Extracting Problem Structure with LLMs for Optimized SAT Local Search

André Schidler[1, 2], **Stefan Szeider**[1]

[1]Algorithms and Complexity Group, TU Wien
[2]Computer Architecture, University of Freiburg
schidler@cs.uni-freiburg.de, sz@ac.tuwien.ac.at

Abstract

Encoding combinatorial problems in terms of propositional satisfiability (SAT) enables utilization of highly efficient SAT solvers for combinatorial search. Local search preprocessing accelerates the SAT solver's search by providing high-quality starting points, a technique implemented in several modern SAT solvers. However, existing preprocessing methods employ generic strategies that fail to exploit the structural patterns inherent in problem encodings. This position paper proposes a novel paradigm wherein Large Language Models (LLMs) analyze problem encoding implementations to synthesize specialized preprocessing algorithms. The LLMs examine Python-based code to identify structural patterns, enabling the automatic generation of encoding-specific local search procedures. These procedures operate across all instances sharing the same encoding scheme rather than requiring instance-specific customization. Our preliminary empirical evaluation demonstrates effective automated algorithm synthesis for structure-aware SAT preprocessing, serving as a foundation for similar approaches across multiple domains of combinatorial optimization.

Introduction

The development of effective combinatorial search algorithms historically requires substantial domain expertise and manual design effort. This process creates an inherent barrier to advancing solver technology across various problem domains where structural patterns in problem encodings remain unexploited. We propose that automated algorithm synthesis through Large Language Models (LLMs) represents a transformative approach to this challenge, fundamentally altering how specialized search algorithms are developed.

The current paradigm relies on human experts to analyze problem structures and design algorithms that exploit these patterns but operates under significant constraints: it requires rare domain expertise spanning both theoretical computer science and implementation techniques, scales poorly to diverse problem structures, and filters algorithm design through human conceptual frameworks that may miss non-intuitive optimizations.

SAT preprocessing exemplifies these limitations. Current SAT solvers such as CaDiCaL (Biere 2019) and CryptoMiniSat (Soos 2020) implement preprocessing techniques that improve solver performance (Balint and Manthey 2013) but employ general-purpose strategies that fail to exploit the structural patterns inherent in SAT encodings of real-world problems. These encodings contain latent patterns derived from graph topologies, connectivity constraints, and domain-specific relationships that become obscured during transformation to Conjunctive Normal Form (CNF).

Our proposed paradigm differs fundamentally from existing code generation approaches by targeting algorithm synthesis rather than implementing predefined approaches, focusing on extracting problem structure from encoding implementations and generating algorithms with theoretical guarantees.

Recent advances support the technical feasibility of this paradigm. LLMs have demonstrated capabilities in algorithm implementation through systems like AlphaCode (Li et al. 2022) and CodeGen (Nijkamp et al. 2023). Machine learning techniques have shown efficacy in learning heuristic policies (Selsam and Bjørner 2019) and guiding search procedures (Yolcu and Póczos 2019). However, these approaches primarily focus on implementing predefined algorithms or solving instance-specific problems rather than the algorithmic design process itself.

Our proposed paradigm differs from existing code generation approaches in three fundamental ways: (1) it targets algorithm synthesis rather than implementation of predefined approaches, (2) it focuses on extracting and exploiting the problem structure from encoding implementations, and (3) it aims to generate algorithms with theoretical guarantees rather than heuristic solutions.

To demonstrate the viability of this paradigm, we implemented a framework that employs LLMs to analyze PySAT encoding implementations and generate specialized local search strategies for SAT preprocessing. The model identifies structural elements in the encoding and incorporates them into strategies that target the encoding scheme rather than specific problem instances. Our tests show these generated strategies solve problems that standard methods cannot handle, while still working well on easier cases.

This shift in algorithm development has broad consequences beyond just better SAT solving. It lowers the ex-

pertise barrier for creating specialized algorithms. It allows for quick adaptation to new problem structures without manual coding. It also provides a consistent framework for algorithm development that works across different domains.

