Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 25, Num. 04 (2018) 57-73

RESEARCH ARTICLE

Exact Algorithms for the Graph Coloring Problem

Algoritmos Exatos para o Problema da Coloração de Grafos

Alane Marie de Lima[1]*, Renato Carmo[2]

Abstract: The graph coloring problem is the problem of partitioning the vertices of a graph into the smallest possible set of independent sets. Since it is a well-known NP-Hard problem, it is of great interest of the computer science finding results over exact algorithms that solve it. The main algorithms of this kind, though, are scattered through the literature. In this paper, we group and contextualize some of these algorithms, which are based in Dynamic Programming, Branch-and-Bound and Integer Linear Programming. The algorithms for the first group are based in the work of Lawler, which searches maximal independent sets on each subset of vertices of a graph as the base of his algorithm. In the second group, the algorithms are based in the work of Brelaz, which adapted the DSATUR procedure to an exact version, and in the work of Zykov, which introduced the definition of Zykov trees. The third group contains the algorithms based in the work of Mehrotra and Trick, which uses the Column Generation method.

Keywords: Graph Theory — Graph Coloring — Exact Algorithms

Resumo: O problema de coloração de grafos consiste em particionar os vértices de um grafo na menor quantidade possível de conjuntos independentes. Por tratar-se de um problema NP-Difícil conhecido, é de grande interesse da computação encontrar resultados sobre algoritmos exatos para sua solução. Entretanto, os principais dentre estes algoritmos estão espalhados pela literatura. Neste artigo, agrupamos e contextualizamos alguns destes algoritmos, a saber, soluções baseadas em Programação Dinâmica, Branch-and-Bound e Programação Linear Inteira. Os algoritmos do primeiro grupo são baseados no trabalho de Lawler, que busca conjuntos independentes maximais em cada subconjunto de vértices de um grafo como base de seu algoritmo. No segundo grupo, os algoritmos são baseados no trabalho de Brelaz, que adaptou a heurística DSATUR para uma versão exata, e no trabalho de Zykov, que introduziu o conceito de árvores de Zykov. O terceiro grupo contém algoritmos baseados no trabalho de Mehrotra e Trick, que utilizaram o método Geração de Colunas.

Palavras-Chave: Teoria dos Grafos — Coloração de Grafos — Algoritmos Exatos

[1,2] *Informatics, Federal University of Parana, Brazil*
*__Corresponding author__: amlima@inf.ufpr.br
DOI: https://doi.org/10.22456/2175-2745.80721 • **Received:** 01/03/2018 • **Accepted:** 05/06/2018

1. Introduction

Graph coloring is the problem of assigning colors to the vertices of a graph in such a way that neighbor vertices are assigned with different colors. The problem of coloring a graph with the minimum possible number of colors is a fundamental $\mathcal{NP}$-Hard problem with a number of applications of interest such as timetabling, code optimization and seating plans [1].

Information on different approaches for this and related computational problems are scattered throughout the literature. We survey some of these approaches discussing their strengths and weaknesses.

The algorithms we discuss are based on three main approaches, namely, Dynamic Programming, Branch-and-Bound and Linear Programming. Besides these, we also discuss the DSATUR algorithm for graph coloring.

The text is organized as follows. In Section 1.1 we briefly state some definitions and the notation used. Section 2 discusses algorithms based in Dynamic Programming. Section 3 discusses algorithms based in Branch-and-Bound. Section 4 discusses algorithms based in Linear Programming. Finally, Section 5 contains the conclusions.

1.1 Definitions and Notation

Given a set S and an integer k, we denote by $\binom{S}{k}$ the set of subsets of S of size k.

A graph G is a pair $(V(G), E(G))$ where $V(G)$ is a finite

set (the vertices of G) and $E(G) \subseteq \binom{V(G)}{2}$ (the edges of G). Note that, according to these definitions, graphs in this text are simple, that is, have no multiple edges or loops.

Two vertices u and v are *neighbors* in G if $\{u,v\} \in E(G)$. The *degree* of vertex v in G is the number of neighbors of v in G and is denoted by $d_G(v)$.

The induced subgraph of G by a set $S \subseteq V(G)$ is denoted by $G[S]$ and the graph $G-S$ is the graph $G[V(G)-S]$. If $v \in V(G)$, we write $G-v$ instead of $G-\{v\}$.

The set $S \subset V(G)$ is *independent* in G if no vertices in S are neighbors in G and is a *maximal independent set* in G if it is not properly contained in another independent set in G. We denote by $\mathbf{I}(G)$ the set of all maximal independent sets in G.

Given an integer $k \leq |V(G)|$, a *k-coloring* of G is a partition of $V(G)$ into k independent sets in G. Each such set is called a *color* in the coloring. A graph G is k-colorable if it admits a k-coloring. The smallest k for which G is k-colorable called the *chromatic number* of G and is denoted $\chi(G)$. An *optimal coloring* of G is a $\chi(G)$–coloring of G.

1.2 Graph Coloring Problems

We define the following $\mathcal{NP}$-Hard problems:

Graph Coloring
Instance : a graph G
Answer : a $\chi(G)$-coloring of G

Chromatic Number
Instance : a graph G
Answer : $\chi(G)$

k-coloring
Instance : a graph G and $k \in \mathbb{N}$
Answer : YES, if we can color G with no more than k colors; NO, otherwise.

Graph coloring problems are polynomially solvable when the given graph G is 2-colorable. The problems are also efficiently solvable for some graph classes, where we highlight here the perfect graphs [2].

2. Dynamic Programming Algorithms

In this section, we discuss dynamic programming algorithms for the graph coloring problem. Dynamic Programming is the approach of computing the answer to an instance of a problem by computing and combining the answers to "sub-instances" of that instance (see, for example, [3, chapter 15]).

We start with an algorithm from Lawler [4] (Section 2.1) which has $\mathcal{O}^*(2.4423^n)$ running time. Then we discuss an algorithm from Eppstein [5] (Section 2.2) which, through a modification of the idea of Lawler's algorithm, improves this bound to $\mathcal{O}(2.4150^n)$. Next (Section 2.3), we discuss a modification of Eppstein's algorithm in Byskov [6] yielding an $\mathcal{O}(2.4023^n)$ algorithm.

All the above algorithms require exponential space. In Section 2.4 we discuss an $\mathcal{O}(5.283^n)$ running time algorithm from Bodlaender and Kratsch [7] requiring polynomial space.

Unless otherwise stated, the proofs of the results in sections 2.1, 2.2 and 2.3 are based on those in [6].

2.1 Lawler's Algorithm

Lawler[4] was the first to propose a dynamic programming algorithm for the graph coloring problem, as described in Algorithm 4. One may view his algorithm as based in the following result.

Theorem 1 (Wang[8])**.** *Every graph has an optimal coloring in which (at least) one of the colors is a maximal independent set.*

Proof. Let $\mathcal{C} = \{P_1, \ldots, P_k\}$ be an optimal coloring of G and let I be a maximal independent set containing P_1. Then $\{I, P_2 \setminus I, \ldots, P_k \setminus I\}$ is an optimal coloring of G and one of its colors is a maximal independent set. □

It follows from Theorem 1 that if G is a graph and $S \subseteq V(G)$, then $\chi(G[S])$ is the minimum among $1+\chi(G[S \setminus I])$ over all maximal independent sets I in $G[S]$, that is,

$$\chi(G[S]) = \begin{cases} 0, & \text{if } S = \emptyset, \\ 1 + \min\{\chi(G[S \setminus I]) : I \in \mathbf{I}(G[S])\}, & \text{otherwise.} \end{cases}$$

Algorithm 4: LAWLER(G)

```
Input: A graph G
Result: The chromatic number of G
n ← |V(G)|
X ← array indexed from 0 to 2^n − 1
X[0] ← 0
For S ← 1 to 2^n − 1
    s ← f(S)
    X[s] ← ∞
    For I ∈ I(G[S])
        i ← f(S \ I)
        If X[i] + 1 < X[s]
            X[s] ← X[i] + 1
Return X[2^n − 1]
```

For each $S \subseteq V(G)$, the chromatic number of $G[S]$ is stored in $X[f(S)]$. The function $f\colon 2^{|V(G)|} \to \{0, \ldots, 2^n - 1\}$ indexes the subsets of $V(G)$ in such a way that $f(X) < f(S)$ for all $X \subset S$ (for instance, by returning $f(S) = \sum_{v_i \in S} 2^i - 1$, where $V(G) = \{v_0, \ldots, v_{n-1}\}$ is an ordering of $V(G)$).

It should be noted that the inner loop in Algorithm 4 performs a non-trivial task, namely, enumerating all maximal independent sets of a graph. As Lawler [4] itself notes, this

can be done in time $\mathcal{O}(nmI)$ for a graph with n vertices, m edges and I maximal independent subsets [9].

Theorem 2. *Algorithm 4 runs in time $\mathcal{O}(2.4423^n nm)$ and space $\Theta(2^n)$, if the input is a graph with n vertices and m edges.*

Proof. Let G be a graph with n vertices and m edges. For each $S \subseteq V(G)$, let $T(S)$ denote the time spent in the execution of the inner loop in Algorithm 4 using the algorithm from Tsukiyama[9]. Then we know that there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that, whenever $|S| \geq n_0$,

$$T(S) \leq c\,|S|\,|E(G[S])|\,|\mathbf{I}(G(S))| \leq cnm3^{|S|/3},$$

because a graph on k vertices can have at most $3^{k/3}$ maximal independent sets [10].

