


SAT-Based Local Search for Plane Subgraph Partitions

André Schidler  

TU Wien, Austria

Abstract

The Partition into Plane Subgraphs Problem (PPS) asks to partition the edges of a geometric graph with straight line segments into as few classes as possible, such that the line segments within a class do not cross. We discuss our approach *GC-SLIM*: a local search method that views PPS as a graph coloring problem and tackles it with a new and unique combination of propositional satisfiability (SAT) and tabu search, achieving the fourth place in the 2022 CG:SHOP Challenge.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases graph coloring, plane subgraphs, SAT, logic, SLIM, local improvement, large neighborhood search

Digital Object Identifier 10.4230/LIPIcs.SoCG.2022.74

Category CG Challenge

Supplementary Material *Software (Source Code)*: <https://github.com/ASchidler/coloring>

archived at `swh:1:dir:2b7057f17495a9a12cf7de4f857037c9ab0d6654`

Dataset (Results): <https://doi.org/10.5281/zenodo.6352601>

Funding FWF (P32441, W1255) and the WWTF (ICT19-065).

1 Introduction

Expressing the Partition into Plane Subgraphs Problem (PPS) in terms of graph coloring allows us to use decades of research on this important and well-researched NP-hard problem. While recent research investigated how to color massive graphs with several million vertices [11, 12], closer analysis of the used instances show that they are large but also very sparse. In contrast, the conflict graphs of this year’s challenge are smaller in size, with up to 73000 vertices, but have an edge density of up to 62% leading to conflict graphs with over 1.5 billion edges. Many established methods for graph coloring do not perform well on such dense graphs, making local search methods very appealing.

We present our approach *GC-SLIM*, a combination of propositional satisfiability (SAT) and tabu search based on the SAT-based local improvement (SLIM) meta-heuristic [9, 13, 14, 15, 16]. While a SAT-encoding of graph coloring can compute colorings for 26 of the 225 challenge instances, GC-SLIM scales to the largest challenge instances, improves upon established tabu search, and placed 4th overall and 2nd among student submission [7, 8, 10, 17].

2 Preliminaries

In this paper, we only consider PPS in terms of graph coloring, i.e., we consider the conflict graph $G' = (V', E')$, containing a vertex for each line segment, and two vertices are adjacent if the corresponding line segments intersect [8]. Since GC-SLIM performs local search, we assume a given k -coloring $c : V(G') \rightarrow \{1, \dots, k\}$. We use the shorthands $S_\ell = \{v \in V(G') \mid c(v) = \ell\}$, $[k] = \{1, \dots, k\}$, and $N(v) = \{w \mid \{v, w\} \in E(G')\}$.



© André Schidler;

licensed under Creative Commons License CC-BY 4.0

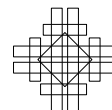
38th International Symposium on Computational Geometry (SoCG 2022).

Editors: Xavier Goaoc and Michael Kerber; Article No. 74; pp. 74:1–74:8

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



We compute the initial solution using DSATUR [6], one of the best greedy heuristics for graph coloring. DSATUR colors one vertex after another, assigning each vertex the smallest possible color that avoids monochromatic edges. DSATUR always colors the most constrained vertex next, i.e., the vertex that has the fewest viable colors available.

GC-SLIM extends the tabu search *PARTIALCOL* [5]. The idea of *PARTIALCOL* is to focus on eliminating a single color ℓ through a series of *swaps*: given a vertex v colored with ℓ and another color ℓ' , a swap changes v 's color to ℓ' and the color of all vertices in $N(v) \cap S_{\ell'}$ to ℓ . Whenever ℓ' does not occur in v 's neighborhood, the number of vertices colored with ℓ decreases. Otherwise, ℓ is propagated in the graph in the hope of success with a later swap. *PARTIALCOL* picks ℓ' among the least prevalent colors in the neighborhood of v , i.e., that minimizes $|N(v) \cap S_{\ell'}|$, with ties broken arbitrarily. The choice of ℓ' is further informed by a list of tabus for each vertex. This list contains all colors a vertex had in the last few iterations, these cannot be used for swaps to avoid getting stuck in local optima.