The field of combinatorial search should invest research effort in automated algorithm synthesis as it represents a fundamental change in how we approach hard computational problems. This research direction works alongside human expertise rather than replacing it, potentially expanding the range of problems we can solve efficiently and changing the fundamental way specialized search algorithms are created and improved.

Problem Formulation

We address the automatic generation of problem-specific local search procedures for integration with CDCL SAT solvers. Current local search algorithms exhibit two primary limitations: they function as general-purpose mechanisms without knowledge of underlying encoding schemes, and empirical evidence indicates they often struggle to find satisfying assignments independently (Li and Li 2012; Cai and Zhang 2021; Cai et al. 2022). Hybrid approaches that combine local search with CDCL solvers demonstrate superior performance on complex combinatorial instances (Cai et al. 2022), which suggests that encoding-specific local search methods designed for hybrid integration would yield enhanced performance on difficult instances.

The manual creation of specialized algorithms requires substantial effort and typically concentrates on established approaches. This focus is necessary because manually developing specialized algorithms with diverse approaches is generally infeasible. We investigate whether LLMs can automatically generate diverse local search prototypes that exploit encoding structures, thereby expanding the range of available search strategies.

We evaluate local search performance by setting the CDCL solver's default phase values according to the assignment produced by the local search procedure. The solver's default phase determines the initial truth value assigned to variables during the search process. If the local search successfully finds a satisfying assignment, the CDCL solver terminates immediately. Generally, the closer a local search assignment approximates a satisfying assignment, the faster the CDCL solver resolves the instance. Performance measurement involves comparing solver runtime across different local search initializations.

Our evaluation includes three combinatorial problems with increasing encoding complexity: *Graph Coloring* with its straightforward CNF representation (Gelder 2008), *Directed Feedback Vertex Set (DFVS)* with its reachability-based encoding requiring $O(|V|^3)$ clauses (Janota, Grigore, and Manquinho 2017), and *Bounded Depth Decision Trees (BDDT)* with a complex encoding representing decision paths (Shati, Cohen, and McIlraith 2021). This problem set allows systematic exploration of our approach's capabilities across varying levels of encoding complexity.

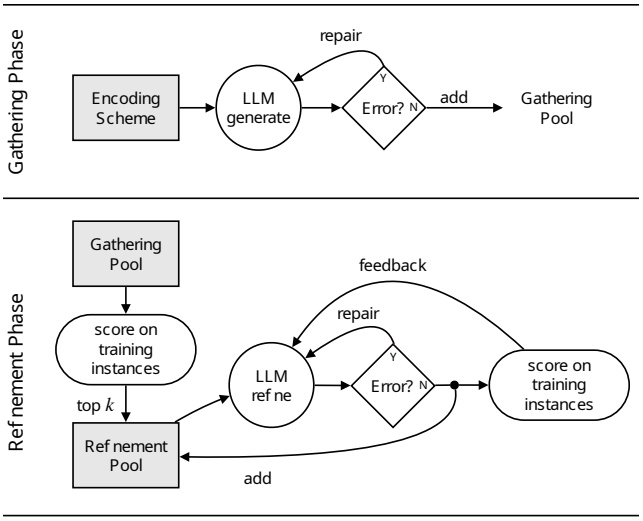


Figure 1: Schematic of the whole approach from the encoding scheme to the pool of refined local search functions. From this pool we take the top-performing functions for the evaluation on test instances.

Methodology

Our framework comprises two distinct phases: a Gathering Phase and a Refinement Phase (see Figure 1 for a schematic overview).[1]

We standardize encoding schemes using PySAT (Ignatiev, Morgado, and Marques-Silva 2018), which expresses encodings through understandable programming constructs and abstracts cardinality constraints behind a single function.

In the **Gathering Phase**, we provide the LLM with the PySAT encoding scheme without problem identification. The LLM generates local search functions that (1) perform local search specifically for the provided encoding, (2) return a partial variable assignment accessible via variable identifiers, (3) complete within a specified timeout, and (4) exploit instance structure and encoding characteristics. We verify each function by executing it on a simple instance with a 30-second timeout, with repair attempts for errors until validation or a predefined limit.

The **Function Evaluation** compares performance by executing each function on training instances. Rankings are determined by three criteria in descending priority: timeout frequency, SAT solver timeout frequency using the function's assignments, and average runtime for successful executions.

