Clause Learning and New Bounds for Graph Coloring*

Emmanuel Hebrard[1] and **George Katsirelos**[2]
[1]LAAS-CNRS, Université de Toulouse, CNRS, France
[2]UMR MIA-Paris, INRA, AgroParisTech, Université Paris-Saclay, Paris, France
hebrard@laas.fr, gkatsi@gmail.com

Abstract

Graph coloring is a major component of numerous allocation and scheduling problems.

We introduce a hybrid CP/SAT approach to graph coloring based on exploring Zykov's tree: for two non-neighbors, either they take a different color and there might as well be an edge between them, or they take the same color and we might as well merge them. Branching on whether two neighbors get the same color yields a symmetry-free tree with complete graphs as leaves, which correspond to colorings of the original graph.

We introduce a new lower bound for this problem based on Mycielskian graphs; a method to produce a clausal explanation of this bound for use in a CDCL algorithm; and a branching heuristic emulating Brelaz on the Zykov tree.

The combination of these techniques in a branch-and-bound search outperforms Dsatur and other SAT-based approaches on standard benchmarks both for finding upper bounds and for proving lower bounds.

1 Introduction

A *coloring* of a graph is a labeling of its vertices such that adjacent vertices have distinct labels. Let a labeling of the graph $G = (V, E)$ be a mapping from its set of vertices V to the integers. A labeling c such that $c(v) \neq c(u)$ for every edge $(uv) \in E$ is a coloring of G, and its cardinality is $|\{c(v) \mid v \in V\}|$. The *chromatic number* $\chi(G)$ of a graph G is the cardinality of its smallest coloring.

The problem of finding a minimum coloring of a graph is NP-hard, but has numerous applications. For instance, devices on nearby locations should not be assigned the same frequency to avoid interferences. The chromatic number of this distance-induced graph is therefore the minimum span of frequencies that is required [Aardal *et al.*, 2007; Park and Lee, 1996]. In compilers, finding an optimal register allocation can be cast as a coloring problem on an *interference* graph of value live ranges [Chaitin *et al.*, 1981].

One of the oldest and most successful technique for coloring a graph is Brelaz' *Dsatur* algorithm [Brélaz, 1979]: when branching, it assigns the lexicographically least color to the vertex with highest *degree of saturation*, that is, the highest number of assigned colors within its neighborhood $N_G(v)$ in G, breaking ties using the number of adjacent vertices. This heuristic is often used within a branch-and-bound algorithm with one variable per vertex whose domain is the set of possible colors. The standard approach for computing a bound in these algorithms is to compute a heuristic approximation of the clique number $\omega(G)$ of the graph G (e.g., the size of a maximal clique) since $\omega(G) \leq \chi(G)$. This bound is known to be weak for some polynomially recognizable classes of graphs, such as Mycielskian graphs, which are triangle-free graphs with arbitrarily large chromatic number [Mycielski, 1955]. Moreover, within the search tree explored using Brelaz' heuristic, the clique has to be found only among vertices with degree of saturation equal to the number of colors in the current partial solution (i.e., adjacent to at least one vertex of every color used so far). Finally, this formulation exhibits value interchangeability [Walsh, 2008]. One common way to break this symmetry is to arbitrarily color a clique, and never branch on colors larger than $k + 1$ when extending a solution with k colors [Van Hentenryck *et al.*, 2003; Mehrotra and Trick, 1996; Zhou *et al.*, 2014].

Satisfiability offers an attractive approach to coloring, in part because it is trivial to encode the problem. In satisfiability, we express problems with Boolean variables $\mathbf{X}$. We say that a literal l is either a variable x or its negation $\overline{x}$. Constraints are disjunctions of literals, written interchangably as sets of literals or as disjunctions, which are satisfied by an assignment if it assigns at least one literal to true. In order to encode graph coloring with satisfiability, one typically relies on *color* variables x_{vi}, where x_{vi} being true means vertex v takes color i. For every edge (uv), there is a binary clause $\overline{x}_{vi} \vee \overline{x}_{ui}$ for every color i. Then, if K is the maximum number of colors, then there is a clause $\bigvee_{1 \leq i \leq K} x_{ui}$. Refinements to this encoding include Van Gelder's log encoding versions, where x_{vj} is true if the j-th bit of the binary encoding of the color taken by vertex v is 1 [Van Gelder, 2008]. How-

*This is an abridged version of "Clause Learning and New Bounds for Graph Coloring" that won the CP 2018 Best Paper Award.