3 Method

We use a combination of tabu search and SAT-solving based on the SLIM method [9, 13, 14, 15, 16]. This method improves a heuristic solution through a series of local improvements, where each local improvement is accomplished by solving a smaller *local instance* with a SAT solver. Key to this method is a way to extract local instances that achieves overall improvement. For GC-SLIM the goal is eliminating color k from a given k -coloring c , thereby changing c to a $(k - 1)$ -coloring. Note that colors are interchangeable. GC-SLIM is then called again with the new coloring, until no more improvements are possible.

We use list coloring for the local instances, as it enables us to color a subgraph of G' in a way that remains compatible with the coloring outside the subgraph.

► **Definition 1** (List Coloring). *Given a k -annotated graph (G^*, L) , where $L : V(G^*) \rightarrow 2^{[k]}$, a k -list coloring is a k -coloring c for G^* , such that for all $v \in V(G^*)$ it holds that $c(v) \in L(v)$.*

Given a set $X \subseteq V(G')$ and a coloring c for G' , we create a list coloring instance (G^*, L) : $G^* = G'[X]$, the subgraph induced by X , and for each $v \in X$, we let $L(v) := [k] \setminus \{c(w) \mid w \in N(v) \setminus X\}$. Given a list coloring c' for G^* , we combine c' and c by changing $c(v)$ to $c'(v)$ for all $v \in X$. Afterwards, c is still a coloring for G' , as the lists ensure no monochromatic edges between X and $V(G^*) \setminus X$. If we are able to solve all local instances for vertices colored with k , we eventually eliminate color k from c .

We can solve the list coloring problem with a simple SAT encoding. We use for each vertex $v \in V(G^*)$ and color $\ell \in L(v)$ the variable $c_{v,\ell}$ which is true if and only if v can take color ℓ . We encode that each vertex needs a color with $\bigwedge_{v \in V(G^*)} \bigvee_{\ell \in L(v)} c_{v,\ell}$. Further, we ensure that adjacent vertices have different colors: $\bigwedge_{\{u,v\} \in E(G^*), \ell \in L(u) \cap L(v)} \neg c_{u,\ell} \vee \neg c_{v,\ell}$.

Finding good local instances is surprisingly difficult. While it was possible to find k -colorings for many local instances, eventually we would always come upon a vertex that could not be recolored, no matter how we created the local instance. We overcame this issue with inspiration taken from *PARTIALCOL*.

GC-SLIM does not try to eliminate k from the local instance. Instead, our method is satisfied with any k -coloring that is different and that minimizes the number of k -colored vertices. Given a set X of vertices that defines the local instance (G^*, L) , we perform the following two changes on L : (i) for all vertices $v \in S_k$ we remove k from $L(v)$, thereby forcing all k -colored vertices to change their color, and (ii) for all vertices $v \in X \setminus S_k$, we add k to $L(v)$. Since we eventually eliminate k , monochromatic edges using k are not an issue. Finally,

we constrain the number of vertices colored by k using a cardinality constraint [3]. Whenever we find a coloring for the local instance, we reduce the cardinality of the constraint. This follows the idea of swaps, where no vertex remains colored with k if possible, and otherwise k is propagated through the graph.

