Computers & Operations Research 57 (2015) 41–50

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A DSATUR-based algorithm for the Equitable Coloring Problem ☆

Isabel Méndez-Díaz [a,*], Graciela Nasini [b,c], Daniel Severín [b,c]

[a] Departamento de Ciencias de la Computación, Facultad de Ciencias Exactas y Naturales – FCEyN, Universidad de Buenos Aires, Intendente Guiraldes 2160 (Ciudad Universitaria, Pabellón 1), Argentina
[b] FCEIA, Universidad Nacional de Rosario, Argentina
[c] CONICET, Argentina

ARTICLE INFO

Available online 6 December 2014

Keywords:
Equitable coloring
DSatur
Exact algorithm

ABSTRACT

This paper describes a new exact algorithm for the Equitable Coloring Problem, a coloring problem where the sizes of two arbitrary color classes differ in at most one unit. Based on the well known DSatur algorithm for the classic Coloring Problem, a pruning criterion arising from equity constraints is proposed and analyzed. The good performance of the algorithm is shown through computational experiments over random and benchmark instances.

1. Introduction

There exists a large family of combinatorial optimization problems having relevant practical importance, besides its theoretical interest. One of the most representative problem of this family is the *Graph Coloring Problem* (GCP), which arises in many applications such as scheduling, timetabling, electronic bandwidth allocation and sequencing problems.

Given a simple graph $G=(V,E)$, where V is the set of vertices and E is the set of edges, a *coloring of* G is an assignment of colors to vertices such that the endpoints of any edge have different colors. A *k-coloring of* G is a coloring that uses k colors. The GCP consists of finding the minimum number k such that G admits a k-coloring. This minimum number of colors is called the *chromatic number* of G and is denoted by $\chi(G)$.

It is well known that GCP models some scheduling problems. The simplest version considers assignments of workers to a given set of tasks. Pairs of tasks may conflict each other, meaning that they should not be assigned to the same worker. The problem is modeled by building a graph containing a vertex for every task and an edge for every conflicting pair of tasks. A coloring of this graph represents a conflict-free assignment and the chromatic number of the graph is exactly the minimum number of workers needed to perform all tasks.

However, an extra constraint could be required to ensure the uniformity of the distribution of workload employees. The addition of this extra *equity* constraint gives rise to the *Equitable Coloring Problem* (ECP), introduced in [1] and motivated by an application concerning *garbage collection* [2]. Other applications of the ECP concern *load balancing problems* in multiprocessor machines [3] and results in *probability theory* [4]. An introduction to ECP and some basic results are provided in [5].

Formally, an *equitable k-coloring* (or just k-eqcol) of a graph G is a k-coloring satisfying the *equity constraint*, i.e. the size of two color classes cannot differ by more than one unit. The *equitable chromatic number* of G, $\chi_{eq}(G)$, is the minimum k for which G admits a k-eqcol. The ECP consists of finding $\chi_{eq}(G)$.

Computing $\chi_{eq}(G)$ for arbitrary graphs is proved to be NP-Hard and just a few families of graphs are known to be easy such as complete n-partite, complete split, wheel and tree graphs [5].

There exist some differences between GCP and ECP that make the latter harder to solve. It is known that the chromatic number of an unconnected graph G is the maximum among the chromatic numbers of its components. Algorithms that solve GCP can take advantages of the property mentioned above (e.g. [6]) by solving GCP on each component, which is less CPU intensive than address the problem on the whole graph. Moreover, one can preprocess the graph in order to reduce its size and, consequently, the time of optimization. For example, choosing two non-adjacent vertices with the same neighborhood, known as twin vertices, and deleting one of them. The chromatic number of the graph remains the same after deletion, since the deleted vertex can inherit the color of the other one. None of these recipes can be applied when solving ECP. For instance, let G be the graph of Fig. 1a and G' be the graph compounded of two disjoint copies of G. Then, $\chi_{eq}(G')=2$ but $\chi_{eq}(G)=3$. Also, let H' be the graph of Fig. 1b. Clearly, v and v' are twin vertices. Let H' be H after v is deleted. We have $\chi_{eq}(H')=2$ but $\chi_{eq}(H)=3$.

There are very few tools in the literature related to ECP resolution. Two constructive algorithms called Naive and SubGraph

☆ Partially supported by Grants PIP-CONICET 241, PICT 2011-0817 and UBACYT 20020100100666.

* Corresponding author.
E-mail addresses: imendez@dc.uba.ar (I. Méndez-Díaz), nasini@fceia.unr.edu.ar (G. Nasini), daniel@fceia.unr.edu.ar (D. Severín).

http://dx.doi.org/10.1016/j.cor.2014.11.014
0305-0548/

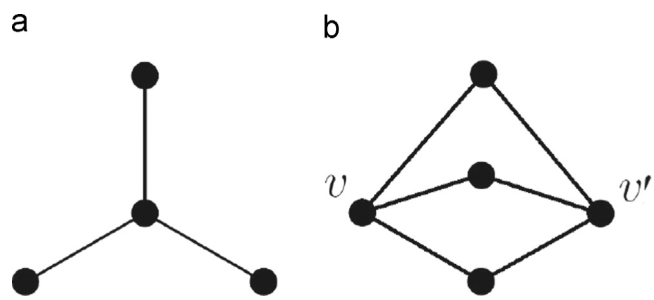

Fig. 1. An example.

were given in [5] to generate greedily an equitable coloring of a graph and, as far as we know, two integer linear programming approaches are available. The first one is a Branch-and-Cut algorithm, called B&C-LF_2 [7], which is based on the asymmetric representative formulation for GCP described in [8]. The other one [9] adapts to ECP the formulation and techniques used by Méndez–Díaz and Zabala for GCP in [6], studies its polyhedral structure and derives families of valid inequalities. Some of them have shown to be very effective as cutting planes in preliminary computational experiments.

Regarding GCP, we can find good exact algorithms which are not based on IP techniques. One of the most well known example is DSatur, proposed by Brélaz in [10]. This Branch-and-Bound algorithm has been referred in the literature several times and is still used by its simplicity, its efficiency in medium-sized graphs and the possibility of applying it at some stage in metaheuristics or in more complex exact algorithms like Branch-and-Cut ones [6]. Recently, it was shown that a modification of DSatur performs relatively well compared with many state-of-the-art algorithms based on IP techniques, showing superiority in random instances [11].

This fact encourages us to research how to modify a DSatur-based solver in order to address the ECP, which is the goal of this paper. Our approach exploits arithmetical properties inherent in equitable colorings and combines them with the techniques originally developed by Brown [12] and Brélaz [10] for DSatur, and improved by Sewell [13] and San Segundo [11]. We call it EqDSatur. A preliminary version of this algorithm with weaker pruning rules than the one analyzed in this work was already presented in [14].

The paper is organized as follows. Section 2 gives a brief summary of known DSatur-based algorithms for GCP. Section 3 shows the background math for our pruning rule. Section 4 describes an implementation of EqDSatur. Section 5 discusses methods for obtaining lower and upper bounds of the equitable chromatic number. Section 6 reports computational experiments carried out to tune up the behavior of EqDSatur, and compares our algorithm against other ones from the literature. Finally, Section 7 gives final conclusions.

We now introduce some notations and definitions employed throughout the paper. For any positive integer k, $[k]$ denotes the set $\{1, 2, \ldots, k\}$. Given a graph $G = (V, E)$, we assume the set of vertices is $V = [n]$. A graph for which every vertex is adjacent to each other is called a *complete graph*. Given $S \subset V$, we denote by $G[S]$ the subgraph of G *induced by* S. A set $Q \subset V$ is a *clique* of G if $G[Q]$ is a complete graph.

Given $u \in V$, the *neighborhood* of u is the set of vertices adjacent to u and is denoted by $N(u)$. The *closed neighborhood* of u, $N[u]$, is the set $N(u) \cup \{u\}$. The *degree of* u, $d(u)$, is the cardinality of $N(u)$. The maximum degree of vertices in G is denoted by $\Delta(G)$.

