UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO LEONARDO RODRIGUES BELO

ALGORITMOS ENUMERATIVOS PARA O PROBLEMA DA B-COLORAÇÃO E APLICAÇÕES RELACIONADAS

RUSSAS

2019

PEDRO LEONARDO RODRIGUES BELO

ALGORITMOS ENUMERATIVOS PARA O PROBLEMA DA B-COLORAÇÃO E APLICAÇÕES RELACIONADAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Márcio Costa Santos

Coorientadora: Prof. Ms. Tatiane Fernandes Figueiredo

RUSSAS

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

R615a Rodrigues Belo, Pedro Leonardo.
Algoritmos enumerativos para o problema da b-coloração e aplicações relacionadas / Pedro Leonardo Rodrigues Belo. – 2019.
38 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Ciência da Computação, Russas, 2019.
Orientação: Prof. Dr. Márcio Costa Santos.
Coorientação: Profa. Ma. Tatiane Fernandes Figueiredo.

1. B-coloração. 2. Algoritmo enumerativo. 3. Otimização combinatória. I. Título.

CDD 005

PEDRO LEONARDO RODRIGUES BELO

ALGORITMOS ENUMERATIVOS PARA O PROBLEMA DA B-COLORAÇÃO E APLICAÇÕES RELACIONADAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Márcio Costa Santos (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Ms. Tatiane Fernandes Figueiredo (Coorientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bonfim Amaro Júnior
Universidade Federal do Ceará (UFC)

“Is this what it feels like to end ?

I do not know, for this is not our end.”

(Kindred - League of Legends)

RESUMO

Uma *b-coloração* de um grafo G com k cores é uma coloração própria dos vértices de G tal que, em cada classe de cor existe um vértice que possui vizinhos com todas as demais $k-1$ cores. Esse vértice é chamado de *b-vértice*. Neste trabalho propõe-se um algoritmo enumerativo, ou seja, que lista todas as soluções viáveis para o problema da b-coloração e outro algoritmo que faz uso de cortes para encontrar o número b-cromático de um grafo. Como alguns algoritmos, muitas vezes fornecem apenas a solução ótima ou um solução viável, a mesma pode não ser capaz de resolver o problema. Já com todas as soluções viáveis em mãos, não há a necessidade de buscar maneiras de se obter apenas outra solução. Ambos os algoritmos foram testados em grafos no padrão DIMACS, e se demonstraram bastante eficazes. Os testes foram realizados em grafos com cinco densidades diferentes, e foram executados em ambos os algoritmos. O algoritmo responsável por encontrar o número b-cromático, que é o mesmo da enumeração completa, porém com cortes, teve seu tempo de execução bastante diminuído, demonstrando que as técnicas de poda são eficientes.

Palavras-chave: B-coloração. Algoritmo enumerativo. Otimização combinatória

ABSTRACT

A *b-coloring* of a graph G with k colors, is a proper coloring of the vertices of G, in such a way that in every color class there is a vertex adjacent to all other $k-1$ colors classes, such vertex is called *b-vertex*. In this paper is proposed an enumerative algorithm for the *b-coloring problem*, which lists all viable solutions for the problem and an algorithm that finds the b-cromatic number of a graph using prunning techniques. Some algorithms often give only the optimal or one viable solution, and that solution may be not capable to solve the problem. With all the solutions found, there is no need to search for other ways to find just one another solution. Both algorithms were tested with graphs using the DIMACS pattern, and they showed to be very effective. The tests were peformed using graphs with five different densities, and they were executed on both algorithms. The algorithm responsable to find the b-cromatic number, which is the same as the full enumeration but uses prunning, had its time reduced drasticaly, showing that the prunning techniques are effective.

Keywords: B-coloring. Enumerative Algorithm. Combinatorial Optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

G	Um grafo qualquer
V	Conjunto de vértices de um grafo
E	Conjunto de arestas de um grafo
$\omega(G)$	Tamanho máximo que uma clique em um grafo G
$\alpha(G)$	Coeficiente de estabilidade
$deg(v)$	Grau de um vértice v
$\delta(G)$	Grau minimo de um grafo G
$\Delta(G)$	Grau máximo de um grafo G
$N(V)$	Vizinhança de um vértice v
$\overline{N}(G)$	Antivizinhança de um vértice v
$\chi(G)$	Número cromático
$b(G)$	Número b-cromático
$m(G)$	Grau-m

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo resolver o problema da b-coloração e encontrar o número b-cromático através de uma enumeração utilizando a técnica de força bruta. O problema da b-coloração já vem sido bastante explorado no tocante da teoria dos grafos, porém pouco explorado na área de otimização.

A b-coloração procura colorir um grafo G de tal forma que maximize o número de b-vértices. Esses b-vértices serão os representantes das k cores que serão utilizadas na coloração, cada um possui apenas uma cor e a sua vizinhança deve possuir as demais k-1 cores, para que seja uma b-coloração apropriada.

Problemas que envolvem a estruturação de cadeias de proteínas e cadeias de DNA pode ser resolvidos com a ajuda da b-coloração, afim de encontrar a estrutura que seja mais adequada. Ambas estruturas (proteínas e DNA) podem ser representadas por grafos, onde cada átomo que as compõem representam os vértices e as ligações químicas entre eles representam as arestas. Com isso, resolvendo o problema de b-coloração, o mesmo pode ser aplicado com intuito de encontrar a estrutura mais adequada.

