Matheuristic variants of DSATUR for the vertex coloring problem

Nicolas Dupin

Matheuristic variants of DSATUR for the vertex coloring problem

Nicolas Dupin[1]

Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France ;
nicolas.dupin@univ-angers.fr

Abstract. If DSATUR heuristic was proven efficient for many instances of the vertex coloring problem, this paper aims to analyze inefficiency causes of DSATUR heuristic, extending DSATUR with matheuristic operators. Using an Integer Linear Programming formulation allows to have larger local greedy optimization in DSATUR construction scheme. Matheuristics allow also to initialize DSATUR heuristics and matheuristics with cliques or a partial optimal coloring. Dual bounds are also obtained, improving lower bounds implied by cliques computed by matheuristics. Computational analyses are provided, highlighting the strengths and weaknesses of DATUR heuristic and matheuristics.

Keywords: Matheuristic, Vertex Coloring; DSATUR; cliques; dual heuristic.

1 Introduction

The vertex coloring problem (VCP) is one of the most widely studied and popular optimization problems in graph theory, for theoretical insights and practical applications like planning problems or interference avoidance in telecommunications, with many works in exact and heuristic methods to design efficient solvers for VCP [19]. VCP is a NP-hard problem [14].

Exact methods can solve VCP to optimality with hundreds of vertices for difficult instances with Integer Linear Programming (ILP) techniques [13, 7]. First compact ILP formulation, as an assignment problem, is quite inefficient with symmetry issues, and was improved with cutting planes [21]. A more recent compact formulation, based on representatives, breaks symmetries and improve the LP relaxation [4]. The most efficient ILP resolution techniques use an extended formulation with column generation (CG) algorithms [13] or solve VCP after transformation into maximum weight stable set problems [7] without CG.

Among the best seminal constructive heuristics for VCP, two adaptive greedy algorithms, namely DSATUR [3] and RLF [18], are the most efficient. Some previous works analyzed inefficiency issues of such constructive heuristics [6, 17]. Note that an exact tree search method was derived from DSATUR, and was proven to be efficient [12, 22].

Among recent trends to hybridize heuristics, matheuristics rely on exact methods to design heuristics that scales better than exact approaches, may provide results on design of meta-heuristics [2, 11], or furnish both lower and upper

bounds, with dual bounds provided by varied relaxations [1, 9]. This paper investigates such methodology to develop matheuristic variants of DSATUR for VCP. The goal is not only to improve DSATUR heuristic, but also to understand better inefficiency issues of DSATUR. Note that matheuristics were used recently for VCP by [5] and for B-coloring [20].

2 Problem statements

2.1 Definitions and notation

Let $G = (V, E)$ be an undirected graph on a finite vertex set V and edge set E. Cardinalities of V and E are denoted $n = |V|$ and $m = |E| < n(n-1)/2$. We write $V = \{v_1, \ldots, v_n\}$, and denote $I = [\![1; n]\!] = [1; n] \cap \mathbb{Z}$. An edge $e \in E$ linking vertices v_i and v_j is denoted $e = (v_i, v_j)$ with $i < j$. Note that for VCP there is no sense to consider loop edges (v_i, v_i) or multiple edges. For some equations, we consider unordered notation $\text{ngb}(i, j) = 1$ if vertices v_i and v_j are linked by an edge, i.e. $(v_{\min(i,j)}, v_{\max(i,j)}) \in E$, otherwise $\text{ngb}(i, j) = 0$. For $i \in I$, δ_i denotes the set of indexes corresponding to neighbors of vertex v_i in G:

$$\delta_i = \{j \in I,\ \text{ngb}(i, j) = 1\} \tag{1}$$

The *degree* of vertex v_i is denoted $d_i = |\delta_i|$.

A k-coloring of G is an assignment of k colors to vertices V such that adjacent nodes do not share the same color. VCP minimizes the number of color k to have a k-coloring of G, the optimum is the *chromatic number* of graph G. Formally, a proper k-coloring is denoted (c) where $c_i \in [\![1; k]\!]$ denotes the color of vertex v_i, fulfilling:

$$\forall i < j,\ \ (v_i, v_j) \in E \implies c_i \neq c_j \tag{2}$$

A *clique* in G is a subset of vertices $C \subset V$ that forms a complete sub-graph of G, i.e. each couple of vertices of C is linked by an edge in G. The cardinality of any clique gives a first lower bound for the chromatic number, an optimal coloring necessarily implies different colors for the clique.

