From Cliques to Colorings and Back Again

Marijn J.H. Heule ✉ ⓘ
Carnegie Mellon University, United States

Anthony Karahalios ✉ ⓘ
Carnegie Mellon University, United States

Willem-Jan van Hoeve ✉ ⓘ
Carnegie Mellon University, United States

─── **Abstract** ───────────────

We present an exact algorithm for graph coloring and maximum clique problems based on SAT technology. It relies on four sub-algorithms that alternatingly compute cliques of larger size and colorings with fewer colors. We show how these techniques can mutually help each other: larger cliques facilitate finding smaller colorings, which in turn can boost finding larger cliques. We evaluate our approach on the DIMACS graph coloring suite. For finding maximum cliques, we show that our algorithm can improve the state-of-the-art MaxSAT-based solver IncMaxCLQ, and for the graph coloring problem, we close two open instances, decrease two upper bounds, and increase one lower bound.

1 Introduction

Given a graph, the *vertex coloring problem* asks to label each vertex of the graph with a color such that adjacent vertices have different labels, using the minimum number of colors (the *coloring number*). A closely related problem is the *maximum clique problem*, which asks to find a subset of vertices that are pairwise adjacent, of maximum size (the *clique number*). Both are NP-hard combinatorial optimization problems at the heart of practical applications including scheduling, timetabling, and network analysis [11, 36].

Many different algorithms have been proposed to solve vertex coloring and maximum clique problems in practice. One stream of research focuses on dedicated exact and heuristic algorithms (e.g., Cliquer [19] and DSATUR [1]), while another stream uses generic methodologies, such as integer programming and column generation (e.g., [7, 17, 18]), constraint programming (e.g., [4, 21]), or Boolean satisfiability (e.g., [6, 13, 32]). An important milestone for these developments was the second DIMACS challenge on cliques, coloring, and satisfiability that was launched in 1993 [11]. To our knowledge, for the DIMACS graph coloring challenge, several instances remain unsolved and in the past eight years only a few instances were closed: wap01a in 2021 [26], 5-FullIns_4 in 2021 [31], and 4-FullIns_5 in 2014 [14, 38]. Our method solves these instances as well (and quickly). Similarly, only a few improved bounds have been found that do not close instances: C2000.9 in 2021 [33] and DSJC250.1 in 2020 [20]. Before this, many instances were closed around 2012 [4, 8, 15, 27–29, 35] and earlier.

In this work, we first revisit the performance of Boolean satisfiability (SAT) solvers on graph coloring and maximum clique problems. The best known maximum clique solver called IncMaxCLQ [13] is based on MaxSAT technology, which is able to close all but four
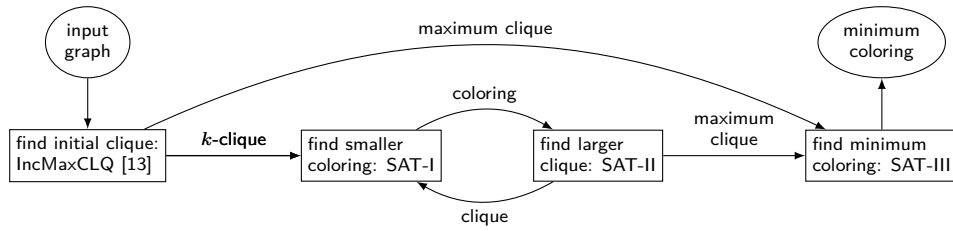
■ **Figure 1** Illustration of the CliColCom algorithm to find a maximum clique and a minimum vertex coloring. The solver IncMaxCLQ is based on MaxSAT. The SAT-I encoding uses a given clique to quickly find colorings. The SAT-II encoding uses these colorings to find a larger clique. Once the maximum clique is found, encoding SAT-III is used to find the minimum coloring.

instances of the DIMACS Clique benchmark suite and find maximum cliques for all but eight instances of the DIMACS Coloring benchmark suite. For graph coloring, the best known solver is the branch-and-bound hybrid CP/SAT solver gc-cdcl [6]. We show that a direct encoding coupled with either the local search SAT solver DDFW [9] or CaDiCaL [3] provides surprisingly strong results. For the 69 DIMACS coloring instances where the coloring number equals the clique number, combining IncMaxCLQ and one of DDFW or CaDiCaL solves 54 instances in under ten minutes.