Summing over all $S \subseteq V(G)$ we get

$$\begin{aligned}
\sum_{S\subseteq V(G)} T(S) &\leq \sum_{S\subseteq V(G)} cnm3^{|S|/3} = cnm \sum_{S\subseteq V(G)} 3^{|S|/3} \\
&= cnm \sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 3^{|S|/3} = cnm \sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 3^{i/3} \\
&= cnm \sum_{i=0}^{n} \binom{n}{i} 3^{i/3} = cnm \sum_{i=0}^{n} \binom{n}{i} \left(3^{1/3}\right)^i \\
&\leq cnm \left(1+3^{1/3}\right)^n \leq cnm\,(2.4423)^n.
\end{aligned}$$

Hence we can conclude that the execution time of Algorithm 4 with G as input is $\mathcal{O}(nm\,(2.4423)^n)$. □

2.2 Eppstein's Algorithm

Lawler's algorithm (Algorithm 4) had the best upper bounds for the graph coloring problem until Eppstein [5] proposed two modifications. The first one is the preprocessing of the 3-colorable subgraphs of the input graph. The other one is filling in vector X (the dynamic processing table) in a different order which allows for skipping the processing of maximal independent sets beyond a certain size.

Let us start with the following result.

Theorem 3 (Madsen, Nielsen and Skjernaa [11])**.** *Let G be a graph and let $J' \subseteq V(G)$ be such that $G[J']$ is a maximal k-colorable subgraph of G. For every $0 \leq k_1 < k$ there is a set $J \subseteq J'$ such that $G[J]$ is a maximal k_1-colorable subgraph of $G[J']$ and $G[J' \setminus J]$ is a maximal $(k-k_1)$-colorable subgraph of $G-J$.*

Proof. Let G be a graph and let $J' \subseteq V(G)$ be such that $G[J']$ is a maximal k-colorable subgraph of G. Given $k_1 \leq k$, let $\{I_1,\ldots,I_k\}$ be an optimal coloring of $G[J']$ in such a way that $|I_1| \geq |I_2| \geq \ldots \geq |I_k|$ and $J := \bigcup_{i=1}^{k_1} I_i$ is the largest possible. Then $G[J]$ is a maximal k_1-colorable subgraph of $G[J']$. Because $G[J']$ is maximal, there can be no vertex $v \in V(G) \setminus J'$ that can be added either to $G[J]$ or to $G[J' \setminus J]$ so that $G[J' \cup \{v\}]$ remains k-colorable. Hence, $G[J' \setminus J]$ is a maximal $(k-k_1)$-colorable subgraph in $G-J$.

□

It is possible to prove (see [12, sec. 3.2.2]) that if $k_1 = k-1$ then $G[J' \setminus J]$ is a maximal independent set in $G-J$. The proof is similar to the one of Theorem 3 with the chosen optimal coloring being one in which I_k has the smallest possible size.

From Theorem 3, we have that if $G[J']$ is a maximal k-colorable subgraph of G, then it has an optimal coloring $\{I_1,\ldots,I_k\}$ such that I_k has the smallest size as possible. Then we have a set J such that $J = J' \setminus I_k$ and $G[J]$ is $(k-1)$-colorable. The same way as $G[J']$, the graph $G[J]$ has a color I_{k-1} in one of its optimal colorings such that I_{k-1} has the smallest size as possible. Hence,

$$|J| = \sum_{i=1}^{\chi(G[J])} |I_i| \geq \sum_{i=1}^{\chi(G[J])} |I_{k-1}| = |I_{k-1}|\chi(G[J])$$

Since $|I_k| \leq |I_{k-1}|$, then $|I_k| \leq |J|/\chi(G[J])$. Hence, for each $S \subseteq V(G)$, the value of $\chi(G[S \cup I])$ is the minimum among $1+\chi(G[S])$ over all maximal independent sets $I \leq |S|/\chi(G[S])$, that is,

$$\chi(G[S \cup I]) : I \in \mathbf{I}(G-S) = \begin{cases} 0, & \text{if } S \cup I = \emptyset, \\ 1+\min\{\chi(G[S])\}, & \text{otherwise.} \end{cases}$$

The chromatic number of $G[S]$ is stored in $X[f(S)]$, for each $S \subseteq V(G)$. The function $f(S)$ is the one defined in Section 2.1. The function $c(S)$ associates each S to its corresponding graph $G[S]$.

Algorithm 5: EPPSTEIN(G)

```
Input: A graph G
Result: The chromatic number of G
n ← |V(G)|
X ← array indexed from 0 to 2^n − 1
X[0] ← 0
For S ← 1 to 2^n − 1
    i ← f(S)
    Run the algorithm of Beigel and Eppstein [13] in
      G[S].
    If χ(G[c(S)]) ≤ 3
        X[i] ← χ(G[c(S)])
    Else
        X[i] ← ∞
For S ← 1 to 2^n − 1
    i ← f(S)
    If 3 ≤ X[i] < ∞
        For I ∈ I(G − S) such that |I| ≤ |S|/X[S]
            j ← f(S ∪ I)
            If X[i] + 1 < X[j]
                X[j] ← X[i] + 1
Return X[2^n − 1]
```

Theorem 4. *Algorithm 5 runs in time $\mathcal{O}(2.4150^n)$ and space $\mathcal{O}(2^n)$.*

Proof. Let G be a graph with n vertices and m edges. For each $S \subseteq V(G)$, let $T(S)$ denote the time spent in the preprocessing of the 3-colorable subgraphs using the algorithm from Beigel and Eppstein[13], which runs in $\mathcal{O}(1.3289^n)$. Then we know that there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that, whenever $|S| \geq n_0$,

$$T(S) \leq c1.3289^{|S|}$$

Summing over all $S \subseteq V(G)$ we get

$$\begin{aligned}
&\sum_{S\subseteq V(G)} T(S) \leq \sum_{S\subseteq V(G)} c1.3289^{|S|} = c \sum_{S\subseteq V(G)} 1.3289^{|S|} \\
&= c\sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 1.3289^{|S|} = c\sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 1.3289^{i} \\
&= c\sum_{i=0}^{n} \binom{n}{i} 1.3289^{i} \leq c\,(1+1.3289)^n \leq c\,(2.3289^n)\,.
\end{aligned}$$

Let $T'(S)$ be the time to process the maximal independent sets of size at most k in $G[S]$, for $k \in \mathbb{N}$. Then, there exist $c' > 0$ and $n_1 \in \mathbb{N}$ such that, whenever $|S| \geq n_1$,

$$\begin{aligned}
&\sum_{S\subseteq V(G)} T'(S) \leq \sum_{S\subseteq V(G)} c'3^{\frac{4|S|}{3}-(n-|S|)}4^{(n-|S|)-3\frac{|S|}{3}} \\
&= c'\sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 3^{\frac{4|S|}{3}-(n-|S|)}4^{(n-|S|)-3\frac{|S|}{3}} \\
&= c'\sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 3^{\frac{4i}{3}-(n-i)}4^{(n-i)-3\frac{i}{3}} \\
&= c'\sum_{i=0}^{n} \binom{n}{i} 3^{\frac{7i}{3}-n}4^{n-2i} \leq c'\left(\frac{4}{3}\right)^n \left(1+\frac{3^{\frac{7}{3}}}{4^2}\right)^n \\
&\qquad\qquad \leq c'\left(\frac{4}{3}+\frac{3^{4/3}}{4}\right)^n \leq c'(2.4150^n)
\end{aligned}$$

because the number of maximal independent sets of size at most k, for $k \in \mathbb{N}$, is

$$\lfloor n/k \rfloor^{(\lfloor n/k \rfloor+1)k-n}(\lfloor n/k \rfloor+1)^{n-\lfloor n/k \rfloor k}$$

and Eppstein[5] proves that these maximal independent sets can be found in $\mathcal{O}(3^{4k-n}4^{n-3k})$.

Hence, the execution time of Algorithm 5 is $\mathcal{O}(2.4150^n)$. □

2.2.1 Eppstein's Algorithm for an Optimal Coloring

Eppstein [5] proposed Algorithm 6 for finding an optimal coloring of a graph G.

Let $\chi(G) = k$. The algorithm searches for a maximal k'-colorable graph $G[T]$ in G, for $k' \in \{1,2,..,k\}$ and $T \subset V(G)$. If $k' = k-1$, then by Theorem 3, the vertices in $G-T$ constitute a maximal independent set I that can be removed from the graph. This process is repeated for the remaining vertices of G until an empty set be found.

Algorithm 6: EPPSTEINOPTCOLOR(G)

Input: A graph G
Result: An optimal coloring of G
$X \leftarrow$ array calculated in Algorithm 5
$S \leftarrow V(G)$
For $T \leftarrow 2^n - 1$ *to* 0
  $s \leftarrow f(S)$
  $t \leftarrow f(T)$
  $i \leftarrow f(S \setminus T)$
  If $T \subset S$ *and* $X[i] = 1$ *and* $X[t] = X[s] - 1$ *then*
    Set the same color to every vertex of $S \setminus T$.
    $S \leftarrow T$

The vertices subsets T and S are represented as binary arrays as in previous algorithms. Function $f(S)$ is the one defined in Section 2.1.

Since the inner loop of the algorithm is $\mathcal{O}(2^n)$ and each instruction inside of it is constant in time, then Algorithm 6 is $\mathcal{O}(2.4150^n)$ (because of the processing of the array X).

2.2.2 Beigel and Eppstein Algorithm for the 3-coloring

The algorithm used in the stage of preprocessing is the one of Beigel and Eppstein [13], which has complexity $\mathcal{O}(1.3289^n)$. The algorithm is based in a reduction from the Graph Coloring problem to a particular restriction of the Constraint Satisfaction problem (CSP) named $(3,2)$–CSP.