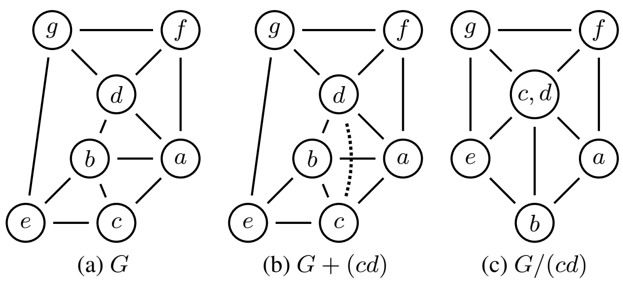


Figure 1: Zykov reccurrence

ever, the use of modern SAT solving techniques like restarting [Gomes *et al.*, 1998; Huang, 2007] and clause learning [Marques-Silva and Sakallah, 1999] are not straightforward to combine with symmetry breaking such as that of van Hentenryck et al. [Van Hentenryck *et al.*, 2003]. They can only be easily combined with starting from an arbitrary coloring to a clique, but that is incomplete. The `color6` solver [Zhou *et al.*, 2014] uses symmetry breaking branching but forgoes restarting to maintain complete symmetry breaking.

On the other hand, the search tree induced by Zykov's deletion-contraction recurrence [Zykov, 1949] has no color symmetry and using the clique number as lower bound is easier and more powerful than in the color variable formulation.

Let $G/(uv)$ be the graph where u and v are *contracted*: the two vertices are identified to a single vertex $r(u) = r(v) = u$, every edge (vw) is replaced by $(r(v)w)$ and self edges are discarded. Conversely, let $G + (uv)$ be the graph where the edge (uv) is added. The Zykov reccurrence is thus:

$$\chi(G) \quad = \quad \min\{\chi(G/(uv)), \chi(G + (uv))\} \tag{1}$$

Indeed, in a minimum coloring of G, either u and v have distinct colors and therefore it is also a coloring of $G + (uv)$, or they have the same color and it is a coloring of $G/(uv)$.

Example 1 *Figure 1 illustrates the Zykov reccurrence. From the graph G in Figure 1a, we obtain the graph $G{+}(cd)$ shown in Figure 1b by adding the edge (cd) and the graph $G/(cd)$ shown in Figure 1c. One of these two graphs has the same chromatic number as G.*

This branching scheme was successfully used in a branch-and-price approach to coloring [Mehrotra and Trick, 1996]. In the context of satisfiability, Schaafsma et al. showed that a clause encoding of Zykov formulation is not efficient [Schaafsma *et al.*, 2009]. For every non-edge (uv), the *edge* variable e_{uv} stands for the decision of contracting the vertices ($e_{uv} = 1$), or adding the edge ($e_{uv} = 0$). A difficulty is that a cubic number of clauses are required, three for every triplet u, v, w, in order to forbid that exactly two of the variables e_{uv}, e_{uw} and e_{vw} are true. This encoding proved too heavy and as a result less efficient than the formulations using color variables. However, Schaafsma et al. introduced a novel and clever way of taking advantage of Zykov's idea: when learning a clause involving color variables, one can compactly encode all symmetric clauses using a single clause that only uses edge variables and propagates the same as if all the symmetric clauses were present.

We propose a constraint programming Zykov-based formulation of coloring in section 2. We use the idea of integrating constraint programming into clause learning satisfiability solvers by simply having each propagator label each pruning or failure by a clausal *reason* or *explanation* [Katsirelos and Bacchus, 2005; Ohrimenko *et al.*, 2007] to alleviate the cost of keeping the edge variables consistent (section 2.1) and to integrate a lower bound based on either cliques (section 2.2) or a more general bound based on Mycielskians (section 2.3). Together with effective branching heuristics (section 2.4), we get a solver that clearly outperforms the state of the art in satisfiability-based coloring (section 3).

