New Integer Linear Programming Models for the Vertex Coloring Problem

Adalat Jabrayilov(✉) and Petra Mutzel

Department of Computer Science, TU Dortmund University, Dortmund, Germany
{adalat.jabrayilov,petra.mutzel}@tu-dortmund.de

Abstract. The vertex coloring problem asks for the minimum number of colors that can be assigned to the vertices of a given graph such that each two neighbors have different colors. The problem is NP-hard. Here, we introduce new integer linear programming formulations based on partial-ordering. They have the advantage that they are as simple to work with as the classical assignment formulation, since they can be fed directly into a standard integer linear programming solver. We evaluate our new models using Gurobi and show that our new simple approach is a good alternative to the best state-of-the-art approaches for the vertex coloring problem. In our computational experiments, we compare our formulations with the classical assignment formulation and the representatives formulation on a large set of benchmark graphs as well as randomly generated graphs of varying size and density. The evaluation shows that the partial-ordering based models dominate both formulations for sparse graphs, while the representatives formulation is the best for dense graphs.

Keywords: Graph coloring · Vertex coloring
Integer linear programming

1 Introduction

The vertex coloring problem (VCP) belongs to the classical optimization problems. This problem asks for the minimum number of colors, which are assigned to the vertices of a graph such that no two adjacent vertices get the same color. The minimum number of colors is called *chromatic number* and denoted by χ. Computing the chromatic number of a graph is NP-hard [9]. Since the vertex coloring problem has many applications, e.g., register allocation, scheduling, frequency assignment and timetabling, there is a vast amount of literature on this problem (see, e.g., [18] for a survey). However, in contrast to other classical combinatorial optimization problems such as the Travelling Salesman Problem, where large instances can be solved to optimality, this is not true for the VCP.

There are two main lines of research based on integer linear programming (ILP) formulations for VCP. The natural formulation introduces binary variables that assign colors to vertices. A vertex v in the graph $G = (V, E)$ is assigned color i iff the corresponding binary variable $x_{v,i}$ gets value 1. This *assignment formulation* has the advantage that it is simple and easy to use. Since the number of variables and constraints of this ILP model is polynomial in $|V|$, it can be fed directly into a (commercial) integer linear programming solver such as *SCIP* [1], *LP_Solve*, *Cplex*, or *Gurobi*. Due to the inherent symmetry in the model (the colors are not distinguishable) only small instances can be solved to optimality. However, additional constraints can be added by which the symmetry can be reduced. The second approach has been suggested by Mehrotra and Trick [19] and is based on the observation that each color class defines an independent set (no two vertices in the set are adjacent) in the graph. The variables correspond to independent sets and the ILP model searches for the minimum number of these independent sets that cover the graph. Since the number of independent sets can be of exponential size, the solution approach is based on column generation. Solution algorithms based on this *Set Covering Formulation* are of complex nature. Mehrotra and Trick [19] suggest a branch-and-price algorithm for solving the ILP model. Both ILP formulations have been studied and improved by additional ideas in the literature leading to complex branch-and-cut algorithms using additional classes of constraints, special branching schemes, separation procedures, and special procedures for providing good upper bounds. The computational studies in the literature show that none of the ILP models dominates the other one.

ILP formulations based on partial-ordering have shown to be practically successful in the area of graph drawing [14]. Here, we suggest a new ILP formulation based on partial-ordering for the vertex coloring problem. It has the advantage that it is as simple to work with as the assignment formulation, since it is of polynomial size and can be fed directly into a standard integer linear programming solver. We further suggest a hybrid ILP formulation which combines the advantages of the assignment formulation with those of the partial-ordering model.

We evaluate the new models using the ILP solver *Gurobi* and show that our new simple approaches dominate the assignment formulation on the tested benchmark sets and are a good alternative to the best state-of-the-art approaches for the vertex coloring problem. We also present the first experimental comparison with the representatives formulation which has been suggested by Campelo et al. [3,4]. Since it introduces variables for every pair of non-adjacent vertices, this formulation seems to be advantageous for dense graphs. Our computational results support this observation.

2 State-of-the-Art

Eppstein [8] has shown that it is possible to solve the vertex coloring problem by enumerating all maximal independent sets in the graph in time $O((4/3 + 3^{4/3}/4)^{|V|})$ which is about $2.4150^{|V|}$. In practice, the successful approaches are much faster than that. There are two main directions followed by exact algorithms based on ILP models for the problem: the ILP-based assignment model

(Sect. 2), and the ILP-based set covering formulation. There are also a lot of experimental evaluations of these methods. Altogether they have not shown a superiority of one of these lines of research. From those, the assignment model is the simplest one, since it can directly be fed into a standard ILP-solver. Another simple ILP formulation is the so-called representatives ILP model. It seems that there is no experimental evaluation concerning this model. In the literature there also exist alternative ILP models and alternative approaches (e.g., based on Constraint Programming [11]). However, the aim of this work is to concentrate on simple ILP formulations that are competitive with the best state-of-the-art approaches. There is also a vast literature on heuristic approaches. For a detailed overview of heuristic and exact approaches, see the survey by Malaguti and Toth [18] and Burke et al. [2].

