of parallel task processing and malleable SAT solving: In an experiment with 6,400 cores, when solving 400 tasks simultaneously, the difficult tasks gradually receive more and more computing resources that become available through the solving of easier tasks. This leads to performance improvements and, consequently, much higher efficiency than with rigid scheduling strategies.

3.3 Proving

Today's sequential SAT solvers are reliable in the sense that, if required, they can output a *proof of unsatisfiability* [30] for a formula. Such a proof grants full confidence in the claimed result that the formula is unsatisfiable: There are several proof checking tools that are formally verified down to the machine code level and that can be used to independently validate a proof [31]–[33]. Efficient parallel SAT solvers have, so far, lacked the option to produce a proof (cf. [34]), which severely limited their use for critical applications. An important example is the formal verification of software or hardware components, where unsatisfiability of a formula typically implies that the encoded correctness property is *always preserved* within the chosen bounds [35]. Therefore, the unsatisfiability of a formula is often a critical property, and an explicit proof of it can be indispensable.

We present the first practical and scalable approach for generating such proofs in parallel SAT solving. A key technology for our approach is the clause-based proof format LRAT [31], where each proof step corresponds to a learned conflict clause and the dependencies between the proof steps are explicitly expressed using globally unique IDs for learned clauses. Our solution orchestrates many sequential solvers as described above, which now individually output their learned clauses into partial proof files. Once one of the solvers finds unsatisfiability, our distributed algorithm essentially *rewinds* the solving process by having each solver traverse its own proof in reverse order and trace back all transitive dependencies of the final conclusion step (see Figure 3). In particular, each clause exchange operation is reversed: the IDs of all remote clauses recognized as necessary are redistributed to their respective origin processes, which then trace back their dependencies further. A distributed protocol aggregates all required proof steps into a central proof file, which can then be independently validated.

In experiments in a cloud environment, our approach generated proofs up to 80 GB in size that were still feasible to validate under reasonable overhead. Even in a calculation with 1,600 solver threads, proof construction is roughly as fast as distributed solving itself, while proof verification (a strictly sequential operation) takes on average four times the running time of solving. Overall, the observed performance represents a substantial improvement over any previous attempts to generate proofs with clause-sharing solvers [25], [34]. The presented approach therefore marks an important first step towards rendering scalable SAT solving accessible for critical applications.

3.4 Planning

In the final sections of the dissertation, we conduct an extensive case study on applied SAT solving. Specifically, we present a SAT-based approach for *Totally Ordered Hierarchical Task Network* (TOHTN) planning – a popular branch of symbolic AI that provides rich opportunities for modeling complex hierarchical planning tasks [36]. A problem instance includes an initial *world state* as well as an initial sequence of *tasks*. Each task is either *primitive* or

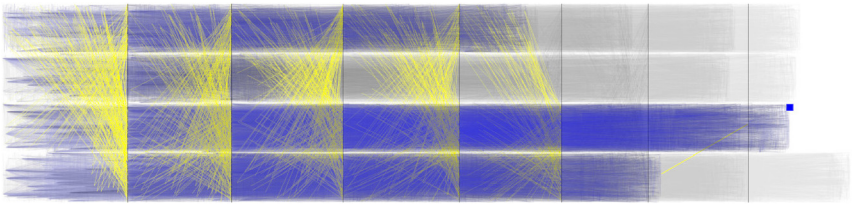

Figure 3: Visualization of a small (four-core, two-second) proof of unsatisfiability. Each row corresponds to a solver; each horizontal/diagonal line represents a dependency from one clause to another; vertical separators indicate a clause sharing operation; the square represents the conclusion indicating unsatisfiability; and highlighted (yellow, blue) lines indicate the parts of the proof identified as required to reach the conclusion.

hierarchical. A primitive task corresponds to an atomically executable operation with preconditions and postconditions relative to the world state. In contrast, a hierarchical task is successively decomposed into a sequence of subtasks by applying conditional reduction rules. All these objects surrounding the planning model are typically parameterized: If a planning instance, for example, includes sets of trucks T and locations L, a parameterized task *drive* (t, x, y) can be instantiated to move a truck $t \in T$ from location $x \in L$ to location $y \in L$. Initially, only the initial world state and initial tasks are *ground*, i.e., without free parameters. Many approaches to TOHTN planning, including all SAT-based methods [37]–[40], rely on a preprocessing step that fully instantiates all relevant objects to then work with a simplified representation. In many cases, this so-called *grounding* leads to a combinatorial explosion of input data even before the actual planning begins [41].