A *stable set* is a set of vertices of G no two of which are adjacent. We denote by $\alpha(G)$ the *stability number* of G, i.e. the maximum cardinality of a stable set of G. Given $S \subset V$, we also denote by $\alpha(S)$ the stability number of $G[S]$.

A *partial k-partition* of G, denoted by $\Pi = (C_1, C_2, \ldots, C_n)$, is a collection of disjoint sets such that $\cup_{j=1}^{k} C_j \subset V$ and $C_j = \varnothing$ if and only if $j \geq k+1$. We write $k(\Pi)$ to refer the number of non-empty sets in Π. We denote by $U(\Pi)$ the set of vertices not covered by the sets of Π, i.e. $U(\Pi) = V \setminus \cup_{j=1}^{k} C_j$. If $U(\Pi) = \varnothing$ we say that Π is a *k-partition*. Given $v \in V \setminus U$, we denote by $\Pi(v)$ the number of the set to which v belongs, i.e. $v \in C_{\Pi(v)}$.

A *partial k-coloring* of G is a partial k-partition $\Pi = (C_1, C_2, \ldots, C_n)$ of G such that each C_j is a stable set of G. In this context, $U(\Pi)$ is called the *set of uncolored vertices* of a partial k-coloring Π. If $U(\Pi) = \varnothing$ we say that Π is a *k-coloring*.

Given $v \in V$ and a partial k-coloring Π, let $D_\Pi(v)$ be the set of different colors assigned to the adjacent vertices of v, i.e. $D_\Pi(v) = \{\Pi(w) : w \in N(v) \setminus U(\Pi)\}$. The *saturation degree* of v in Π, $\rho_\Pi(v)$, is the cardinality of $D_\Pi(v)$ and the *set of available colors* of v, $F_\Pi(v)$, is the set of unused colors in the neighborhood of v, i.e. $F_\Pi(v) = [n] \setminus D_\Pi(v)$.

Given a partial k-partition Π, $u \in U(\Pi)$ and $j \in [k+1]$ we denote by $\Pi + \langle u, j \rangle$ to the partial partition obtained by adding u to C_j.

We say that a partial k-partition (or partial k-coloring) $\Pi = (C_1, C_2, \ldots, C_n)$ *can be extended* to a k'–partition (or k'–coloring) if there exists a k'–partition (or k'–coloring) $\Pi' = (C'_1, C'_2, \ldots, C'_n)$ which can be obtained from Π by successive applications of the operator "+". A direct consequence is that $k \leq k'$ and $C_j \subset C'_j$ for all $j \in [k]$.

We say that a k-partition or k-coloring $\Pi = (C_1, C_2, \ldots, C_n)$ of G is *equitable* if it satisfies the equity constraint, i.e.

$$\big| |C_i| - |C_j| \big| \leq 1, \quad \text{for } i, j \in [k].$$

An equitable k-coloring is also called k-eqcol for the sake of simplicity.

2. An overview of DSatur-based algorithms for GCP

The idea behind an enumerative algorithm such as DSatur is to determine early whether it is possible to extend a partial coloring to a proper coloring so that uncolored vertices are painted with available colors. In this way, the enumerative procedure avoids to explore partial colorings that will not lead to an optimal coloring, and therefore would be needlessly enumerated.

DSatur is based on a generic enumerative scheme proposed by Brown [12], outlined as follows:

Input: G a graph, Π_0 an initial partial coloring of G and Π^* an initial coloring of G.

Output: Π^* an optimal coloring of G, UB the chromatic number of G.

Algorithm: Set $UB \leftarrow k(\Pi^*)$. Then, execute Node(Π_0).

Node (Π):

Step 1. If $U(\Pi) = \varnothing$, set $UB \leftarrow k(\Pi)$, $\Pi^* \leftarrow \Pi$ and return.
Step 2. Select a vertex $u \in U(\Pi)$.
Step 3. For each color $j \in [\min\{k(\Pi)+1, UB-1\}]$ such that $j \in F_\Pi(u)$:
  Set $\Pi' \leftarrow \Pi + \langle u, j \rangle$.
  If $F_{\Pi'}(v) \cap [UB-1] \neq \varnothing$ for all $v \in U(\Pi')$, execute Node (Π').

The previous scheme only works when the initial partial coloring Π_0 can be extended to an optimal coloring. A suitable Π_0 can be computed as follows: if $Q = \{v_1, v_2, \ldots, v_q\}$ is a maximal clique of G, it is known that a q-partial coloring Π_0 such that $\Pi_0(v_i) = i$ for all $i \in [q]$ can be extended to a $\chi(G)$–coloring.

Indeed, we must know a maximal clique Q and an initial coloring Π^* in advance. Moreover, we must state the rule for choosing vertex u in Step 2 and the order in which colors from $F(u)$ have to be evaluated. From now on, we call to these criteria *vertex selection strategy* (VSS) and *color selection strategy* (CSS).

Brélaz proposed the algorithm DSatur [10] by obtaining a maximal clique Q and an initial coloring Π^* with greedy heuristics (one is SLI given in [15] and the other is contributed by himself). The vertex selection strategy, which we call DSATUR-VSS, selects the uncolored vertex with the largest saturation degree. In case of a tie, select the vertex with the largest degree. More specifically, let ρ be the maximum saturation degree of Π and T be the so called *set of candidate vertices*:

$$T = \{u \in U(\Pi) : \rho_{\Pi}(u) = \rho\}.$$

DSATUR-VSS chooses $u \in T$ that maximizes $d(u)$. In the case that more than one vertex in T has the maximum degree, unite them according to some predetermined order, e.g. its number in V.

Sewell [13] suggested a modified tie breaking rule for choosing u from the set T, called Celim (CELIM-VSS). It consists of selecting from the set of vertices tied at maximum saturation degree, the one with the maximum number of common available colors in the neighborhood of uncolored vertices. That is, choose $u \in T$ such that the value

$$celim(u) = \sum_{j \in F_{\Pi}(u)} |\{v \in N(u) \cap U(\Pi) : j \in F_{\Pi}(v)\}|$$

is the highest.

Let us note that, while DSATUR-VSS attempts to estimate future color availability through the degree of vertices, CELIM-VSS also contemplates the impact of coloring a vertex over the uncolored vertices yet. Although CELIM-VSS is more CPU intensive than DSATUR-VSS, fewer nodes are evaluated and, in the case of medium and high density instances, less time is required to reach the optimality.

A further improvement in the vertex selection strategy was recently proposed by San Segundo [11]. The criterion chooses the vertex $u \in T$ that maximizes the value

$$pass(u) = \sum_{j \in F_{\Pi}(u)} |\{v \in N(u) \cap T : j \in F_{\Pi}(v)\}|.$$

By comparing it with Sewell's criterion we may observe that CELIM-VSS attempts to minimize the number of subproblems by systematically reducing available color at deeper levels of the search tree. By constrast, San Segundo's criterion restricts this computation to the neighbors in the set of tied vertices, reducing color domains of vertices which are already known to have the least number of available colors, and so therefore more likely to require a new color at deeper levels of the search tree.

At an early stage of enumeration, the set T has many vertices and the computation of $pass(u)$ induces an overload in the strategy that, in some cases, worsens the overall performance. In order to prevent this overload, a threshold called TH is introduced by the author. If $k(\Pi) - \rho \leq TH$, he chooses from the set T, the vertex u whose value of $pass(u)$ is the highest. Otherwise, he chooses the vertex u whose degree is the highest just like DSATUR-VSS. This strategy is called Pass (PASS-VSS). Several values of this threshold were tested in [11] and $TH=3$ was settled as the best option.

This approach proved to be quite competitive with other exact algorithms for GCP from the literature.