2 Clause-learning Approach

In our approach, similar to that of Schaafsma et al., we use a model which leads to the exploration of the tree resulting from application of the Zykov recurrence. We have one Boolean variable e_{uv} for each non-edge of the input graph, that is for every $(uv) \notin E$. When e_{uv} is true, the vertices v and u are contracted, hence assigned the same color, and are separated otherwise, hence assigned different colors. We somewhat abuse notation in the sequel and write clauses using variables e_{uv} even when $(uv) \in E$ and assume that the variable is set to false at the root of the search tree.

With every partial assignment A to the edge variables, we can associate a graph G_A, with $G_\emptyset = G$. For non-empty assignments it is the graph that results from contracting all vertices u, v of G for which A contains e_{uv} and adding an edge between all pairs of vertices u, v of G for which A contains $\overline{e}_{uv}$. When e_{uv} and e_{vw} are both true, this means that we contract u and v and then contract w and $r(v)$ and similarly for false literals. The operation of contracting vertices is associative and commutative, so we get the same graph G_A regardless of the order in which we process the literals in A.

Equality is transitive, so if e_{uv} and e_{vw} are true, then so is e_{uw}. Similarly, if e_{uv} is true and e_{vw} is false, then e_{uw} must also be false. We enforce this using the constraint:

$$\textsc{Triangle}(\{e_{uv} \mid (uv) \notin E\}) \tag{2}$$

GAC can be achieved by a decomposition of size $O(|V|^3)$:

$$(\overline{e}_{uv} \vee \overline{e}_{vw} \vee e_{uw}) \qquad \forall \text{ distinct } u, v, w \in V \tag{3}$$

Enforcing unit propagation on this decomposition therefore takes $O(|V|^3)$ time, amortized over a branch of the search tree. In our implementation, we have opted instead for a dedicated propagator for this, described in section 2.1, whose complexity over a branch is only $O(|V|^2)$.

The model also includes a constraint:

$$\textsc{Coloring}(\{e_{uv} \mid (uv) \notin E\}, k) \tag{4}$$

which is satisfied by colorings with fewer than k colors. This constraint is clearly NP-hard. We describe two incomplete propagators for it in sections 2.2 and 2.3. The first computes either the well known clique lower bound (section 2.2) and the second a novel, stronger, bound (section 2.3). If that bound meets or exceeds k, the propagator fails and produces an explanation. Neither of these bounds is cheap to compute, hence the propagator runs at a lower priority than unit propagation and the $\textsc{Triangle}$ constraint.

2.1 Triangle Consistency Propagation

The propagator for the TRIANGLE constraints works as follows: for each vertex v, we keep a bag $b(v)$ to which it belongs. Initially, $b(v) = \{v\}$ for all v. When we set e_{uv} to true, we set $e_{u'v'}$ to true for all $v' \in b(v), u' \in b(u)$. We also set $e_{u'v'}$ to false for all $v' \in b(v)$, $u' \in N(b(u) \setminus N(b(v))$.[1] Finally, we set $B = b(u) \cup b(v)$ and update $b(v') = B$ for all $v' \in B$. In the case where we set e_{uv} to false, we set $e_{u'v'}$ to false for all $v' \in b(v), u' \in b(u)$.

A small but important optimization is that if the propagator is invoked for e_{uv} becoming true (resp. false) but u and v are already in the same bag (resp. already separated) then it does nothing. This ensures that it touches each non-edge exactly once, hence its complexity is quadratic over an entire branch. This is also optimal, since in the worst case every non-edge must be set either as a decision or by propagation.

This propagator uses the clauses of the decomposition of TRIANGLE as explanations. The mapping from actions that it performs to explanations is fairly straightforward, using the vertices involved in the literal that woke the propagator as "pivots". For example, if $b(v) = \{v, v'\}$, $b(u) = \{u, u'\}$ and it is woken on the literal e_{uv}, it sets $e_{uv'}$ using $(\overline{e}_{vv'} \vee \overline{e}_{uv} \vee e_{uv'})$ as the reason and then $e_{u'v'}$ using $(\overline{e}_{uv'} \vee \overline{e}_{uu'} \vee e_{u'v'})$.

2.2 Clique-based Lower Bound

An important advantage of the edge-variable based model is that computing a lower bound for the current subproblem is as easy as for the entire problem.