For constructive heuristics, we define a *partial k-coloring* (c) where $c_i \in [\![1; k]\!] \cup \{-1\}$. if $c_i > 0$, c_i is the color of vertex v_i , otherwise $c_i = -1$ denotes that vertex v_i is not colored yet. A partial k-coloring (c) fulfill:

$$\forall i < j,\ \ (v_i, v_j) \in E \implies (c_i \neq c_j \text{ or } c_i = c_j = -1) \tag{3}$$

For a given partial k-coloring (c), we consider for each vertex v_i the *saturation table* S_i as the set of the assigned colors of the neighbors of v_i. The *saturation (degree)* denotes $s_i = |S_i|$.

$$S_i = \bigcup_{j \in \delta_i}^{n} \{c_j\} \setminus \{-1\} \tag{4}$$

We denote with $\succcurlyeq$ the total order among vertices, as the hierarchic order comparing firstly the saturation degrees and then the degree:

$$\forall v_i, v_j \in V,\quad v_i \succcurlyeq v_j \iff s_i > s_j \text{ or } (\ s_i = s_j \text{ and } d_i \geqslant d_j\ ) \tag{5}$$

2.2 Compact ILP formulations

This section presents two compact ILP formulations for VCP.

Assignment-based ILP model Having k a maximum number of colors, following ILP model minimizes the number of colors to cover G, if the chromatic number is at most k, otherwise the infeasibility of this ILP proves there exists no k-coloring for G. Two types of binary variables are used. On one hand, assignment variables $z_{i,c} \in \{0,1\}$ are defined for $i \in I$ and $c \in [\![1;k]\!]$, $z_{i,c} = 1$ if and only if vertex v_i is assigned to color c. On the other hand, availability variables $y_c \in \{0,1\}$ are defined $c \in [\![1;k]\!]$, $y_c = 1$ if and only if color c is used. It gives rise to following ILP:

$$\min \sum_{c=1}^{k} y_c \tag{6}$$

$$s.t: \quad \sum_{c=1}^{k} z_{i,c} = 1 \quad ,\forall i \in I \tag{7}$$

$$z_{i,c} + z_{j,c} \leqslant y_c \ ,\forall (v_i, v_j) \in E, \forall c \in [\![1;k]\!] \tag{8}$$

Objective function (6) counts with variables y the number of colors used. Constraints (7) ensure that each vertex is colored. Constraints (8) expresses incompatibility of neighbor vertices to have the same color.

Having as initial value k an upper bound of the chromatic number, given by a primal heuristic, or simply $k = |V|$ ensures the feasibility to compute the chromatic number and an optimal assignment of colors solving this last ILP. However, for efficiency issues with ILP resolution, value of k should be as small as possible. Symmetries in this encoding, for instance by permutation of colors, is a bottleneck to solve large VCP with a Branch&Bound (B&B) tree search.

Representatives ILP model A compact formulation, based on representatives, breaks symmetries and improves the LP relaxation [4]. Binary variables $z_{i,i'} \in \{0,1\}$, are defined for all $i, i' \in V$ with $i \leqslant i'$, $z_{i,'} = 1$ if and only if vertices v_i et $v_{i'}$ have the same color, and i is the minimum index of its color. It induces following asymmetric ILP formulation:

$$\min_{z} \sum_{i=1}^{n} x_{i,i} \tag{9}$$

$$s.c: \quad \sum_{i' \leqslant i} x_{i',i} \geqslant 1 \quad ,\forall i \in I \tag{10}$$

$$x_{j,i} + x_{j,i'} \leqslant x_{j,j} \ \forall (v_i, v_{i'}) \in E, \forall j \leqslant i, \tag{11}$$

Objective function (9) counts with variables $x_{i,i}$ the number of colors used. Constraints (10) ensure that each vertex i is colored: either $x_{i,i} = 1$ or there

exists a representative $i' < i$ such that $x_{i',i} = 1$. Constraints (11) expresses incompatibility of neighbor vertices to have the same color, and that a variable $x_{i',i} = 1$ implies that $x_{i',i'} = 1$.

2.3 Standard DSATUR algorithm

DSATUR colors the vertices one after another, assigning the first color available or adding a new color. The order of traversal follows order $\succcurlyeq$. Among the uncolored vertices, the selected vertex to color maximizes firstly the saturation and then the degree in a lexicographic way. Coloring a new vertex updates saturation, the order of traversal is not defined a priori, DSATUR is an adaptive greedy algorithm. Algorithm 1 writes the standard version of DSATUR, iterating with a partial coloring till the graph is fully colored.

Algorithm 1: Standard DSATUR algorithm

Input: $G = (V, E)$ a non-empty and non-oriented graph
Initialization:
  define partial coloring c with $c_i := -1$ for all $i \in I$
  define saturation table S with $S_i := \emptyset$ for all $i \in I$
  initialize set $U := V$, and color $k := 0$
  while $U \neq \emptyset$
    find $u \in U$, a maximum of $\succcurlyeq$ in U.
    if $|S_u| = k$ **then** k:=$k + 1$ // a new color is added
    compute $c_i := \min S_u$ // assign color to u
    remove u from U
    for all $i \in \delta_u \cap U$, $S_i = S_i \cup \{c_i\}$ // update saturation
  end while
return color k and (c) a k-coloring of G

3 DSATUR matheuristic variants

This section provides variants of DSATUR with matheuristic extensions of operators of Algorithm 1.