Regarding the color selection strategy, as far as we know, all DSatur-based implementations merely consider the set of available colors in ascending order: first evaluate color 1, then color 2, and so on. We call it DSATUR-CSS.

Considering the good performance of DSatur-based algorithms for GCP, it is natural to derive an algorithm for ECP consisting of the previous Brown's scheme by changing the initial coloring in the initialization by an equitable coloring, and checking whether Π is an equitable coloring in Step 1. In summary, this simple algorithm, which we call TrivialEqDSatur, only applies the equity constraint at the leafs of the search tree in the hope that the resulting coloring is equitable. This may cause TrivialEqDSatur to explore vast regions of the search tree that will not lead to equitable colorings.

Nevertheless, the exploration of useless nodes could be avoided by checking, at each node, whether a partial coloring can be extended to an equitable coloring. In the next section, we study necessary and sufficient conditions for a partial coloring to be extended to an equitable coloring and how to implement it as part of a DSatur-based algorithm.

3. A pruning rule for the ECP

We now study arithmetical properties of the sizes of color classes in equitable colorings and how to combine them in order to propose a pruning rule for our algorithm.

From now on, for a partial k-partition $\Pi = (C_1, C_2, \ldots, C_n)$, let $M(\Pi)$ be the largest color class in Π, $T(\Pi)$ be the index of color classes in Π with size $M(\Pi)$, and $t(\Pi)$ be the cardinality of $T(\Pi)$, i.e. $M(\Pi) = \max\{|C_j| : j \in [k]\}$, $T(\Pi) = \{j \in [k] : |C_j| = M(\Pi)\}$ and $t(\Pi) = |T(\Pi)|$.

The following result fully characterizes when a partial partition can be extended to an equitable partition.

Theorem 1. *Let Π be a partial k-partition, $M = M(\Pi)$ and $t = t(\Pi)$. Then, Π can be extended to an equitable partition if and only if*

$$n \geq (M-1) \cdot k + t \tag{1}$$

Proof. Clearly, if Π can be extended to an equitable partition Π', then the classes from $T(\Pi)$ in Π' must have at least M vertices. Consequently, the classes from $[k] \setminus T(\Pi)$ in Π' must have at least $M-1$ vertices. Then, $n \geq M \cdot t + (M-1) \cdot (k-t)$ which is equivalent to (1).

On the other hand, if (1) holds then $U(\Pi)$ has enough vertices for the following procedure to get an equitable k-partition: add one by one the remaining uncolored vertices to the smallest non-empty class at each step. □

Formula (1) allows us to obtain another way of characterizing equitable colorings besides the traditional definition:

Corollary 2. *Let Π be a k-coloring of G, $M = M(\Pi)$ and $t = t(\Pi)$. Then, Π is a k-eqcol if and only if (1) holds.*

Proof. By Theorem 1, if (1) holds then Π is extended to the equitable k-partition Π itself. Since Π is already a coloring, Π is a k-eqcol. The converse is analogous. □

If we wonder when a partial coloring can be extended to an equitable coloring, it is clearly that condition (1) is necessary. However, if we know a lower bound of χ_{eq}, the condition can be tightened:

Corollary 3. *Let Π be a partial k-coloring, $M = M(\Pi)$, $t = t(\Pi)$ and LB be a lower bound of $\chi_{eq}(G)$. If Π can be extended to an equitable coloring, then*

$$n \geq (M-1) \cdot \max\{k, LB\} + t \tag{2}$$

Proof. In the case that $k \geq LB$, (2) holds by Theorem 1. Hence, we assume $k < LB$. If Π can be extended to an equitable k'–coloring Π', we have that $k' \geq \chi_{eq}(G) \geq LB$ and classes from $T(\Pi)$ in Π' must have at least M vertices. Consequently, classes from $[LB] \setminus T(\Pi)$ in Π' must have at least $M-1$ vertices. Therefore, $n \geq M \cdot t + (M-1) \cdot (LB - t)$ and (2) holds. □

We include the condition given in the previous result as a pruning rule in the Brown's scheme. Below, we sketch our approach called EqDSatur:

INPUT: G a graph, Π_0 an initial partial coloring of G, Π^* an initial equitable coloring of G and LB a lower bound of $\chi_{eq}(G)$.

OUTPUT: Π^* an optimal equitable coloring of G, $UB = \chi_{eq}(G)$.

ALGORITHM: Set $UB \leftarrow k(\Pi^*)$. Then, execute NODE(Π_0).

NODE (Π):

Step 1. If $U(\Pi) = \varnothing$, set $UB \leftarrow k(\Pi)$, $\Pi^* \leftarrow \Pi$ and return.
Step 2. Select a vertex $u \in U(\Pi)$.
Step 3. For each color $j \in [\min\{k(\Pi)+1, UB-1\}]$ such that $j \in F_\Pi(u)$:
Set $\Pi' \leftarrow \Pi + \langle u, j \rangle$.
If $n \geq (M(\Pi')-1) \cdot \max\{k(\Pi'), LB\} + t(\Pi')$ and $F_{\Pi'}(v) \cap [UB-1] \neq \varnothing$ for all $v \in U(\Pi')$, execute NODE (Π').

The following theorem shows that EQDSATUR works:

Theorem 4. *If Π_0 can be extended to a $\chi_{eq}(G)$-eqcol then EqDSatur GIVES THE VALUE OF $\chi_{eq}(G)$ into the variable UB and an optimal equitable coloring into Π^* after its execution.*

Proof. In the case that (2) does not hold, the node corresponding to Π' is not called since Π' can not be extended to an equitable coloring according to Corollary 3. Therefore, the algorithm does not prune nodes that could reach an optimal equitable coloring.

Also, each coloring reached at Step 1 is indeed an equitable coloring, due to Corollary 2 and the fact that the current coloring satisfies (2). □

4. Implementation of EqDSatur

It is clear that the scheme proposed previously is barely helpful if we do not know how to implement it in a efficient way.

Below, we propose a detailed fast implementation of EQDSATUR. Indentations are meaningful and mark the scope of the operations involved. All sets listed in the implementation are represented by global binary-valued arrays. Global variable k is the number of colors of the current partial partition.

INPUT: G a graph, Π^* an initial eqcol of G and LB a lower bound of $\chi_{eq}(G)$.

OUTPUT: Π^* an optimal eqcol of G, $UB = \chi_{eq}(G)$.

ALGORITHM:

Set $UB \leftarrow k(\Pi^*)$.
Create a partial coloring Π such that $C_i \leftarrow \{v_i\}$ for all $i \in [q]$, where $Q = \{v_1, v_2, \dots, v_q\}$ is a maximal clique of G.
Set $U(\Pi) \leftarrow V/Q$ and $k \leftarrow q$.
Execute NODE$(1, q)$.

NODE(M,t):

Step 1. If $U(\Pi) = \varnothing$, set $UB \leftarrow k$, $\Pi^* \leftarrow \Pi$ and return.
Step 2. Select a vertex $u \in U(\Pi)$.
Step 3. For each $j \in [\min\{k+1, UB-1\}]$ such that $j \in F_\Pi(u)$:
    Set $size \leftarrow |C_j|$.
    If $j \leq k$, do:
        If $size = M$, set $t' \leftarrow 1$ and $M' \leftarrow M+1$.
        If $size = M-1$, set $t' \leftarrow t+1$ and $M' \leftarrow M$.
        If $size \leq M-2$ set $t' \leftarrow t$ and $M' \leftarrow M$.
    If $j = k+1$, do:
        If $M = 1$, set $t' \leftarrow t+1$ and $M' \leftarrow M$.
        If $M \geq 2$, set $t' \leftarrow t$ and $M' \leftarrow M$.
Set $previous\_k \leftarrow k$.
Set $k \leftarrow \max\{j, k\}$.
If $n \geq (M'-1) \cdot \max\{k, LB\} + t'$, do:
        Set $C_j \leftarrow C_j \cup \{u\}$.
        Set $U(\Pi) \leftarrow U(\Pi)/\{u\}$.
        Execute NODE (M', t').
        Set $U(\Pi) \leftarrow U(\Pi) \cup \{u\}$.
        Set $C_j \leftarrow C_j/\{u\}$.
    Set $k \leftarrow previous\_k$.