Given positive integers a and b, we define the (a,b)-CSP problem as follows.

(a,b)-CSP

Instance :
a triple (X,D,R), of disjoint finite sets which are called, respectively, the set of variables, the set of values and the set of constraints. Each constraint in R a pair (t,f), where t is a b-tuple of variables and f is a relation of b values from D.
Answer : a valuation of the variables that does not violate any of the constraints..

Beigel and Eppstein [13] use backtracking and polynomial time reductions of an instance to solve the CSP. Lemma 1 contains one of these reductions, which is the main one among them. We describe Lemma 1, which was stated and proved by the authors, and we give a brief idea of the 3-coloring algorithm.

The value that will be assigned to each variable is limited to at most a elements of D. The constraints describe the combinations of values that each b-uple cannot have simultaneously. In the case of the instances of the graph coloring problem, we can describe them as $(3,2)$-CSP instances where each variable represents a vertex and the set of values corresponds to the set of colors that will be assigned to each vertex. Since we are solving the 3-coloring problem, then each variable is limited to at most 3 colors of D. Besides, each constraint will have

two variables because it corresponds to an edge of the given graph and the respective vertices that cannot have the same color. Figure 1 shows an example of this reduction, where $X = \{A,B,C,D\}$, the domain of the values is $\{0,1,2\}$ and

$$R = \begin{array}{l} \{(A=1,B=1),(A=2,B=2),(A=3,B=3),\\ (A=1,C=1),(A=2,C=2),(A=3,C=3),\\ (B=1,C=1),(B=2,C=2),(B=3,C=3),\\ (B=1,D=1),(B=2,D=2),(B=3,D=3),\\ (C=1,D=1),(C=2,D=2),(C=3,D=3)\} \end{array}$$

The colors 0,1 and 2 are represented by the "black", "dots" and "grid" patterns, respectively.

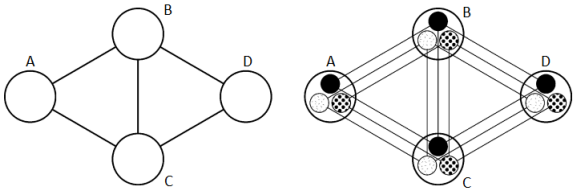

Figure 1. Example of a 3-coloring (adapted from Beigel and Eppstein [13])

Lemma 1. *Given a (X,D,R) instance of the $(a,2)$-CSP problem, let $v \in X$ be a variable such that only two colors of D are allowed to v. We can get a (X',D',R') instance from (X,D,R) with one less variable, such that (X',D',R') does not contain v and any optimal solution to this instance is also optimal for (X,D,R).*

Proof. Let $x,y \in X$ be two variables of (X,D,R). Let v be a variable limited to only two values h and i of D. Without loss of generality, let $\{(y = w),(v = h)\}$ and $\{(x = z),(v = i)\}$ be constraints of R such that w and z are values of D. If y and x receive colors w and z at the same time, respectively, then there will be no possible color to be assigned to v. Therefore, we can avoid this adding the constraint $\{(y = w),(x = z)\}$. Let (X',D',R') be the instance obtained from (X,D,R) with this new constraint and without the variable v. Hence, any optimal solution to (X',D',R') is also an optimal solution for (X,D,R) setting h or i to v. □

The algorithm of Beigel and Eppstein [13] for the 3-coloring problem turns a graph G as input into a graph G' (as represented in Figure 1), that corresponds to the reduction of the original instance into a CSP instance. The main idea of the algorithm consists on finding a subset $T \subset V(G')$, such that T is small in relation to $|V(G')|$ and it has a large set of neighbors, that we denote by N. Besides, $G[T]$ must be a tree. Supposing that the original graph G is 3-colorable, then we can ensure that each neighbor of a vertex in T is limited to 1 or 2 values of the domain in G'.

We color all the vertices in T choosing one of its $3^{|T|}$ possible proper colorings. Each $v \in N$ will be limited to at most two possible colors of the domain, since each one of these vertices has a colored neighbor in T. By Lemma 1, the vertices in N can be removed from the CSP instance. The resulting subgraph formed by $V(G') \setminus \{T \cup N\}$ constitutes a $(3,2)$-CSP instance that is solved by a backtracking algorithm proposed by Beigel and Epsstein [13]. The valuation of the variables in the optimal solution of this instance corresponds to the color assignment of the vertices in the original graph.

This algorithm has complexity $\mathcal{O}(1.3289^n)$.

2.3 Byskov's Algorithm

Byskov's algorithm [6] (Algorithm 8) is very similar to the one of Eppstein [5]. It also searches for the 3- colorable subgraphs of G and for all the maximal independent sets $I \subseteq G-S$. The improvement consists on searching for the 4-colorable subgraphs of G after finding the 3-colorable ones. This modification leads to an algorithm of complexity $\mathcal{O}(2.4023^n)$ (Theorem 5), which has the best results for the worst case analysis of dynamic programming algorithms for the graph coloring problem.

Algorithm 8: BYSKOV(G)

Input: A graph G
Result: The chromatic number of G
$n \leftarrow |V(G)|$
$X \leftarrow$ an array indexed from 0 to 2^n-1
$X[0] \leftarrow 0$
For $S \leftarrow 1$ *to* 2^n-1
  Run the algorithm of Beigel and Eppstein [13] in $G[S]$ to find the 3-colorable subgraphs of G like Eppstein's algorithm [5].
For $I \in \boldsymbol{I}(G)$
  For *all* $S \subseteq (V(G) \setminus I)$
    $i \leftarrow f(S)$
    If $X[i] = 3$
      $j \leftarrow f(S \cup I)$
      If $X[j] > 4$
        $X[j] \leftarrow 4$
For $S \leftarrow 1$ *to* 2^n-1
  $i \leftarrow f(S)$
  If $4 \leq X[i] < \infty$
    For $I \in \boldsymbol{I}(G-S)$ *such that* $|I| \leq |S|/X[S]$
      $j \leftarrow f(S \cup I)$
      If $X[j] > X[i]+1$
        $X[j] \leftarrow X[i]+1$
Return $X[2^n-1]$

Theorem 5. *Byskov's algorithm [6] runs in time $\mathcal{O}(2.4023^n)$ and space $\mathcal{O}(2^n)$, if the input is a graph G with n vertices and m edges.*

Proof. Let $I_k(G)$ be the set of all maximal independent sets of G that have size at most k. The time to find the 4-colorable subgraphs $G[S]$ of G corresponds to

$$\sum_{I \subseteq G} \sum_{S \subseteq (V(G) \setminus I)} 1 = \sum_{k=1}^{n} \sum_{I_k \in I_k(G)} \sum_{S \subseteq (V(G) \setminus I_k)} 1 = \sum_{k=1}^{n} |I_k(G)| 2^{n-k}$$

According to Byskov [14], the maximum number of maximal independent sets that have size at most k in a graph is

$$d^{(d+1)k-n}(d+1)^{n-dk}, \text{ for any } d \geq 3 \text{ such that } d \in \mathbb{N}_*$$

This bound is tight when $n/d \leq k \leq n(d+1)$. So, we have

$$\sum_{k=1}^{n} |I_k(G)|2^{n-k} \leq \sum_{k=1}^{n} d^{(d+1)k-n}(d+1)^{n-dk}2^{n-k}.$$

We denote $y(n,k) = d^{(d+1)k-n}(d+1)^{n-dk}2^{n-k}$. The maximum point of this function is attained when $k = n/5$, so we can divide the function in this point:

$$y(n,k) = \sum_{k=1}^{\lfloor n/5 \rfloor} y(n,k) + \sum_{k=\lfloor n/5 \rfloor+1}^{n} y(n,k).$$

Setting $k = n/5$, we obtain $d = 5$ in the left part, since $n/d \leq k \leq n/5$. In the other part, we obtain $d = 4$, since $n/5 < k \leq n/(d+1)$. Hence, we have

$$\sum_{k=1}^{\lfloor n/5 \rfloor} y(n,k) = \sum_{k=1}^{\lfloor n/5 \rfloor} 5^{6k-n}6^{n-5k}2^{n-k}.$$

$$\sum_{k=\lfloor n/5 \rfloor+1}^{n} y(n,k) = \sum_{k=\lfloor n/5 \rfloor+1}^{n} 4^{5k-n}5^{n-4k}2^{n-k}.$$

Both of these sums are $\mathcal{O}(2.4023^n)$. Then, the time to find all the 4-colorable subgraphs of G is $\mathcal{O}(2.4023^n)$.

The running time of the last loop of the algorithm corresponds to $\mathcal{O}(2.3814^n)$. The proof is similar to the second part of Eppstein's algorithm [5] (Theorem 4), but the bound of the sum is $4^{5k-n}5^{n-4k}$ and $|I| \leq |S|/4$. □

2.4 Bodlaender and Kratsch Algorithm

As we mentioned before, the difference from the algorithm of Bodlaender and Kratsch [7] in relation to the other ones is that although it is much less efficient in time, polynomial memory is required in their algorithm (Algorithm 9). They defined the Lemma 2 as the base of their work.

Lemma 2. *Let G be a graph such that $n = |V(G)|$ and let $0 < \alpha < 1$. For all $S \subseteq V(G)$, the chromatic number of $G[S]$ corresponds to*

$$\chi(G[S]) = \begin{cases} 1 + \min\{\chi(G-S)\}, \textit{ for all } S \subseteq V(G) \\ \textit{if } |S| \geq \alpha n \textit{ and } S \textit{ is a maximal independent set.} \\ \\ \min\{\chi(G[S]) + \chi(G-S)\}, \textit{ for all } S \subseteq V(G) \\ \textit{such that } (n-\alpha n)/2 \leq |S| \leq n/2 \end{cases} \quad (1)$$

Proof. Let $P = (P_1, P_2, \ldots, P_k)$ be an optimal coloring of G. Let P_i be a color such that $|P_i| \geq \alpha n$ for some $i \in \{1, \ldots, k\}$. Either P_i is a maximal class (and then $S = P_i$) or it is a subset of another maximal set P_i' (and then $S = P_i'$). Then, the chromatic number of G is equal to 1+min$\{\chi(G-S)\}$ (Theorem 1).