Assignment-Based ILP Model. The classical ILP model for a graph $G = (V, E)$ is based on assigning color i to vertex $v \in V$. For this, the assignment variables $x_{v,i}$ are defined for each vertex v and color $i \in \{1, \dots, H\}$ with $x_{v,i} = 1$ if vertex v is assigned to color i and $x_{v,i} = 0$ otherwise. H is an upper bound of the number of colors (e.g., the result of a heuristic) and is at most $|V|$. For modelling the objective function, an additional binary variable w_i for each $i \in \{1, \dots, H\}$ is needed which gets value 1 iff color i is used in the coloring. The model is given by:

$$\text{(ASS-S)}\ \min \textstyle\sum_{1 \le i \le H} w_i \tag{1}$$

$$\text{s.t.}\ \textstyle\sum_{i=1}^{H} x_{v,i} = 1 \quad \forall v \in V \tag{2}$$

$$x_{u,i} + x_{v,i} \le w_i \ \forall (u, v) \in E,\ i = 1, \dots, H \tag{3}$$

$$x_{v,i}, w_i \in \{0, 1\} \ \forall v \in V,\ i = 1, \dots, H \tag{4}$$

The objective function minimizes the number of used colors. Equations (2) ensure that each vertex receives exactly one color. For each edge there is a constraint (3) making sure that adjacent vertices receive different colors. This model has the advantage that it is simple and easy to use. It can be easily extended to generalizations and/or restricted variants of the graph coloring problem. Since the number of variables is quadratic in $|V|$ (bounded by $H(|V| + 1)$) and the number of constraints is cubic in $|V|$ (exactly $|V| + H|E| = O(|V||E|)$ constraints of type 2 and 3), it can directly be used as input for a standard ILP solver. The main drawback of this model is the inherent symmetry, since there are $\binom{H}{\chi}$ possibilities to select χ from H colors thus leading to exponentially (in the number of colors) many equivalent solutions. In order to remove this type of symmetry, Mendez-Diaz and Zabala [20,21] suggest to extend (ASS-S) through the following set of constraints:

$$w_i \le \sum_{v \in V} x_{v,i} \qquad i = 1, \dots, H \tag{5}$$

$$w_i \le w_{i-1} \qquad i = 2, \dots, H \tag{6}$$

We call this extended model (ASS). These constraints ensure that the color i is only assigned to some vertex, if color $i-1$ is already assigned to another one. Moreover, they present several sets of constraints that arose from their studies of the associated polytope. In order to solve the new strengthened ILP model, they developed a branch-and-cut algorithm.

Representatives ILP Model. A vertex coloring divides the vertices into disjoint color classes. Campêlo et al. [3,4] suggested a model in which each color class is represented by exactly one vertex. For this, they suggest to introduce a binary variable x_{uv} for each non-adjacent pair of vertices $u, v \in V$ which is 1 if and only if the color of v is represented by u. Additional binary variables x_{uu} indicate if u is the representative of its color class. Let $\bar{N}(u)$ be the set of non-adjacent vertices to u. The constraints are as follows:

$$\text{(REP)} \quad \min \textstyle\sum_{u \in V} x_{uu} \tag{7}$$

$$\textstyle\sum_{u \in \bar{N}(v) \cup v} x_{uv} \geq 1 \ \forall v \in V \tag{8}$$

$$x_{uv} + x_{uw} \leq x_{uu} \quad \forall u \in V,\ \forall e = (v, w) \in G[\bar{N}(u)] \tag{9}$$

$$x_{uv} \in \{0,1\} \qquad \forall \text{ non-adjacent vertex pairs } \ u, v \ \text{ or } \ u = v \tag{10}$$

Inequalities (8) require that for any vertex $v \in V$, there must exist a representative which can be v itself or some vertex from $\bar{N}(v)$. Inequalities (9) state that a vertex u cannot be the representative for both endpoints of an edge (v, w) and that in the case that $x_{uv} = 1$ (u is the representative of vertex v) also the variable x_{uu} must take value 1. The advantage of this model is its simplicity and its compactness. It has exactly $|\bar{E}| + |V|$ variables and up to $|V| + |V||E|$ many constraints, where $\bar{E}$ is the set of non-adjacent vertex pairs of $G = (V, E)$, i.e., $\bar{E} = (V \times V) \setminus E$. With growing density of the graphs, the number of constraints increases but the number of variables decreases and converges towards $|V|$. In [3], Campelo et al. mention that the symmetry in this model may lead to problems with branch-and-bound based solvers. The reason for this lies in the fact that within a color class any of the vertices in this class can be the representative. In order to circumvent this, the authors define an ordering on the vertices and require that in each color class only the vertex with the smallest number is allowed to be the representative of this class. The ILP model arising from this requirement is called the *asymmetric representatives formulation* (AREP). The authors [3] study the polytopes associated with both representative formulations. They suggest to add constraints based on cliques, odd-holes, anti-holes, wheels, and independent sets in order to strengthen the model. Moreover, they provide a comparison with the set covering based formulation. In [5], Campos et al. study the asymmetric representatives formulation and the corresponding polytope. Their results lead to complete characterizations of the associated polytopes for some specific graph classes. Up to our knowledge, no computational experiments have been published in the literature.