We do not describe implementation details of how to update $F_\Pi(v)$ for the sake of readability, but it can be found in [11]. On the other hand, details of how to compute the clique Q and the initial equitable coloring are discussed in Section 5.

It is not hard to see that variables M and t are indeed the cardinality of the largest class and the number of color classes with size M in the current partial coloring. The update of these variables as well as $U(\Pi)$, C_j and k is performed in constant time.

Updating M and t, and checking (2) is cheap but not free. So, it becomes important to analyze if the usage of this pruning rule pays off in terms of CPU time. This task is performed in Section 6 through empirical experimentation.

5. Lower and upper bounds of $\chi_{eq}(G)$

In order to initialize EQDSATUR, it is necessary to compute bounds of the equitable chromatic number. In this section, we discuss how to obtain such values and we report some computational experiments related to them. We remark that, in particular, the lower bound LB remains constant during the enumeration, so it is essential that the value of LB be as best as possible.

5.1. Computation of lower bounds

Clearly, every equitable coloring of G is also a classic coloring of G so every lower bound of $\chi(G)$ can be used as a lower bound of $\chi_{eq}(G)$. In particular, the size of any maximal clique of G is a known lower bound of $\chi(G)$ and $\chi_{eq}(G)$. There are several ways suggested in the literature to obtain such cliques. The easiest method is, for a given graph G and a given vertex v, a greedy algorithm that includes v as the first vertex of the clique and then selects the vertex adjacent to the clique with highest degree in each step until no more vertices can be added to the clique. Furthermore, one may apply this method to different initial vertices v and choose the largest clique. In the case that two cliques of the same size are found, it is advisable to follow a suggestion made by Sewell [13]: retain the clique Q that maximizes $\sum_{q \in Q} d(q)$. The clique found with this criterion will lead to smaller initial sets $F(v)$ since those colors used by the clique will not be available for vertices v adjacent to some vertex in the clique. Let us call FINDCLIQUE(G) to this algorithm.

Let us notice that the distance between $\chi(G)$ and $\chi_{eq}(G)$ can be as far as we want. Such is the case with star graphs $K_{1,m}$ [1] (i.e. a graph $K_{1,m}$ is composed of a vertex v and a stable set S of size m such that v is adjacent to every vertex in S):

$$\chi_{eq}(K_{1,m}) - \chi(K_{1,m}) = (\lceil m/2 \rceil + 1) - 2 = \lceil m/2 \rceil - 1.$$

Therefore, it becomes essential to find other lower bounds for $\chi_{eq}(G)$ besides a maximal clique of G. Lih and Chen [16] proved that

$$\chi_{eq}(G) \geq \left\lceil \frac{n+1}{\alpha(V/N[v])+2} \right\rceil$$

for any $v \in V$. However, it requires to know the stability number of $G[V/N[v]]$, an NP-Hard problem [17]. Nevertheless, a relaxation of this value can be used instead. It is known that the cardinality of a partition in cliques of a graph is an upper bound for the stability number of that graph. Let PC_v be the cardinality of a partition in cliques of $G[V/N[v]]$. The lower the size of the partition is, the tighter the bound becomes. Let us call EQLOWBOUND(G) to the algorithm that computes the number:

$$\max\left\{\lceil\frac{n+1}{PC_v+2}\rceil : v \in V\right\},$$

where PC_v is obtained by the following greedy heuristic. Initially, let G_v be the graph $G[V/N[v]]$. We compute a maximal clique of G_v and then we delete those vertices from G_v that belong to the clique found. This simple procedure is repeated until G_v becomes empty, and PC_v is the number of cliques found.

We want to emphasize that both procedures (FindClique and EqLowBound) could be improved, thus obtaining better bounds of χ_{eq} but at the expense of spending more CPU time.

Table 1
Comparison of bounds.

n	p	Lower bound		Upper bound		% Rel. gap
		LB_{FC}	LB_{ELB}	$UB_{(3)}$	UB_{NV}	
125	0.1	4.03	3	21.13	7.67	46.5
125	0.3	6.2	5.13	49.9	20.13	68.37
125	0.5	9.1	9	75.8	33.03	71.3
125	0.7	14.13	17.2	99.3	46.67	62.53
125	0.9	31	40.67	119.7	68.33	40.1
250	0.1	4.23	3	38.43	12.27	64.87
250	0.3	6.97	6	93.67	38.17	81.13
250	0.5	10.33	11.03	146.33	65.77	82.53
250	0.7	16.33	22.63	193.47	92.87	75.23
250	0.9	38.33	63	236.7	137.17	53.17
500	0.1	4.9	4	69.3	22.5	77.9
500	0.3	7.73	7	180.2	72.23	88.83
500	0.5	11.46	13	281.9	129.57	89.5
500	0.7	18.6	28.43	378.67	184.8	84.1
500	0.9	46.57	93.63	467.57	286.1	66.73

Table 2
Tests on different vertex selection strategies.

n	p	% Solved			Av. UB		
		DSATUR	CELIM	PASS	DSATUR	CELIM	PASS
70	0.1	100	100	100	4	4	4
70	0.3	100	100	100	7.93	7.93	7.93
70	0.5	93	97	**100**	12.03	11.93	**11.83**
70	0.7	97	**100**	**100**	17.53	**17.3**	**17.3**
70	0.9	100	100	100	29.2	29.2	29.2
80	0.1	100	100	100	4.23	4.23	4.23
80	0.3	**100**	97	**100**	**8.43**	8.53	**8.43**
80	0.5	87	87	**93**	13.47	13.47	**13.2**
80	0.7	53	50	**70**	20.1	20.2	**19.53**
80	0.9	100	100	100	31.7	31.7	31.7
90	0.1	100	100	100	5	5	5
90	0.3	100	100	100	9	9	9
90	0.9	100	100	100	34.2	34.2	34.2

Table 3
Tests on different vertex selection strategies.

n	p	Av. nodes			Av. time		
		DSATUR	CELIM	PASS	DSATUR	CELIM	PASS
70	0.1	216	**168**	208	0	0	0
70	0.3	401,862	253,181	**171,448**	0.1	0.1	0.07
70	0.5	6,116,237	5,134,138	**4,702,843**	6.61	7.76	6.7
70	0.7	21,048,794	**11,175,213**	12,020,710	28.2	**23.5**	**21**
70	0.9	249,682	145,481	**138,057**	0.17	0.17	0.1
80	0.1	1132	**967**	5186	0	0	0
80	0.3	31,102,992	17,153,530	**15,495,305**	27.7	25.2	**22.7**
80	0.5	540,416,906	333,631,281	**192,172,556**	601	574	**324**
80	0.7	821,110,267	480,890,653	959,670,395	1263	1162	1817
		675165908†	**308,409,086**†	410,011,950†	1035†	**749**†	791†
80	0.9	5,513,947	**3,098,276**	3,817,790	8.2	7.6	6.57
90	0.1	4521	3186	**2875**	0	0	0
90	0.3	83,857,234	58,179,096	**32,510,740**	86.8	88.6	**52**
90	0.9	144,093,673	**71,388,770**	73,185,398	305	218	**161**

5.2. Computation of upper bounds

A known upper bound for $\chi_{eq}(G)$ is $\Delta(G)+1$ [18], but a slightly better one can be derived from a result stated in [19]: "every graph satisfying $d(u)+d(v) \le 2r+1$ for every edge (u,v), has a $(r+1)$-eqcol". From this result, it is straightforward to obtain the following relationship:

$$\chi_{eq}(G) \le \left\lceil \frac{\max\{d(u)+d(v) : (u,v) \in E\}-1}{2} \right\rceil + 1. \tag{3}$$

Another way for finding an initial upper bound is via heuristics. In our implementation, we adopt Naive [5] which is a heuristic that works well and produces good solutions. Basically, Naive generates a classic coloring with the algorithm *SL* [15] and then re-color vertices from the biggest color class to the smallest color class. When it is not possible, a new color is assigned to some vertex from the biggest class. The re-coloring procedure is repeated until an equitable coloring is reached.