We therefore concentrate on two cases - finding stronger colorings for instances where we quickly have a maximum clique, and improving both cliques and colorings for instances where we do not quickly find a maximum clique, which are often those where the coloring number does not equal the clique number. We propose an algorithm, named *CliColCom*, (derived from 'cliques, colorings, and communication') that consists of four sub-algorithms with an inner loop that alternates between finding cliques of larger size and colorings with fewer colors (see Fig. 1). Specifically, we use cliques to define a symmetry-breaking predicate based on a variable ordering for the coloring problem (using encoding SAT-I), similar to ones used by Van Gelder [32] and Velev [34]. Conversely, we use colorings to formulate the maximum clique problem (using encoding SAT-II), similar to the MaxSAT encoding by Li [13]. We continue this alternating process until a maximum clique is found, which serves as input to the final sub-algorithm that finds a minimum coloring (using encoding SAT-III). This approach can be viewed as a new form of communication between SAT solvers. While one way that SAT solvers communicate is through exchanging learned clauses like in portfolio-based parallel SAT [5], we demonstrate how SAT solvers can also pass solutions back and forth, using the other solver's solution in its problem's clauses.

We show that CliColCom can find larger cliques than IncMaxCLQ for two of the eight DIMACS Coloring instances that IncMaxCLQ cannot solve, and for the vertex coloring problem closes two open instances (wap02a, wap08a), improves one lower bound (r1000.1c), and improves two upper bounds (wap03a, wap04a).

The rest of this paper is organized as follows. In Section 2 we provide formal definitions and notation for graph coloring and maximum clique problems. Section 3 presents the details of our algorithm. In Section 4 we provide an overview of the used tools. The experimental evaluation is presented in Section 5, and we conclude in Section 6.

2 Graph Coloring and Maximum Clique Problems

We first recall the definitions of cliques and colorings [22]. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A *k-clique* is a subset of k vertices that are pairwise

adjacent. A *maximum clique* of G is a clique in G of maximum size. The size of a maximum clique is called the *clique number* of G.

An *independent set* is a subset of vertices that are pairwise non-adjacent. A *k-coloring* of G is a partition of V into k independent sets $V_1, V_2, \ldots, V_k$. The independent sets represent the *color classes* of the coloring. The *coloring number* of G is the size of a coloring that uses the minimum number of colors.

The existence of a k-clique proves a lower bound of k on the clique number, and a k-coloring proves an upper bound of k on the coloring number. To prove the dual bounds, one must show that a $k+1$-clique and $k-1$-coloring do not exist. The existence of a k-clique proves a lower bound of k for the coloring number. Both the vertex coloring and max clique problems are NP-hard, so computational results of algorithms are of interest [12, 16, 36].

3 CliColCom Algorithm

In this section we present an exact algorithm for graph coloring that also contains an exact algorithm for the maximum clique problem. It consists of four sub-algorithms as shown in Fig. 1. The algorithm for the maximum clique problem is obtained by omitting sub-problem SAT-III. We next describe each of the sub-algorithms below.

3.1 IncMaxCLQ: Find an Initial Clique

The input to the first sub-algorithm is a graph $G = (V, E)$. To obtain an initial clique, we run an exact MaxSAT solver called IncMaxCLQ [13] with a time limit; we use one second in our experiments. If IncMaxCLQ finds a maximum clique and proves optimality, then we immediately go to step SAT-III using this maximum clique. Otherwise the clique returned by IncMaxCLQ will be used for the SAT-I encoding.

3.2 SAT-I: Find a Coloring

The next sub-algorithm called SAT-I takes as input a graph $G = (V, E)$, a k-clique C, and an upper bound $b \geq k$. Its purpose is to find a coloring of good quality. When we first enter this sub-algorithm, we determine b by running the DSATUR graph coloring heuristic. In subsequent calls, b will be the best known coloring number.

The SAT-I encoding is optimized for local search solvers and asks for the existence of a b-coloring of G. It has two sets of constraints: 1) each vertex has at least one color; and 2) adjacent vertices are colored differently. The direct encoding uses color variables $x_{v,i}$ which denote that vertex $v \in V$ has color $i \in \{1, \ldots, b\}$. The first constraint consists simply of a single clause of b literals per vertex:

$(x_{v,1} \vee \cdots \vee x_{v,b})$ for $v \in V$.