The **Refinement Phase** focuses on improving high-performing functions from the Gathering Phase. For each selected function, the LLM implements modifications while preserving the core approach. Each variant undergoes evaluation with comparative performance feedback provided to the LLM. Based on performance changes, the LLM either continues similar refinements or explores alternative

[1]More details can be found in the long version (https://arxiv.org/abs/2501.14630) and the supplementary material (https://doi.org/10.5281/zenodo.14732109).

approaches, with substantial modifications requested when performance plateaus.

Experimental Evaluation

In this section, we evaluate the quality of the generated local search functions. Further, we also explore how well the different parts of our approach contribute to this quality. [2]

Setup We use the OpenAI models o1-mini-2024-09-12 and gpt-4o-2024-11-20, as well as Anthropic claude-3-5-sonnet-20241022. Further, we use Cadical 1.9.5 (Biere et al. 2020, 2024) as the SAT solver and PySAT 1.8.dev13 (Ignatiev, Morgado, and Marques-Silva 2018).

We use one benchmark set per encoding scheme, where the respective training sets consist of those instances that the SAT solver solves within 10 to 60 seconds and test sets contain all instances requiring longer solving times. This results in 10 training and 38 testing instances for graph coloring[3] (Sun et al. 2021), 18 training instances and 124 test instances for DFVS (Zhou 2016; Kiesel and Schidler 2022, 2023), and 9 training instances and 32 test instances for BDDT (Bessiere, Hebrard, and O'Sullivan 2009; Olson et al. 2017; Narodytska et al. 2018; Verwer and Zhang 2019; Avellaneda 2020; Schidler and Szeider 2024). Function Evaluation runs local search for 60 seconds, initializes the default phases, and runs the SAT solver for another 120 seconds.

Gathering In the Gathering Phase, we generate 50 local search function per LLM model and encoding scheme. All three LLM models correctly identified the purpose of the graph coloring and BDDT encoding schemes, while misunderstanding the DFVS encoding scheme. The three models differ in their ability to repair errors. Hence, GPT o1-mini, which requires the fewest iterations to repair errors, generated the 50 functions faster and required fewer tokens, than the slower Claude Sonnet and even slower GPT 4o.

Refinements We pick for each problem and LLM model the five best local search functions. Each local search function is then refined 19 times, resulting in 20 versions of each function, 100 local search functions per model and problem and 300 local search functions per problem. Figure 2 shows that the refinements achieve significant change over time.

Test Instance Performance We evaluate the functions in the Refinement Pool using the test instances. We first compute a baseline by running the SAT solver without local search for one hour per instance. From the 20 versions per function, we pick the base version (no refinements) and two of its best refinements. Each function is then given 15 minutes to find an assignment. The assignment is used to initialize the default phases before running the SAT solver for up to one hour. We give the results for the best function per LLM model in Figure 3. The 15 minutes are expected to become negligible in a non-prototypical and native implementation, e.g., using C++.

[2]Code and Results are in the Supplementary Materials, more details on the experiments in the appendix.

[3]TreewidthLib instances were provided by Fichte, Lodha, and Szeider (2017)

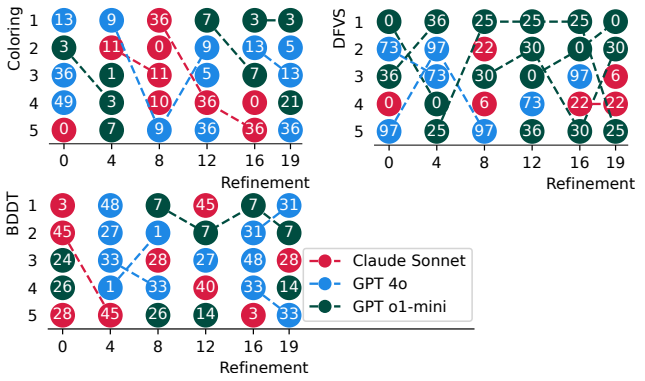


Figure 2: The top five local search functions for different problems over different refinement iterations.