3.1 Initialization

Several initialization strategies can be used before processing the standard DSATUR constructive heuristic. Initialization consists of defining an initial partial coloring and computing the saturation table for the uncolored vertices. Several strategies can be provided:

- `maxDeg`: only one vertex is colored, one having the highest degree. Only the neighbors of this vertex have a saturation set to 1.

- `col-`n: one considers n vertices having the highest degrees, and color them solving to optimality ILP (9)-(11) restricted to these n vertices. Note that n is a controlled parameter, that is set so that the ILP is solvable quickly.
- `clq`: one find a large clique in the graph, and color this clique with different colors. Finding a maximum clique being NP-hard, this initialization requires a heuristic. In Appendix, a constructive matheuristic dealing with maximum clique problems of fixed size is presented.
- `clq-col-`n: initialization strategy `clq` is firstly operated. Then `col-`n strategy is operated for n vertices having the highest saturation and degrees, considering the clique. Colors of these n vertices are either colors used for the clique, or new colors. If ILP formulations of section 2 can be used, fixing variables of the clique to their assignment or defining them as representatives of old colors, Section 3.2 provides an ILP formulation for VCP with existing colors.

Note that `maxDeg` is equivalent to standard DSATUR as written in Algorithm 1: no initialization induces that the first node to be selected in the loop is one with the highest degree. Many initialization are possible with `maxDeg`, there can be many vertices having the maximum degree. With `col-`n strategies, there is more depth in the initialization, to analyze if first iterations or standard DSATUR algorithm induce bad decisions. Initializing saturation with a clique does not induce a heuristic choice, it is an exact pre-processing, each color of the clique shall be different. The initialization of the saturation table is more advanced with a clique than with a single vertex.

3.2 Local optimization with larger neighborhoods

Let (c) be a partial k-coloring. Let k be the current number of colors in c, let C be the set of colored vertices, and U a subset of un-colored vertices:

$$C = \{i \in I, c_i > 0\} \tag{12}$$

$$U \subset \{i \in I, c_i = -1\} \tag{13}$$

In this section, we define an ILP formulation to assign a color for vertices indexed in U while preserving the colors that are assigned in C. Following formulation hybridizes assignment-based ILP formulation for the existing colors, and the representative-based formulation for new colors. Binary variables $x_{u,u'}$ are defined here only for $u < u' \in U$, considering subset of edges $E_U = \{(v_u, v_{u'})\}_{u<u' \in U}$. Binary variables $z_{u,l}$ to assign previous colors are defined for $u \in U$ and $l \in [\![1; k]\!]$ such that there is no neighbor u that have color l in (c). Variables $z_{u,l}$ are defined for all defined for $u \in U$ for $l \in K_u$ where:

$$K_u = \{l \in [\![1; k]\!], \forall i \in C, c_i = l \implies \text{ngb}(i, j) = 0)\} \tag{14}$$

It induces following ILP formulation to color the vertices indexed in U:

$$
\begin{aligned}
& \min_{z} \sum_{u \in U} x_{u,u} & & \\
s.t: \quad & z_{i,l} + z_{i',l} \leqslant 1 & & \forall (v_i, v_{i'}) \in E_U, \forall l \in [\![1;k]\!] \\
& x_{u,i} + x_{u,i'} \leqslant x_{u,u} & & \forall (v_i, v_{i'}) \in E_U, \forall u \in U, u \leqslant i \\
& \sum_{i' \in U : i' \leqslant i} x_{i',i} + \sum_{l \in K_u} z_{i,l} \geqslant 1 & & \forall u \in U
\end{aligned}
\tag{15}
$$

3.3 General algorithm

Algorithm 2: Matheuristic DSATUR variants

Input: $G = (V, E)$ a non-empty and non-oriented graph
Parameters:
• an initialization strategy $\mathcal{S}$ (from section 3.1) ;
• $o \in \mathbb{N}$, $o > 1$;
• $r \in \mathbb{N}$.
Initialization:
initialize colored set C, and color k with strategy $\mathcal{S}$.
initialize $W := V \setminus C$.
update partial coloring c and saturation table S with strategy $\mathcal{S}$.
while $W \neq \emptyset$
sort W with order $\succcurlyeq$.
define U_1 as the o first elements after sorting.
define U_2 as the elements of rank $o+1$ and $\min(|W|, o+r)$ after sorting.
solve ILP (15) with C and $U = U_1 \cup U_2$.
$k := k + OPT$ where OPT is the optimal value of the last ILP.
if $o + r \leqslant |W|$ **then** $U_1 = U$ **end if**
set $W := W \setminus U_1$
assign colors c_u of the ILP for $u \in U_1$
end while
return color k and (c) a k-coloring of G