We present the first SAT-based TOHTN planning approach that retains the parameterized problem description and, as such, avoids the described preprocessing. In the above example, we do not instantiate a task for all $|T| \times |L|^2$ possible parameter combinations (as prior approaches do) but encode the parameterized task along with its parameters t, x, y and their domains directly in propositional logic. Our integrated planning approach alternates between a cautious instantiation of the planning hierarchy, its encoding in propositional logic, and the actual SAT solving. The generated formulas are often one to two orders of magnitude smaller compared to previous SAT-based approaches. Accordingly, our approach scored the second place in the International Planning Competition 2020, behind an incomplete algorithm [42] whose greedy search was the more lightweight for most problems. Our approach was able to solve the most problems and find the shortest plans in the competition [43].

Finally, we present a method for processing hierarchical planning problems on massively parallel hardware by using our scheduling framework with malleable SAT solving. In an experiment with 2,348 cores, our system simultaneously handled 587 planning problems. Depending on the planning domain considered and specifically the difficulty of the constructed SAT instances, the results range from minimal overhead (approx. 5 %) to significant acceleration ($\leq$24.8). Thus, our application study also serves as a successful stress test for our contributions to scalable SAT solving in the context of a challenging application. Another important practical contribution of this study is that we extended our distributed SAT solver to support *incremental SAT solving* [44], i.e., interactive solving procedures over modifiable formulas, which was a key technology to achieve low response times for individual SAT solving calls.

4 Outlook

SAT solving has long been an important cornerstone of reliable and dependable computing – efficient and provably correct verification tools, for instance, can present a crucial foundation for pressing societal challenges such as ensuring the soundness of critical software, models, and infrastructures. This can include questions of cybersecurity, safe robotic behavior, or fair AI-based decision-making. In particular, I believe that complementing today's thriving sub-symbolic AI methods (machine learning, large language models) with formal, precise, and provably correct reasoning techniques is a major challenge of modern computer science [45]. While recent advances in sub-symbolic AI have been driven by high-performance computing, we have made some successful contributions to allowing formal reasoning techniques to also effectively exploit large-scale computational resources. My hope is that this continues to empower the many applications of automated reasoning and thus leads to a more widespread adoption of formal methods in modern computer science disciplines, which may, in turn, boost subsequent research on those methods.

A crucial aspect of scaling up automated reasoning to distributed systems is to preserve the high level of trust and dependability that the best available sequential solutions offer. Parallel and distributed SAT solvers in particular have long been considered much less trustworthy than their

sequential counterparts, and rightfully so (due to more complex technology stacks, less comprehensive testing, and the lack of proofs). Our research on distributed proof production has made an important first step to render large-scale SAT solving suitable even for the most safety – or security-critical tasks that arise from applications.

At a very practical level, the open-source software system MALLOB resulting from our research has been consistently scoring top ranks at the International SAT Competition 2020–2024 and represents the state of the art in large-scale SAT solving since 2020. As a result, our system was called "*by a wide margin, the most powerful SAT solver on the planet*" by Amazon VP and distinguished scientist B. Cook in 2022 [46]. Our system has thus established itself as a world-leading automated reasoning system and gained a significant amount of attention, which paved the road to fruitful international cooperations. Together with its powerful scheduling capabilities, its support for incremental SAT solving, and its pioneering role in terms of proof production, the MALLOB framework is likely to remain relevant for scientific as well as industrial actors.