The results show that the hybrid approach changes the result, leading to success on previously unsolved instances and timeouts on previously solved instances. The concrete shift in solved instances depends on the LLM model and the underlying combinatorial problem. Overall, the generated local search functions allow solving additional instances and are, therefore, a good tool to tackle unsolved instances.

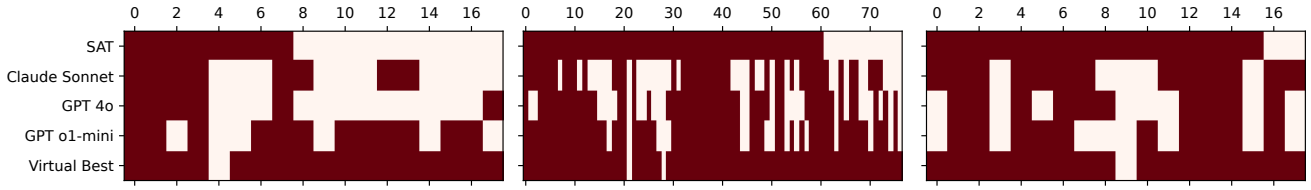
Discussion

Our results show that it is indeed possible to automatically generate and evaluate local search prototypes using LLMs. In this part of our evaluation, we want to discuss aspects of our approach that are of interest for adapting our approach.

Test and Training Correlation The automatic evaluation relies on some automated way of ranking the different local search functions. Hence, we are interested in how well the training results predict the testing results. Figure 4 visualizes this. The score for a function on an instance is determined by the function's running time on that instance divided by the best known running time on that instance. We give the average over all instances. Hence, a score of 1 indicates that the function achieved the best running time on all instances and a score of 0 indicates a timeout on all instances. The correlation is strongest for DFVS, becomes weaker for BDDT, and is almost non-present for coloring. DFVS has the most training instances and the training instances come from several different sources, giving the training set diversity. In contrast, coloring instances tend to be either easy or hard and few instances fit into the 10 to 60 second range. This leads to many test instance from the same source and low diversity. BDDT also has fewer test instance than DFVS, but the datasets are very different from each other. The selection of training instances is, therefore, very important for finding the best performing local search functions.

Code Diversity We manually reviewed the code generated by the LLMs in an effort to judge how much the code varies, as well as the overall quality of the code. Due to the large number of generated local search functions—over 900—we cannot review all the code. Hence, we focus on trends.

The LLMs manage to create prototypes and these proto-



(a) Coloring (b) DFVS (c) BDDT

Figure 3: Instances solved by either the SAT solver alone (*SAT*), or the SAT solver in conjunction with a local search method (remaining rows). For each LLM model, we show the single best local search method, and the *Virtual Best* shows which instances the SAT solver could solve in conjunction with any of the generated local search functions.

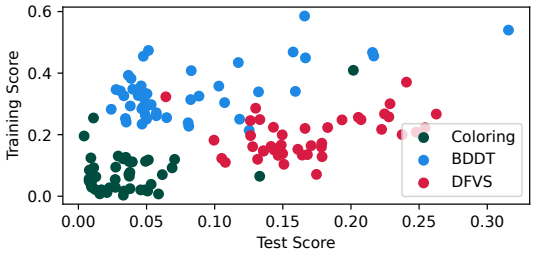


Figure 4: Function scores on the training instances and test instances. Each marker represents one local search function.

types are rarely performance-optimized. The functions generally compute a score to determine the best change to the assignment. This is often done repeatedly by iterating over the whole instance, not caching any results, resulting in a major performance bottleneck. We address this specific issue via prompt and the LLMs can often fix it, but performance bottlenecks that are specific to single local search functions cannot be addressed via generic prompt. Hence, we expect that the local search functions can be improved even more with a performance-oriented implementation. Further, potential issues should be assessed before the Gathering Phase by generating a small set of initial local search functions and manually reviewing them. These issues can then be addressed in the prompt.