In order to find a large clique we use the following greedy algorithm: from an initially empty set of cliques, vertices are explored and added to all the cliques admitting it, or put in a new singleton clique if none admit it. We then pick the largest among these cliques as our lower bound.

If the lower bound meets or exceeds the upper bound k, the propagator reports a conflict. We construct a clausal conflict as follows: each vertex v of the current graph is the result of the contraction of 1 or more vertices of the original graph. In keeping with the notation for the triangle consistency propagator, we call this the *bag* $b(v)$. We arbitrarily pick one vertex $r(v)$ from the bag of each vertex v in the largest clique C, and set the explanation to the clause: $\bigvee_{v,u \in C} e_{r(v)r(u)}$.

2.3 Mycielski-based Bound

Although being a useful bound in practice, the clique number is both hard to compute and gives no guarantees on the quality of the bound. We propose here a new lower bound inspired by *Mycielskian graph*.

Definition 1 (Mycielskian graph [Mycielski, 1955]) *The Mycielskian graph $\mu(G)$=($\mu(V)$,$\mu(E)$) of G =(V,E) is defined as follows:*

- *$\mu(V)$ contains every vertex in V, and $|V| + 1$ additional vertices, constituted of a set $U = \{u_i \mid v_i \in V\}$ and another distinct vertex w.*
- *For every edge $v_iv_j \in E$, $\mu(E)$ contains v_iv_j, v_iu_j and u_iv_j. It also contains all the edges between U and w.*

[1]We write $N(S)$ for $\bigcup_{u \in S} N(u)$.

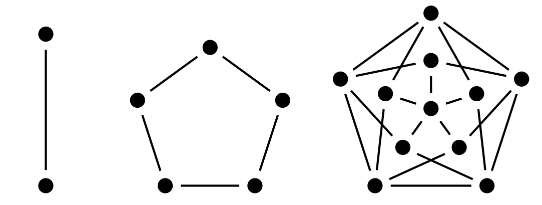

(a) M_2 (b) M_3 (c) M_4

Figure 2: $M_2 = \mu(\emptyset)$, $M_3 = \mu(M_2)$ and $M_4 = \mu(M_3)$

The Mycielskian $\mu(G)$ of a graph G, has the same clique number, however its chromatic number is $\chi(G) + 1$. Indeed, consider a coloring of $\mu(G)$. For any vertex $v_i \in V$, we have $N(v_i) \subseteq N(u_i)$, and therefore we can safely use the same color v_i as for u_i. If follows that at least $\chi(G)$ colors are required for the vertices in U, and since $N(w) = U$, then w requires a $\chi(G) + 1$-th color. Mycielski introduced these graphs to demonstrate that triangle-free graphs can have arbitrarilly large chromatic numbers, hence the clique number does not approximate the chromatic number.

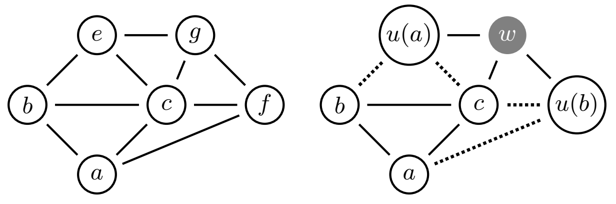
The principle of our bound is a greedy procedure that can discover embedded "pseudo" Mycielskians. Indeed, the class of embedded graphs that we look for is significantly broader than set of "pure" Mycielskians $\{M_2, M_3, M_4, \ldots\}$. First, we look for a partial subgraph. Therefore, trivially, Mycielskians with extra edges also provide valid lower bounds. Moreover, we use as starting point a (potentially large) clique. Finally, the method we propose can also find Mycielskians modulo some vertex contractions. Clearly, those are also valid lower bounds since contracting vertices is equivalent to adding equality constraints to the problem.