5.3. Quality of the bounds

As we said above, it is important to bear in mind that the CPU time assigned to the procedures yield the bounds and how much they will impact in the enumerative algorithm. Since these procedures are fast heuristics, we are not sure whether they yield quality bounds. Next, we analyze them through experimentation.

This experiment and all the further ones shown in this paper were carried out on an Intel i5 CPU 750@2.67 GHz with Ubuntu Linux O.S. and Intel C++ Compiler.

We denote by LB_{FC} to the size of the maximal clique returned by FindClique, LB_{ELB} to the lower bound computed by EqLowBound, $UB_{(3)}$ to the upper bound given by (3) and UB_{NV} to the number of colors of the equitable coloring returned by Naive.

Random instances are generated from two parameters: the number of vertices n and the probability p that an edge is included in the graph. Let us note that p is approximately equal to the density of the random graph, i.e.

$$\frac{2|E|}{n(n-1)}.$$

Table 1 summarizes the average of the bounds over 450 randomly generated instances of different sizes (each row of the table corresponds to 30 instances). Columns 1 and 2 show the number of vertices n and probability p of the evaluated instances. Columns 3 and 6 display the average of LB_{FC}, LB_{ELB}, $UB_{(3)}$ and UB_{NV}, and Column 7 is the average of percentage of relative gap, i.e.

$$\frac{100(\min\{UB_{NV}, UB_{(3)}\} - \max\{LB_{FC}, LB_{ELB}\})}{\min\{UB_{NV}, UB_{(3)}\}}.$$

As we can see from Table 1, LB_{ELB} is particularly useful for medium and high density graphs. The time spent in the computation of the bounds (less than a second) can be considered negligible compared to the duration of the enumerative algorithm. Therefore, it is reasonable to have on hand both lower bounds and choose the best one for each case.

Regarding $UB_{(3)}$, it seems to be useless compared to UB_{NV}. Moreover, we did not find any instance such that $UB_{NV} \geq UB_{(3)}$ showing that Naive algorithm is enough to provide good upper bounds.

It is worth mentioning that medium density graphs present the worst average of relative gap. Unfortunately, this issue is transported to the enumerative algorithm making these instances the hardest to solve.

Table 4
Tests on different color section strategies.

n	p	% Solved			Av. UB		
		DSATUR	BCCOL	ORDER1	DSATUR	BCCOL	ORDER1
70	0.1	100	100	100	4	4	4
70	0.3	100	100	100	7.93	7.93	7.93
70	0.5	100	100	100	11.8	11.8	11.8
70	0.7	**100**	93	**100**	**17.3**	17.8	**17.3**
70	0.9	**100**	77	**100**	**29.2**	30.8	**29.2**
80	0.1	100	100	100	4.23	4.23	4.23
80	0.3	**100**	93	**100**	**8.43**	8.63	**8.43**
80	0.5	93	93	93	**13.2**	13.3	**13.2**
80	0.7	70	**73**	70	**19.5**	20.3	**19.5**
80	0.9	**100**	90	**100**	**31.7**	32.5	**31.7**
90	0.1	100	100	100	5	5	5
90	0.3	100	100	100	9	9	9
90	0.9	**100**	80	**100**	**34.2**	36.5	**34.2**

We also evaluated the heuristics on a set of 64 benchmark instances, of which 60 are from a subset of DIMACS COLORLIB library [20] and the remaining 4 are Kneser graphs [21]. Both COLORLIB and Kneser graphs were already used by other authors for evaluating equitable coloring algorithms (c.f. [7]).

Results are given in Tables 8 and 9. Columns 1–4 show the name of the instance, its number of vertices and edges, and its equitable chromatic number (a question mark "?" means $\chi_{eq}(G)$ is unknown so far). Columns 5–9 display the value of the lower bounds, the upper bounds and the percentage of relative gap. Values marked in boldface mean they match with $\chi_{eq}(G)$.

Similarly to the previous experiment, heuristics took less than 1 s for almost all instances. The worst case was `latin_sq_10` which took 4 s.

Let us note that optimality is reached in 6 instances, namely `anna`, `games120`, `homer`, `huck`, `jean` and `le450_25b`. Naive also is able to compute the optimal solution in 10 instances (`mug*_*`, `*-Insertions_*`, `myciel4` and `kneser7_3`). On the other hand, FindClique reaches the best lower bound in 12 instances (`zeroin.i.1`, `queen7_7`, `queen8_12`, `mulsol.i.1`, `school1_nsh`, `fpsol2.i.1`, `le*_*` and `inithx.i.1`) while EqLowBound reaches it only for `david`.

We conclude that heuristics presented in this section are reasonably fast, simple to implement, and suitable to provide good quality bounds to an exact algorithm.

6. Computational experiments

In this section, we make computational experiments in order to find the best strategies for EqDSatur and compare it against other exact algorithms. We work with random graphs with $n \in \{70, 80\}$ and $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, and with $n=90$ and $p \in \{0.1, 0.3, 0.9\}$. For each combination of n and p, we generate $T=30$ instances and we analyze the performance of our algorithm by considering the following indicators:

- *Percentage of solved instances* (% solved): An instance is considered "solved" when the time needed to reach the optimal value is at most 2 h. The percentage of solved instances is the value $100.|S|/T$ where S is the set of solved instances.
- *Average of the best upper bound reached* (Av. UB): It is the average of the upper bound obtained after the enumeration, over all T instances.
- *Average of nodes evaluated* (Av. nodes): It is the average of nodes evaluated of the search tree over the set of solved instances S.

Table 5
Tests on different color section strategies.

n	p	Av. nodes			Av. time		
		DSATUR	BCCOL	ORDER1	DSATUR	BCCOL	ORDER1
70	0.1	**208**	**208**	311	0	0	0
70	0.3	171,448	**139,351**	166,733	0.07	0.03	0.07
70	0.5	**4,702,843**	34,141,331	10,586,393	**6.7**	35.9	14.3
70	0.7	**12,020,710**	116,668,568	**12,120,596**	**21**	109	25.1
70	0.9	**138,057**	11,987,145	**138,058**	**0.1**	10.3	**0.13**
80	0.1	5186	**932**	1006	0	0	0
80	0.3	**15,495,305**	17,812,892	**15,394,052**	22.7	21.9	23
80	0.5	192,172,555	270,041,809	**179,275,549**	324	379	**318**
80	0.7	959,670,395	810,826,785	941,362,879	1817	1179	1807
		1052136994†	**923999573**†	1028274591†	1978†	**1259**†	1963†
80	0.9	**3,817,790**	42,009,653	**3,818,024**	**6.57**	37.7	**6.8**
90	0.1	**2875**	5907	2909	0	0	0
90	0.3	**32,510,740**	47,623,951	44,847,604	**52**	74.1	65.6
90	0.9	**73,185,398**	164,689,901	**73,184,947**	**161**	253	168

Table 6
Comparison between TrivialEqDSatur and EqDSatur.

n	p	% Solved		Av. UB		Av. nodes		Av. time	
		Triv.	EqDS	Triv.	EqDS	Triv.	EqDS	Triv.	EqDS
70	0.1	100	100	4	4	264,721	**208**	0	0
70	0.3	100	100	7.93	7.93	168,862,113	**171,448**	32.7	**0.07**
70	0.5	100	100	11.8	11.8	88,287,477	**4,702,843**	29.9	**6.7**
70	0.7	100	100	17.3	17.3	37,918,448	**12,020,710**	29	**21**
70	0.9	100	100	29.2	29.2	2,776,802	**138,057**	1.07	0.1
80	0.1	100	100	4.23	4.23	130,316,183	**5186**	20.6	**0**
80	0.3	100	100	8.43	8.43	345,842,251	**15,495,305**	99.6	**22.7**
80	0.5	83	**93**	13.6	**13.2**	1,614,284,274	**192,172,556**	705	**324**
80	0.7	67	**70**	19.6	**19.5**	897,961,603	959,670,395	1665	1817
						897,961,603†	828,204,878†	1665†	1585†
80	0.9	100	100	31.7	31.7	122,216,644	**3,817,790**	54	**6.57**
90	0.1	100	100	5	5	15,428,656	**2875**	2.47	**0**
90	0.3	100	100	9	9	124,572,212	**32,510,740**	75.9	**52**
90	0.9	100	100	34.2	34.2	75,124,470	**73,185,398**	169	**161**

- *Average of time elapsed* (Av. time): It is the average of time in seconds needed to solve each instance, over the set of solved instances S.