3 Partial-Ordering Based ILP Models

3.1 A Pure Partial-Ordering Based ILP Model: POP

Our new binary model considers the vertex coloring problem as a partial-ordering problem (POP). We assume that the H colors $(1, \ldots, H)$ are linearly ordered. Instead of directly assigning a color to the vertices, we determine a partial order of the union of the vertex set and the set of ordered colors. For this, we determine the relative order of each vertex with respect to each color in the color ordering. More specific: for every color i and every vertex $v \in V$ our variables provide the information if v is smaller or larger than i. We denote these relations by $v \prec i$ or $v \succ i$, resp. In other words, the colors and the vertices build a partially ordered set in which all pairs of the form (v, i) with $v \in V, i = 1, \ldots, H$ are comparable. We define the following POP variables:

$$\forall v \in V,\ i = 1, \ldots, H : \quad y_{i,v} = \begin{cases} 1 & v \succ i \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall v \in V,\ i = 1, \ldots, H : \quad z_{v,i} = \begin{cases} 1 & v \prec i \\ 0 & \text{otherwise.} \end{cases}$$

If vertex v has been assigned to color i, then v is neither smaller nor larger than i and we have $y_{i,v} = z_{v,i} = 0$. The connection with the assignment variables x from the (ASS) model is as follows:

$$x_{v,i} = 1 - (y_{i,v} + z_{v,i}) \qquad \forall v \in V,\ i = 1, \ldots, H \tag{11}$$

We select an arbitrary vertex $q \in V$ and formulate our new binary program:

$$\begin{array}{llll}
\text{(POP)} & \min 1 + \sum_{1 \le i \le H} y_{i,q} & & (12) \\
\text{s.t.} & z_{v,1} = 0 & \forall v \in V & (13) \\
 & y_{H,v} = 0 & \forall v \in V & (14) \\
 & y_{i,v} - y_{i+1,v} \ge 0 & \forall v \in V,\ i = 1, \ldots, H-1 & (15) \\
 & y_{i,v} + z_{v,i+1} = 1 & \forall v \in V,\ i = 1, \ldots, H-1 & (16) \\
 & y_{i,u} + z_{u,i} + y_{i,v} + z_{v,i} \ge 1 & \forall (u,v) \in E,\ i = 1, \ldots, H & (17) \\
 & y_{i,q} - y_{i,v} \ge 0 & \forall v \in V,\ i = 1, \ldots, H-1 & (18) \\
 & y_{i,v}, z_{v,i} \in \{0,1\} & \forall v \in V,\ i = 1, \ldots, H & (19)
\end{array}$$

Lemma 1. *The integer linear programming formulation (POP) is correct: Any feasible solution of the ILP corresponds to a feasible vertex coloring and the value of the objective function corresponds to the chromatic number of G.*

Proof. All original vertices need to be embedded between the colors 1 and H. Constraints (13) and (14) take care of this. By transitivity, a vertex that is larger than color $i+1$ is also larger than i (constraints (15)). Constraints (16) express that each vertex v is either larger than i (i.e. $y_{i,v} = 1$) or smaller than $i+1$ and not both. These constraints jointly with constraints (15) ensure that each vertex will be assigned to exactly one color, i.e. there is no color pair $i \neq j$ with $y_{i,v} = z_{v,i} = 0$ and $y_{j,v} = z_{v,j} = 0$. We show this by contradiction. Let $y_{i,v} = z_{v,i} = 0$. In the case $j < i$, as $z_{v,i} = 0$ we have $y_{i-1,v} = 1$ by (16). Therefore we have $y_{j,v} = 1$ for each $j \leq i-1$ by (15) which is a contradiction to $y_{j,v} = 0$. In the case $j > i$, as $y_{i,v} = 0$ we have $y_{k,v} = 0$ for each $k \geq i$ by (15). Therefore we have $z_{v,k+1} = 1$ by (16) leading to $z_{v,j} = 1$ for each $j \geq i+1$ which is a contradiction to $z_{v,j} = 0$. Constraints (17) prevent assigning the same color i to two adjacent vertices u and v. Constraints (18) take care of the fact that our chosen vertex q will be assigned to the largest chosen color. So if q is not larger than color i then this will be true for all other vertices $v \in V \setminus \{q\}$. Because of this constraint, the objective function indeed minimizes the number of assigned colors since it sums up the number of colors smaller than q. In order to get the number of chosen colors, we need to add one for the color assigned to q. We say that q *represents the chromatic number* of G.

Comparison with the assignment model (POP) has $2 \cdot H|V|$ binary variables and about $4|V|H + 2|V| + |E|H$ constraints. Notice that the Eqs. (13), (14) and (16) can be used to eliminate $(H+1)|V|$ variables. Hence the reduced model has $(H-1) \cdot |V|$ variables, and thus $H + V$ variables less than (ASS), which has $H(|V|+1)$ variables.

Mendez-Diaz and Zabala [20] mention that the classical branching rule (to branch on fractional assignment variables by setting them to 1 in one subproblem and to 0 in another subproblem) produces quite unbalanced enumeration trees. This is the case because of setting $x_{v,i} = 1$ implies $x_{v,j} = 0$ for all $j \neq i$, while setting $x_{v,i} = 0$ does not provide any further information. The model (POP) does not have this problem, since setting a POP-variable $y_{i,v} = 0$ implies $y_{j,v} = 0$ for all j with $j > i$ and setting $y_{i,v} = 1$ implies $y_{j,v} = 1$ for all j with $j < i$ because of (15).