On a personal note, I am currently continuing my research on scalable automated reasoning as a Young Investigator Group[1] leader within a pilot programme of the Helmholtz Association (*Core Informatics at Karlsruhe Institute of Technology*, KiKIT). Building upon the insights, results, and software arising from my doctoral studies, our current research aims at transferring scalable SAT technology to related essential automated reasoning tools, such as SAT solving with objective optimization (MaxSAT) or first-order logic solving for verification purposes (SMT). We are also actively continuing to research scalable methods for producing and processing proofs of unsatisfiability [47], with the ultimate goal of obtaining a formally verified distributed SAT solver while still retaining state-of-the-art solving performance.

Our successes surrounding scalable SAT solving have been recognized by the SAT Association, awarding my dissertation with the inaugural *Fahiem Bacchus Award* (for an outstanding satisfiability-related 2022–2023 thesis). Most recently, the Dissertation Award jointly conferred by the German Informatics Society, Austrian Computer Society, and Swiss Informatics Society not only represents a significant acknowledgment of our work so far but also provides immense support and motivation for our subsequent research going forward. I am deeply grateful for these recognitions and would like to express my thanks to everyone who invested work in the process, including in particular the GI selection jury chaired by Rüdiger Reischuk. I am looking forward to pursuing further advancements in the field of scalable automated reasoning.

Research ethics: Not applicable.
Informed consent: Not applicable.
Author contributions: The author has accepted responsibility for the entire content of this manuscript and approved its submission.
Use of Large Language Models, AI and Machine Learning Tools: LLMs and AI-driven translation were used to translate text across languages and to slightly polish some language.
Conflict of interest: The author states no conflict of interest.
Research funding: Some of this work was performed on the supercomputers ForHLR and HoreKa funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research. The author gratefully acknowledges the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de). This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (http://dx.doi.org/10.13039/501100007601, grant agreement No. 882500).
Data availability: The datasets generated and/or analyzed during the presented work are available at Zenodo: https://doi.org/10.5281/zenodo.10184679.

References

[1] D. Schreiber, "Scalable SAT solving and its application," Ph.D. thesis, Karlsruhe Institute of Technology (KIT), 2023.
[2] P. Sanders and D. Schreiber, "Decentralized online scheduling of malleable NP-hard jobs," in *Proc. International European Conference on Parallel Processing (Euro-Par)*, 2022, pp. 119—135.
[3] D. Schreiber and P. Sanders, "Scalable SAT solving in the cloud," in *Theory and Applications of Satisfiability Testing (SAT)*, 2021, pp. 518—534.
[4] M. Dawn, D. Schreiber, M. J. H. Heule, B. Kiesl-Reiter, and M. W. Whalen, "Unsatisfiability proofs for distributed clause-sharing SAT solvers," in *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2023, pp. 348—366.
[5] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. ACM Symposium on Theory of Computing*, 1971, pp. 151—158.
[6] J. K. Fichte, D. Le Berre, M. Hecher, and S. Szeider, "The silent (R)evolution of SAT," *Commun. ACM*, vol. 66, no. 6, pp. 64—72, 2023.
[7] Y. Vizel, G. Weissenbacher, and S. Malik, "Boolean satisfiability solvers and their applications in model checking," *Proc. IEEE*, vol. 103, no. 11, pp. 2021—2035, 2015.

1 https://satres.kikit.kit.edu.