Algorithm 2 is a general version for an extended DSATUR matheuristic. Initialization can be any strategy defined in section 3.1. The remaining of the Algorithm simultaneously colors o vertices, solving an ILP (15) with $o+r$ vertices and the previously assigned colors. In the standard version of DSATUR heuristic, we have $o = 1$ and $r = 0$. Having $r > 0$ ensures more depth in the local decision making with the possibility to reoptimize these variables after, as in [5, 11]. Having $r = 0$ could lead to threshold effects. To solve efficiently local optimization, there are $o + r$ new vertices to color using ILP formulation (15), this parameter should be fixed according to the capability of the ILP solver to solve VCP problems for this size. Note that the r vertices that can be re-optimized are not necessarily chosen for the next iteration as the saturation table is updated with the o fixed colors.

3.4 Dual bounds

As in [1, 9], some DSATUR matheuristics allow to have both lower and upper bounds, with dual bounds provided by varied relaxations [1, 9]. Firstly, we recall that the cardinality of any clique gives a first lower bound, an optimal coloring (as any proper coloring) implies different colors for the clique. Algorithm 3 in Appendix A gives thus first dual bounds for VCP, after the first phase to initialize DSATUR with a clique.

After a clique initialization, any dual bounds of the ILP resolution of (15) assigning $n = o + r$ colors, either in `clq-col-`n initialization or in the first iteration of in Algorithm 2 after `clq` initialization, is a dual bound for VCP, relaxing the constraints corresponding to the unoptimized nodes, and without any heuristic reduction of the original problem (which is not true once colors are fixed in a subset that is not a clique). As in [9], dual bounds can be obtained with several relaxations computations of ILP (15): an exact ILP resolution of such ILP restricted with small values of $n = o + r$, larger values of N with computations of LP relaxation, or intermediate dual bounds with truncated ILP resolution with intermediate values of n. Note that such dual heuristics may take advantage of exact reduction techniques as in [15] to compute more efficiently dual bounds for smaller and equivalent VCP problems or to have a more relevant selection of the subset of n nodes considered in the relaxation.

3.5 Towards randomization and multi-start?

As in [20], one may extend Algorithm 2 for a multi-start initialization and use randomization. DSATUR standard algorithm can be randomized with the perturbation of the order of traversal of vertices, with the choice of the color to assign, not necessarily the first one. In Algorithm 2, the color to assign is optimized regarding the depth of traversal, so that such randomization does not make sense anymore. The depth in the order of traversal induces also few impact to have local perturbations of the order of traversal. It makes sense only to break ties for the threshold effects of the algorithm. A more promising randomization is to consider several large cliques to have several initialization.

For the computation of dual bounds, two randomization may be considered: the initial clique and also the selection of the $n = o + r$ nodes for the second phase computation of dual bounds using the ILP solver.

4 Computational results

Computational experiments were processed using a computer Intel(R) Core(TM) i7-6700, 3.40GHz, running Linux Lubuntu 20.4, using up to 4 threads and 32 Gb of RAM memory. CPLEX version 20.1 was used for ILP resolution. Algorithms were coded in Julia programming language version 1.7.3, using the JuMP library version 1.1.1 to call ILP solvers and LightGraphs version 1.3.5 for graphs. Without specific precision, we use CPLEX 20.1 with its default parameter, except

parameters `CPX_PARAM_EPAGAP` = 0.99999 to stop computation to optimality knowing the objective function is integer, a time limit of 30 seconds maximum for each computation (which was not reached in most of the following results). CPLEX allows to set optimization parameters to sizes $n = 123$ for clique depth search in Algorithm 3 and size $n = o + r = 80$ in Algorithm 2, to have partial ILP computations solvable to optimality in at most few seconds. Note that CBC can also be used with JuMP, to have an open source code, for these cases, we set $n = 100$ and $o + r = 60$.

4.1 Instances

For this study, we consider a subset of 53 DIMACS instances removing instances that are easy for DSATUR, where DSATUR and the matheuristics gives the BKS that is proven optimal. These instances are highlighted in Appendix B, with their characteristics and their best known lower and upper bounds. For comparing primal heuristics, we used for this paper the instances without the exact pre-processign reduction from [15]. For the selected difficult instances, only 13 are reduced by [15], which could lead to easy instances, which was used for the computations of dual bounds. Original and reduced instances for VCP are available at https://github.com/Cyril-Grelier/gc_wvcp_cp.

4.2 Standard DSATUR with varied initialization

Table 1 presents for different initialization of DSATUR the total number of colors used for the 53 selected instances, the gaps to the Best Known Solutions (BKS), the number of instances where BKS is equaled, and the other columns are compared with the standard DSATUR Algorithm: #worse and #better counts the number of instances where the considered Algorithms have different values with BKS, respectively worse and better solutions, and quartiles are considered with the absolute gap from DSATUR to the corresponding algorithm, negative values means that the corresponding algorithm has better value than standard DSATUR, quartiles allows to appreciate the dispersion of the results.