Este trabalho está organizado da seguinte maneira, no capítulo 2, é feita uma breve revisão sobre os conceitos básicos necessários para um melhor entendimento deste trabalho. No capítulo 3, o problema é explorado em detalhes. No capítulo 4, alguns trabalhos voltados para o problema de b-coloração em teoria dos grafos, e também é apresentado o primeiro modelo de programação linear inteira para o problema de b-coloração. No capítulo 5, é demonstrado como a enumeração para resolver o problema de b-coloração foi efetuada, assim como as estruturas e técnicas utilizadas. No capítulo 6, os resultados obtidos são discutidos afim de demonstrar a viabilidade das técnicas utilizadas neste trabalho. E por fim no capítulo 7,

2 PRELIMINARES

Algumas definições básicas que são necessárias ao leitor para o bom entendimento do tema são apresentadas neste capítulo. São utilizadas notações padrão de teoria dos grafos e programação matemática, o leitor que possui conhecimento básico nestas áreas pode se dirigir diretamente ao capítulo 3. Entretanto para o leitor não familiarizado e por questão de completude do texto, estas noções básicas serão apresentadas a seguir.

2.1 Teoria dos Grafos

Um grafo simples $G = (V,E)$ consiste de um conjunto de vértices V, não vazio e finito, e um conjunto de pares não-ordenados E, e finito, sobre este conjunto V, chamado de conjunto de arestas. Faça n e m serem as cardinalidades dos conjuntos V e E, respectivamente. Dois vértices i e j são adjacentes se, existe uma aresta que liga o vértice i ao vértice j.

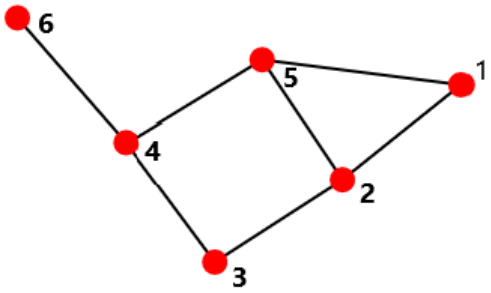

Figura 1 – Exemplo de um grafo, onde $V = \{1,2,3,4,5,6\}$ e $E = \{(1,2),(1,5),(2,5),(2,3),(3,4),(4,5),(4,6)\}$

Fonte: O Autor

Uma *clique* em um grafo G é um subconjunto de vértices S, sendo que para todo par de vértices distintos $u, v \in S$, exista uma aresta e entre eles (ARORA; BARAK, 2009). Ao maior k tal que o grafo G tem uma clique com k *vértices*, damos o nome de *número de clique* de G e o denotamos por $\omega(G)$.

Um *conjunto estável* de $G = (V,E)$ é um conjunto de vértices S, no qual nenhum par de vértices de S é adjacente. O *coeficiente de estabilidade* de G é $\alpha(G) = |max\{S \subseteq V |\ S$ é um conjunto estável$\}|$, onde $|S|$ denota a cardinalidade de S (KORSHUNOV, 1974).

Para cada vértice $v \in V$, faça $N(v)$ ser a vizinhança de v, i.e., o conjunto de vértices adjacentes a v. Dizemos que v *domina* w se $N(w) \subset N(v)$ (MALAGUTI; MONACI; TOTH, 2011). A *anti-vizinhança* de um vértice v é denotada por $\overline{N}(v)$, e representa o conjunto de

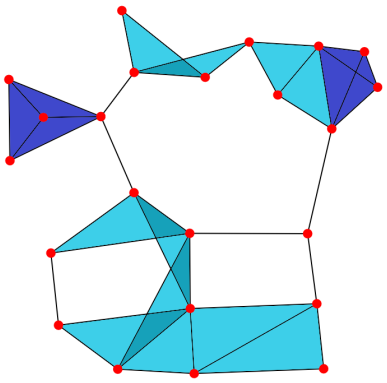

Figura 2 – Exemplo de um grafo com 23 cliques com 1 vértice, 42 cliques com 2 vértices, 19 cliques com 3 vértice (triângulos azuis claro e escuro) e 2 cliques com 4 vértices (áreas azuis). Neste exemplo, $\omega(G) = 4$.
Fonte: O Autor

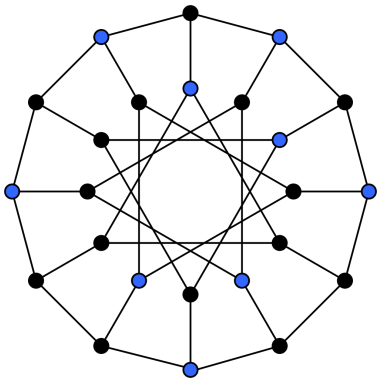

Figura 3 – Os pontos azuis forma um conjunto independente máximo. Neste exemplo $\alpha(G) = 9$
Fonte: <https://upload.wikimedia.org/wikipedia/commons/b/bd/Independent_set_graph.gif>

vértices não adjacentes a v em G. O *grau* de um vértice v em um grafo G é cardinalidade de $N(v)$. O grau de um vértice v é denotado por $deg(v)$. O grau máximo de um grafo G, é denotado por $\Delta(G) = max\{deg(v)|v \in V(G)\}$ em G e o grau mínimo de um grafo, é denotado por $\delta(G) = min\{deg(v)|v \in V(G)\}$ em G.