[8] J. Erlacher, F. Mendel, and M. Eichlseder, "Bounds for the security of ASCON against differential and linear cryptanalysis," in *IACR Transactions on Symmetric Cryptology*, 2022, pp. 64—87.
[9] G. H. I. de Azevedo, A. A. Pessoa, and A. Subramanian, "A satisfiability and workload-based exact method for the resource constrained project scheduling problem with generalized precedence constraints," *Eur. J. Oper. Res.*, vol. 289, no. 3, pp. 809—824, 2021.
[10] A. Darwiche, "Three modern roles for logic in AI," in *Proc. ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2020, pp. 229—243.
[11] D. Liu, C. Yu, X. Zhang, and D. Holcomb, "Oracle-guided incremental SAT solving to reverse engineer camouflaged logic circuits," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 433—438.
[12] H. Brown, L. Zuo, and D. Gusfield, "Comparing integer linear programming to SAT-solving for hard problems in computational and systems biology," in *International Conference on Algorithms for Computational Biology*, 2020, pp. 63—76.
[13] M. J. H. Heule, O. Kullmann, and V. Marek, "Solving and verifying the boolean pythagorean triples problem via cube-and-conquer," in *Theory and Applications of Satisfiability Testing (SAT)*, Springer, 2016, pp. 228—245.
[14] D. Schreiber, "Skalierbares SAT Solving und dessen Anwendung," in *Ausgezeichnete Informatikdissertationen 2023 (Band 24)*, Bonn, Gesellschaft für Informatik e.V., 2024, pp. 281—290.
[15] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: methods & evaluation," *Artif. Intell.*, vol. 206, pp. 79—111, 2014.
[16] D. G. Feitelson, "Job scheduling in multiprogrammed parallel systems," IBM Research Report, 1997.
[17] A. Gupta, B. Acun, O. Sarood, and L. V. Kalé, "Towards realizing the potential of malleable jobs," in *Proc. International Conference on High Performance Computing (HiPC)*, IEEE, 2014, pp. 1—10.
[18] M. Dörr, "K-means in a malleable distributed environment," Bachelor's thesis, Karlsruhe Institute of Technology (KIT), 2022.
[19] N. Wilhelm, "Malleable distributed hierarchical planning," Master's thesis, Karlsruhe Institute of Technology (KIT), 2022.
[20] C. P. Gomes and B. Selman, "Algorithm portfolios," *Artif. Intell.*, vol. 126, nos. 1—2, pp. 43—62, 2001.
[21] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: a parallel SAT solver," *J. Satisf. Boolean Model. Comput.*, vol. 6, no. 4, pp. 245—262, 2010.
[22] J. Marques-Silva, I. Lynce, and S. Malik, "CDCL SAT solving," in *Handbook of Satisfiability*, IOS Press, 2021, pp. 131—153.
[23] T. Balyo, P. Sanders, and C. S. HordeSat, "A massively parallel portfolio SAT solver," in *Theory and Applications of Satisfiability Testing (SAT)*, 2015, pp. 156—172.
[24] T. Balyo and C. Sinz, "Parallel satisfiability," in *Handbook of Parallel Constraint Reasoning*, Springer, 2018.
[25] M. Fleury and A. Biere, "Scalable proof producing multi-threaded SAT solving with Gimsatul through sharing instead of copying clauses," in *Proc. Pragmatics of SAT*, 2022.
[26] A. Biere, "Yet another local search solver and Lingeling and friends entering the SAT competition 2014," in *Proc. SAT Competition*, 2014, p. 65.
[27] A. Biere, "CaDiCaL, lingeling, plingeling, treengeling and YalSAT entering the SAT competition 2018," in *Proc. SAT Competition*, 2018, pp. 14—15.
[28] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, kissat, paracooba, plingeling and treengeling entering the SAT competition 2020," in *Proc. SAT Competition*, 2020, p. 50.
[29] N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, and M. Suda, "SAT competition 2020," *Artif. Intell.*, vol. 301, 2021, Art. no. 103572. https://doi.org/10.1016/j.artint.2021.103572.
[30] M. J. H. Heule, "Proofs of unsatisfiability," in *Handbook of Satisfiability*, IOS Press, 2021, pp. 635—668.
[31] L. Cruz-Filipe, M. J. H. Heule, W. A. Hunt, Jr., M. Kaufmann, and P. Schneider-Kamp, "Efficient certified RAT verification," in *Proc. International Conference on Automated Deduction (CADE)*, 2017, pp. 220—236.
[32] S. Gocht, C. McCreesh, and J. Nordström, "VeriPB: the easy way to make your combinatorial search algorithm trustworthy," in *Workshop From Constraint Programming to Trustworthy AI at the 26th International Conference on Principles and Practice of Constraint Programming (CP'20)*, 2020.
[33] P. Lammich, "Efficient verified (UN)SAT certificate checking," *J. Autom. Reason.*, vol. 64, no. 3, pp. 513—532, 2020.
[34] M. J. H. Heule, N. Manthey, and T. Philipp, "Validating unsatisfiability results of clause sharing parallel SAT solvers," in *Proc. Pragmatics of SAT*, 2014, pp. 12—25.
[35] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods Syst. Des.*, vol. 19, no. 1, pp. 7—34, 2001.
[36] P. Bercher, R. Alford, and D. Höller, "A survey on hierarchical planning — one abstract idea, many concrete realizations," in *Proc. International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019, pp. 6267—6275.
[37] G. Behnke, D. Höller, and S. Biundo, "totSAT — totally-ordered hierarchical planning through SAT," *Proc. AAAI Conf. Artif. Intell.*, vol. 32, pp. 6110—6118, 2018.
[38] G. Behnke, D. Höller, and S. Biundo, "Finding optimal solutions in HTN planning — a SAT-based approach," in *Proc. International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019, pp. 5500—5508.
[39] D. Schreiber, D. Pellier, H. Fiorino, and T. Balyo, "Efficient SAT encodings for hierarchical planning," in *Proc. International Conference on Agents and Artificial Intelligence (ICAART)*, 2019, pp. 531—538.
[40] D. Schreiber, D. Pellier, H. Fiorino, and T. Balyo, "Tree-REX: SAT-based tree exploration for efficient and high-quality HTN planning," in *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 2019, pp. 382—390.
[41] J. Wichlacz, A. Torralba, and J. Hoffmann, "Construction-planning models in minecraft," in *Proc. ICAPS Workshop on Hierarchical Planning*, 2019.
[42] M. C. Magnaguagno, F. Meneguzzi, and L. de Silva, "HyperTensioN — a three-stage compiler for planning," in *Proc. International Planning Competition*, 2021, pp. 5—8.
[43] D. Schreiber, "Lifted logic for task networks: TOHTN planner Lilotane in the IPC 2020," in *Proc. International Planning Competition*, 2021, pp. 9—12.
[44] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543—560, 2003.
[45] A. Platzer, "Intersymbolic AI: interlinking symbolic AI and subsymbolic AI," in *International Symposium on Leveraging Applications of Formal Methods*, 2024, pp. 162—180.