As already discussed, the original (ASS-S) model has inherent symmetries, which can be resolved by additional constraints leading to the (ASS) model. In the new (POP) model, this type of symmetry does not occur.

3.2 A Hybrid Partial-Ordering Based ILP Model: POP2

Our second ILP formulation is a slight modification of the first model and can be seen as a hybrid of the models (POP) and (ASS). It is the consequence of the observation that with growing density the (POP) constraint matrix contains more nonzero elements than the (ASS) constraint matrix. This is due to the constraints (17), which are responsible for the valid coloring of adjacent vertices, and contain twice as many nonzero coefficients as the corresponding (ASS) constraints (3). Therefore, we substitute (17) by (11) and the following constraints:

$$x_{u,i} + x_{v,i} \leq 1 \qquad \forall (u,v) \in E,\ i = 1, \ldots, H. \tag{20}$$

This reduces the number of nonzero coefficients from $4|E|H$ to $2|E|H + 3|V|H$ giving a reduction ratio of about two in dense graphs. Although we added $|V|H$

new assignment variables, the dimension of the problem remains unchanged, since the new variables directly depend on the POP-variables by equality (11).

4 Computational Experiments

In our experiments we are interested in answering the following questions:

– (H1): Do our new partial-ordering based ILP formulations dominate the classical assignment ILP model (ASS) on a set of benchmark instances?
– (H2): Does one of the two partial-ordering models dominate the other one?
– (H3): How do the simple models behave compared to the state-of-the-art algorithms on a benchmark set of graphs?
– (H4): Does (AREP) dominate the other approaches on dense instances?

4.1 Implementation

The preprocessing techniques (a)–(d) are widely used (e.g., [11,12,17,18,20,21]):

(a) A vertex u is *dominated* by vertex v, $v \neq u$, if the neighborhood of u is a subset of the neighborhood of v. In this case, the vertex u can be deleted, the remaining graph can be colored, and at the end u can get the color of v.
(b) To reduce the number of variables we are interested in getting a small upper bound H for the number of needed colors.
(c) Since any clique represents a valid lower bound for the vertex coloring problem one can select any maximal clique and precolor it.
(d) In the case of equal lower and upper bounds the optimal value has been found, hence no ILP needs to be solved.

We extended (c) as follows:

(e) In (ASS), (POP), and (POP2) we can fix more variables if we try to find the clique Q with $\max(|Q|H + |\delta(Q)|)$, where $\delta(Q) := \{(u,v) \in E\colon |\{u,v\} \cap Q| = 1\}$. The first term $|Q|H$ is due to the fact that we can fix H variables for each vertex in Q. After precoloring of some vertex $u \in Q$, the neighbors v of u cannot receive the same color as u, e.g. if the assignment variable $x_{u,i} = 1$ then $x_{v,i} = 0$. Hence we can fix one variable for each edge $(u,v) \in \delta(Q)$.

To represent the chromatic number in (POP) and (POP2), we pick any vertex q from the clique Q found in the preprocessing. The remaining vertices from the clique are precolored with colors $1, \cdots, |Q| - 1$.

We have implemented the simple models (ASS), (POP), (POP2), (REP) and (AREP) using the Gurobi-python API. To find large maximal cliques in (c) and (e), we used a heuristic from the python library http://networkx.readthedocs.io. The source codes are available on our benchmark site[1]. As mentioned in Subsect. 3.1, in our implementation of (POP) and (POP2) we used (13), (14) and (16) and eliminated all z variables.

[1] https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks.

4.2 Test Setup and Benchmark Set of Graphs

To solve the models, we used the Gurobi 6.5 single-threadedly on Intel Xeon E5-2640, 2.60 GHz, with 64 GB of memory and running Ubuntu Linux 14.04. The `dfmax` benchmark [7], which is used in the DIMACS challenge to compare the results obtained with different machines, needs on our machine 5.54 s for `r500.5`.

We considered two benchmark sets, which are available at [15]. To compare the new approach with the state-of-the-art algorithms from the literature, we used subsets of the DIMACS [24] benchmark sets. From 119 DIMACS graphs we have chosen the hard instances according to the Google benchmark site [10] and the `GPIA` graphs, which are obtained from a matrix partitioning problem to determine sparse Jacobian matrices. The second benchmark set consists of 340 randomly generated graphs $G(n,p)$, which have n vertices and an edge between each vertex pair with probability p. This set contains also two subsets:

set70: 100 instances: 20 graphs with $n = 70$ and $p = 0.1, 0.3, 0.5, 0.7, 0.9$.
sparse100: 240 instances: 20 graphs for each $n = 80, 90, 100$ and for each $p = 0.1, 0.15, 0.2, 0.25$.