2.2 Coloração

Dado um grafo $G = (V,E)$, uma coloração própria é uma atribuição de cores a seus vértices de tal maneira que, vértices adjacentes recebem cores distintas. Uma k-coloração de G é uma coloração própria que utiliza um conjunto de k cores, uma k-coloração também pode ser vista como uma partição de V em k conjuntos estáveis. Cada subconjunto estável de uma k-coloração é chamado de *classe de cor* (MALAGUTI; MONACI; TOTH, 2011). Ao menor inteiro k para o qual o grafo $G = (V,E)$ admite uma k-coloração, chamamos de número cromático de G e o denotamos por $\chi(G)$. O problema de coloração consiste em dado um grafo de entrada,

determinar $\chi(G)$, ou seja, determinar o menor número de cores necessárias para colorir o grafo.

O problema de coloração faz parte dos problemas clássicos da classe NP-Difícil, esses problemas são difíceis de serem resolvidos tanto pelo ponto de vista teórico, quanto computacional.

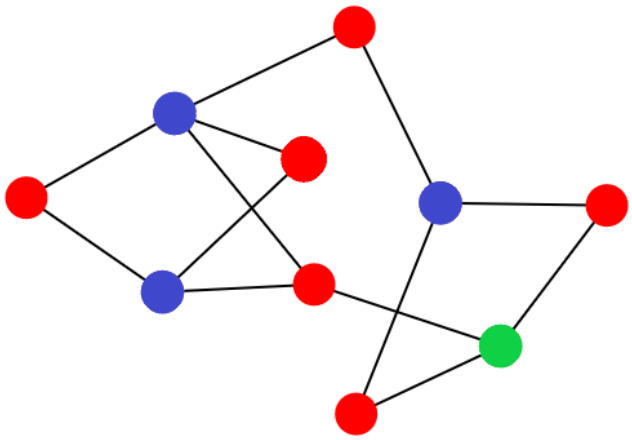

Figura 4 – Coloração de um grafo que utiliza 3 cores.
Fonte: O Autor

2.3 Algorítimos Exatos e Enumerativos

Algoritmos exatos em ciência da computação e em pesquisa operacional, são algoritmos que sempre fornecem a solução ótima para um determinado problema. O exemplo mais emblemático de um algoritmo exato é um algoritmo que enumera todas as soluções possíveis, este tipo de algoritmo é chamado de enumerativo. Esse tipo de algoritmo antigamente não era uma boa alternativa, devido ao fato que ele checa todas as possíveis soluções e retorna a ótima, e os computadores da época não eram capazes de resolve os problemas em um tempo viável. Mas com o passar do tempo, os computadores ganharam mais poder computacional e capacidade de memória, tornando esses algoritmos cada vez mais viáveis de serem utilizados.

2.4 Problemas de otimização

Muitos problemas importantes e práticos podem ser expressos como problemas de otimização. Tais problemas envolvem encontrar a melhor solução dentro de um conjunto de soluções de acordo com um critério pré-definido, representando por um função que valora cada uma das possíveis soluções. Esta é chamada de função objetivo e o conjunto de soluções é

chamado de conjunto viável para o problema. De maneira geral podemos definir um problema de otimização como $min_{x \in \mathscr{X}} f(x)$ onde f é a função objetivo e $\mathscr{X}$ o conjunto viável.

2.5 Programação matemática

Programação matemática é uma abordagem para resolver problemas de otimização que visa modelar as decisões presentes no problema como variáveis matemáticas e utilizar igualdades e desigualdades para modelar as relações entre estas decisões.

2.6 Problemas inteiros

Um problema de programação inteira é um problema de otimização modelado com programação matemática no qual algumas ou todas as variáveis estão restringidas ao conjunto dos números inteiros, entretanto as restrições e função objetivo são lineares. Neste contexto, o conjunto das possíveis decisões não é um conjunto convexo, mas é formado por diversos pontos.

2.7 Busca em profundidade

O algoritmo de busca em profundidade realiza uma busca sem informações, que se expande através do primeiro nó filho da árvore de busca, e se aprofunda cada vez mais, até que o alvo de busca seja encontrado ou se chegue em um nó folha.

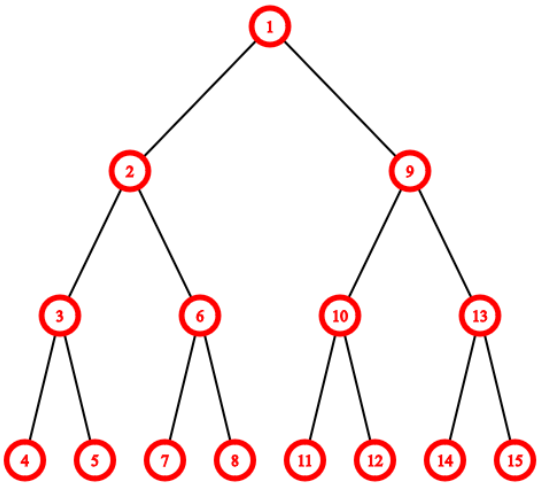

Figura 5 – A numeração em cada nó representa a ordem que o algoritmo faz durante a busca.
Fonte: O Autor

2.8 Cortes de poda

Chamamos de cortes de poda, uma restrição ou conjunto de restrições que permite eliminar da busca por uma solução ótima um nó da árvore de decisão. Com a utilização destes cortes, a complexidade geral do algoritmo é reduzida. Ao se utilizar as técnicas de cortes, o espaço de busca do resultado deve ser diminuído sem alterar a acurácia do resultado.