Let $N_G(v)$ be the neighborhood of v in the graph G. Suppose that we have a partial subgraph $H = (V_H, E_H)$ of G such that $\chi(H) \geq k$. This can be for example a clique of size k. We define $S_v = \{u \mid N_H(v) \subseteq N_G(u)\}$. Now, suppose that there exists a vertex w with at least one neighbor in every set S_v, i.e., such that $w \in \cap_{v \in V_H} N_G(S_v)$ and let $u(v)$ be any element of S_v such that $u(v) \in N_G(w)$ and $U = \{u(v) \mid v \in V\}$, then:

Lemma 1 (Proof in full version) *Let V' be $V \cup U \cup \{w\}$ and E' be $E \cup \bigcup_{v \in V} N_H(v) \times u(v) \cup \bigcup_{u \in U}\{(u, w)\}$. Then $H' = (V', E')$ is such that $\chi(H') \geq k + 1$.*

Example 2 *Figure 3a shows the graph $G/(cd)$ obtained by contracting vertices c and d in the graph G of Figure 1. Consider the clique $\{a, b, c\}$. We have $S_a = \{a, e\}$, $S_b = \{b, f\}$ and $S_c = \{c\}$. Furthermore, $N_G(\{a, e\}) \cap N_G(\{b, f\}) \cap N_G(\{c\}) = \{b, c, g\} \cap \{a, e, c, g\} \cap \{a, b, e, g, f\} = \{g\}$, from which we can conclude that this graph has chromatic number at least 4. As shown in Figure 3b, the clique $\{a, b, c\}$ can be extended to a Mycielskian with a first layer $U = \{e, c, f\}$ and an extra vertex $w = g$.*

Given a partial subgraph $H = (V_H, E_H)$ of the graph G (with $\chi(H) \geq k$), we greedily extend it into a larger partial subgraph $H' = (V'_H, E'_H)$, following the above principles, and continue extending H' until the rules below do not apply:



(a) $G/(cd)$ (b) $\mu(\{a, b, c\}) = G/(cd)$

Figure 3: Embedded Mycielski

		cdcl			color6			CPLEX			Dsatur		
		opt	ub	lb	opt	ub	lb	opt	ub	lb	opt	ub	lb
DSJ	14	0.07	76.0	30.7	0.07	77.6	28.9	0.07	86.1	29.8	0.00	77.9	27.6
FullI	14	1.00	6.8	6.8	0.21	6.8	5.1	0.86	6.9	6.4	0.00	6.8	4.9
Insert	11	0.27	5.2	2.5	0.36	5.2	2.8	0.36	5.2	3.6	0.00	5.2	2.0
abb	1	1.00	9.0	9.0	0.00	14.0	8.0	0.00	14.0	8.0	0.00	10.0	6.0
ash	3	1.00	4.0	4.0	0.67	4.7	3.7	0.33	5.7	3.3	0.00	4.3	3.0
flat	6	0.00	73.8	11.7	0.00	74.3	10.7	0.00	79.7	10.7	0.00	74.8	9.7
fpsol2	1	1.00	65.0	65.0	0.00	65.0	59.0	1.00	65.0	65.0	1.00	65.0	65.0
inithx	1	1.00	54.0	54.0	0.00	54.0	43.0	1.00	54.0	54.0	1.00	54.0	54.0
latin	1	0.00	116.0	90.0	0.0	125.0	90.0	0.00	159.0	90.0	0.00	129.0	90.0
le450	10	0.50	15.2	13.0	0.10	15.6	13.0	0.30	19.1	13.0	0.20	16.0	11.7
miles	5	1.00	34.8	34.8	0.00	36.4	33.4	1.00	34.8	34.8	1.00	34.8	34.8
mug	4	1.00	4.0	4.0	1.00	4.0	4.0	1.00	4.0	4.0	0.00	4.0	3.0
myciel	5	1.00	6.0	6.00	0.80	6.00	4.80	0.60	6.00	5.00	0.00	6.00	2.00
qg	4	0.75	66.00	57.5	0.25	63.2	57.5	0.25	72.5	57.5	0.25	59.5	57.5
queen	13	0.46	12.08	10.8	0.00	15.9	10.6	0.38	12.5	10.8	0.23	12.0	10.6
school	1	1.00	14.0	14.0	0.00	26.0	14.0	1.00	14.0	14.0	0.00	14.0	13.0
wap	8	0.12	46	41.2	0.00	47.6	40.0	0.00	51.1	40.0	0.00	48.0	30.4
will	1	1.00	7.0	7.0	0.00	10.0	6.0	1.00	7.0	7.0	0.00	7.0	6.0

Table 1: Comparison with the state of the art: by benchmark

At each iteration, we compute the sets S_v and the set $W = \bigcap_{v \in V_H} N_G(S_v)$ of nodes with at least one neighbor in every S_v. The procedure stops if W is empty. Otherwise, we pick any element $w \in W$ and for every vertex $v \in V_H$, we extract the corresponding vertex $u(v)$ in S_v which is adjacent to w. The graph H can then be extended as shown in Lemma 1.