4.3 Experimental Evaluation

Table 1 shows the results for the DIMACS benchmark set. The new approach is compared with the assignment and the representatives models and with other state-of-the-art algorithms [6,17,20,21] from the literature. The table contains 37 of the 68 hard instances, which can be solved by at least one of the considered algorithms. Columns 1–3 show the instance names and sizes. Column 4 describes the hardness of the instances according to the Google site [10]. Columns 5–16 display the lower and upper bounds as well as the running times of the simple models (AREP), (POP), (POP2) and (ASS) that have been obtained within a time limit of one hour by the ILP-solver. An entry tl indicates that the time limit is reached. The times are provided in seconds for solving the reduced ILPs after prepocessing. The preprocessing is done with python and took a few seconds for most instances and not more than one minute. Columns 5–7 show the results of (AREP) with preprocessings (a)–(d). (AREP) does not need (b) directly, but indirectly due to (d). Columns 8–16 show appropriate results for (POP), (POP2) and (ASS). Since (e) can reduce the number of assignment and POP variables, we implemented (ASS), (POP) and (POP2) also with (e) instead of (c). While both versions (ASS)+(c) and (ASS)+(e) solved 21 instances, the average runtime of solved instances by (ASS)+(e) (171.42 s) was smaller than (ASS)+(c) (405.02 s). Also for (POP) and (POP2) it turned out that (e) is better than (c). Table 1 displays the results of (POP), (POP2), (ASS) corresponding to (e) only (due to space restrictions). Since all the solved instances in the recent paper [6] and by (REP) can be solved by (AREP), the table displays the results of the latter only. Note, that the instance `4-Insertions_3` (taking 8 h by [6]) exceeds the 1 h time limit taking 3641 s by (AREP).

Table 1. Results for the hard instances from DIMACS benchmark set

instance	$\|V\|$	$\|E\|$	class	AREP			POP			POP2			ASS+(e)			[20]	[21]	[17]
				lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time			
1-FullIns_4	93	593	NP-m	5.0	5.0	0.57	5.0	5.0	0.01	5.0	5.0	0.02	5.0	5.0	0.01	0.1		tl
1-FullIns_5	282	3247	NP-?	6.0	6.0	279.46	6.0	6.0	7.42	6.0	6.0	2.04	6.0	6.0	2.71	tl		tl
2-FullIns_4	212	1621	NP-m	6.0	6.0	0.64	6.0	6.0	0.04	6.0	6.0	0.03	6.0	6.0	0.02	tl	4	tl
2-FullIns_5	852	12201	NP-?	6.0	7.0	tl	7.0	7.0	6.19	7.0	7.0	86.48	7.0	7.0	17.66	tl		tl
3-FullIns_3	80	346	NP-m	6.0	6.0	0.01	6.0	6.0	0.00	6.0	6.0	0.00	6.0	6.0	0.00	0.1		2.9
3-FullIns_4	405	3524	NP-?	7.0	7.0	2.07	7.0	7.0	0.05	7.0	7.0	0.04	7.0	7.0	0.05	tl		tl
3-FullIns_5	2030	33751		7.0	8.0	tl	8.0	8.0	15.19	8.0	8.0	39.31	8.0	8.0	32.15	tl		tl
4-FullIns_3	114	541	NP-m	7.0	7.0	0.01	7.0	7.0	0.01	7.0	7.0	0.01	7.0	7.0	0.00	3		3.4
4-FullIns_4	690	6650	NP-?	8.0	8.0	1.27	8.0	8.0	0.08	8.0	8.0	0.04	8.0	8.0	0.04	tl		tl
4-FullIns_5	4146	77305		7.0	9.0	tl	9.0	9.0	11.65	9.0	9.0	19.22	9.0	9.0	93.98	tl		tl
4-Insertions_3	79	156	NP-m	3.0	4.0	tl	4.0	4.0	11.71	4.0	4.0	19.18	4.0	4.0	64.10	4204		tl
5-FullIns_3	154	792	NP-m	8.0	8.0	0.01	8.0	8.0	0.01	8.0	8.0	0.01	8.0	8.0	0.00	20		4.6
5-FullIns_4	1085	11395	NP-?	9.0	9.0	3.73	9.0	9.0	0.08	9.0	9.0	0.08	9.0	9.0	0.06	tl		tl
ash608GPIA	1216	7844	NP-m	$-\infty$	$+\infty$	tl	4.0	4.0	42.50	4.0	4.0	62.42	4.0	4.0	854.60	692		2814.8
ash958GPIA	1916	12506	NP-m	$-\infty$	$+\infty$	tl	4.0	4.0	109.15	4.0	4.0	122.92	4.0	6.0	tl	tl	4236	tl
DSJC125.5	125	3891	NP-h	14.0	20.0	tl	11.0	20.0	tl	13.0	22.0	tl	13.0	21.0	tl	tl		18050.8
DSJC125.9	125	6961	NP-h	44.0	44.0	1.13	35.0	50.0	tl	42.0	44.0	tl	42.0	45.0	tl	tl		3896.9
DSJC250.9	250	27897	NP-h	72.0	72.0	1367.70	39.0	$+\infty$	tl	45.0	$+\infty$	tl	40.0	89.0	tl	tl		tl
DSJR500.1c	500	121275	NP-h	85.0	85.0	0.10	77.0	$+\infty$	tl	83.0	86.0	tl	78.0	$+\infty$	tl	tl		288.5
DSJR500.5	500	58862	NP-h	$-\infty$	$+\infty$	tl	115.0	$+\infty$	tl	122.0	122.0	551.92	115.0	$+\infty$	tl	tl		342.2
le450_15a	450	8168	NP-m	15.0	21.0	tl	15.0	16.0	tl	15.0	15.0	690.84	15.0	15.0	963.46	tl		0.4
le450_15b	450	8169	NP-?	15.0	19.0	tl	15.0	15.0	3473.07	15.0	15.0	827.73	15.0	15.0	1283.05	tl		0.2
le450_15c	450	16680	NP-?	$-\infty$	450.0	tl	15.0	$+\infty$	tl	15.0	$+\infty$	tl	15.0	25.0	tl	tl		3.1
le450_15d	450	16750	NP-?	$-\infty$	450.0	tl	15.0	26.0	tl	15.0	26.0	tl	15.0	$+\infty$	tl	tl		3.8
le450_25c	450	17343	NP-?	$-\infty$	450.0	tl	25.0	30.0	tl	25.0	31.0	tl	25.0	$+\infty$	tl	tl		1356.6
le450_25d	450	17425	NP-?	25.0	450.0	tl	25.0	30.0	tl	25.0	31.0	tl	25.0	$+\infty$	tl	tl		66.6
le450_5a	450	5714	NP-?	$-\infty$	450.0	tl	5.0	9.0	tl	5.0	5.0	25.65	5.0	5.0	62.12	tl		0.3
le450_5b	450	5734	NP-?	$-\infty$	450.0	tl	5.0	8.0	tl	5.0	5.0	171.03	5.0	5.0	194.73	tl		0.2
mug100_1	100	166	NP-m	4.0	4.0	0.62	4.0	4.0	0.38	4.0	4.0	0.13	4.0	4.0	0.20	60		14.4
mug100_25	100	166	NP-m	4.0	4.0	0.71	4.0	4.0	0.72	4.0	4.0	0.48	4.0	4.0	0.49	60		12
qg.order40	1600	62400	NP-m	$-\infty$	$+\infty$	tl	40.0	45.0	tl	40.0	40.0	594.96	40.0	46.0	tl	tl		2.9
qg.order60	3600	212400	NP-?	$-\infty$	$+\infty$	tl	60.0	68.0	tl	60.0	62.0	tl	$-\infty$	68.0	tl	tl		3.8
queen10_10	100	1470	NP-h	10.0	12.0	tl	10.0	12.0	tl	10.0	12.0	tl	10.0	12.0	tl	tl		686.9
queen11_11	121	1980	NP-h	11.0	13.0	tl	11.0	13.0	tl	11.0	13.0	tl	11.0	13.0	tl	tl		1865.7
school1_nsh	352	14612	NP-m	14.0	14.0	891.92	14.0	14.0	24.66	14.0	14.0	13.25	14.0	14.0	30.38	0		17
wap05a	905	43081	NP-m	$-\infty$	$+\infty$	tl	40.0	$+\infty$	tl	50.0	50.0	1317.36	41.0	$+\infty$	tl	tl		293.2
wap06a	947	43571	NP-?	$-\infty$	$+\infty$	tl	40.0	$+\infty$	tl	40.0	$+\infty$	tl	40.0	$+\infty$	tl	tl		175
solved:						15			19			**25**			21		9+2	**25**
avg. time						170.00			194.89			182.36			171.42		843.6	1196.3