We complement this change in goal with our method for constructing local instances: starting from a single vertex $v \in S_k$, $X_0 = \emptyset$ and $X_1 = \{v\}$, GC-SLIM constructs $X = \bigcup X_i$ iteratively such that the two invariants $X \cap S_k = \{v\}$ and $|X| \leq b$ hold, i.e., v is the only vertex with color k and $|X|$ does not exceed a budget b . The set $X_{i>1}$ is constructed by adding specific neighbors of each vertex $w \in X_{i-1} \setminus X_{i-2}$, the vertices added in the last iteration. For each w , GC-SLIM adds the neighbors colored in the m least prevalent colors to X_i . The process stops when no more vertices can be added without exceeding the budget. We call m the *branching limit*, as the whole process resembles breadth-first search where the breadth in each step is limited by m . The budget ensures that the local instances stay small enough to be tackled by a SAT solver. Further, we use $\min_{\ell \in [k-1]} |c_\ell \cap N(v)|$ as the upper bound on the number of vertices colored with k in the local instance. This corresponds to a normal swap and ensures that GC-SLIM does not perform worse than PARTIALCOL.

Similar to tabu search, GC-SLIM stores the last colors of a vertex and disregards them during the construction of X , even if they are the least prevalent. We also considered removing tabu colors from the lists of the local instances. This yielded worse results as it restricts the possibilities for the SAT solver too much.

3.1 Hyperparameters

There are several hyperparameters that can severely impact GC-SLIM's performance.

The *timeout for the SAT solver* determines the time the solver has to find a solution. A large number allows for more improvements but may waste time without finding a solution. Depending on the instance and the concrete timeout, 25% to 50% of the SAT calls time out. Generally, a small timeout of 5 seconds performs well initially as it finds improvements fast, and a higher timeout can reveal more improvements once the low timeout fails.

The *iteration limit* determines the number of local instances GC-SLIM generates per color. Higher numbers increase the chance of success, but may waste time if unsuccessful. Again, low limits are good initially and higher iteration limits perform better in later stages.

The *branching limit* controls the breadth versus depth of the exploration when generating the local instance. Varying this parameter leads to different results, where smaller values of 2 and 3 find the most improvements, and values up to 15 reveal further improvements.

GC-SLIM adjusts the *budget* for the local instance automatically. Starting from an initial budget of 300 vertices, after any three consecutive SAT solver calls that time out, the budget decreases by 60 vertices. Whenever three consecutive SAT solver calls are successful, the budget increases by 60 vertices. In practice, the budget varies between 60 vertices – the lower limit – for very dense graphs and over 2000 vertices for sparse graphs.

3.2 Parallelisation

GC-SLIM uses multithreading as follows. Each thread tries to eliminate a color on its own and threads only synchronize if one of them succeeds. At this point, the successful thread shares the improved coloring and each thread starts again picking a color to eliminate.

We use multithreading in two ways: (i) to eliminate more colors in the same amount of time, and (ii) to try different hyperparameter settings, potentially on the same colors.

4 Experiments

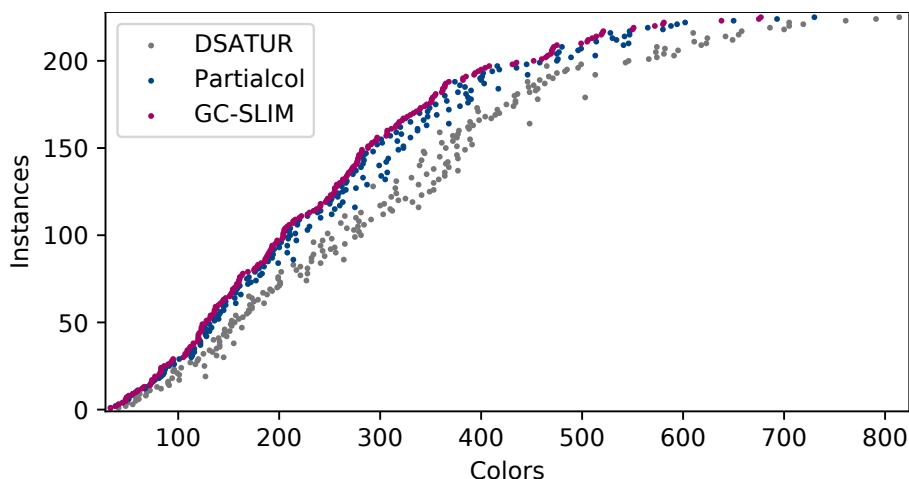
We discuss both the results from the competition and experiments with a specific time limit. GC-SLIM was implemented in C++, compiled with gcc 7.5.0, and run under Ubuntu 18.04.