The LLMs usually provide their implementation goal in the comments. According to these comments, the local search functions implement over 50 different meta-heuristics. Unfortunately, many of the implementations do not work as described in the comments and sometimes do not work at all. Interestingly, Claude Sonnet often tries to combine different meta-heuristics, while the GPT models try to implement a single approach. The local search function generated by GPT 4o resemble more general approaches, while the GPT o1-mini and Claude Sonnet generated function have clear adaptations due to our prompts.

Issues are usually filtered away due to bad performance on the training set. Surprisingly, even wrong implementations can sometimes deliver good results. Hence, we found several such issues in even the top-performing local search functions. Consequently, there is room for even better performance when efficiently implementing the prototypes, but a proper analysis of the code is necessary beforehand.

Analysis of Generated Search Strategies We also analyzed the ideas present in the best local search functions. Since the graph coloring encoding is relatively simple, the best local search functions use ideas from general graph coloring tabu search. In the case of DFVS, the local search functions reflect that the LLM models do not fully understand the encoding scheme, as the best local search functions overall lack working encoding scheme specific mechanisms. Hence, the best local search function implements WalkSAT. The local search functions for BDDT are the most interesting ones, as they use very encoding scheme specific approaches that are novel. Overall, the quality and specificity of the local search functions for BDDT shows how well the LLMs can adapt the code to even complicated encoding schemes, as long as the LLM model understands them.

Conclusion

We demonstrated that it is possible to automatically generate effective local search algorithms by having LLMs analyze SAT encoding schemes. Our key innovation lies in targeting the encoding methodology rather than specific problem instances, allowing the generated strategies to work across all problems sharing the same encoding pattern. This scheme-centric approach produced diverse search strategies whose performance correlated with the LLMs' comprehension of the underlying encoding structures.

Our work shows that the traditional manual design pipeline can be augmented or partially automated through encoding-aware LLM analysis. This perspective shift applies to the broader field of combinatorial optimization, where problem structure often remains underutilized due to the expertise barriers in algorithm design.

The observed correlation between encoding comprehension and algorithm quality indicates that improvements in LLM reasoning capabilities will directly translate to more sophisticated algorithmic synthesis. This places our approach at the intersection of several research frontiers: automated algorithm design, structure-aware optimization, and neural-symbolic integration.

We contend that the combinatorial search community should investigate this form of algorithm synthesis as a complementary approach to traditional algorithm design methodologies, potentially democratizing access to high-performance specialized algorithms across problem domains while revealing structural insights that might elude human analysis.

Acknowledgments

This work was supported by Austrian Science Fund (FWF) grants 10.55776/P36420 and 10.55776/COE12, as well as an Amazon Research Award (Fall/2023).

References

Avellaneda, F. 2020. Efficient Inference of Optimal Decision Trees. In *Proceedings of AAAI 2020*. AAAI Press.

Balint, A.; and Manthey, N. 2013. Boosting the Performance of SLS and CDCL Solvers by Preprocessor Tuning. In *POS@SAT*, volume 29 of *EPiC Series in Computing*, 1–14. EasyChair.

Bessiere, C.; Hebrard, E.; and O'Sullivan, B. 2009. Minimising Decision Tree Size as Combinatorial Optimisation. In *Proceedings of CP 2009*, 173–187. Berlin, Heidelberg: Springer Berlin Heidelberg.

Biere, A. 2019. CaDiCaL at the SAT Race 2019. In *PSAT Race 2019: Solver and Benchmark Descriptions*, 8–9.

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024. CaDiCaL 2.0. In *Proceedings of CAV 2024*, volume 14681 of *LNCS*, 133–152. Springer.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling, and Treengeling Entering the SAT Competition 2020. In *SAT Competition 2020: Solver and Benchmark Descriptions*, 51–53.

Brélaz, D. 1979. New Methods to Color the Vertices of a Graph. *Commun. ACM*, 22(4): 251–256.

Cai, S.; and Zhang, X. 2021. Deep Cooperation of CDCL and Local Search for SAT. In *Proceedings of SAT 2021*, volume 12831 of *Lecture Notes in Computer Science*, 64–81. Springer.

Cai, S.; Zhang, X.; Fleury, M.; and Biere, A. 2022. Better Decision Heuristics in CDCL through Local Search and Target Phases. *J. Artif. Intell. Res.*, 74: 1515–1563.