Table 1 shows that `col-`n strategies are disappointing, leading to worse results in average than standard DSATUR. On the contrary, `clq` initialization improves significantly standard DSATUR. Using `col-`n strategies after clique initialization improves also significantly standard DSATUR. Note that for instance `lr1000.1.col`, a BKS is found by DSATUR and `clq` initialization, not for the other approaches, which explains this instance is considered in the selected pool of difficult instances for DSATUR.

It is interesting that `clq-col-`80 and `clq` improve DSATUR solutions on different instances, Considering the best results of both algorithms in the row "Best `clq`" of Table, as if we consider both approaches in parallel as in [10, 8], it provides an additional significative improvement. In the row, "Best `clq`+DSATUR" we consider the best result including also DSATUR, to analyze the complementarity with the original approach. A very slight improvement is observed, as well as considering all the approaches in "Best+DSATUR" row, or removing only

Table 1. Comparison of DSATUR matheuristics with different initialization of saturation table. Results parallelizing several strategies are also provided.

	#colors	gap	#BKS	#worse	#better	Q1	Q2	Q3
`maxDeg`	3240	32.03 %	1	0	0	0	0	0
`col-60`	3251	32.48 %	1	19	16	-1	0	1
`col-80`	3250	32.44 %	2	20	16	-1	0	1
`clq-col-80`	3214	30.97 %	2	18	17	-1	0	1
`clq`	3209	30.77 %	4	13	19	-1	0	0
Best `clq`	3181	29.63 %	6	7	26	-1	0	0
Best `clq`+DSATUR	3174	29.34 %	6	0	26	-1	0	0
Best–DSATUR	3163	28.89 %	6	3	34	-2	-1	0
Best+DSATUR	3160	28.77 %	6	0	34	-2	-1	0
BKS	2454	0.00 %	53	0	52	-14	-5	-3

the . DSATUR standard approach in row "Best–DSATUR". These last results highlight that for three instances, `le450_5b`; `queen11_11` and `queen15_15`, none of the other initialization improves or equals standard DSATUR. This section validates to consider both `clq-col-80` and `clq` strategies, in a multi-start of parallel heuristic, and the power of using cliques for DSATUR variants. If the solutions of standard DSATUR have been improved, the gaps from the BKS remain very significant.

4.3 DSATUR with larger local optimization

Table 2 has the same shape as Table 1 to compare DSATUR extended matheuristics to the standard Algorithm 1. Parameters o, r are the ones in Algorithm 2, standard version of DSATUR, implemented with Algorithm 1, corresponds to $o = 1$ and $r = 0$. This allows to analyze the impact of a larger depth in local optimization and the part of vertices to reoptimize for a better efficiency.

Table 2. Comparison of DSATUR matheuristics with different initialization of saturation, and values of optimization parameters o and r in Algorithm 2

Init satur	o	r	#colors	gap	#BKS	#worse	#better	Q1	Q2	Q3
`maxDeg`	1	0	3240	32,03 %	1	0	0	0	0	0
`col-80`	1	0	3250	32,44 %	2	20	16	-1	0	1
`col-80`	20	60	3181	29,63 %	6	12	30	-3	-1	0
`col-80`	40	40	3218	31,13 %	5	20	26	-2	0	1
`col-80`	80	0	3322	35,37%	2	35	13	0	1	2
`clq`	1	0	3209	30,77 %	4	13	19	-1	0	0
`clq`	40	40	3155	28,57%	10	9	32	-3	-1	0
Best Clq			3134	27,71 %	10	4	37	-3	-1	0
Best-DSATUR			3125	27,34 %	10	3	40	-3	-2	-1
Best+DSATUR			3122	27,22 %	10	0	40	-3	-2	-1
BKS			2454	0,00 %	53	0	52	-14	-5	-3

Table 2 does not provide results with `maxDeg` initialization, first iteration of Algorithm 2 induce a similar saturation than using `col-`n initialization. Using `clq` initialization, results are very stable considering parameters value $(o, r) \in \{(20, 60); (40, 40); (60, 20); (80, 0)\}$. Using `col-`80 initialization, setting parameter values $(o, r) \in \{(20, 60); (40, 40); (60, 20)\}$ improves DSATUR standard algorithm, this is not the case with $(o, r) = (80, 0)$. Coherently with [5, 11], it is important to have a significant part of variables that can be reoptimized to avoid bad choices due to threshold effects. A drawback of increasing r value (and decreasing o value to keep value $o + r$ stable), is that computation times are increasing. $(o, r) = (40, 40)$ is a good compromise between solution quality and computation time.