Columns 17 and 18 are taken from [20, 21] and show the running times of two (ASS)-based branch-and-cut algorithms suggested by Mendez-Diaz and Zabala. Column 18 ([21]) contains only the additional solved instances, i.e. the instances which have not been solved in [20]. Column 19 is taken from [17] and shows the running times of a set-covering-based branch-and-price algorithm suggested by Malaguti et al. [17]. Notice that the comparison of running times is not quite fair, since [21] and [17] report that their machines need 24 s and 7 s, respectively, for the benchmark [7] instance r500.5, while our machine needs 5.54 s. However, [20, 21] and [17] used 2 h and 10 h as time limit respectively, while we only used 1 h. Nevertheless, it is interesting to see the number of optimal solved instances by each algorithm. This is displayed in the row *"solved"*. The table shows the average time for optimally solved instances in the last row.

We can see that the model (POP2) as well as the set-covering-based approach [17] have solved the highest number of instances (25 out of 68) to provable optimality. Although (POP2) was run on an approximately 1.26 ($= 7s/5.54\,\text{s}$) times faster machine its average runtime is about 6.56 ($= 1196.3\,\text{s}/182.36\,\text{s}$) times faster than that described in [17]. The models (ASS)+(e), (POP), (AREP), (REP) have solved 21, 19, 15, 13 instances respectively, while the algorithms [20, 21] only solved 11 ($= 9 + 2$) instances. It seems that the hybrid model (POP2)

Table 2. Results of the simple models for the `GPIA` graphs from DIMACS set

				Time	AREP			POP			POP2			ASS+(c)			ASS+(e)				
instance	$\|V\|$	$\|E\|$	class	limit	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	old lb	old ub
will199GPIA	701	6772	NP-s	1h	-∞	$+\infty$	tl	7	7	6.68	7	7	11.35	7	7	6.98	7	7	7.24		
ash331GPIA	662	4181	NP-s	1h	-∞	$+\infty$	tl	4	4	11.94	4	4	16.07	4	4	3.45	4	4	74.46		
ash608GPIA	1216	7844	NP-m	1h	-∞	1215	tl	4	4	43.06	4	4	62.98	4	4	607.82	4	4	855.11		
ash958GPIA	1916	12506	NP-m	1h	-∞	$+\infty$	tl	4	4	108.57	4	4	124.75	4	6	tl	4	6	tl		
abb313GPIA	1555	53356	NP-?	3h	-∞	853	tl	8	10	tl	**9**	10	tl	8	12	tl	8	11	tl	8[20]	9[17]