[46] B. Cook, “Automated reasoning’s scientific frontiers,” Amazon Science, 2021. Available at: https://www.amazon.science/blog/automated-reasonings-scientific-frontiers.
[47] D. Schreiber, “Trusted scalable SAT solving with on-the-fly LRAT checking,” in *Theory and Applications of Satisfiability Testing (SAT)*, 2024, pp. 25:1—25:19.

Bionote

Dominik Schreiber
Karlsruhe Institute of Technology, Institute of Information Security and Dependability, Karlsruhe, Germany
dominik.schreiber@kit.edu
https://orcid.org/0000-0002-4185-1851

Dominik Schreiber is a Young Investigator Group leader at the Karlsruhe Institute of Technology (KIT) in Germany. His research focuses on automated reasoning, particularly propositional satisfiability (SAT) solving, with an emphasis on parallel and distributed systems. He completed his doctoral studies on Scalable SAT Solving and its Application at Peter Sanders’ Algorithm Engineering group at KIT (2018—2023), following double-degree studies at INP Grenoble and KIT (2016—2018). He has served as a Program Committee member of ICAPS, ECAI and SAT, organized competitions (IPC) and workshops (ICAPS HPlan), and offered courses on SAT Solving and Planning & Scheduling. He is an alumnus of the German Academic Scholarship Foundation (Studienstiftung des dt. Volkes).