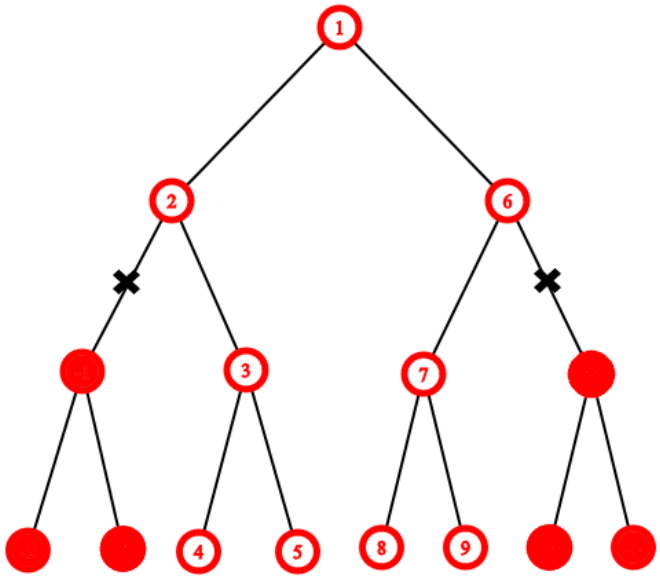

Figura 6 – Ordem em que os nós são visitados com a utilização de cortes.
Fonte: O Autor

3 PROBLEMA

Neste capítulo o problema a ser tratado neste trabalho será descrito formalmente.

3.1 Problema da b-coloração

Uma *B-coloração* de um grafo *G* com *k* cores é uma coloração própria para os vértices de *G* tal que, em cada classe de cor existe um vértice que possui vizinhos com todas as demais *k-1* cores, esse vértice é chamado de *b-vértice* (KRATOCHVíL; TUZA; VOIGT, 2002, traduzido pelo autor).

O *número b-cromático* $b(G)$ de um grafo *G* é o máximo *k* para que *G* possua uma *b-coloração* com *k* cores (IRVING; MANLOVE, 1999).

Este trabalho é dedicado a determinar o número b-cromático para um grafo dado de entrada. Para encontrar o número b-cromático, todas as b-colorações válidas são encontradas através de uma enumeração, e logo após, encontramos a solução com o maior número de b-vértices e determinamos o número b-cromático.

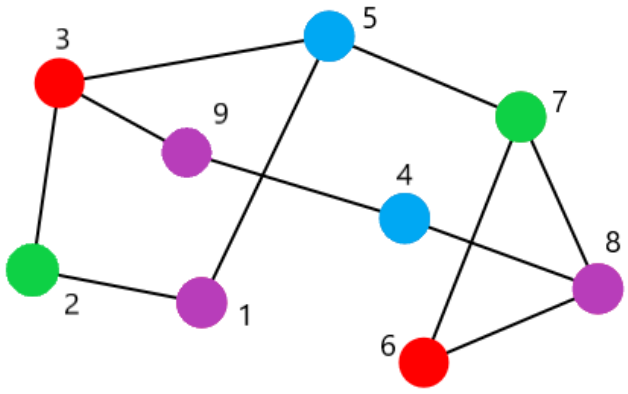

Figura 7 – Exemplo de uma b-coloração, onde os vértices 3,5,7 e 8 representam b-vértices. Este grafo utiliza 4 cores para a b-coloração.
Fonte: O Autor

3.2 Uso da b-coloração voltado para a saúde

A *b-coloração* foi usada por Elghazel *et al.* (2006) nos hospitais da França para otimizar o processo de *clustering* e como uma alternativa para o *Diagosis Related Groups* (DRG). Foram usados alguns dados sobres os pacientes como: natureza dos tratamentos, exames médicos, diagnósticos e também dados sobre o próprio paciente para a diferenciação dos clusters.

A *b-coloração* foi usada para determinar os grupos que formariam os *clusters*. *Clustering* (agrupamento) é o processo de dividir um conjunto de objetos em grupos, ou *clusters*, de tal maneira que todos os objetos de um determinado grupo possuam características semelhantes, e objetos de grupos distintos não possuam características semelhantes.

3.3 Motivação

A *b-coloração* pode ser utilizada em diversas aplicações de diferentes áreas, alguns trabalhos que podem ser encontrados são: bioinformática (problemas de cadeia de DNA), Elghazel *et al.* (2006) na saúde (agrupamento de estadias de hospitais), Dekar e Kheddouci (2008) em serviços web (composição de algoritmos baseados em b-coloração) e Gaceb *et al.* (2008) em automatização (reconhecimento de documentos para os correios).