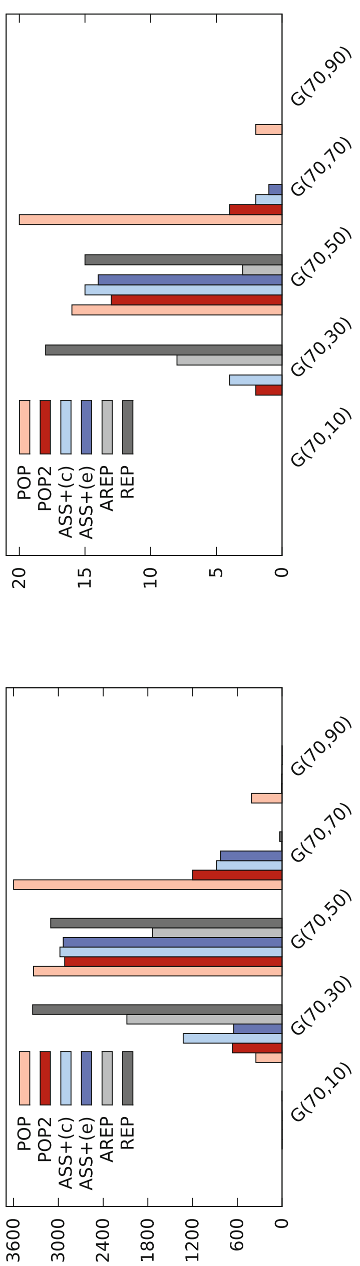
combines the advantages of the pure (POP) model and assignment models. A closer look at the single instances shows that it solved all instances which are solved by (POP) or (ASS). As indicated in our theoretical analysis in Sect. 3.2 the hybrid model dominates the pure model for denser instances.

Table 2 shows the results for the simple models for all five DIMACS `GPIA` graphs as well as old lower and upper bounds for the instance `abb313GPIA`. Since (POP) and (POP2) solved all `GPIA` graphs except `abb313GPIA` within a time limit of 1 h, we decided to increase the time limit for this graph to 3 h. (POP2) achieved a lower bound of 9 in 7769 sec thus improving the best lower bound found. To our knowledge, this new model is the first one solving all of the remaining four `GPIA` graphs.

In order to study the behaviour of the implemented simple models for instances with varying size and density, we first used the benchmark *set70*, for which the results are displayed in Fig. 1. Figure 1(a) shows the average runtime for each set $G(70, p)$ for each density $p = 10, 30, 50, 70, 90$, while Fig. 1(b) shows the number of unsolved instances for each set. (POP) and (ASS)+(e) were able to solve all instances of densities 10 and 30, where the average runtime of (POP) is about 2 times smaller than that of (ASS)+(e). Also here the model (ASS) with the new preprocessing technique (e) is significantly faster than with (c). From density 50 on, the (POP) model seems to have more problems than the other models. The (ASS)+(c), (ASS)+(e) and the (POP2) model deliver similar quality with similar running times for the denser graphs. From density 50 on, the (AREP) model clearly dominates all other models. Since (POP) was the fastest model on the sparse graphs of *set70*, we decided to evaluate larger sparse graphs and generated the set *sparse100*. The corresponding results are shown in Fig. 2. For the larger instances, both partial-ordering based models (POP) and (POP2) dominate the other models. The representative model already gets problems with the smallest instances in the set.

We conclude our work with answering our questions from the beginning:

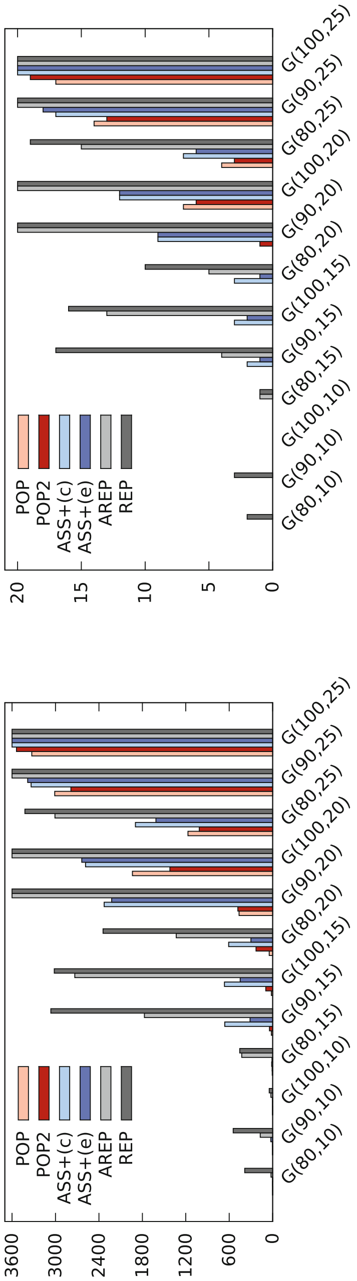
- (H1): (POP2) is able to solve more DIMACS instances to provable optimality than the assignment model. This is not true for the (POP) model. On the random graphs the models (POP) and (POP2) dominate the assignment model on graphs with density $p \leq 30$ and $p \leq 50$, respectively.
- (H2): The (POP2) model dominates the original model (POP) on the harder DIMACS instances as well as on the tested dense random graphs. The explanation lies in the fact discussed in Sect. 3.2. It seems that the (POP2) model combines the advantages of (POP) which is better for sparse graphs and (ASS) which is better for dense graphs.