Otherwise, if all $|P_i| < \alpha n$ for $i \in \{1, \ldots, k\}$, then there is a subset $S \subseteq V(G)$ where every color of P is either in S or in $G-S$. Then, the chromatic number of G corresponds to $\min\{\chi(G[S]) + \chi(G-S)\}$ (Theorem 3).

Let $|P_1| \leq |P_2| \leq \cdots \leq |P_k|$ be an ordering of the colors. Choosing the first q colors of this ordering such that $S = P_1 + P_2 + \cdots + P_q$ and $|S| \leq n/2$ for some $q \in \{1, \ldots, k\}$, we have that either S and $V(G) \setminus S$ have the same size (that is, $n/2$) or $|S| < |V(G) \setminus S|$. In this case, the difference between S and its complement is αn. Hence, $|S| = (n - \alpha n)/2$ and $|V(G) \setminus S| = (n + \alpha)/2$. Then, $n/2 \geq |S| \geq (n - \alpha)/2$. □

Algorithm 9: $\chi(G, \alpha)$

```
Input: A graph G and 0 < α < 1
Result: The chromatic number k of G
n ← |V(G)|
k ← n
For all S ⊆ V(G)
    If S is maximal independent set |S| ≥ αn
        If k > 1 + χ(G − S, α)
            k ← 1 + χ(G − S, α)
    If (n − αn)/2 ≤ |S| ≤ n/2
        If k > χ(G[S], α) + χ(G − S, α)
            k ← χ(G[S], α) + χ(G − S, α)
Return k
```

The Algorithm 9 has time $\mathcal{O}(5.283^n)$ and this value was obtained for $\alpha = 0.19903$.

3. Branch-and-Bound Algorithms

Branch-and-bound algorithms solve an optimization problem by a systematic enumeration of its possible solutions. The state space search constitutes a tree where each branch forms a possible solution. A point of a solution, that is, a node of the tree, is branched only if its value is less or equal than a global upper bound[1]. Algorithms of this kind for the graph coloring problem are based in the work of Brelaz [15] and Zykov [16], described in subsections 3.1 and 3.2, respectively.

3.1 Brelaz's Algorithm

The Branch-and-Bound algorithm of Brelaz [15] is based on his DSATUR greedy procedure for determining an upper bound to the chromatic number. The author defined the *degree of saturation* of a vertex, which is the number of different colors that are assigned to its neighbors in a coloring.

[1] We are dealing with minimization problems here. In maximization problems, the point of the solution is branched only if its value is greater or equal than a global lower bound.

Each color has an index $i \in \mathbb{N}$. At each step of the heuristic, the vertex with the greatest degree of saturation is selected to receive the color such that i is minimum. If more than one vertex has the same degree of saturation, then the vertex with the greatest number of neighbors is chosen as a tie breaker. If a tie occurs in this case again, then one of the tied vertices is chosen at random. A description to the heuristic is given in Algorithm 10.

Algorithm 10: DSATUR(G)

Input: A graph G
Result: A coloring ζ of G
$n = |V(G)|$
$\zeta \leftarrow \emptyset$
I_i is color of G, for $1 \leq i \leq n$.
Get an ordering $(v_1, v_2, \ldots, v_n)$ for $V(G)$, such that $d_G(v_i) \geq d_G(v_{i+1})$, for $i \in \{1, \ldots, n\}$.
Add v_1 to I_1.
For *all* $i \leftarrow 1$ *to* n
$I_i \leftarrow \emptyset$
While *there are uncolored vertices*
Select the uncolored vertex v that has the greatest degree of saturation. If there are more than one vertex with the greatest degree of saturation, then choose the one that has the greatest $d_G(v)$. If a tie occurs again, then choose one of these vertices at random.

$j \leftarrow 1$
While v *is uncolored and* $k \leq n$
If $N_G(v) \cap I_j = \emptyset$
Add v to I_j.
Add I_j to ζ.
Else
$j \leftarrow j+1$
Return ζ

The exact version of the DSATUR routine is an adaptation of Brown's algorithm [17], which we describe in Algorithm 11. Further modifications have been done by Sewell [18], San Segundo [19] and Furini, Gabrel and Ternier [20].

In the adaptation of Algorithm 11 as the exact DSATUR, the greedy DSATUR routine is executed at first to find an initial coloring and an upper bound k. If G is a graph, then $v \in V(G)$ is the vertex that was assigned with the color that has index k in this initial solution. The algorithm tries to improve the upper bound coming back to a vertex u that was colored before v in the previous solution and that can be assigned with a different color l, such that $l < k$. The algorithm then selects the other vertices according to the DSATUR criteria to recolor.

Every time the algorithm finds a complete coloring that uses less than k colors, the upper bound is updated and the algorithm tries to improve the current solution again. Otherwise, if some vertex in a partial solution cannot be colored with a color that have index less than k, then it is not necessary to continue this coloring. The algorithm comes back to other point of the solution tree to recolor the vertices. If this point is the root node of the tree, then the execution stops.

Figure 3 is an example of the Branch-and-Bound DSATUR on the graph of the Figure 2. Each node of the tree and its predecessors forms a partial solution. For example: the nodes where the vertices A, F and B were assigned with colors 1,2 and 2, respectively, forms a 2-coloring. The value q is a lower bound for a partial solution and U_i is the set of colors that can be assigned to the vertex i, that is, the colors in $\{1,2,\ldots,q+1\}$ that was not used by a colored neighbor of i and that are not scratched in the Figure 3.

In Figure 3, for instance, an initial upper bound $k = 4$ and a 4-coloring was found in the steps 1 to 8. Since the vertex E was the one that got the color 4 in the step 6, then the algorithm verifies if there is another color that could be assigned to the vertex D. Since there is no other possible color, then the algorithm checks if there is another color for vertex C. In this case, the color 3 is a feasible one. The algorithm recolor the other vertices from this point.

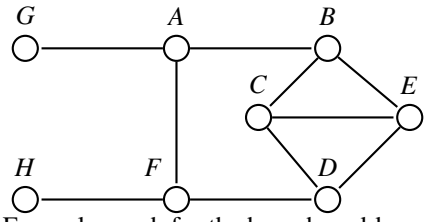

Figure 2. Example graph for the branch-and-bound DSATUR algorithm

The most recent adaptation of Brelaz's algorithm [15] can be found in the work of Sewell [21] and San Segundo [19]. They proposed new tie breakers in the step of choosing the vertex with the greatest degree of saturation.

The algorithm of Sewell [21] chooses the vertex that shares the greatest amount of available colors with its neighbors in the uncolored subgraph, while the algorithm of San Segundo [19] chooses the vertex that shares the greatest amount of colors with the vertices that have the same greatest degree of saturation. Furini, Gabrel and Ternier[20] also made an adaptation of Brelaz's algorithm, although their modification was done in the updating step of the global upper bound for the chromatic number.

Tests of the algorithms of Brelaz [15] and Sewell [21] have been done by San Segundo [19]. The algorithms were executed in random graphs with at most 80 vertices and in some DIMACS [22] instances. The author concluded that the three algorithms have similar results, except in graphs of densities up to 0.7, which [19] is more efficient. In almost all DIMACS instances, San Segundo's algorithm [19] had the best performance. Results in [20] are similar to the ones in [19]. For the random graphs instances, the algorithm of Furini, Gabrel and Ternier [20] had better results to graphs from 75 to 80 vertices, but the computational costs in relation

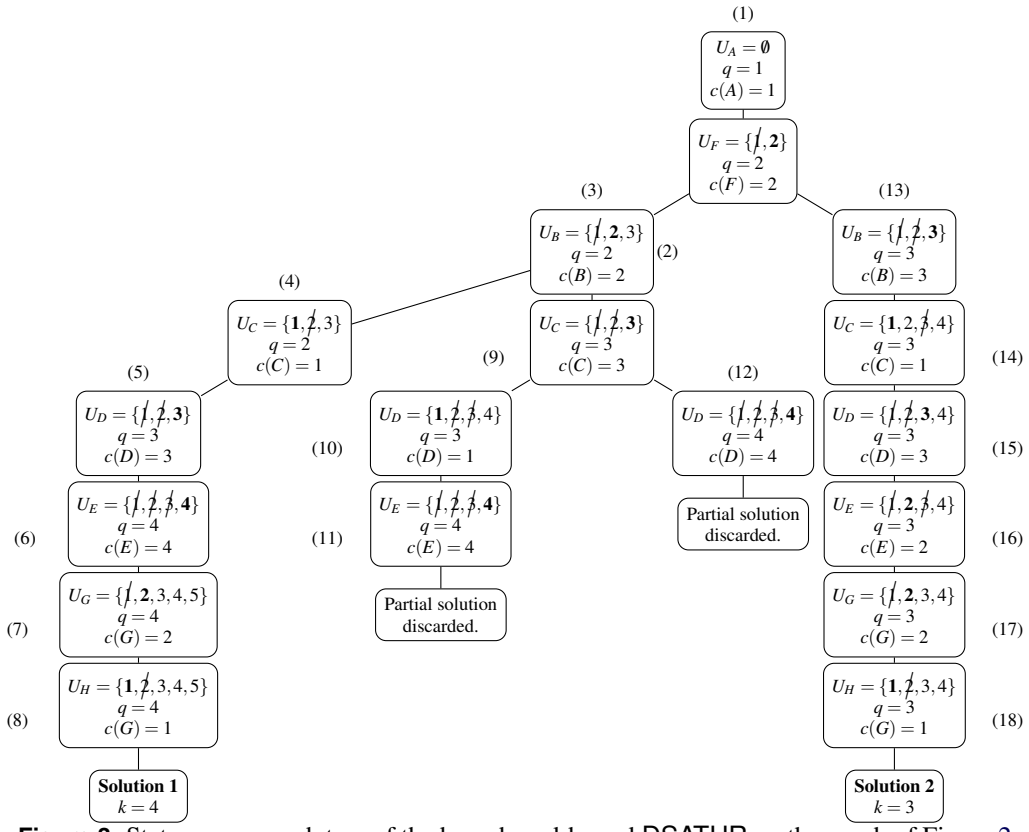

Figure 3. State space search tree of the branch-and-bound DSATUR on the graph of Figure 2

to [19] is small.