Neste trabalho o CPLEX é utilizado como um testador para a verificação de uma coloração válida. Os algoritmos *FullEnum* e *SolveMax* que serão explorados no capítulo 5, apenas fazem chamadas ao CPLEX quando se chega em uma folha (possível solução), ou seja, ambos funcionam corretamente independente da forma em que as possíveis soluções são verificadas. Em trabalhos futuros é visado a substituição do CPLEX por uma heurística capaz de efetuar a b-coloração.

A meta deste trabalho é resolver o problema de b-coloração de maneira a obter diferentes soluções com o maior número de b-vértices possíveis. E uma vez com todas as soluções obtidas através da enumeração, será possível obter o número b-cromático de um grafo G qualquer. Essas colorações serão aplicadas no contexto da área da saúde, em cadeias de proteínas e cadeias de DNA, no qual a estrutura de ambas podem ser representadas por um grafo, e em seguida podemos determinar uma b-coloração para definir qual será a estrutura mais apropriada para ser usada em sua formação.

4 TRABALHOS RELACIONADOS

Neste capítulo será feita uma revisão da literatura existente sobre o problema. De antemão, avisamos o leitor que esta revisão é focada em métodos para resolver o problema de b-coloração como um problema de otimização. Neste contexto, temos poucos resultados publicados na literatura, visto que este é um problema novo, considerando o aspecto da otimização. Apesar de pouco tratado sobre o ponto de vista da otimização, este é muito estudado sobre o prisma de teoria dos grafos.

Lima e Carmo (2018), fazem um agrupamento de algoritmos exatos para o problema da coloração de grafos. Alguns dos algoritmos apresentados pelos autores são: Algoritmo de Brelaz, que é um algoritmo de Branch and Bound baseado no algoritmo DSATUR, que também é de autoria de Brelaz. Algoritmo de Lawler, que utiliza das técnicas de programação dinâmicas para encontrar o número cromático de um grafo e o algoritmo de Mehrotra e Trick, que utiliza da técnica de *Branch and price* para encontrar conjunto independente ponderado máximo de um grafo através de sua coloração, uma adaptação desse problema também pode ser encontrada no artigo dos autores Malaguti, Monaci e Toth (2011).

Irving e Manlove (1999) apresentam um limite superior para o problema do número *b-cromático* de um grafo G, o *grau-m*. O *grau-m* de G, denotado por $m(G)$, é o maior inteiro para o qual existe um conjunto de $m(G)$ vértices de G, com grau maior ou igual a $m(G)-1$ (IRVING; MANLOVE, 1999). Irving e Manlove (1999) também provam que encontrar o número *b-cromático* é NP-Difícil. Este é um trabalho de grande influência no tocante no problema de b-coloração, por isso é citado aqui, mesmo não sendo um artigo que trata o problema sobre o prisma de otimização.

Existem poucos trabalhos que abordam a b-coloração no prisma da otimização, um deles é o trabalho de Koch e Marenco (2018). Koch e Marenco (2018) apresentam o primeiro modelo exato voltado para o problema da b-coloração. Apesar de também apresentar heurísticas para encontrar as soluções, o trabalho de Koch e Marenco (2018) é uma abordagem exata para o problema da *b-coloração*.

O modelo de 4.1 a 4.7 foi apresentado pelos autores de Koch e Marenco (2018).

$$max \sum_{c \in C} d_c \tag{4.1}$$

$$\sum_{c \in C} x_{vc} = 1 \qquad \forall\, v \in V \tag{4.2}$$

$$x_{vc} + x_{wc} \leq 1 \qquad \forall vw \in E, \forall\, c \in C \tag{4.3}$$

$$z_{vc} \leq \sum_{w \in N(v)} x_{wd} \qquad \forall\, v \in V, \forall\, c,d \in C, c \neq d \tag{4.4}$$

$$d_c \leq \sum_{v \in V} z_{vc} \qquad \forall\, c \in C \tag{4.5}$$

$$x_{vc}, z_{vc} \in \{0,1\} \qquad \forall\, v \in V, \forall\, c \in C \tag{4.6}$$

$$d_c \in \{0,1\} \qquad \forall\, c \in C \tag{4.7}$$

A formulação de (4.1) a (4.7) faz uso das variáveis binarias x_{vc} e z_{vc} $\forall\, v \in v$ e $\forall\, c \in C$, onde C denota o conjunto de cores viáveis, com a seguinte interpretação, $x_{vc} = 1$ se e somente se o vértice v possuir a cor c, e $z_{vc} = 1$ se e somente se $x_{vc} = 1$ e v for um *b-vértice* na coloração induzida por x. Por fim temos a variável d_c ($c \in C$), tal que $d_c = 1$ se existir algum *b-vértice* com a cor c.

A função objetivo (4.1) faz com que o modelo maximize o número de b-vértices, as restrições (4.2) garantem que o vértice v possua apenas uma cor, as restrições (4.3) garantem que nós adjacentes não possuam a mesma cor, as restrições (4.4) garantem que z_{vc} seja escolhido caso x_{vc} for um b-vértice, as restriçõs (4.5) garantem que d_c seja escolhido caso $z_{vc} = 1$, as restrições (4.6) e (4.7) são de integralidade.