(a) Average Execution Time [s]

(b) Number of unsolved instances

Fig. 1. Comparison of the simple models on the benchmark *set70*



(a) Average Execution Time [s]

(b) Number of unsolved instances

Fig. 2. Comparison of the simple models on the benchmark *sparse100*

- (H3): The simple models are able to solve very hard instances from the DIMACS benchmark set. A comparison with the computational results of the state-of-the-art algorithms (such as [12,13,16–23]) shows that the quality of the suggested algorithms is about the same (also see Tables 1 and 2). Some of the approaches are able to solve some of the instances faster, but they are slower on other instances.
- (H4): The representatives model does clearly dominate the other models on dense instances. This can be seen on the denser instances of the DIMACS graphs and on the series of random graphs with increasing density.

Acknowledgements. This work was partially supported by DFG, RTG 1855.

References

1. Achterberg, T., Berthold, T., Koch, T., Wolter, K.: Constraint integer programming: a new approach to integrate CP and MIP. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 6–20. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68155-7_4
2. Burke, E.K., Mareček, J., Parkes, A.J., Rudová, H.: A supernodal formulation of vertex colouring with applications in course timetabling. Ann. Oper. Res. **179**(1), 105–130 (2010)
3. Campêlo, M.B., Campos, V.A., Corrêa, R.C.: On the asymmetric representatives formulation for the vertex coloring problem. Discrete Appl. Math. **156**(7), 1097–1111 (2008)
4. Campêlo, M.B., Corrêa, R.C., Frota, Y.: Cliques, holes and the vertex coloring polytope. Inf. Process. Lett. **89**(4), 159–164 (2004)
5. Campos, V., Corrêa, R.C., Delle Donne, D., Marenco, J., Wagler, A.: Polyhedral studies of vertex coloring problems: The asymmetric representatives formulation. ArXiv e-prints, August 2015
6. Cornaz, D., Furini, F., Malaguti, E.: Solving vertex coloring problems as maximum weight stable set problems. Disc. Appl. Math. **217**(Part 2), 151–162 (2017)
7. Benchmarking machines and testing solutions (2002). http://mat.gsia.cmu.edu/COLOR02/BENCHMARK/benchmark.tar
8. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algorithms Appl. **7**(2), 131–140 (2003)
9. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco, CA, USA (1979)
10. Gualandi, S. Chiarandini, M.: Graph coloring instances (2017). https://sites.google.com/site/graphcoloring/vertex-coloring
11. Gualandi, S., Malucelli, F.: Exact solution of graph coloring problems via constraint programming and column generation. INFORMS J. Comput. **24**(1), 81–100 (2012)
12. Hansen, P., Labbé, M., Schindl, D.: Set covering and packing formulations of graph coloring: algorithms and first polyhedral results. Discrete Optim. **6**(2), 135–147 (2009)
13. Held, S., Cook, W., Sewell, E.: Maximum-weight stable sets and safe lower bounds for graph coloring. Math. Program. Comput. **4**(4), 363–381 (2012)
14. Jabrayilov, A., Mallach, S., Mutzel, P., Rüegg, U., von Hanxleden, R.: Compact layered drawings of general directed graphs. In: Hu, Y., Nöllenburg, M. (eds.) GD 2016. LNCS, vol. 9801, pp. 209–221. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50106-2_17

15. Jabrayilov, A., Mutzel, P. (2017). https://ls11-www.cs.tu-dortmund.de/mutzel/colorbenchmarks
16. Johnson, D.S., Trick, M. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. AMS, Providence (1996)
17. Malaguti, E., Monaci, M., Toth, P.: An exact approach for the vertex coloring problem. Discrete Optim. **8**(2), 174–190 (2011)
18. Malaguti, E., Toth, P.: A survey on vertex coloring problems. Int. Trans. Oper. Res. **17**, 1–34 (2010)
19. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS J. Comput. **8**(4), 344–354 (1996)
20. Méndez-Díaz, I., Zabala, P.: A branch-and-cut algorithm for graph coloring. Discrete Appl. Math. **154**(5), 826–847 (2006)
21. Méndez-Díaz, I., Zabala, P.: A cutting plane algorithm for graph coloring. Discrete Appl. Math. **156**(2), 159–179 (2008)
22. Segundo, P.S.: A new DSATUR-based algorithm for exact vertex coloring. Comput. Oper. Res. **39**(7), 1724–1733 (2012)
23. Sewell, E.: An improved algorithm for exact graph coloring. In: Johnson and Trick [16], pp. 359–373
24. Trick, M.: DIMACS graph coloring instances (2002). http://mat.gsia.cmu.edu/COLOR02/