3.2 Zykov's Algorithm

The algorithm of Zykov [16] (Algorithms 12 and 13 [23]) is based in his recurrence which states that, if G is a graph and for any pair of vertices $x, y \in V(G)$ that do not share an edge, an optimal coloring of G can either assign the same color to x and y or not (Theorem 6).

In this section, we first show the basis of Zykov's algorithm, which he called *a Zykov tree*. Then, we discuss about the analysis of the worst case of the algorithm (Theorem 7), based in [23], and some of the results found in the literature.

3.2.1 Zykov Trees

Zykov [16] has stated we can obtain two new graphs from G, for a given pair of vertices $x, y \in V(G)$ that are not neighbors. One of these graphs will contract x and y into a single vertex, while the other will create an edge between x and y. The chromatic number of G corresponds to the minimum chromatic number of one of these graphs. In other words, an optimal coloring of the first graph assign the same color to x and y, while an optimal coloring of the second graph assign different colors to x and y.

Definition 1. *For two vertices $x, y \in V(G)$ that do not share and edge, a* contraction *in G produces a new graph G'_{xy} given by*

$$V(G'_{xy}) = V(G) \setminus \{x,y\} \cup \{z\}$$

$$E(G'_{xy}) = \left\{ \begin{array}{l} \{u,v\} \in E(G) \text{ such that } x \notin \{u,v\} \text{ and } y \notin \{u,v\} \\ \cup \\ \{u,z\} \text{ such that } \{u,x\} \in E(G) \text{ or } \{u,y\} \in E(G) \end{array} \right\}$$

For two vertices $x, y \in V(G)$ that do not share and edge, an addition *in G produces a new graph G''_{xy} given by*

$$V(G''_{xy}) = V(G)$$
$$E(G''_{xy}) = E(G) \cup \{\{x,y\}\}$$

Theorem 6. *The chromatic number of G is given by the recurrence*

$$\chi(G) = \min\left\{\chi(G'_{xy}), \chi(G''_{xy})\right\}$$
$$\text{such that } x, y \in V(G) \text{ and } \{x,y\} \notin E(G)$$

The recurrence of Theorem 6 builds a binary tree called *Zykov tree* where its leaf nodes are cliques and the chromatic number of the graph is given by the smallest clique of the tree.

Zykov trees have exactly one branch formed only by contraction operations, where the size of the clique of this branch is an upper bound q for the chromatic number. This bound is updated each time a better coloring than the current one is

Algorithm 11: BROWN(G)

Input: A graph G
Result: The optimal coloring c of G
$n \leftarrow |V(G)|$.
Get an ordering $(v_1, v_2, \ldots, v_n)$ to $V(G)$, such that $d_G(v_i) \geq d_G(v_{i+1})$ for all $i \in \{1, \ldots, n\}$.
Set color 1 to v_1.
$i \leftarrow 2$
$k \leftarrow n$
$q \leftarrow 1$
$U_1 \leftarrow \emptyset$
$l_1 \leftarrow 1$
$updateU \leftarrow TRUE$
While $i > 1$
//Solutions are generated while the root of the solution tree is not reached.
If $updateU = TRUE$
Calculate the set U_i, such that U_i has the colors in $\{1, 2, \ldots, q+1\}$ minus the ones that are not used by the neighbors of v_i.
If $U_i = \emptyset$
$i \leftarrow i - 1$
$q \leftarrow l_i$
$updateU \leftarrow FALSE$
Else
Choose $j \in U_i$, such that j has the minimum value as possible, and set color j to the vertex v_i.
Delete the color j of the set U_i.
If *the color j is smaller than the upper bound k*
If *the color j is greater than the lower bound q*
$q \leftarrow q + 1$
If $i = n$
Store the current solution and set $k \leftarrow q$.
Find the smallest index j such that the color of v_j is equals to k.
$i \leftarrow j - 1$
$q \leftarrow k - 1$
$updateU \leftarrow FALSE$
Else
$l_i \leftarrow q$
$i \leftarrow i + 1$ //a new vertex is selected to be colored
$updateU \leftarrow TRUE$
Else
$i \leftarrow i - 1$
$q \leftarrow l_i$
$updateU \leftarrow FALSE$
Return *A function* $c : V(G) \rightarrow \{1, \ldots, k\}$

found. In a Branch-and-Bound version of Zykov's algorithm, operations of contractions and additions will happen only in graphs that do not have a q-clique on its structure. Figure 4 shows an example of the Algorithms 12 and 13 on the graph represented in the tree's root.

Algorithm 12: COLOR(G)

Data: A graph G
Result: The minimum value q
$n \leftarrow |V(G)|$
If *G is a complete graph*
$q \leftarrow \min\{n, q\}$
else if *G does not have a q-clique* **then**
Choose $x, y \in V(G)$ such that $\{x, y\} \notin E(G)$.
Color(G'_{xy}) //vertex contraction
Color(G''_{xy}) //edge addition
Return q

Algorithm 13: ZYKOV(G)

Data: A graph G
Result: The chromatic number of G
$n \leftarrow |V(G)|$
$\chi(G) \leftarrow$ Color(G)
Return $\chi(G)$

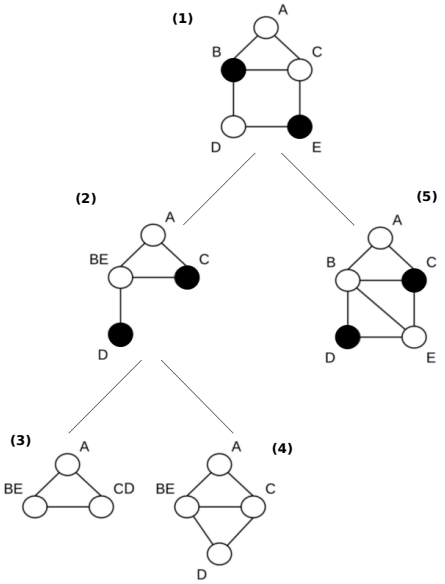

Figure 4. Pruned Zykov tree

Theorem 7. *An algorithm based on Zykov trees has complexity $\mathcal{O}(2^{n^2})$ and space $\mathcal{O}(n^2(n+m))$, if the input is a graph with n vertices and m edges.*

Proof. The height of a Zykov tree is at most $\overline{m}$, which is the number of complementary edges so that the graph be complete. Each level i of the tree has 2^i nodes, so the size of the tree is

$$\sum_{i=0}^{\overline{m}} 2^i = \frac{2^{\overline{m}+1}-1}{2-1} = 2^{\bar{m}+1} - 1 = 2^{\frac{n(n-1)}{2}+1} - 1$$

The amount of memory necessary in algorithms of this kind is $\mathcal{O}(n^2(n+m))$, since the whole graph is stored on each level of the recursion. □

Corneil and Graham [24] adapted Zykov's algorithm to find the pair of vertices that are not neighbors in a q-cluster, which is a dense subgraph of G. Their algorithm uses an amount $\mathcal{O}(n^3)$ of memory.

McDiarmid [25], on the other hand, focused on doing a more detailed worst case analysis of Zykov's algorithms, concluding that this algorithms are not more efficient than the ones that are based on generating the maximal independent sets of a graph. He concluded that for almost all graphs, a Zykov tree has size $\mathcal{O}(e^{cn\sqrt{\log n}})$, where c is a constant $c > 1$.

4. Integer Linear Programming Algorithms

In this section, we present the algorithms that solve graph coloring instances as integer linear programs, which are based in the previous work of Mehrotra and Trick [26]. We first show some definitions of Linear Programming and formulations of the graph coloring problem as integer linear programs (ILP).

Mehrotra and Trick [26] use the Branch-and-Price method, which combines the methods of Branch-and-Bound and Column Generation. We describe their algorithm and discuss about its most recent adaptations.

4.1 Integer Linear Programming

When we have an optimization problem where a linear function subject to a set of linear constraints is given, then we have a *linear program*. When the objective is to find the smallest value for the function, then we have a linear minimization program.

Let $n, m \in \mathbb{N}^*$, $A \in \mathbb{Q}^{n\times m}$ and the arrays $c \in \mathbb{Q}^n$, $x \in \mathbb{Q}^n_+$ and $b \in \mathbb{Q}^m$ represented as columns. c^T denotes the array c transposed. The canonical and the standard formulations of a linear program are defined below.