The computed bound k is equal to $\chi(H)$ plus the number of successful iterations.

Complexity. We need $O(|V_H| \times |V|)$ bitset operations for computing the sets S_v, and $O(|V_H|^2)$ time for computing the vertices $u(v)$ and extending the graph. Typically, the number of iterations is very small. In the worst case, it cannot be larger than $\log |V|$ since the number of vertices in H is (more than) doubled at each iteration. It follows that there are at most $2|V|$ iterations, and therefore, the worst case time complexity is $O(|V|^2)$ bitset operations (hence $O(|V|^3)$ time).

Explanation. Similarly to the clique based lower bound, the explanations that we produce here correspond to the set of all edges in the graph H: $\bigvee_{(v,u) \in E_H} e_{uv}$

2.4 Branching Heuristic

In order to get behavior similar to that of Brelaz' heuristic in the edge variable model, we proceed as follows: we pick a maximal clique C in the current graph. We pick the vertex v that maximises $|N(v) \cap C|$, breaking ties by highest $|N(v) \setminus C|$, and an arbitrary vertex $u \in C \setminus N(v)$ We then set e_{uv} to true. The maximal clique implicitly defines a coloring, which can be used to compute a saturation degree. If the assignments e_{uv} are refuted for all $u \in C$, then v is adjacent to all vertices in C and so $C \cup \{v\}$ is a larger clique, which corresponds to using a new color in Brelaz' heuristic.

3 Experimental Evaluation

We implemented our approach, `cdcl`, using MiniCSP[2] as the underlying solver.[3] It uses Brelaz branching, and applies the Mycielskian bound only after a failure.

We compared with the state-of-art SAT-based solver `color6` [Zhou *et al.*, 2014], a very efficient clause-learning algorithm for graph coloring proposed recently by Zhou et al. Similarly to our approach, it is based on a SAT solver (namely `zChaff`), however, it uses the color-based formulation. It was shown to outperform the state of the art on many instances. As `color6` solves satisfiability instances only (testing whether a coloring with a specific number of colors exists), we implemented a branch-and-bound wrapper on top of it, denoted `color6`. We used the lower and upper bounds computed by our approach (respectively the maximal clique algorithm described in section 2.2 and a greedy run of Brelaz) as initial bounds for `color6`.

Moreover, we also compared with an implementation of `Dsatur` by Trick, and an integer programming formulation in `CPLEX`. The model we used for `CPLEX` is the trivial one using binary color variables (one for each vertex and each color), and one binary inequality per edge. However, observe that `CPLEX` actually computes maximal cliques in its preprocessing, so providing it with clique inequalities would have been useless. Moreover, we initialized the upper bound with the same method as for `color6`, and also arbitrarily fixed the colors of one maximal clique in order to break symmetries.

We used 125 benchmark instances from Trick's graph coloring webpage (http://mat.gsia.cmu.edu/COLOR/color.html) [Trick, 2002]. In the subsequent tables, however, we omit 22 of these instances that were trivial for every approach we used (i.e., that solved by every method to optimality). Every method was run with a time limit of one hour and a memory limit of 3.5GB on 4 nodes, each with 35 Intel Xeon CPU E5-2695 v4 2.10GHz cores running Linux Ubuntu 16.04.4.

The results in Tables 1 are class averages and the number of instances in each class is given next to the class name. We show the ratio of instances for which a proof of optimality was found ('opt'), as well as the average upper bound ('ub') and lower bound ('lb'), for every method. The best results for each criterion are highlighted using colors. `cdcl` is better on all but three classes of instances: `Insert`, `qg` (quasigroup) and `queen`. Moreover, it finds the same coloring as the other methods in the `Insert` class, and computes strictly more proofs of optimality than other solvers in the the two other classes. Finally, on many classes it is strictly better than the second best solver, considering at least one criterion.

Table 2 shows results aggregated across all instances. We

[2]Sources at: https://bitbucket.org/gkatsi/minicsp.