The servers we used during the competition had two Xeon E5-2640v CPUs, each with 10 cores running at 2.4 GHz. The experiments ran on nodes with two AMD EPYC 7402 CPUs, each with 24 cores running at 2.8 GHz. Each run was limited to 64 GB of memory and 24 hours runtime. We used all available cores and no other experiments ran in parallel.

GC-SLIM uses the SAT solvers Glucose 3 [2] and Cadical 1.5.0 [1, 4]. Cadical is the default solver and when varying the hyperparameters, we also varied solvers.

4.1 Competition

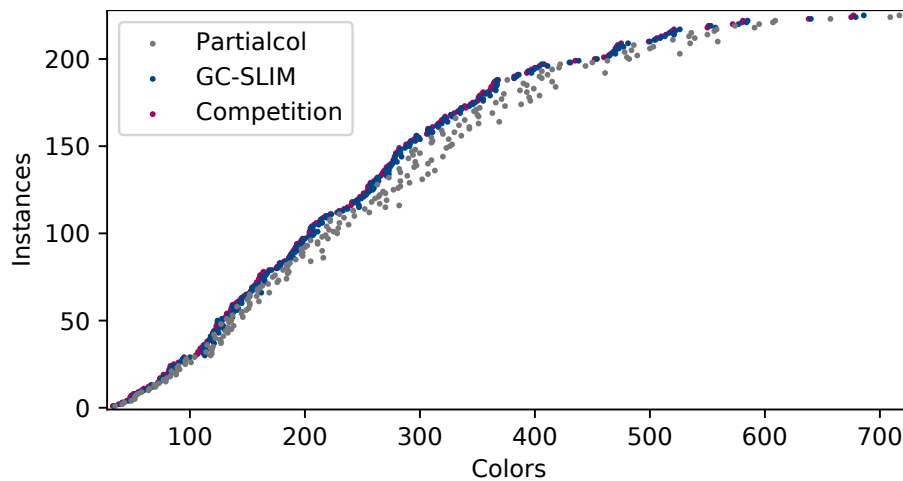
During the competition, we kept the current best coloring for each instance in a repository. This allows us to give a rough timeline of the changes, particularly on the impact of GC-SLIM. Figure 1 shows the initial number of colors, the number of colors achieved with PARTIALCOL right before introducing GC-SLIM, and the final number of colors.



■ **Figure 1** Comparison of the best colorings at different phases of the competition. The instances are ordered by the number of colors in the best result.

In the first phase, we used PARTIALCOL with varied parameters. In each iteration, the color for swaps was initially the least prevalent color, until PARTIALCOL failed to find improvements. Then, we added small random values to the color counts to diversify the exploration of the search space, until eventually, we picked random colors, which enabled further improvements. At the end of this phase, PARTIALCOL could not eliminate any color within 10 million swaps.

In the last phase, we used several runs of GC-SLIM per instance. Each run had up to twelve hours to eliminate one or more colors. For each run, we varied hyperparameters and SAT solvers. Towards the end, we used multithreading for instances with a large gap between upper bound and a clique lower bound. As Figure 1 shows, GC-SLIM was able to improve the colorings of most instances. On average, GC-SLIM removed over 50 additional colors per instance. Unfortunately, developing GC-SLIM was a long process and the final version finished too close to the deadline to achieve the best possible results.



■ **Figure 2** Comparison of 24 hour runs of tabu search and GC-SLIM. Our competition results are added as a reference. The instances are ordered by the number of colors in the best coloring.