Initializing with `clq` provides again the best results, $(o, r) = (40, 40)$ improves significantly DSATUR with both the standard and the clique initialization. Combining $(o, r) = (40, 40)$ and standard DSATUR construction $(o, r) = (1, 0)$ allows an additional improvement. This highlights that standard DSATUR algorithm has good properties, that can be broken with more depth in local optimization. Coherently with [8], using larger neighborhoods in greedy constructive heuristics improves in average the solution quality. However, even with an ensemble of such constructive heuristics that can be computed in parallel, a significant gap remains to the BKS.

4.4 Dual bounds

In Appendix B, best known lower bounds reported, mainly after [16] and in some cases the chromatic number is known by construction or specific reasoning. To compute the dual bounds, we used the exact reduction from [15] for the 13 instances where a reduction is obtained, as shown in Tables 4 and 5.

For the DIMACS instances, Algorithm 3 is very efficient to find large cliques, and gives already the maximum clique size which is also the chromatic number for 28 out of the 53 selected instances with parameter $n \in \{100, 125\}$. This often occurs also for the easy instances that were removed from the dataset for this study as easy instances for DSATUR. We removed thus these instances for results of dual bounds, these case being specific and easy to compute optimal dual bounds. Note that having $n = 100$ or $n = 125$ produced the same results with Algorithm 3, computations with $n = 125$ are slightly longer. For the 25 remaining instances, we report dual bounds and computation times obtained after a clique computation with Algorithm 3 computations with $n = 100$ with following parameters to analyze the compromise between the number of vertex to consider in the ILP (15):

- $n = o + r = 80$ and a time limit of $300s$, with bounds at the root node of the B&B tree and in truncated resolution time, or the optimal value.
- $n = o + r = 125$ and a time limit of $900s$, with bounds at the root node of the B&B tree .
- $n = o + r = 200$ and a time limit of $3600s$ for B&B tree search.

Table 3. Comparaison of the lower bounds obtained by initial clique computation and ILP refinements to the BKS and BKLB, LB are reported as well as computation times. For ILP refinements, the time to compute the clique is not counted, these times are additional time to improve the lower bounds given by clique

	UB	LB	LB			t		
	BKS	BKLB	clq	$n = 125$	$n = 200$	clq	$n = 125$	$n = 200$
C2000.5	145	99	15	20	21	185	163	3600
C4000.5	259	107	17	21	22	252	99	3600
dsjc125.1	5	5	4	5	5	0,2	81,8	118
dsjc125.5	17	17	10	14	14	14	131,4	3600
dsjc125.9	44	44	34	43	44	33	1	1
dsjc250.1	8	7	4	6	5	0,7	186,4	3600
dsjc250.5	28	26	12	16	17	160	206	3600
dsjc250.9	72	71	41	56	70	53	1,5	105
dsjc500.1	12	9	5	5	5	5	4	3600
dsjc500.5	48	43	13	17	19	167	61	3450
dsjc500.9	126	123	51	65	79	50	0,4	274
dsjc1000.1	20	10	6	6	6	38	8,6	3600
dsjc1000.5	83	73	14	19	20	175	172,6	3600
dsjc1000.9	222	215	59	73	86	80	2,3	15
dsjr500.1c	85	85	76	77	79	47	3	11
dsjr500.5	122	122	114	122	122	5	0,6	10
flat300_26_0	26	26	11	15	16	167	217	3600
flat300_28_0	28	28	12	15	16	160	259	3600
flat1000_50_0	50	50	13	17	19	175	186	3600
flat1000_60_0	60	60	13	17	19	178	135	3600
flat1000_76_0	76	76	14	18	19	166	179	3600
latin_square	97	90	90	90	90	32	0,2	12
r1000.1c	98	96	87	88	88	143	73	9
r1000.5	234	234	213	214	220	81	8,5	19
TOTAL	1965	1716	928	1039	1101			
Average						95	87	2033

Table 3 shows that the computation of dual bounds with ILP (15) induced improvements of the cliques given in input for most of the instances. For `latin_square` instance, a clique of size 90 is found easily, it is the actual best Lower Bound known (BLBK) for VCP, the ILP computations of improved dual bounds do not improve this bound. Note that in experiments of [16], computations could take around 30 days to have dual bounds for very difficult problems, especially for instances C2000.5 and C4000.5. For such large and difficult instance, where the best cliques knows are far from the BKS and BLBK, our dual bounds are quite limited, which is also the cases for [13].

Table 3 shows it is preferable and computable to tackle problems with $n = o + r = 200$. The higher the value of n is, the higher the optimal solution of ILP (15), and truncated ILP resolution remains efficient to computer better dual bounds than the ones computed optimally with smaller subproblems. For three instances, namely `dsjr500.5` `dsjc125.1` and `dsjc125.9`, optimal lower bounds are found quickly. Globally, more improvements are observed on dense graphs (suffixed by .5 and .9 indicating the density) than on sparse ones (suffixed by .1), the more difficult instances were no improvement is observed are sparse graphs. Note that half of ILP (15) with $n = 200$ are solved to optimality in less than one hour, sometimes very quickly, clique initialization may be helpful to speed up such ILP computations, whereas for other instances some ILP computations of size $n = 125$ are not solvable in one hour.