We report them on tables, where each row corresponds to a different combination of n and p, and each column displays the value of an indicator for the strategy to be compared. In general, best values are marked in boldface. We do not evaluate combinations $n=90$ with $p \in \{0.5, 0.7\}$ since DSatur-based algorithms (including ours) solves few instances in those cases and comparisons become rough. The total number of instances amounts to 390.

When we compare two strategies A and B, it may happen that the instances solved by A and B are different and the comparison of the averages of nodes and time may be ambiguous or unfair. In those cases, we consider these averages over the set of instances solved by both strategies: if S_A and S_B are the set of solved instances for A and B respectively, we also compute the average of nodes and time over the set $S_A \cap S_B$. These values are reported with a mark "†".

6.1. Vertex selection strategy

The following experiment compares an implementation of EqDSatur with the three vertex selection strategies mentioned in Section 2 namely DSATUR-VSS, CELIM-VSS and PASS-VSS. Tables 2 and 3 resume the results.

As we can see, PASS-VSS has been able to solve more instances than the other strategies. Also, PASS-VSS performs better in terms of time. Nevertheless, DSATUR-VSS and CELIM-VSS reports less time than PASS-VSS for graphs of 80 vertices and $p=0.7$. Since PASS-VSS has solved more instances than the other two strategies, we have added an extra row marked with "†" reporting averages for the three strategies over the instances that the three strategies have been able to solve simultaneously. Here, CELIM-VSS seems to be a little better than PASS-VSS. In our opinion, it is not worth considering these small improvements at the expense of solving fewer instances.

Our conclusion is that PASS-VSS is the right choice for our algorithm.

6.2. Color selection strategy

We contemplate four options:

- *DSATUR-CSS*. Consider the set of available colors in ascending order.
- *BCCOL-CSS* [6]. First consider the new color $(k+1)$ and then the set of available colors in ascending order.
- *ORDER1-CSS*. Sort color classes of Π according to their size in ascending order: $|C_{i_1}| \leq |C_{i_2}| \leq \ldots \leq |C_{i_k}|$. Then consider colors in the following order: $i_1, i_2, \ldots, i_k, k+1$.
- *ORDER2-CSS*. Do the same as in ORDER1-CSS but considering colors in the following order: $k+1, i_1, i_2, \ldots, i_k$.

BCCOL-CSS is implemented as part of the branching strategy in the Branch-and-Cut BC-Col and the idea is that it tends to find feasible colorings quickly, albeit not good since it introduces new colors to reach them. ORDER1-CSS is inspired in the heuristic presented in [22]. This rule tends to balance the sizes of color classes and finds equitable colorings early. The downside is that a QuickSort must be performed on each node. ORDER2-CSS is a mix between ORDER1-CSS and BCCOL-CSS. Since we have noticed that it does not perform as well as the others, we do not report it.

Results for DSATUR-CSS, BCCOL-CSS and ORDER1-CSS are resumed in Tables 4 and 5.

We first analyze the differences between the classical strategy DSATUR-CSS and BCCOL-CSS, where the latter performs quite well for $n=80$ and $p=0.7$. We have noticed that both strategies do not solve the same instances, hence the discrepancy between solved instances (70% and 73% respectively) and average of *UB* (19.5 and 20.3 respectively), so we have added an extra row reporting averages over the instances that both strategies have been able to solve simultaneously. Although, by inspecting the extra row, BCCOL-CSS solves the "common" instances 57% faster than DSATUR-CSS, the performance of BCCOL-CSS is worse for most of the remaining rows.

Regarding ORDER1-CSS, we can note that there are few differences between this strategy and DSATUR-CSS. Both strategies solves the same instances and reaches the same *UB* for every non-solved graph. The time used by DSATUR-CSS is slightly less than ORDER1-CSS for graphs of 70 and 90 vertices. For $n=80$ and $p \in \{30, 50\}$, ORDER1-CSS evaluates 7% and 2% less nodes respectively than DSATUR-CSS. Since ORDER1-CSS performs a QuickSort at each node, the differences in time among these strategies fall to 2% and 0.6% respectively.

We choose DSATUR-CSS for our implementation of EqDSatur, but ORDER1-CSS may be considered as an alternative strategy anywise.

Table 7
Performance of EqDSatur and CPLEX on random graphs.

n	p	% Solved		Av. LB		Av. UB			Av. time	
		CPX	EqDS	Init.	CPX	Init.	CPX	EqDS	CPX	EqDS
60	0.1	100	100	3.23	4	4.9	4	4	0	0
60	0.3	100	100	5.23	7.03	10.7	7.03	7.03	506	0
60	0.5	63	100	7.63	10.6	17.1	11	10.8	1825	0.6
60	0.7	90	100	12.7	15.7	22.7	15.8	15.7	942	1.6
60	0.9	100	100	22.8	26	32.3	26	26	1	0
70	0.1	100	100	3.5	4	5.03	4	4	0.03	0
70	0.3	50	100	5.33	7.5	12.6	8	7.93	4005	0.07
70	0.5	0	100	8.13	11	19.1	12.8	11.8	–	6.7
70	0.7	20	100	13.9	16.7	26.6	18.2	17.3	2360	21
70	0.9	100	100	25.1	29.2	37.7	29.2	29.2	258	0.1
80	0.1	100	100	3.67	4.23	5.63	4.23	4.23	1.33	0
80	0.9	90	100	27.1	31.6	42.1	31.7	31.7	659	6.57
100	0.1	100	100	3.9	5	6.87	5	5	15.5	0
120	0.1	50	100	4	5	7.73	5.5	5.1	1673	2.57

6.3. TrivialEqDSatur vs. EqDSatur

Our next experiment consists of comparing TrivialEqDSatur and EqDSatur implementations in order to verify whether the pruning rule given in Section 3 is efficient. We recall that TrivialEqDSatur is a simple modification of the standard DSatur that checks whether the colorings at the leafs of the search tree are equitable or not. Both algorithms use the same selection strategies previously chosen and the same bounds given by the heuristics proposed in Section 5 (although TrivialEqDSatur does not take advantage of the value of *LB*). Table 6 resumes the results.

We have noticed that every instance solved by TrivialEqDSatur has been solved by EqDSatur too, but not conversely. This fact led us to insert an extra row in the table for the case $n=80$ and $p=0.7$, where we report the average of nodes evaluated and time elapsed of EqDSatur for those instances that have been solved by TrivialEqDSatur.

We can observe that EqDSatur outperforms TrivialEqDSatur for all the indicators.

Table 8
COLORLIB instances (part 1).