Note that this only enforces that there is at least one color per vertex instead of exactly one color per vertex. The latter would include clauses of the form $(\overline{x}_{v,i} \vee \overline{x}_{v,j})$ for $1 \leq i < j \leq b$. These clauses however are known to be "blocked" and top-tier SAT solvers eliminate them [10].

The second constraint uses the following clauses:

$(\overline{x}_{u,i} \vee \overline{x}_{v,i})$ for $(u, v) \in E, i \in \{1, \ldots, b\}$.

To break the color symmetry, we add unit clauses that assign a different color to each vertex in the given clique C, which is a common practice [32].

The encoding is used as follows. We start with bound $b-1$ and run a local search solver for a limited time (or number of flips). If no coloring is found within the limit, we report the previously found b-coloring. Otherwise, we decrease b by one unit and repeat. We thus return the best coloring we can find within a limited time. Note that if the encoding for $b = |C|$ is satisfiable, then we have found the coloring number of G.

3.3   SAT-II: Find a Larger Clique

The third sub-algorithm uses the fact that a vertex coloring is a partition of the graph into independent sets. The coloring from SAT-I is used to define these independent sets. For a graph $G = (V, E)$ and a p-partition $\{V_1, \ldots, V_p\}$ of V into independent sets, the encoding SAT-II asks whether there exists a clique of size c, where $c \leq p$.

This encoding uses clique variables $v_{i,s}$, which denote that the s-th vertex in V_i is part of the clique. Apart from the clique variables, the encoding uses relaxation variables r_i denoting that no vertex from partition V_i is used in the clique. The clauses have the following form:

$$(r_i \vee v_{i,1} \vee \cdots \vee v_{i,|V_i|}) \text{ for } i \in \{1, \ldots, p\}.$$

Additionally we have constraints between partitions enforcing that two vertices from different partitions cannot be in a clique if there is no edge between them in the graph:

$$(\overline{v}_{i,s} \vee \overline{v}_{j,t}) \text{ for } 1 \leq i < j \leq p, s \in \{1 \ldots, |V_i|\}, t \in \{1, \ldots, |V_j|\}, (v_{i,s}, v_{j,t}) \notin E.$$

We could have included similar clauses for pairs of vertices within a partition. However, these clauses are blocked as well and top-tier solvers would remove them.

Finally, we have a constraint stating that at most $k = p - c$ of the relaxation variables can be assigned to true. We use the sequential counter encoding proposed by Sinz to enforce the 'at most $p - c$' constraint [25]. This encoding introduces $O(pk)$ auxiliary variables and $O(pk)$ additional clauses.

The sub-algorithm starts with $c = |C| + 1$ where C is the largest clique found so far by either IncMaxCLQ or SAT-II itself. We solve the encoding with an exact CDCL solver (see Section 4). If the formula is unsatisfiable, then the largest clique has size $c - 1$. Otherwise, we have found a clique C' of size c and continue by increasing the bound $c \mathrel{+}= 1$ and repeating this sub-algorithm. If the SAT-II encoding cannot be solved within a certain time limit, we return to sub-algorithm SAT-I to find a smaller vertex coloring, using the improved clique C'. Due the time limits imposed on SAT-I and SAT-II, we could in theory repeatedly solve them with the same clique and the same coloring. For that reason, we increase the time limit for SAT-II with a multiplicative factor when the coloring and the clique have not changed, so that sub-algorithm SAT-II becomes exact. Therefore, SAT-II will eventually return a maximum clique, unless a global time limit on the overall algorithm is exceeded.

3.4   SAT-III: Find an Optimal Coloring

The final sub-algorithm uses encoding SAT-III, which generalizes SAT-I and is optimized for CDCL solvers. The sub-algorithm takes as input a graph $G = (V, E)$, a k-clique C, and a lower bound b. The first part of the encoding is identical to the SAT-I encoding with G, C, and b as input.