5 Conclusions and perspectives

This paper studied matheuristic variants of DSATUR, to improve its standard version, and also to help understanding the strengths and weaknesses of this well-known heuristic. Initializing DSATUR with a large clique, using a simple greedy matheuristic, is a very significant improvement of the standard initialization with one vertex of maximal degree. Having larger optimization in the greedy construction is efficient when some vertices can be re-optimized to avoid threshold effects. However, improvement of DSATUR is slight, significant gaps to BKS remain using DSATUR constructive matheuristics. Dual bounds are also provided, highlighting also the interest of cliques for DSATUR. With a newly introduced ILP formulation, dual bounds implied by cliques can be improved in short and long computation times.

These results offer several perspectives. Firstly, exact version of DSATUR [22, 12] could be improved using cliques for branching in the tree search algorithm, dual and primal heuristics can be used and parametrized to prune some nodes in this tree search. Secondly, dual bound can be improved using other exact techniques for dual bounds, as [13, 7, 16]. Thirdly, perspectives are to extend similarly RLF as matheuristics.

References

1. M. A. Boschetti, A. N. Letchford, and V. Maniezzo. Matheuristics: survey and synthesis. *Int Trans Oper Res*, 30(6):2840–2866, 2023.

2. M. A. Boschetti and V. Maniezzo. Matheuristics: using mathematics for heuristic design. *4OR*, 20(2):173–208, 2022.
3. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
4. M. Campêlo, V. A. Campos, and R. C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discret. Appl. Math*, 156(7):1097–1111, 2008.
5. R. C. Chandrasekharan and T. Wauters. A constructive matheuristic approach for the vertex colouring problem. In *13th International Conference on the Practice and Theory of Automated Timetabling-PATAT*, volume 1, 2021.
6. M. Chiarandini, G. Galbiati, and S. Gualandi. Efficiency issues in the rlf heuristic for graph coloring. In *Proceedings of the 9th Metaheuristics International Conference, MIC*, pages 461–469, 2011.
7. D. Cornaz, F. Furini, and E. Malaguti. Solving vertex coloring problems as maximum weight stable set problems. *Discret. Appl. Math*, 217:151–162, 2017.
8. N. Dupin, R. Parize, and E. Talbi. Matheuristics and Column Generation for a Basic Technician Routing Problem. *Algorithms*, 14(11):313, 2021.
9. N. Dupin and E. Talbi. Machine learning-guided dual heuristics and new lower bounds for the refueling and maintenance planning problem of nuclear power plants. *Algorithms*, 13(8):185, 2020.
10. N. Dupin and E. Talbi. Parallel matheuristics for the discrete unit commitment problem with min-stop ramping constraints. *Int Trans Oper Res*, 27(1):219–244, 2020.
11. N. Dupin and E. Talbi. Matheuristics to optimize refueling and maintenance planning of nuclear power plants. *Journal of Heuristics*, 27(1):63–105, 2021.
12. F. Furini, V. Gabrel, and I.-C. Ternier. An improved dsatur-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, 69(1):124–141, 2017.
13. F. Furini and E. Malaguti. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2):130–136, 2012.
14. M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM (JACM)*, 23(1):43–49, 1976.
15. O. Goudet, C. Grelier, and D. Lesaint. New bounds and constraint programming models for the weighted vertex coloring problem. In *Thirty-Second International Joint Conference on Artificial Intelligence*, pages 1927–1934, 2023.
16. S. Held, W. Cook, and E. C. Sewell. Safe lower bounds for graph coloring. In *Integer Programming and Combinatoral Optimization: 15th International Conference, IPCO 2011.*, pages 261–273. Springer, 2011.
17. R. Janczewski, M. Kubale, K. Manuszewski, and K. Piwakowski. The smallest hard-to-color graph for algorithm dsatur. *Discrete Mathematics*, 236(1-3):151–165, 2001.
18. F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489, 1979.
19. E. Malaguti and P. Toth. A survey on vertex coloring problems. *Int Trans Oper Res*, 17(1):1–34, 2010.
20. R. A. Melo, M. F. Queiroz, and M. C. Santos. A matheuristic approach for the b-coloring problem using integer programming and a multi-start multi-greedy randomized metaheuristic. *European Journal of Operational Research*, 295(1):66–81, 2021.
21. I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discret. Appl. Math*, 156(2):159–179, 2008.

22. P. San Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39(7):1724–1733, 2012.

Appendix A: matheuristic to find large cliques and stables

This appendix present the matheuristic that computes a large clique as initialization of Algorithm 2. To ease presentation, we present the matheuristic in Algorithm 3 for the Maximum Independent Set (MIS) problem applied to the complementary graph of G. Indeed, it is equivalent for a subset V to be a clique in the graph G and an independent (or stable) set in the complementary graph.