Fichte, J. K.; Lodha, N.; and Szeider, S. 2017. SAT-Based Local Improvement for Finding Tree Decompositions of Small Width. In *Proceedings of SAT 2017*, volume 10491 of *LNCS*, 401–411. Springer.

Gelder, A. V. 2008. Another look at graph coloring via propositional satisfiability. *Discret. Appl. Math.*, 156(2): 230–243.

Großmann, E.; Heuer, T.; Schulz, C.; and Strash, D. 2022. The PACE 2022 Parameterized Algorithms and Computational Experiments Challenge: Directed Feedback Vertex Set. In *Proceedings of IPEC 2022*, volume 249 of *LIPIcs*, 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437.

Janota, M.; Grigore, R.; and Manquinho, V. 2017. On the Quest for an Acyclic Graph. In *Proceedings of the 24th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2017, Bari, Italy, November 14-15, 2017*, volume 2011 of *CEUR Workshop Proceedings*, 33–44. CEUR-WS.org.

Kiesel, R.; and Schidler, A. 2022. PACE Solver Description: DAGer - Cutting out Cycles with MaxSAT. In *Proceedings of IPEC 2022*, volume 249 of *LIPIcs*, 32:1–32:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Kiesel, R.; and Schidler, A. 2023. A Dynamic MaxSAT-based Approach to Directed Feedback Vertex Sets. In *Proceedings of ALENEX 2023*, 39–52. SIAM.

Li, C. M.; and Li, Y. 2012. Satisfying versus Falsifying in Local Search for Satisfiability - (Poster Presentation). In *Proceedings of SAT 2012*, volume 7317 of *LNCS*, 477–478. Springer.

Li, Y.; Choi, D. H.; Chung, J.; Kushman, N.; Schrittwieser, J.; Leblond, R.; Eccles, T.; Keeling, J.; Gimeno, F.; Lago, A. D.; Hubert, T.; Choy, P.; de Masson d'Autume, C.; Babuschkin, I.; Chen, X.; Huang, P.; Welbl, J.; Gowal, S.; Cherepanov, A.; Molloy, J.; Mankowitz, D. J.; Robson, E. S.; Kohli, P.; de Freitas, N.; Kavukcuoglu, K.; and Vinyals, O. 2022. Competition-Level Code Generation with AlphaCode. *CoRR*, abs/2203.07814.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning Optimal Decision Trees with SAT. In *Proceedings of IJCAI 2018*, 1362–1368. ijcai.org.

Nijkamp, E.; Pang, B.; Hayashi, H.; Tu, L.; Wang, H.; Zhou, Y.; Savarese, S.; and Xiong, C. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *Proceedings of ICLR 2023*. OpenReview.net.

Olson, R. S.; La Cava, W.; Orzechowski, P.; Urbanowicz, R. J.; and Moore, J. H. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1): 36.

Schidler, A.; and Szeider, S. 2024. SAT-based Decision Tree Learning for Large Data Sets. *J. Artif. Intell. Res.*, 80: 875–918.

Selsam, D.; and Bjørner, N. S. 2019. Guiding High-Performance SAT Solvers with Unsat-Core Predictions. In *Proceedings of SAT 2019*, volume 11628 of *LNCS*, 336–353. Springer.

Shati, P.; Cohen, E.; and McIlraith, S. A. 2021. SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features. In *CP*, volume 210 of *LIPIcs*, 50:1–50:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Soos, M. 2020. CryptoMiniSat 5.6.8. In *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*, 37–38.

Sun, W.; Hao, J.; Zang, Y.; and Lai, X. 2021. A solution-driven multilevel approach for graph coloring. *Appl. Soft Comput.*, 104: 107174.

Verwer, S.; and Zhang, Y. 2019. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. In *Proceedings of AAAI 2019*, 1625–1632. AAAI Press.

Yolcu, E.; and Póczos, B. 2019. Learning Local Search Heuristics for Boolean Satisfiability. In *Proceedings of NeurIPS 2019*, 7990–8001.

Zhou, H.-J. 2016. A spin glass approach to the directed feedback vertex set problem. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(7): 073303.