We additionally add clauses to break color symmetries. To this end, we first construct a vertex ordering O, by starting with the vertices in C in arbitrary order. We then iteratively extend the ordering by adding the vertex with the most neighbors in O, breaking ties by

highest degree. We break the color symmetries for the vertices from $k + 1$ to $|O|$ in the ordering. Let v_i denote the i-th vertex in ordering O. The encoding enforces that if none of the first $i - 1$ vertices in O uses the color c, then vertex v_i must have a color less or equal to c. The clauses have the following form:

$$(x_{v_1,c} \vee x_{v_2,c} \vee \cdots \vee x_{v_{i-1},c} \vee \overline{x}_{v_i,d}) \text{ for } d \in \{c + 1, \ldots, b\}, i \in \{k + 1, \ldots, |O|\}.$$

The sub-algorithm uses this encoding as follows: starting with $b = k$, we solve the formula using an exact CDCL solver. If the formula is found to be unsatisfiable, meaning that G requires more than b colors, the bound is increased $b \mathrel{+}= 1$ and we repeat. This is continued until the formula is satisfiable. The final bound equals the coloring number of G.

3.5 Example Run

We illustrate the flow between the sub-algorithms using graph r1000.1c. This instance has clique number 92, while the best known lower and upper bound on the coloring number are 96 and 98, respectively [4]. We first run IncMaxCLQ, which returns a clique of size 82 within one second (which it cannot improve within reasonable time).

We next run the SAT-I encoding with bound $b = 110$ (from the coloring found by DSATUR). The local search solver UBCSAT with the WalkSAT algorithm lowers the upper bound one by one until it reaches $b = 102$ and times out (i.e., reaching a million flips without finding a coloring). Each step takes a few seconds. The 103-coloring is used in the SAT-II encoding. We start with $c = 83$ (the size of the clique + 1). The solver CaDiCaL finds a satisfying assignment in a fraction of a second. This also holds for the bounds $c \in \{84, \ldots, 92\}$. The bound $c = 93$ times out (reaches a million conflicts).

We return to SAT-I using the 92-clique. This helps the local search solver and now it can find a coloring for $b = 102$ and can even lower it to 98 before timing out on $b = 97$. The 98-coloring is used in SAT-II. This time CaDiCaL can prove optimality of $c = 92$ (the $c = 93$ instance is unsatisfiable). Now that the maximum clique has been determined, we switch to SAT-III to determine the coloring number. The clique of size 92 is extended to a vertex ordering. The solver CaDiCaL is used to solve the instances with bounds $b \in \{92, \ldots, 97\}$. The bounds up to $b = 96$ are unsatisfiable, while $b = 97$ times out (24 hours). Therefore, we report an improved lower bound of 97 on the coloring number for this open instance.

4 SAT Solving Paradigms

The best SAT-solving paradigm differs for each of the encodings proposed in the prior section. Because some sub-algorithms work by solving a sequence of SAT instances, the use of MaxSAT solvers could be also be explored. Below we will discuss the ones used during our experiments.

Conflict-Driven Clause Learning. The most effective and well-known exact SAT-solving paradigm is conflict-driven clause learning (CDCL) [24]. In the context of maximum clique and graph coloring, CDCL is mostly effective for unsatisfiability results. In particular, we use this paradigm to increase the lower bound results after the maximum clique was determined. Although the default heuristics in CDCL solvers are in general effective on a broad range of formulas, we observed that using negative branching instead of phase saving improves performance on graph coloring instances. We will use the CDCL solver CaDiCaL during the experiments and turn on negative branching (options `-forcephase=1 -phase=0`).

Local Search. An almost obscure, yet quite effective local search solving paradigm is called Divide and Distribute Fixed Weights (DDFW) [9]. In DDFW, all clauses have weights. The
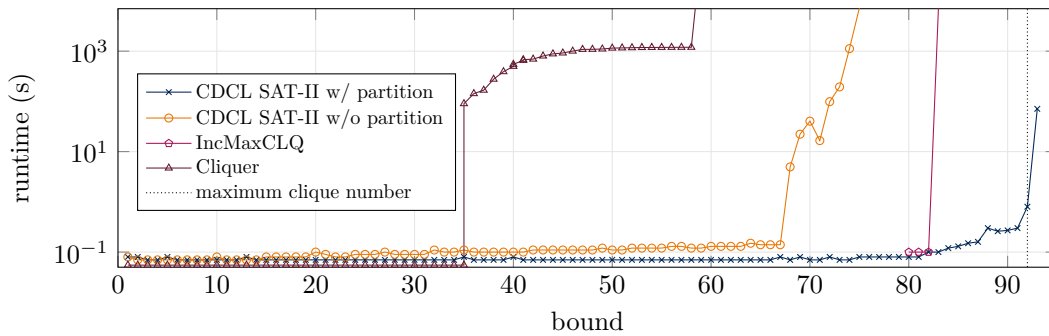
■ **Figure 2** Performance of different maximum clique techniques to compute a large clique of r1000.1c. The method SAT-II uses the partition obtained from a 98-coloring obtained by SAT-I.