Definition 2. *The canonical formulation of a linear program, defined by A, b and c, is given by*

$$\begin{aligned} &\textit{Minimize } z = c^T x \\ &\textit{subject to } Ax \geq b \\ &\qquad x \geq 0 \end{aligned}$$

Definition 3. *The standard formulation of a linear program, defined by A, b and c, is given by*

$$\begin{aligned} &\textit{Minimize } z = c^T x \\ &\textit{subject to } Ax = b \\ &\qquad x \geq 0 \end{aligned}$$

A canonical formulation can be converted into a standard one with the addition of slack variables. More details about it can be checked in [27].

When we have a linear program where the values of x are integer, then we have an *integer linear program*. The value of the non-integer optimal solution of a linear program is a lower bound for its integer optimal solution value, and the difference between these two values is called integrality gap. Solving a linear program is a polynomial problem, and the most known algorithm that solves a linear program is the Simplex Algorithm [28]. On the other hand, solving an integer linear program is a $\mathcal{NP}$-Hard problem, since a Branch-and-Bound algorithm combined to the Simplex is necessary to solve it optimally [29].

4.1.1 ILP formulations for the graph coloring problem

For a graph G where $n = |V(G)|$ and $m = |E(G)|$, a first formulation for the graph coloring problem as an ILP is given by Formulation (2), defined by the equations (2a), (2b), (2c), (2d) and (2e).

$$\text{Minimize} \sum_{j=1}^{n} x_j \tag{2a}$$

$$\text{subject to:} \sum_{j=1}^{n} y_{vj} = 1, \quad \text{for all } v \in V(G) \text{ and } j \in \{1,\ldots,n\} \tag{2b}$$

$$y_{vj} + y_{uj} \leq x_j, \quad \text{for all } \{v,u\} \in E(G) \text{ and } j \in \{1,\ldots,n\} \tag{2c}$$

$$y_{vj} \in \{0,1\}, \quad \text{for all } v \in V(G) \text{ and } j \in \{1,\ldots,n\} \tag{2d}$$

$$x_j \in \{0,1\}, \quad \text{for all } j \in \{1,\ldots,n\} \tag{2e}$$

In this formulation, there is one binary variable x_j for each color j indicating if that color is part of a solution or not. Each variable y_{vj} indicates if the color j is assigned to the vertex v. The objective function describes a minimization integer linear program, since an optimal solution has the minimum number of colors assigned to the vertices. The set of constraints in (2b) describes that a vertex can only have one color assigned to it. Each constraint in (2c) describes that vertices that share an edge cannot have the same color.

Formulation (2) has a polynomial amount of constraints and variables, but it generates a factorial set of symmetric solutions. That is, a solution is symmetric when it can be represented as a combination of different valuations. It turns

the state space search tree of the Branch-and-Bound algorithm too large, as Lewis [1] exposes.

Let the equations (3a), (3b) and (3c) be the Formulation (3), proposed by Mehrotra and Trick [26].

$$\text{Minimize} \sum_{S \in \mathcal{S}} x_S \tag{3a}$$

$$\text{subject to:} \sum_{\{S : v \in S\}} x_S \geq 1, \qquad \text{for all } v \in V(G) \tag{3b}$$

$$x_S \in \{0,1\}, \qquad \text{for all } S \in \mathbf{S} \tag{3c}$$

In Formulation (3), $\mathbf{S}$ is the set of the maximal independent sets of a graph and the binary variable x_S indicates whether a set $S \in \mathbf{S}$ is part of a solution or not. The objective function in (3a) indicates that the chromatic number of a graph corresponds to the minimum set cover. Each constraint in (3b) indicates that each vertex v must be contained in at least one maximal independent set S. Although this formulation has an exponential number of variables, the problem of symmetry is avoided.

Fractional coloring A non-integer optimal solution of the Formulation (3) gives a fractional coloring of G. That is, since each maximal independent set represents a color, then each vertex will receive a set of colors instead of just one in a fractional coloring. Besides, the sets of vertices that share and edge will be disjoint. The value of an optimal fractional coloring is called *fractional chromatic number*, denoted by $\chi_f(G)$.

For example, let S_1 and S_2 be two maximal independent sets and let $x_{S_1} + x_{S_2} \geq 1$ be the constraint for a vertex $v \in V(G)$. If x_{S_1} and x_{S_2} have values 0.5, for example, then it means these colors contributes to 0.5 each in the fractional chromatic number, and the vertex v has both S_1 and S_2 assigned to it.

The integrality gap between $\chi_f(G)$ and $\chi(G)$ is $\mathcal{O}(\log n)$ [30], although finding $\chi_f(G)$ is also an $\mathcal{NP}$-Hard problem.

4.2 Column Generation

Sometimes we cannot escape the fact that some formulations have a large set of variables, such as the Formulation (3). They are still used in literature because they may avoid the problem of symmetry in solutions [31]. One of the problems of this kind of formulation is that many variables may not be in the optimal solution. A way around it is to start solving the instance with a small set of the variables and add the others as necessary.

The Column Generation method uses this approach to solve linear programs with a large set of variables. The *column* in the method's name refers to the Simplex algorithm in tableau format, where each column is associated to a variable.

A Column Generation based algorithm decomposes the linear program in two, named Master Problem and Pricing Problem. The first one is a more restricted reformulation of the original linear program, while the second one is the linear program that determines the variables that will be gradually added to the Master Problem. Both of the problems can be obtained by the Dantzig-Wolfe Linear Decomposition [28]. A description to this decomposition is given by Andrade, Miyazawa and Xavier [32].

For a given linear program in the standard form, the main idea of a Column Generation based algorithm is to find an initial set of the variables such that the Master Problem has at least one viable solution from this set. The Master Problem is solved with this set using the Simplex Algorithm. This is called the Restricted Master Problem (RMP).

Before showing a general description to a Column Generation algorithm, we review the pricing step of the Simplex Algorithm, since the Pricing Problem comes from this stage.

The solution found by the Simplex Algorithm separates the indices of the variables in two sets B and N. The set B has dimension m and the variables associated to this set is called the basic variables (or the basis B). The set N is the set of the indices that are not in B, and the variables associated to it are called the non-basic ones. We denote the set of the basic and non-basic variables as x_B and x_N, respectively. We also denote the sets of the costs of the basic and non-basic variables as c_B and c_N, respectively. Each variable in x_B has value greater or equal than 0 and every variable in x_N is equal to 0.

The matrix A can be rewritten as $A = [A_B|A_N]$, where $A_B = \{A_i\}_{i \in B}$, $A_N = \{A_i\}_{i \in N}$ and A_B is non-singular. We obtain the following equivalence:

$$Ax = b \iff A_B x_B + A_N x_N = b \iff$$

$$\iff A_B^{-1} A_B x_B + A_B^{-1} A_N x_N = A_B^{-1} b \iff x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

Rewriting the objective function substituting x_B for the expression found above, we have

$$\begin{aligned} z = c_B^T x_B + c_N^T x_N &= c_B^T (A_B^{-1} b - A_B^{-1} A_N x_N) + c_N^T x_N \\ &= c_B^T A_B^{-1} b + x_N (c_N^T - c_B^T A_B^{-1} A_N) \end{aligned}$$

The pricing step of the Simplex Algorithm, then, will find the non-basic variable that has the minimum reduced cost to enter the basis, that is, the minimum negative value of $c_N^T - c_B^T A_B^{-1} A_N$. Only the non-basic variables are examined since the basic variables has its reduced cost equals to zero. We denote A_B^{-1} as λ.

For a given linear program that has an optimal solution, let n be its number of variables. The Column Generation is generically described as follows (Algorithm 14).

A Branch-and-Bound algorithm that uses the Column Generation method is called a Branch-and-Price algorithm, and it is illustrated in the Figure 5. When a branching step occurs, it means that constraints are added to a linear program to force the value of a variable to be integer (details can be checked in [27]). In the Figure 5, X is the set of the linear programs that are in the state space search tree to be solved.