Como o objetivo deste trabalho é a construção de um algoritmo enumerativo, alguns estudos foram feitos em métodos enumerativos, para que melhores resultados possam ser obtidos. Um trabalho que pode ser citado é o de Östergård. (2002), que apresenta um novo algoritmo para encontrar a clique máxima dos graus dos vértices, do menor para o maior. Além de empregar técnicas de paralelismo e cortes para limitar o espaço de busca.

5 ENUMERAÇÃO

Como o objetivo deste trabalho é construir um algoritmo que seja capaz de enumerar todas as soluções para uma b-coloração, é feita uma enumeração sobre uma estrutura de árvore de decisão binária, onde cada ramificação a direita de um nó representa a não inclusão e uma ramificação a esquerda representa a inclusão de um vértice para representar um b-vértice de uma cor. Esta árvore, entretanto, nunca é gerada por nosso algoritmo de enumeração, ao invés disso, é utilizado um algoritmo similar a busca em profundidade para percorrer toda essa árvore e decisão e construí-la de maneira implícita. Ao final desta enumeração, obtemos além de todas as soluções viáveis, o número b-cromático para um grafo dado de entrada.

Observe que esta árvore de decisão escolhe os b-vértices para cada classe de cor, assim as folhas representam os conjuntos de vértices que devem se tornar b-vértices. Note que isso não significa que temos uma b-coloração válida, ainda é necessário checar se este conjunto de vértice pode conduzir a uma b-coloração válida ou não. Mais adiante será discutido como este problema problema é resolvido. A seguir serão apresentados os algoritmos enumerativos, um que faz uso de cortes de poda e outro que não faz uso de cortes de poda. Para tornar a explicação mais coerente dividimos a introdução desse algoritmo em duas partes: o algoritmo que realiza a enumeração total e o algoritmo que realiza apenas o cálculo do número b-cromático.

A função responsável por construir todas as folhas da árvore é a *Enumeration*. O primeiro nó contém a ordem no qual os vértices serão visitados e será a raiz da árvore, e os demais formarão o restante da árvore a medida que forem necessários. Antes de descrever como essa busca é realizada, será descrito a estrutura de dado utilizada.

Para representar cada nó da árvore, foi definida a estrutura *TreeNode*, que também é responsável pela construção implícita da árvore. Cada *TreeNode* possui uma lista de b-vértices, uma lista de candidatos (vértices que ainda serão visitados), e dois booleanos, que dizem se o nó já gerou os filhos esquerdo e direito. A lista de b-vértices indica quais vértices representam b-vértices, a lista de candidatos indica qual o próximo vértice dará sequência a enumeração. Partindo da raiz, que possui como b-vértices uma lista vazia, e todos os vértices como candidato, temos três forma de iteração:

- *Iterações que geram filhos a esquerda*

  É criado um novo *TreeNode* que é possui seu booleano marcado como falso, indicando que o mesmo é filho a esquerda. Sua lista de b-vértices é composta por todos os b-vértices de seu pai e adicionalmente o primeiro vértice da lista de candidatos de seu pai. A lista de

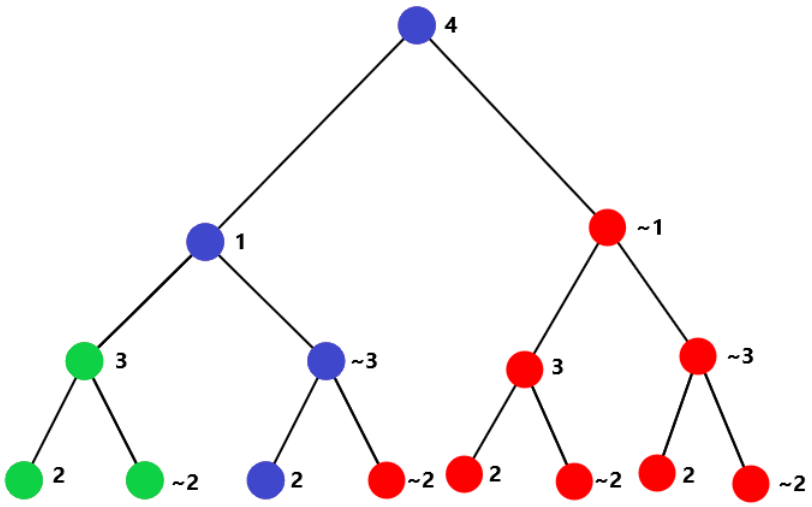

Figura 8 – Grafo com 4 vértices e a ordem = {4,1,3,2}. Os vértices em azul representam os nós que se encontram na pilha e são os b-vértices atuais, os verdes representam os nós que já foram visitados e os em vermelhos os nós que ainda serão visitados. Obs: Apenas os nós em azul se encontram em memória.
Fonte: O autor.

candidatos deste *TreeNode* é composta por todos os vértices da lista de candidatos de seu pai, com exceção do primeiro vértice. Após sua criação esse *TreeNode* é armazenado na pilha, uma posição acima de seu pai. Esta iteração corresponde ao algoritmo 1.