algorithm flips variable assignments if the weighted sum of the satisfied clauses improves. If no such variable exists (i.e., a local minimum is reached), then a random falsified clause is selected which is increased in weight by pulling weight of its neighboring satisfiable clauses. This is repeated until a satisfying assignment is found. We use the implementation of DDFW in UBCSAT [30] for the experiments with SAT-III.

A well-known local search algorithm is WalkSAT [23]. Given a random assignment, the algorithm picks a random falsified clause and flips one of its literals to satisfy the clause. This is repeated until a satisfying assignment is found. WalkSAT is much more greedy compared to DDFW. This is helpful to reduce upper bounds in SAT-I. However, it is not effective to find a coloring for graphs when the coloring number equals the clique number. We use the implementation of WalkSAT in UBCSAT.

5 Experiments

We tested the performance of our method on solving both the maximum clique and vertex coloring problems on the 137 DIMACS Graph Coloring instances. This benchmark consists of a variety of instances with different sizes and densities - some random graphs and some from real world problems. We chose this sets of instances even for maximum clique performance because the DIMACS Maximum Clique and BHOSLIB [37] instances are almost all solved. The source code and log files of the experiments are available in the repository `https://github.com/marijnheule/clicolcom`. We will note when we run experiments on one of two different CPUs: Intel Xeon 2.33GHz CPU or AMD EPYC 7742 CPU [2].

5.1 Maximum Clique Results

As a baseline, we first ran IncMaxCLQ which solved 129 instances to optimality within one hour on the Intel Xeon 2.33GHz CPU. IncMaxCLQ failed to produce and prove a maximum clique for only eight graphs: C2000.5, C2000.9, C4000.5, latin_square_10, DSJC500.9, DSJC1000.9, DSJR500.1c, and r1000.1c. For the last two instances, the largest found cliques were of size 78 and 82, respectively. Our method is able to compute the maximum clique of them in a few minutes: 83 and 92, respectively. We are not aware of any other tool that can compute (and prove optimality) of the maximum cliques for these two instances.

Figure 2 illustrates the effectiveness of the SAT-II encoding. It shows for the open instance r1000.1c the runtimes of various techniques to find a large clique. We only find the maximum clique of size 92 with the SAT-II encoding that uses a 98-coloring obtained via SAT-I using
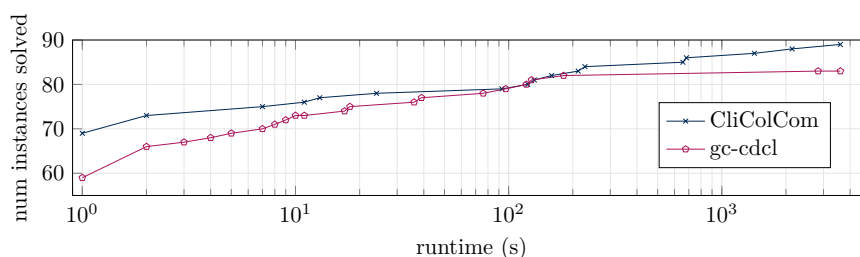
◾ **Figure 3** Performance profile of the number of DIMACS graph coloring instances gc-cdcl and CliColCom can prove to optimality over time.

local search. The instance with $b = 93$ is unsatisfiable, hence the larger runtime. IncMaxCLQ can compute a clique of size 82, while Cliquer gets only to 58. Without a coloring (i.e., each vertex is an independent set), SAT-II performance poorly and finds cliques up until size 74.