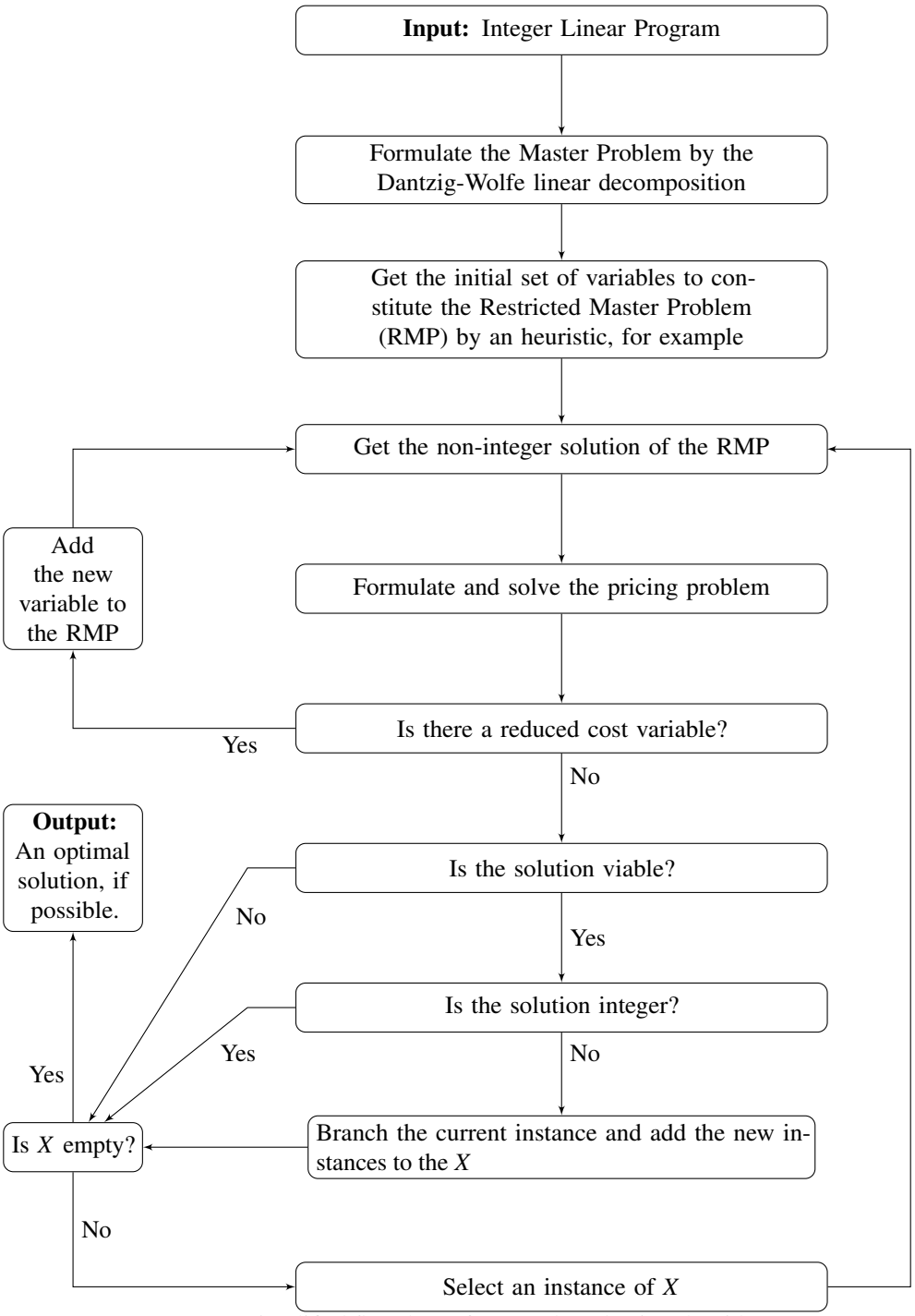

Figure 5. Simplified flowchart of a Branch-and-Price algorithm

Algorithm 14: COLUMN GENERATION

Data: The $m \times n$ matrix A and the arrays b and c of dimensions m and n, respectively.
Result: The optimal solution of the linear program defined by A, b and c, such that $Ax = b$ and $c^T x$ is minimum.
Let P be the linear program formulated from A, b and c.

Get the Master Problem of P from the Dantzig-Wolfe linear decomposition.

Get the Restricted Master Problem RMP.

Do

Solve the RMP by the Simplex Algorithm.

Define the set of non-basic variables as the set that has the non-basic variables found in the non-integer solution of the RMP and the variables that are not in the RMP yet. Formulate the Pricing Problem (PP) from the pricing step of the Simplex Algorithm, that is,

$$\overline{c} = c_N^T - \lambda^T A_N$$

$\overline{c}_i \in \overline{c}$ such that $\overline{c}_i$ is minimum and $i \in N$

Solve the PP. If $\overline{c}_i < 0$, then add the variable found in PP to the RMP. Otherwise, the solution of the RMP is also an optimal solution to the linear program given as input.

while *there is* $\overline{c}_i < 0$ *in* $\overline{c}$;
Return x

4.3 Mehrotra and Trick Algorithm

The first exact algorithm for the graph coloring that use the Branch-and-Price method was proposed by Mehrotra and Trick [26]. The Master Problem is composed by Formulation (3) without the integrality constraints.

The Restricted Master Problem is described in Formulation (4), which is defined by the equations (4a) and (4b). To get the initial set of variables $\overline{\mathbf{S}} \in \mathbf{S}$ for the Restricted Master Problem, the authors use an heuristic procedure for the Maximum Weighted Independent Set (MWIS) problem. We describe this heuristic in Algorithm 15. We show that the Pricing Problem of their algorithm is equivalent to solve the Maximum Weighted Independent Set problem, in fact. We finally describe the rule proposed by the authors in the branching step of the algorithm. The most recent adaptations of Mehrotra and Trick [26] can be found in the work of Malaguti and Toth [33] and Gualandi and Malucelli [34].

4.3.1 The Pricing Problem

As we mentioned before, Mehrotra and Trick [26] find the initial set of variables to the Restricted Master Problem, defined as follows.

$$\text{Minimize} \sum_{s \in \overline{S}} x_S \tag{4a}$$

$$\text{subject to:} \sum_{\{S : v \in S\}} x_S \geq 1, \qquad \text{for all } v \in V(G) \tag{4b}$$

$$x_S \geq 0, \qquad \text{for all } S \in \overline{\mathbf{S}}$$

The authors use an heuristic to the Maximum Weighted Independent Set problem (MWIS), that we define below. The heuristic is described in Algorithm 15.

Maximum Weighted Independent Set (MWIS)
Input: a graph G and a function $w : V(G) \to \mathbb{Q}$ such that $w(v)$ is named *weight* of the vertex $v \in V(G)$.
Answer: a maximum weighted independent set, that is, an independent set $I \subseteq V(G)$ such that $\sum_{v \in I} w(v)$ is maximum.

Algorithm 15: GREEDY HEURISTIC FOR THE $MWIS(G)$

Data: A graph G
Result: A maximal weighted independent set
$I \leftarrow \emptyset$
Do
Choose a vertex $v \in V(G)$ of maximum weight.
Add v to I.
Remove $N_G(v)$ from $V(G)$.
while $V(G) \neq \emptyset$;
Return I

As we mentioned before, the Pricing Problem is obtained by the pricing step of the Simplex Algorithm, that is,

$$\min \left\{ c_N^T - \lambda^T A_N \right\}.$$

More details of the relation between the dual and the pricing problems can be checked in [35].

In the Master Problem defined in Formulation (3), each coefficient in c_N^T has value 1 and each variable is associated to a column of A_N, which describes an independent set S that is represented as an array $z \in \{0,1\}^n$. Each row of the Master Problem is associated to a vertex $v \in V(G)$, so we can denote λ_v as the dual value obtained by v in the non-integer solution of the RMP. Hence, we have that the Pricing Problem corresponds to $\min\left\{1-\lambda^T z\right\}$, that is,

$$\min\left\{1-\sum_{v\in V(G)}\lambda_v z_v\right\} \equiv 1-\max\left\{\sum_{v\in V(G)}\lambda_v z_v\right\}$$

Formulation (5), defined by (5a) and (5b), describes the Pricing Problem, which is in fact the Maximum Weighted Independent Set problem. If the optimal solution of this formulation is greater than 1, then the independent set S that will be added to the RMP is constituted by each vertex v where $z_v = 1$, that is, $S = \{v \in V(G) \text{ such that } z_v = 1\}$. Otherwise, there is no independent set that can improve the current solution of the Master Problem.

$$\text{Maximize} \sum_{v\in V(G)}\lambda_v z_v \tag{5a}$$

$$z_u + z_v \leq 1, \qquad \text{for all } \{u,v\} \in E(G) \tag{5b}$$

$$z_v \in \{0,1\}, \qquad \text{for all } v \in V(G)$$

4.3.2 The Branching Rule

The branching step follows the Zykov rule described in Subsection 3.2. For an instance of the state space search tree, let S_1 and S_2 be two maximal independent sets and x_{S_1} and x_{S_2} the variables associated to these sets, such that at least one of them got a non-integer value. Since the non-integer optimal solution of the Master Problem represents a fractional coloring, then there are two vertices $u, v \in V(G)$ such that, without loss of generality, $u \in S_1 \cap S_2$ and $v \in S_1 \setminus S_2$. It means that whether u can receive the same color as the vertices in S_1 (and so does v) or it can receive the same color as the vertices of S_2. Then, one of the new instances to be added to the state space search tree will have a constraint where u and v are contracted in one vertex, while the other new instance will have a constraint where u and v have an edge between them. In fact, the constraint will be updated, so it is not necessary to create new constraints to the new instances.

4.3.3 Further Work

Mehrotra and Trick [26] observe that improvements in their method can be done in the algorithm that solves the MWIS problem. In their work, they propose a recurrence where given a graph G and a vertex $v \in V(G)$, the MWIS can either exclude v or have v and the vertices that are not neighbors of v, that is,

$$MWIS(G) = \max\left\{MWIS(G-v), MWIS(G[\{v\} \cup \overline{N}_G(v)])\right\}$$

The most recent adaptations of their algorithm can be found in the works of Malaguti and Toth [33] and Gualandi and Malucelli [34]. The first one uses a tabu search based heuristic to find the MWIS in the pricing step. This heuristic is executed for a fixed number of iterations. If this number is attained and a column were not found, then the instance of the MWIS is solved as a linear program. Gualandi and Malucelli [34], on the other hand, use Constraint Programming in the pricing step.

San Segundo [19] and Furini, Gabrel and Ternier [20] compare their algorithms with the implementation of Malaguti and Toth [33], and they conclude that the Branch-and-Price based algorithms are currently the most efficient ones for hard DIMACS instances, while the algorithms based on the exact version of the DSATUR are the best ones for random graphs with 60 to 80 vertices and densities between 0.1 and 0.9.

5. Conclusion

In this work, we presented the most recent exact algorithms for the graph coloring problem, which are the ones based in Dynamic Programming, Branch-and-Bound and Integer Linear Programming. The algorithms of the first group have an worst case analysis, while the other ones have only an experimental analysis. We briefly introduce the information required to the comprehension of the exact algorithms that solve graph coloring instances. We also observe that the algorithms based in Branch-and-Price and Branch-and-Bound DSATUR are the most promising for the graph coloring problem, where the most recent are based in the previous work of Mehrotra and Trick [26] for the first group and Brelaz [15] for the second group. Tables 1 and 2 show an overview of the algorithms presented in this work.