Name	Vert.	Edges	χ_{eq}	Lower bound		Upper bound		% Rel. gap.
				LB_{FC}	LB_{ELB}	$UB_{(3)}$	UB_{NV}	
miles750	128	2113	31	30	11	64	33	9.09
miles1000	128	3216	42	40	17	87	47	14.89
miles1500	128	5198	73	69	43	107	74	6.76
zeroin.i.1	211	4100	49	**49**	3	111	51	3.92
zeroin.i.2	211	3541	36	30	4	141	51	41.18
zeroin.i.3	206	3540	36	30	4	141	49	38.78
queen6_6	36	290	7	6	5	20	10	40
queen7_7	49	476	7	**7**	6	24	12	41.67
queen8_8	64	728	9	8	8	28	18	55.56
queen8_12	96	1368	12	**12**	11	33	20	40
queen9_9	81	1056	10	9	8	32	15	40
queen10_10	100	1470	?	10	10	36	18	44.44
anna	138	493	11	**11**	3	61	**11**	0
david	87	406	30	11	**30**	59	40	25
games120	120	638	9	**9**	5	14	**9**	0
homer	561	1628	13	**13**	2	89	**13**	0
huck	74	301	11	**11**	6	40	**11**	0
jean	80	254	10	**10**	3	30	**10**	0
1-FullIns_3	30	100	4	3	3	12	7	57.14
2-FullIns_3	52	201	5	4	3	16	9	55.56
3-FullIns_3	80	346	6	5	3	20	7	28.57
4-FullIns_3	114	541	7	6	3	24	12	50
5-FullIns_3	154	792	8	7	3	28	9	22.22
1-FullIns_4	93	593	5	3	3	33	7	57.14
mug88_1	88	146	4	3	3	5	**4**	25
mug88_25	88	146	4	3	3	5	**4**	25
mug100_1	100	166	4	3	3	5	**4**	25
mug100_25	100	166	4	3	3	5	**4**	25
mulsol.i.1	197	3925	49	**49**	4	122	63	22.22
mulsol.i.2	188	3885	?	31	11	157	58	46.55
school1	385	19,095	15	14	9	278	49	71.43
school1_nsh	352	14,612	14	**14**	8	231	40	65

Table 9
COLORLIB instances (part 2) and Kneser graphs.

Name	Vert.	Edges	χ_{eq}	Lower bound		Upper bound		% Rel. gap.
				LB_{FC}	LB_{ELB}	$UB_{(3)}$	UB_{NV}	
fpsol2.i.1	496	11,654	65	**65**	3	253	85	23.53
fpsol2.i.2	451	8691	47	30	5	347	62	51.61
fpsol2.i.3	425	8688	55	30	7	347	80	62.5
1-Insertions_4	67	232	5	2	3	16	**5**	40
2-Insertions_3	37	72	4	2	3	7	**4**	25
3-Insertions_3	56	110	4	2	3	8	**4**	25
4-Insertions_3	79	156	4	2	2	9	**4**	50
DSJC125.1	125	736	5	4	3	22	8	50
DSJC125.5	125	3891	?	9	9	75	27	66.67
DSJC125.9	125	6961	?	30	42	120	66	36.36
DSJC250.1	250	3218	?	4	3	37	13	69.23
DSJC250.5	250	15,668	?	10	11	146	65	83.08
DSJC250.9	250	27,897	?	37	63	235	136	53.68
le450_5a	450	5714	5	**5**	3	41	12	58.33
le450_5b	450	5734	5	**5**	4	41	12	58.33
le450_15a	450	8168	15	**15**	5	89	18	16.67
le450_15b	450	8169	15	**15**	5	91	17	11.76
le450_25a	450	8260	25	**25**	5	118	26	3.85
le450_25b	450	8263	25	**25**	6	107	**25**	0
inithx.i.1	864	18,707	54	**54**	3	503	70	22.86
inithx.i.2	645	13,979	?	30	8	542	158	81.01
myciel4	23	71	5	2	3	9	**5**	40
myciel5	47	236	6	2	3	18	9	66.67
myciel6	95	755	?	2	3	36	11	72.73
flat300_20_0	300	21,375	?	10	11	160	81	86.42
latin_sq_10	900	307,350	?	90	82	684	460	80.43
ash331GPIA	662	4181	4	3	3	24	8	62.5
will199GPIA	701	6772	7	6	4	39	9	33.33
kneser7_2	21	105	6	3	3	11	8	62.5
kneser7_3	35	70	3	2	2	5	**3**	33.33
kneser9_4	126	315	3	2	2	6	4	50
kneser11_5	462	1386	3	2	2	7	4	50

Table 10
Performance of the algorithms on COLORLIB instances and Kneser graphs.

Name	χ_{eq}	Lower bound			Upper bound					Time			
		Init.	CPX	BCLF_2	Init.	CPX	BCLF_2	EqDS	EqDS*	CPX	BCLF_2	EqDS	EqDS*
miles750	31	30	31	31	33	31	31	33	31	0	171	–	0
miles1000	42	40	42	42	47	42	42	47	42	0	267	–	0
miles1500	73	69	73	73	74	73	73	73	73	0	13	0	0
zeroin.i.1	49	49	49	49	51	49	49	49	49	0	50	0	0
zeroin.i.2	36	30	36	36	51	36	36	51	51	2	510	–	–
zeroin.i.3	36	30	36	36	49	36	36	49	49	5	491	–	–
queen6_6	7	6	7	7	10	7	7	7	7	1	1	0	0
queen7_7	7	7	7	7	12	7	7	7	7	0	0	0	0
queen8_8	9	8	9	9	18	9	9	9	9	654	441	6	1
queen8_12	12	12	12		20	12		12	20	5		3079	–
queen9_9	10	9	9		15	11		10	10	–		475	499
queen10_10	?	10	10		18	12		13	11	–		–	–
david	30	30	30	30	40	30	30	30	30	0	13	0	0
1-FullIns_3	4	3	4	4	7	4	4	4	4	0	2	0	0
2-FullIns_3	5	4	5	5	9	5	5	5	5	0	25	1	1
3-FullIns_3	6	5	6	6	7	6	6	7	7	0	85	–	–
4-FullIns_3	7	6	7	7	12	7	7	12	7	0	72	–	–
5-FullIns_3	8	7	8	8	9	8	8	9	9	0	268	–	–
1-FullIns_4	5	3	5		7	5		5	5	28		1404	1412
mug88_1	4	3	4		4	4		4	4	1		109	120
mug88_25	4	3	4		4	4		4	4	0		56	60
mug100_1	4	3	4		4	4		4	4	1		4425	4946
mug100_25	4	3	4		4	4		4	4	1		4978	5595
mulsol.i.1	49	49	49		63	49		49	49	1		0	0
mulsol.i.2	?	31	34		58	39		58	58	–		–	–
school1	15	14	14		49	49		49	49	–		–	–
school1_nsh	14	14	14		40	14		23	40	1840		–	–
fpsol2.i.1	65	65	65		85	65		65	65	11		0	0
fpsol2.i.2	47	30	47		62	62		62	62	–		–	–
fpsol2.i.3	55	30	55		80	80		80	80	–		–	–
1-Insertions_4	5	3	4		5	5		5	5	–		1055	1088
2-Insertions_3	4	3	4		4	4		4	4	0		0	0
3-Insertions_3	4	3	4		4	4		4	4	8		1	2
4-Insertions_3	4	2	4		4	4		4	4	836		1615	1701
DSJC125.1	5	4	5		8	5		5	5	214		0	0
DSJC125.5	?	9	13		27	27		19	19	–		–	–
DSJC125.9	?	42	43		66	47		47	47	–		–	–
DSJC250.1	?	4	5		13	13		9	9	–		–	–
DSJC250.5	?	11	12		65	65		36	65	–		–	–
DSJC250.9	?	63	63		136	136		86	86	–		–	–
le450_5a	5	5	5		12	5		12	10	4558		–	–
le450_5b	5	5	5		12	5		12	12	4305		–	–
le450_15a	15	15	15		18	18		17	17	–		–	–
le450_15b	15	15	15		17	17		16	16	–		–	–
le450_25a	25	25	25		26	25		25	25	54		0	0
inithx.i.1	54	54	54		70	54		55	55	63		–	–
inithx.i.2	?	30	30		158	158		158	158	–		–	–
myciel4	5	3	5	5	5	5	5	5	5	0	5	0	0
myciel5	6	3	6		9	6		6	6	149		0	0
myciel6	?	3	6		11	7		7	8	–		–	–
flat300_20_0	?	11	11		81	81		81	81	–		–	–
latin_sq_10	?	90			460	460		460	460	–		–	–
ash331GPIA	4	3	4		8	8		8	4	–		–	1
will199GPIA	7	6	7		9	9		9	7	–		–	2
kneser7_2	6	3	6	6	8	6	6	6	6	0	6	0	0
kneser7_3	3	2	3	3	3	3	3	3	3	0	2	0	0
kneser9_4	3	2	3	3	4	3	3	3	3	0	809	0	0
kneser11_5	3	2	3		4	3		3	4	84		2128	–

6.4. *Comparing against other exact algorithms*

This subsection is devoted to compare EqDSatur against the Branch-and-Cut B&C-LF_2 described in [7] and the general purpose solver CPLEX 12.4 with the IP formulation given in [9] and the initial bounds computed by the heuristics given in Section 5.