5.2 Comparison with State-of-the-Art Graph Coloring

We run these experiments on an Intel Xeon 2.33GHz CPU. We use gc-cdcl as a baseline because it is the SAT-based solver that performs best on this problem domain.[1]

We ran gc-cdcl for 1 hour and it proved the optimal solution for 83 instances. We compare this to our method in Fig. 3, which shows that we can solve 88 instances in 1 hour. The differences are as follows: gc-cdcl solves myciel7, queens9_9, and qg.order60 and CliColCom does not. CliColCom solves 4-FullIns_5, 1-Insertions_4, DSJR500.1c, le450_15c, le450_15d, wap01a, wap02a, wap06a and gc-cdcl does not.

We observed strong performance of our setup on the wap0* graphs. We therefore ran each instance on a cluster of AMD EPYC 7742 CPUs with 128 seeds. The results are reported in Table 1. We improve the upper bound on four graphs, which includes closing two instances. The DDFW algorithm was crucial to obtain these results. The wap01 instance was recently closed with a method that requires significantly more time [26].

◾ **Table 1** DDFW runtimes in seconds for wap0* instances using 128 seeds (no timeout). The second and third column show the lower and upper bounds. The bold bounds are improvements.

instance	LB	UB	min	mean	max
wap01a	41	41	291.19	736.01	1855.56
wap02a	40	**40**	195.45	382.85	883.02
wap03a	40	**43**	9612.49	15865.50	21963.13
wap04a	40	**41**	29757.11	65609.40	91501.84
wap05a	50	50	1.37	1.59	2.11
wap06a	40	40	9.21	26.44	92.54
wap07a	40	41	211.26	632.63	2207.33
wap08a	40	**40**	1016.65	6742.98	12096.61

---

[1] The paper "An Incremental SAT-Based Approach to the Graph Colouring Problem" published in CP 2019 claims strong computational results. Although these are reported in an aggregated form, they imply that several challenging open instances would have been solved. The GitHub repository linked in the paper was deleted. We contacted the authors, who were unfortunately unable to share the code or reproduce the published results. We therefore omit a comparison with that work.

5.3 Robustness, Variations, and Discussion

Naturally, our algorithm is sensitive to variations in its design. Below we discuss some extensions and variants to provide additional insights.

Robustness. Replacing the CDCL solver by any modern CDCL solver would hardly change runtime. The use of negative branching in CDCL slightly improves performance and is available in most SAT solvers. Increasing the timeout has little to no impact.

For the improved upper bounds of the wap graphs, we tried many local search solvers and only the implementation of DDFW in UBCSAT seems to be able to obtain them. The key aspects that impact the performance are: 1) fix only the clique for local search (full symmetry breaking significantly hurts local search solvers); 2) use full symmetry breaking for CDCL (otherwise unsatisfiable instances become impossible to solve); and 3) use the communication (otherwise hard problems cannot be solved).

Multiple colorings for SAT-II. The presented SAT-II encoding for finding a larger clique uses one vertex coloring in its clauses. This encoding can be extended to use multiple colorings by introducing corresponding sets of literals and clauses for each coloring. Taking DSJC250.9 as an example, we show that using multiple colorings can be beneficial to the runtime. We ran experiments using CaDiCaL for SAT-II that used either 1, 2, or 5 74-colorings to solve for a 43-clique. Using 40 trials for each number of colorings, the mean runtimes were 450, 62, and 202 respectively. This indicates that using two colorings may improve runtimes compared to one coloring, but using five colorings can perform worse than two colorings.

Vertex ordering for SAT-III. Encoding SAT-III for finding a minimum coloring uses a vertex ordering that begins with a maximum clique, assuming that starting with a maximum clique is effective. Although useful in most cases, we observed that this heuristic does not always result in the most effective ordering. For example, consider the graph coloring instances `queen9_9`, i.e., the n-queens instance of size 9. Its largest clique has size 9 (any row, column, or diagonal), while its coloring number is 10. Finding a clique of size 9 and a coloring of size 10 is easy. However, showing the absence of a 9-coloring (unsatisfiable, thus CaDiCaL was used) is hard: solving the SAT-III encoding starting with a border 9-clique (e.g., the top row) requires roughly 3 hours solving. Starting with a diagonal 9-clique reduces the runtime to 400 seconds. But, if one starts with a *non-optimal* 5-clique in the center (the + shaped clique that cannot be extended to a 6-clique), the runtime reduces to 100 seconds.