4.2 24-hour experiment

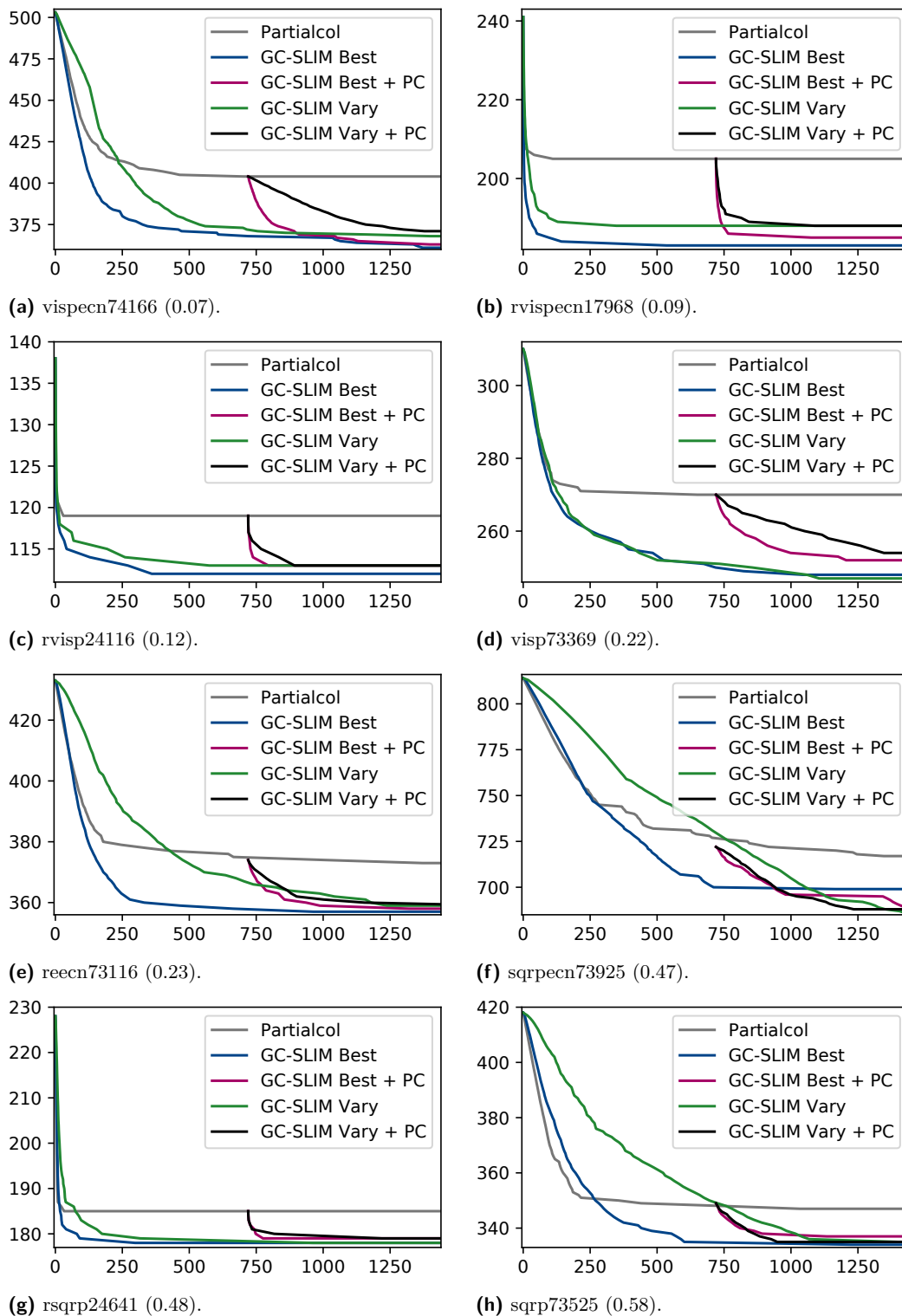
The usefulness of our method is not fully captured by the competition results, as applying local search for several weeks is usually not an option. We therefore ran PARTIALCOL and GC-SLIM with various configurations for 24 hours. For this purpose, we tried different hyperparameter settings in a shorter 5-hour run on ten instances and picked the settings that performed best as *best* configuration. The *varied* configuration uses a different set of hyperparameters in each thread. Both configurations use five parallel threads. These configurations run either alone, or combined with PARTIALCOL, where PARTIALCOL runs for twelve hours and then GC-SLIM for another twelve hours.