We observe that besides the differences between the exact approaches, they show similarities that are not always clear to the researcher. The algorithm proposed by Eppstein [5] for the optimal coloring problem, for example, can be actually extended to the algorithms of Lawler [4] and Byskov [6], without affecting the result of the respective analyses. Similarities between different paradigms also can be found. The Zykov's recurrence, for example, is much less efficient than other approaches, but it is useful in the branching step of Branch-and-Price algorithms, since this kind of rule does not change the original structure of the problem. Besides, we can cite the DSATUR procedure either as a simple heuristic for finding an initial upper bound for the chromatic number of a graph, or as a good exact algorithm for random graphs.

Table 1. Overview of the Dynamic Programming based algorithms

Algorithm	Problem	Results	Space
Lawler (1976)	Chromatic number	$\mathcal{O}(2.4423^n)$	$\Theta(2^n)$
Eppstein (2003)	Chromatic number	$\mathcal{O}(2.4150^n)$	$\mathcal{O}(2^n)$
	Optimal coloring		
Byskov (2004)	Chromatic number	$\mathcal{O}(2.4023^n)$	
	Optimal coloring		
Bodlaender & Kratsch (2006)	Chromatic number	$\mathcal{O}(5.283^n)$	$\mathcal{O}(n \log n)$

Table 2. Overview of the Branch-and-Bound and ILP based algorithms

Design Paradigm	Method	Algorithm	Results	Type of instance
Branch-and-Bound	Backtracking	Brown (1972)	Sewell's algorithm is more efficient than Brown's for graphs with more than 60 vertices.	Random graphs
	Exact DSATUR (Brélaz (1979))	Sewell (1998)		Random graphs and DIMACS instances
			More efficient than Sewell's in dense graphs (density up to 0.7).	Random graphs
		San Segundo (2012)	San Segundo's algorithm is more efficient than Sewell's in almost all instances.	DIMACS instances
			Their algorithm is more efficient than San Segundo's for dense graphs (density from 0.7 to 0.9) and number of vertices from 75 to 80, but the computational cost difference is small. Their algorithm is also more efficient than the ILP algorithms in this type of instance.	Random graphs
		Furini, Gabrel and Ternier (2017)	Similar results to San Segundo's algorithm.	DIMACS instances
	Zykov trees	Corneil e Graham (1973)	Storage bounded in $\mathcal{O}(n^3)$.	Random graphs
		McDiarmid (1979)	Analytical results. For almost all graphs, a Zykov tree has size $\mathcal{O}(e^c n\sqrt{\log n})$.	—
Integer Linear Programming	Branch-and-Price	Mehrotra and Trick (1995)	More efficient results for these instances in relation to the Branch-and-Bound algorithms.	DIMACS instances
	Branch-and-Price and Tabu Search	Malaguti and Toth (2011)		
	Branch-and-Price and Constraint Programming	Gualandi and Malucelli (2012)		

Acknowledgements

This work was partially funded by CNPq (Proc. 428941/2016-8) and CAPES.

Author contributions

This paper is an abridged version of the first author's Master Dissertation [12] which was advised by the second author.

References

[1] LEWIS, R. *A Guide to Graph Colouring: Algorithms and Applications*. 1. ed. Berlin: Springer, 2016. v. 1.

[2] GRÖTSCHEL, M.; LOVÀSZ, L.; SCHRIJVER, A. Polynomial algorithms for perfect graphs. In: BERGE, C.; CHVÀTAL, V. (Ed.). *Topics on Perfect Graphs*. 1. ed. North-Holland, Holland: North-Holland Publishing Company, 1984, (North-Holland Mathematics Studies, v. 88). p. 325–356.

[3] CORMEN, T. H. et al. *Introduction to Algorithms*. 3. ed. Massachusetts, USA: MIT Press, 2009. v. 1. I–XIX, 1–1292 p.

[4] LAWLER, E. A note on the complexity of the chromatic number problem. *Inform. Process. Lett.*, v. 5, n. 3, p. 66–67, 1976.

[5] EPPSTEIN, D. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms Appl.*, v. 7, n. 2, p. 131–140, 2003.

[6] BYSKOV, J. M. Chromatic number in time $\mathcal{O}(2.4023^n)$ using maximal independent sets. *BRICS Rep. Ser.*, v. 9, n. 45, p. 1–9, 2002.

[7] BODLAENDER, H. L.; KRATSCH, D. An exact algorithm for graph coloring with polynomial memory. *UU-CS*, v. 2006, n. 15, p. 1–5, 2006.

[8] WANG, C. C. An algorithm for the chromatic number of a graph. *J. ACM*, v. 21, n. 3, p. 385–391, 1974.

[9] TSUKIYAMA, S. et al. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, v. 6, n. 3, p. 505–517, 1977.

[10] MOON, J.; MOSER, L. On cliques in graphs. *Israel J. Math.*, v. 3, n. 1, p. 23–28, 1965.

[11] MADSEN, B. A.; BYSKOV, J. M.; SKJERNAA, B. On the number of maximal bipartite subgraphs of a graph. *BRICS Rep. Ser.*, v. 9, n. 17, p. 1–10, 2002.

[12] LIMA, A. M. de. *Algoritmos Exatos para o Problema da Coloração de Grafos*. Dissertação (Mestrado) — Universidade Federal do Paraná, Parana, Brazil, 2017.

[13] BEIGEL, R.; EPPSTEIN, D. 3-coloring in time $\mathcal{O}(1.3289^n)$. *J. Algorithm.*, v. 54, n. 2, p. 168 – 204, 2005.

[14] BYSKOV, J. Enumerating maximal independent sets with applications to graph colouring. *Oper. Res. Lett.*, v. 32, n. 6, p. 547–556, 2004.

[15] BRÉLAZ, D. New methods to color the vertices of a graph. *Commun, ACM*, v. 22, n. 4, p. 251–256, 1979.

[16] ZYKOV, A. On some properties of linear complexes. *Mat. Sb. (N.S.)*, v. 24(66), n. 2, p. 418–419, 1962.

[17] BROWN, J. R. Chromatic scheduling and the chromatic number problem. *Manage. Sci.*, v. 19, n. 4-part-1, p. 456–463, 1972.

[18] SEWELL, E. C. A branch and bound algorithm for the stability number of a sparse graph. *INFORMS J. ON COMPUT.*, v. 10, n. 4, p. 438–447, 1998.

[19] SEGUNDO, P. S. A new DSATUR-based algorithm for exact vertex coloring. *Comput. Oper. Res.*, v. 39, n. 7, p. 1724–1733, 2012.

[20] FURINI, F.; GABREL, V.; TERNIER, I.-C. An improved dsatur-based branch-and-bound algorithm for the vertex coloring problem. *Networks*, v. 69, n. 1, p. 124–141, 2017.

[21] SEWELL, E. C. An improved algorithm for exact graph coloring. *DIMACS ser. discrete math. theor. comput. sci.*, v. 26, n. 1, p. 359–373, 1996.

[22] MATHEMATICS, C. for D.; SCIENCE, T. C. *DIMACS Implementation Challenges*. 1994. Online; accessed 04 July 2018. Disponível em: <http://dimacs.rutgers.edu/archive/Challenges/>.

[23] NETO, A. S. A.; GOMES, M. J. N. Problema e algoritmos de coloração em grafos - exatos e heurísticos. *Rev. Sist. Comput.*, v. 4, n. 2, p. 201–115, 2014.

[24] CORNEIL, D. G.; GRAHAM, B. An algorithm for determining the chromatic number of a graph. *SIAM J. Comput.*, v. 2, n. 4, p. 311–318, 1973.

[25] MCDIARMID, C. Determining the chromatic number of a graph. *SIAM J. Comput.*, v. 8, n. 1, p. 1–14, 1979.

[26] MEHROTRA, A.; TRICK, M. A. A column generation approach for graph coloring. *INFORMS J. Comput.*, v. 8, n. 4, p. 344–354, 1995.

[27] MATOUŠEK, J.; GÄRTNER. *Understanding and using linear programming*. 1. ed. Berlin, Germany: Springer, 2007. v. 1. (Universitext, v. 1).

[28] DANTZIG, G. *Linear programming and extensions*. 1. ed. Princeton, NJ: Princeton Univ. Press, 1963. v. 1. (Rand Corporation Research Study, v. 1).

[29] PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. 1. ed. New Jersey, USA: Prentice-Hall, Inc., 1998. v. 1.

[30] LUND, C.; YANNAKAKIS, M. On the hardness of approximating minimization problems. *J. ACM*, v. 41, n. 5, p. 960–981, 1994.

[31] BARNHART, C. et al. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, v. 46, n. 3, p. 316–329, 1998.

[32] ANDRADE, C. E. d.; MIYAZAWA, F. K.; XAVIER, E. C. *Um algoritmo exato para o Problema de Empacotamento Bidimensional em Faixas*. Dissertação (Mestrado) — Instituto de Computação Universidade Estadual de Campinas, São Paulo, Brazil, 2006.

[33] MALAGUTI, E.; MONACI, M.; TOTH, P. An exact approach for the vertex coloring problem. *Discrete Optim.*, v. 8, n. 2, p. 174–190, 2011.

[34] GUALANDI, S.; MALUCELLI, F. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS J. Comput.*, v. 24, n. 1, p. 81–100, 2012.

[35] DESAULNIERS, G.; DESROSIERS, J.; SOLOMON, M. *Column Generation*. 1. ed. Springer US: Springer US, 2006. v. 1. (GERAD 25th anniversary series, v. 1).