Heavy cliques for SAT-III. IncMaxCLQ is very effective in finding a maximum clique. However, we observed that when using a clique to generate a vertex ordering for SAT-III, performance of SAT-III may be enhanced by using the *heaviest* maximum clique, i.e., the maximum clique with the maximum sum of the vertex degrees. For example, for the queen instances the heaviest maximum clique is a diagonal, which we find to generate an ordering leading to better runtime when solving SAT-III compared to using a border row or column. Enhancing IncMaxCLQ to produce such a clique would further strengthen the results.

6 Conclusion

We were able to achieve state-of-the-art performance on the well-known DIMACS Coloring benchmark suite by combining off-the-shelf (Max)SAT-solving tools and a combination of three SAT encodings. Our algorithm, called CliColCom, uses the encodings to alternate between finding larger cliques and smaller colorings until a maximum clique and minimum coloring is found. We closed two open instances of the DIMACS benchmark suite and improved bounds on three others.

References

1 D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

2 Shawn T. Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A. Nystrom. *Bridges-2: A Platform for Rapidly-Evolving and Data Intensive Research*, pages 1–4. Association for Computing Machinery, New York, NY, USA, 2021. URL: `https://doi.org/10.1145/3437359.3465593`.

3 Armin Biere Katalin Fazekas Mathias Fleury and Maximilian Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. *SAT COMPETITION*, 2020:50, 2020.

4 Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

5 Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6(4):245–262, 2010.

6 Emmanuel Hébrard and George Katsirelos. Constraint and satisfiability reasoning for graph coloring. *Journal of Artificial Intelligence Research*, 69:33–65, 2020.

7 S. Held, W. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

8 Stephan Held, William Cook, and Edward C Sewell. Safe lower bounds for graph coloring. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 261–273. Springer, 2011.

9 Abdelraouf Ishtaiwi, John Thornton, Abdul Sattar, and Duc Nghia Pham. Neighbourhood clause weight redistribution in local search for sat. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 772–776, Berlin, Heidelberg, 2005. Springer.

10 Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010.

11 David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

12 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

13 Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 939–946. IEEE, 2013.

14 Shadi Mahmoudi and Shahriar Lotfi. Modified cuckoo optimization algorithm (mcoa) to solve graph coloring problem. *Applied soft computing*, 33:48–64, 2015.

15 Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

16 Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International transactions in operational research*, 17(1):1–34, 2010.

17 A. Mehrotra and M. A. Trick. A Column Generation Approach for Graph Coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

18 I. Méndez-Díaz and P. Zabala. A Branch-and-Cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154:826–847, 2006.

19 Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002.

20   Daniel Porumbel. Projective cutting-planes. *SIAM Journal on Optimization*, 30(1):1007–1032, 2020.

21   Jean-Charles Régin. Using Constraint Programming to Solve the Maximum Clique Problem. In *Proceedings of CP*, pages 634–648, 2003.

22   Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

23   Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*, pages 337–343. AAAI Press / The MIT Press, 1994.

24   João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

25   Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, pages 827–831. Springer, 2005.

26   Wen Sun, Jin-Kao Hao, Yuhao Zang, and Xiangjing Lai. A solution-driven multilevel approach for graph coloring. *Applied Soft Computing*, 104:107174, 2021.

27   Olawale Titiloye and Alan Crispin. Graph coloring with a distributed hybrid quantum annealing algorithm. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 553–562. Springer, 2011.

28   Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

29   Olawale Titiloye and Alan Crispin. Parameter tuning patterns for random graph coloring with quantum annealing. *PloS one*, 7(11):e50060, 2012.

30   Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2004.

31   R. P. van der Hulst. A branch-price-and-cut algorithm for graph coloring. Master's thesis, University of Twente, 2021.

32   Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2008.

33   Willem-Jan van Hoeve. Graph coloring with decision diagrams. *Mathematical Programming*, pages 1–44, 2021.

34   Miroslav N Velev. Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 135–142. IEEE, 2007.

35   Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290, 2012.

36   Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

37   Ke Xu. Bhoslib: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring)–hiding exact solutions in random graphs. web site. *Web site, http://www. nlsde. buaa. edu. en/~ kexu/benchmarks/graphbenchmarks. htm*, 2004.

38   Zhaoyang Zhou, Chu-Min Li, Chong Huang, and Ruchu Xu. An exact algorithm with learning for the graph coloring problem. *Computers & operations research*, 51:282–301, 2014.