[3]Sources at: https://bitbucket.org/gkatsi/gc-cdcl/src/master/.

method	optimal	ub		lb		gap (ub)	gap (lb)
	avg	gavg	avg	gavg	avg	avg	avg
`cdcl`	0.534	15.247	30.107	10.790	18.689	0.091	0.225
`CPLEX`	0.417	16.503	33.388	10.886	18.379	0.401	0.256
`color6`	0.194	16.314	31.233	10.040	17.748	0.320	0.472
`Dsatur`	0.126	15.506	30.495	8.754	16.524	0.145	0.725

Table 2: Comparison with the state of the art: global results

report the average ratio of instance proven optimal ('optimal') in the first column. Then in the second to the fifth columns, we report the arithmetic ('avg') and geometric averages ('gavg') for both the lower and upper bounds. Finally, we report the *mean normalised gap to the best upper bound*, and to the best lower bound.

Overall, `cdcl` is best for all criteria. `CPLEX` is third best for the number of optimality proofs. Although it requires a lot of memory, and is very poor in terms of solution quality, `CPLEX` often gives good lower bounds. This is not so surprising since the linear relaxation is quite potent on this formulation. It should be noted, however, that in many cases it was not able to improve on the initial bounds provided to the model. Finally, `Dsatur`, even though extremely simple, is still a very good method to actually find small colorings and is a close second best for the upper bound.

4 Conclusions

We have presented a CP/SAT hybrid approach to graph coloring. The approach uses a new, sophisticated, lower bound that generalizes the clique bound and is inspired by Mycielskian graphs. We combined it with clause learning and effective primal heuristics for coloring to get a solver that outperforms the previous state of the art in satisfiability-based coloring, constraint programming based coloring, as well as a MIP model of the problem. The main disadvantage of the approach is that it requires one Boolean variable for each non-edge of the graph and hence cannot scale to large sparse graphs.

References

[Aardal *et al.*, 2007] Karen I Aardal, Stan PM Van Hoesel, Arie MCA Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007.

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

[Chaitin *et al.*, 1981] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Comput. Lang.*, 6(1):47–57, January 1981.

[Gomes *et al.*, 1998] Carla Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-1998)*, pages 431–438, 1998.

[Huang, 2007] Jinbo Huang. The Effect of Restarts on the Efficiency of Clause Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, 2007.

[Katsirelos and Bacchus, 2005] George Katsirelos and Fahiem Bacchus. Generalized Nogoods in CSPs. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 390–396, 2005.

[Marques-Silva and Sakallah, 1999] Joao P. Marques-Silva and Karem A. Sakallah. GRASP—a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[Mehrotra and Trick, 1996] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[Mycielski, 1955] Jan Mycielski. Sur le coloriage des graphes. In *Colloq. Math*, volume 3, pages 161–162, 1955.

[Ohrimenko *et al.*, 2007] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = Lazy Clause Generation. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP-2007)*, pages 544–558, 2007.

[Park and Lee, 1996] Taehoon Park and Chae Y Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research society of Japan*, 39(2):258–265, 1996.

[Schaafsma *et al.*, 2009] Bas Schaafsma, Marijn Heule, and Hans van Maaren. Dynamic Symmetry Breaking by Simulating Zykov Contraction. In *12th International Conference on Theory and Applications of Satisfiability Testing (SAT-2009)*, pages 223–236, 2009.

[Trick, 2002] Michael A. Trick, editor. *Computational Symposium on Graph Coloring and its Generalizations (COLOR-2002)*, 2002.

[Van Gelder, 2008] Allen Van Gelder. Another Look at Graph Coloring via Propositional Satisfiability. *Discrete Appl. Math.*, 156(2):230–243, 2008.

[Van Hentenryck *et al.*, 2003] Pascal Van Hentenryck, Magnus Ågren, Pierre Flener, and Justin Pearson. Tractable Symmetry Breaking for CSPs with Interchangeable Values. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 277–282, 2003.

[Walsh, 2008] Toby Walsh. Breaking Value Symmetry. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI-2008)*, pages 880–887, 2008.

[Zhou *et al.*, 2014] 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.

[Zykov, 1949] Alexander A. Zykov. On some properties of linear complexes. *Mat. Sb. (N.S.)*, 24(66)(2):163–188, 1949.