Algorithm 3: Matheuristic greedy computation of large cliques

Input: $G = (V, E)$ a non-empty and non-oriented graph
Parameter: $n > 0$ the maximal size of MIS to solve
Initialization:
  initialize $I := \emptyset$, $R := V$.
  Compute $G' = (R, E^c)$ the complementary graph of G.
  while $R \neq \emptyset$
    define U_1 as the n vertices of G' having the minimal degree.
    solve ILP (16) with vertices in U_1, let S a solution.
    set $I := I \cup S$
    set $R := R \setminus U_1$
    remove from R the neighbors of vertices in S.
    update graph G' removing edges with removed vertices, update degrees.
  end while
return value $|I|$ and set I, clique in the graph G.

Algorithm 3 computes iteratively an independent set based on MIS of fixed size n. Defining with U_1 a subset of V, a maximum independent set in U_1 can be computed using the following ILP formulation, where binary variables $z_v \in \{0, 1\}$, are defined with $z_v = 1$ if and only if vertex $v \in U_1$ is considered in the stable:

$$\begin{aligned} &\max_{z \in \{0,1\}^{|U_1|}} \sum_{v \in U_1} z_v \\ s.c: \;& z_v + z_{v'} \leqslant 1, \forall (v, v') \in E, \end{aligned} \tag{16}$$

Algorithm 3 is an adaptive greedy algorithm: once vertices are added in the current independent set, the next candidate vertices are chosen with the minimum degrees in the updated graph, removing neighbors of selected points that cannot be added in the current stable.

Appendix B: Selected instances and their characteristics

Table 4. Lists of selected instances (part 1/2), with their number of vertices and edges without and with exact reduction, and reference values for lower and upper bounds

	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	LB	UB
reduction [15]	no	no	yes	yes		
C2000.5	2000	999836	2000	999836	99	145
C4000.5	4000	4000268	4000	4000268	107	259
dsjc125.1	125	736	125	736	5	5
dsjc125.5	125	3891	125	3891	17	17
dsjc125.9	125	6961	125	6961	44	44
dsjc250.1	250	3218	250	3218	7	8
dsjc250.5	250	15668	250	15668	26	28
dsjc250.9	250	27897	250	27897	71	72
dsjc500.1	500	12458	500	12458	9	12
dsjc500.5	500	62624	500	62624	43	48
dsjc500.9	500	112437	500	112437	123	126
dsjc1000.1	1000	49629	1000	49629	10	20
dsjc1000.5	1000	249826	1000	249826	73	83
dsjc1000.9	1000	449449	1000	449449	215	222
dsjr500.1	500	121275	12	66	12	12
dsjr500.1c	500	3555	289	40442	85	85
dsjr500.5	500	58862	486	57251	122	122
flat300_26_0	300	21633	300	21633	26	26
flat300_28_0	300	21695	300	21695	28	28
flat1000_50_0	1000	245000	1000	245000	50	50
flat1000_60_0	1000	245830	1000	245830	60	60
flat1000_76_0	1000	246708	1000	246708	76	76
le450_5a	450	5714	450	5714	5	5
le450_5b	450	5734	450	5734	5	5
le450_5c	450	9803	450	9803	5	5
le450_5d	450	9757	450	9757	5	5
le450_15a	450	8168	449	8166	15	15
le450_15b	450	8169	410	7824	15	15
le450_15c	450	16680	450	16680	15	15
le450_15d	450	16750	450	16750	15	15
le450_25c	450	17343	435	17096	25	25
le450_25d	450	17425	433	17106	25	25

Table 5. Lists of selected instances (part 2/2), with their number of vertices and edges without and with exact reduction, and reference values for lower and upper bounds

	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	LB	UB
reduction [15]	no	no	yes	yes		
latin_square	900	307350	900	307350	90	97
queen6_6	450	17343	450	17343	6	6
queen7_7	450	17425	450	17425	7	7
queen8_8	64	728	64	728	9	9
queen8_12	96	1368	96	1368	12	12
queen9_9	81	1056	81	1056	9	9
queen10_10	100	1470	100	1470	10	10
queen11_11	121	1980	121	1980	11	11
queen12_12	144	2596	144	2596	12	12
queen13_13	169	3328	169	3328	13	13
queen14_14	196	4186	196	4186	14	14
queen15_15	225	5180	225	5180	15	15
queen16_16	256	6320	256	6320	16	16
r125.5	125	3838	109	3323	36	36
r250.1c	250	30227	68	2270	64	64
r250.5	250	14849	235	13968	65	65
r1000.1	1000	14378	46	651	20	20
r1000.1c	1000	485090	686	227525	96	98
r1000.5	1000	238267	966	230416	234	234
school1	385	19095	371	18983	14	14
school1_nsh	352	14612	341	14537	14	14