The overall results in Figure 2 – GC-SLIM shows the best result over all configurations – show that GC-SLIM improves upon PARTIALCOL even in this shorter timeframe. Further, the experimental results are comparable to our competition results on many instances: the competition results use only 3 fewer colors on average. This suggests the strong possibility for further improvements, given more time.

In Figure 3 we show how the number of colors develops over time for the largest instances of their respective set. We can see that the majority of improvements are achieved within a short amount of time, and PARTIALCOL quickly struggles, while GC-SLIM is able to find further improvements. Interestingly, the combination of both methods is sometimes able to find more improvements than either method alone.

5 Conclusion

GC-SLIM showed good results both in the experiments and in the competition. Key to the performance of our algorithm is the local instance selection that is closely tied to PARTIALCOL. We hope that our method will serve as a template for combining local search and SAT, and considering other local search methods will lead to further improvements.



■ **Figure 3** Color reduction (y-axis) over time (x-axis in minutes) for different configurations.

References

- 1 Cadical. <http://fmv.jku.at/cadical/>. Accessed: 2022-02-20.
- 2 Glucose. <https://www.labri.fr/perso/lsimon/glucose/>. Accessed: 2022-02-20.
- 3 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. doi:10.1007/978-3-540-45193-8_8.
- 4 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.
- 5 Ivo Blöchliger and Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.*, 35(3):960–975, 2008. doi:10.1016/j.cor.2006.05.014.
- 6 Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, April 1979.
- 7 Loïc Crombez, Guilherme D. da Fonseca, Yan Gerard, and Aldo Gonzalez-Lorenzo. Shadoks approach to minimum partition into plane subgraphs. In *Symposium on Computational Geometry (SoCG)*, pages 71:1–71:8, 2022.
- 8 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum partition into plane subgraphs: The CG:SHOP Challenge 2022. *CoRR*, abs/2203.07444, 2022. arXiv:2203.07444.
- 9 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. SAT-based local improvement for finding tree decompositions of small width. In *Proceedings of SAT 2017*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411. Springer Verlag, 2017.
- 10 Florian Fontan, Pascal Lafourcade, Luc Libralesso, and Benjamin Momège. Local search with weighting schemes for the CG:SHOP 2022 competition. In *Symposium on Computational Geometry (SoCG)*, pages 73:1–73:7, 2022.
- 11 Emmanuel Hebrard and George Katsirelos. A hybrid approach for exact coloring of massive graphs. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 374–390. Springer, 2019. doi:10.1007/978-3-030-19212-9_25.
- 12 Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 517–523. ijcai.org, 2017. doi:10.24963/ijcai.2017/73.
- 13 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4894–4898. ijcai.org, 2017. doi:10.24963/ijcai.2017/689.
- 14 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Learning fast-inference bayesian networks. *Advances in Neural Information Processing Systems*, 34, 2021.
- 15 Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Turbocharging treewidth-bounded Bayesian network structure learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3895–3903. AAAI Press, 2021.

- 16 André Schidler and Stefan Szeider. SAT-based decision tree learning for large data sets. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3904–3912. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16509>.
- 17 Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Conflict-based local search for minimum partition into plane subgraphs. In *Symposium on Computational Geometry (SoCG)*, pages 72:1–72:6, 2022.