- *Iterações que geram filhos a direita*

  Da mesma forma que as iterações a esquerda, também é gerado um novo *TreeNode*, porém o booleano é marcado como verdadeiro. A lista de candidatos deste *TreeNode* é construída da mesma forma que nas iterações a esquerda, e sua lista de b-vértices, o primeiro vértice da lista de candidatos de seu pai , ao contrário dos filhos a esquerda, não irá fazer parte da lista de b-vértices. Esse *TreeNode* também é armazenado na pilha uma posição acima de seu pai. Esta iteração corresponde ao algoritmo 2.

- *Iterações sem filhos*

  Ao se chegar em uma folha, a função *Colorable* é utilizada, essa função é responsável por fazer as chamadas ao CPLEX que determina se o problema possui solução ou não. Uma vez com a solução em mãos, caso a mesma exista, ela será guardada em um arquivo. O retrocesso do algoritmo faz com que todas as possibilidades sejam testadas, obedecendo a seguinte lógica, ao seguir pelo caminho da esquerda o vértice u irá fazer parte da solução, e ao seguir pela direita não. Quando não há mais caminhos a serem seguidos em um nó, o mesmo volta para seu pai e verifica se ainda há caminhos, caso haja, aquele caminho será visitado e um novo ramo será gerado.

Em ambas as iterações esquerda e direita, os candidatos são passados de pai para filho, e a lista dos filhos é composta por todos os candidatos exceto o primeiro de seu pai. Quando um filho é gerado e armazenado na pilha, seja ele esquerdo ou direto, a variável *booleana* correspondente no pai, terá seu valor modificado para verdadeiro, indicando que aquele caminho já foi percorrido. Esses *booleanos* irão servir para o funcionamento correto do retrocesso (*backtracking*).

Algoritmo 1: ConstruirFilhoEsquerdo

Entrada:
Saída: Um novo nó (TreeNode)
Dados:
Resultado: Constrói um TreeNode usando os dados do TreeNode que efetuou a chamada a esta função.

```
1 t ←new TreeNode(this, esquerdo);
2 Marque o filho esquerdo como construído;
3 retorne t;
```

Algoritmo 2: ConstruirFilhoDireito

Entrada:
Saída: Um novo nó (TreeNode)
Dados:
Resultado: Constrói um TreeNode usando os dados do TreeNode que efetuou a chamada a esta função.

```
1 t ←new TreeNode(this, direito);
2 Marque o filho direito como construído;
3 retorne t;
```

5.1 Enumeração completa (Full Enum)

O método que realiza a enumeração completa das soluções, este chama-se *FullEnum*. Como já foi abordado anteriormente, o método utiliza os princípios da busca em profundidade. Esse método considera todas as folhas geradas pela enumeração até mesmo os casos que obviamente não irão gerar um solução adequada, como por exemplo, todos os vértices fixados como b-vértices. A descrição detalhada deste algoritmo é feita no algoritmo (3).

Algoritmo 3: Enumeração completa (fullEnum)

Entrada: Grafo G
Saída: Todas as b-colorações possíveis de G
Resultado: Enumeração completa de um grafo *G* qualquer.
$numNos \leftarrow 0, nivel \leftarrow 0, numFolhas \leftarrow 0$;

```
1  faça
2      se pilha[nivel] não tem filhos então
3          escreva o conjunto de b-vértices;
4          testarColoração(b-vértices, qtdBvertices);
5          incremente numNos;
6          decremente o nível atual da pilha;
       fim
7      se pilha[nivel] tem filho esquerdo então
8          t ← pilha[nivel].construirFilhoEsquerdo();
9          pilha[nivel + 1] ← t;
10         incremente numNos;
11         incremente o nível atual da pilha;
12         continue;
       fim
13     se pilha[nivel] tem filho direito então
14         t ← pilha[nivel].construirFilhoDireito();
15         pilha[nivel + 1] ← t;
16         incremente numNos;
17         incremente o nível atual da pilha;
18         continue;
       fim
19     decremente o nível atual da pilha;
   enquanto nivel != 0;
```

5.2 Solve max

Além da Enumeração completa, o outro método é o *SolveMax*, que faz uso dos cortes de poda, esse método retorna apenas o número b-cromático, diferente da enumeração completa que fornece todas as soluções viáveis. Para não gerar , ou melhor não percorrer, todas as soluções, este algoritmo faz uso de *cortes de poda* serão abordados de uma maneira mais profunda na seção 5.3 e a descrição do algoritmo é detalhada no pseudo-código (4).