In the first experiment, we consider 30 instances for each combination of $n \in \{60, 70\}$ and $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We also consider $n \in \{80, 100, 120\}$ with $p = 0.1$ and $n = 80$ with $p = 0.9$ since CPLEX solves very few medium-density random instances with $n \geq 80$. The total number of instances amounts to 420.

Table 7 summarizes the results, where *LB* and *UB* are averaged over all instances while the time elapsed is averaged over solved instances. A mark "–" is reported when no instance is solved. Columns called "Init." correspond to the bounds computed by the initial heuristics.

We note that our algorithm is able to solve more instances than CPLEX in considerably less time. The differences are more pronounced in medium density instances.

We do not compare EqDSatur directly against B&C-LF_2 since values reported in [7] consider different random instances. Despite this, we remark that B&C-LF_2 has failed to solve any instance with

$n = 70$ and $p \in \{0.3, 0.5\}$ whereas EqDSatur can solve instances of the same size without difficulty.

The last experiment consists of comparing EqDSatur against CPLEX and B&C-LF_2 on DIMACS COLORLIB instances and Kneser graphs proposed in Section 5, except those instances that have been already solved by the initial heuristics. Besides DSATUR-CSS, we also take into account the alternative color strategy ORDER1-CSS.

Table 10 reports the final results. Columns 1 and 2 display the name of the instance and its equitable chromatic number. Columns 3–5 and 6–10 show the bounds given by the initial heuristics and the bounds obtained by each algorithm after its execution. Finally, columns 11–14 show the time needed to solve the instance, or "–" if the algorithm is not able to solve it within the limit of 2 h. Columns called "EqDS" and "EqDS*" correspond to EqDSatur with DSATUR-CSS and ORDER1-CSS respectively.

Results for B&C-LF_2 are taken from [7]. We leave blank when the instance is not mentioned in that paper. We also recall that these results had been obtained with a slightly different platform: an 1.8 GHz AMD-Atlon machine with Linux and XPRESS 2005-a as the linear programming solver.

From the 58 evaluated instances, CPLEX has solved 38, EqDSatur with DSATUR-CSS has solved 29 and with ORDER1-CSS has solved 31. However, some of the instances not solved by both versions of EqDSatur (more precisely, `3-FullIns_3`, `4-FullIns_3` and `5-FullIns_3`) are indeed hard to solve by enumerative schemes, as reported in [11], so in our opinion EqDSatur presents the expected behavior. On the other hand, both versions of EqDSatur outperform CPLEX and B&C-LF_2 in `queen8_8`, and CPLEX in `myciel5` and `queen9_9`. In particular, the version with ORDER1-CSS outperforms B&C-LF_2 in `miles750` and `miles1000`.

Let us note that DSATUR-CSS delivers a faster algorithm than ORDER1-CSS for the set of instances solved by both. Also, it is able to solve `queen8_12` and `kneser11_5` in more than half an hour. Nevertheless, by using ORDER1-CSS, `miles750`, `miles1000`, `ash331GPIA` and `will199GPIA` can be solved without difficulty.

7. Conclusions

In this paper, we present and analyze an exact DSatur-based algorithm for ECP. We propose a pruning rule based on arithmetical properties related to equitable partitions, which has shown to be very effective. We also discuss several color and vertex selection strategies and how to obtain lower and upper bounds of the equitable chromatic number for initializing the algorithm. Finally, several experiments were carried out to conclude that our approach can tackle the resolution of random graphs better than other algorithms found in the literature so far.

References

[1] Meyer W. Equitable coloring. Am Math Mon 1973;80:920–2.

[2] Tucker A. Perfect graphs and an application to optimizing municipal services. SIAM Rev 1973;15:585–90.

[3] Das SK, Finocchi I, Petreschi R. Conflict-free star-access in parallel memory systems. J Parallel Distrib Comput 2006;66:1431–41.

[4] Pemmaraju SV. Equitable colorings extend Chernoff–Hoeffding bounds. Approximation, randomization, and combinatorial optimization: algorithms and techniques. Lecture notes in computer science 2001;2129:285–96.

[5] Furmanczyk H, Kubale M. Equitable coloring of graphs. In: Graph colorings. Providence, Rhode Island: American Mathematical Society; 2004. p. 35–53.

[6] Méndez-Díaz I, Zabala P. A branch-and-cut algorithm for graph coloring. Discret Appl Math 2006;154:826–47.

[7] Bahiense L, Frota Y, Noronha TF, Ribeiro C. A branch-and-cut algorithm for the equitable coloring problem using a formulation by representatives. Discret Appl Math 2014;164:34–46.

[8] Campêlo M, Corrêa R, Campos V. On the asymmetric representatives formulation for the vertex coloring problem. Discret Appl Math 2008;156:1097–111.

[9] Méndez Díaz I, Nasini G, Severín D. A polyhedral approach for the equitable coloring problem. Discret Appl Math 2014;164:413–26.

[10] Brélaz D. New methods to color the vertices of a graph. Commun ACM 1979;22:251–6.

[11] San-Segundo P. A new DSATUR-based algorithm for exact vertex coloring. Comput Oper Res 2012;39:1724–33.

[12] Brown JR. Chromatic scheduling and the chromatic number problem. Manag Sci (Part I) 1972;19:456–63.

[13] Sewell EC. An improved algorithm for exact graph coloring. In: Trick MA, Johnson DS, editors. Cliques, coloring and satisfiability. In: Proceedings of the second DIMACS implementation challenge, vol. 26. American Mathematical Society; 1996. p. 359–73.

[14] Méndez Díaz I, Nasini G, Severín D. An exact DSatur-based algorithm for the equitable coloring problem. Electron Note Discret Math 2013;44:281–6.

[15] Isaacson JD, Marble G, Matula DW. Graph coloring algorithms. In: Graph theory and computing. New York: Academic Press; 1972. p. 109–22.

[16] Lih K-W, Chen B-L. Equitable coloring of trees. J Comb Theory, Ser B 1994;61:83–7.

[17] Garey M, Johnson D. Computers and intractability: a guide to the theory of NP completeness. San Francisco, CA: Freeman; 1979.

[18] Hajnal A, Szemerédi E. Proof of a conjecture of P. Erdös. Comb Theory Appl 1970;2:601–23.

[19] Kierstead HA, Kostochka AV. An ore-type theorem on equitable coloring. J Comb Theory Ser B 2008;98:226–34.

[20] DIMACS COLORLIB library. ⟨http://mat.gsia.cmu.edu/COLOR03⟩.

[21] Chen B-L, Huang K-C. The equitable colorings of Kneser graphs. Taiwan J Math 2008;12:887–900.

[22] Brélaz D, Nicolier Y, de Werra D. Compactness and balancing in scheduling. Math Methods Oper Res 1977;21:65–73.