5.3 Cortes utilizados

O método SolveMax (4) faz uso desses cortes para que possa encontrar o número b-cromático de forma mais eficiente. Esses cortes não necessariamente serão utilizados em todos os ramos da árvore, como por exemplo o corte (5.3.2), que é usado exclusivamente em ramos a

Algoritmo 4: Enumeração com cortes(solveMax)

```
   Entrada: Grafo G
   Saída: Número b-cromático de G
   Resultado: Encontrar o número b-cromático de um grafo.
1  numNos ← 0, nivel ← 0, numFolhas ← 0;
2  faça
3      se pilha[nivel] não possui filhos então
4          escreva o conjunto de b-vértices;
5          testarColoração (b-vertices, qtdBvertices);
6          incremente numNos;
7          decremente o nível atual da pilha;
       fim
8      se pilha[nivel] tem filho esquerdo então
9          t ← pilha[nivel].ConstruirFilhoEsquerdoComCortes();
           se t.qtdBvertices > 0 então
10             pilha[nivel+1] ← t;
11             incremente numNos;
12             incremente o nível atual da pilha;
13             continue;
           fim
       fim
14     se pilha[nivel] tem filho direito então
15         t ← pilha[nivel].ConstruirFilhoDireitoComCortes();
           se t.qtdBvertices > 0 então
16             pilha[nivel+1] ← t;
17             incremente numNos;
18             incremente o nível atual da pilha;
19             continue;
           fim
       fim
20     decremente o nível atual da pilha;
   enquanto nivel != 0;
```

esquerda da árvore. Em alguns casos eles podem não modificar o ramo de forma alguma. Mas nos casos em que eles são utilizados, o processo de enumeração se torna mais rápido devido ao fato de que não será necessário executar o algoritmo do CPLEX em vários ramos que não fornecerão uma solução. Caso não houvesse esses cortes, o algoritmo seria obrigado a testar todas as folhas da árvore em busca de uma solução viável, inclusive os que nunca serão capazes de fornecer uma solução. Os cortes que serão abordados nas próximas subseções fazem com que o problema se torne mais fácil de se resolver e consequentemente otimizando o algoritmo. Serão utilizadas algumas abreviações: C para a lista de candidatos, B para a lista de b-vértices e BS como sendo o número de b-vértices da melhor solução encontrada, observe que o método

SolveMax precisa armazenar este valor.

5.3.1 Remover de C e B, todo vértice de grau $\leq$ BS

Um dos cortes mais simples leva em conta o grau dos nós que estão em C e B, podemos remover de C e B todos os vértices que tem grau menor que BS, uma vez que estes não são b-vértices em uma b-coloração com mais de BS cores. Este será chamado de Corte01.

5.3.2 Remover todo vértice u de C, tal que N(u)$\subseteq$ N(v) e uv$\notin$ E

Caso um vértice vértice $v \in B$ para algum $v \in B$ seja escolhido para ser um b-vértice, e o mesmo possa continuar representando seu papel na solução, nenhum vértice u de sua anti-vizinhança e que esteja contido no conjunto C, pode ser dominado por ele, uma vez que u e v teriam cores distintas e nenhum vizinho de u pode ter a cor de v. Esse corte é válido apenas para filhos gerados a esquerda da árvore, devido ao fato de que os membros de C, serão escolhidos como b-vértices na solução atual. Este corte é chamado de Corte02.

5.3.3 $|C| + |B| \leq BS$

Se o tamanho conjunto de vértices candidatos e b-vértices somados, não possua vértices o suficiente para gerar uma uma nova solução que seja melhor que a solução atual já encontrada, não há por que seguir esse ramo da árvore. Este corte será chamado de Corte03.

5.3.4 Se existir um vértice v$\in$B, tal que d(v) $\leq$ |B|

Se algum vértice escolhido para ser um b-vértice não possuir grau suficiente para ser realmente um b-vértice, o mesmo irá gerar uma solução inválida, esse ramo não deve ser visitado. Este corte será chamado de Corte04.

Os cortes apresentados são utilizados durante a criação dos filhos direito ou esquerdo, para que não seja alocado um nó na pilha que não seja utilizado. O novo nó só é alocado caso o mesmo ainda possui b-vértices em sua lista depois que os cortes forem efetuados. Os pseudo-códigos (5) e (6) demonstram o funcionamento da utilização dos cortes.

Algoritmo 5: ConstruirFilhoEsquerdoComCortes

Entrada: TreeNode pai
Saída: Um TreeNode filho com possíveis cortes em B ou C

```
1  t ← newTreeNode(this, false);
2  Marque o filho esquerdo como gerado.;
3  B ← t.getBvertices();
4  C ← t.getCandidatos();
5  bnum ← Tamanho de B;
6  cnum ← Tamanho de C;
7  Corte01(B, C);
8  Corte02(C);
9  Corte03(B, bnum);
10 Corte04(cnum, bnum, t);
11 retorne t;
```

Algoritmo 6: ConstruirFilhoDiretoComCortes

Entrada: TreeNode pai
Saída: Um TreeNode filho com possíveis cortes em B ou C

```
1  t ← newTreeNode(this,true);
2  Marque o filho esquerdo como gerado.;
3  B ← t.getBvertices();
4  C ← t.getCandidatos();
5  bnum ← Tamanho de B;
6  cnum ← Tamanho de C;
7  Corte01(B, C);
8  Corte03(B, bnum);
9  Corte04(cnum, bnum, t);
10 retorne t;
```

5.4 Teste de Coloração Válida

Para finalizar a descrição de ambos os algoritmos, é necessário detalhar como a verificação de uma coloração válida é realizada. Veja que o algoritmo descrito até agora só enumera possíveis conjuntos de b-vértices, ainda é necessário testar se existe uma coloração para a qual estes são b-vértices.

O modelo de (5.1) até (5.6) é usado como um teste, ele informa se, dado um conjunto de b-vértices, existe ou não uma b-coloração para o grafo. O teste é realizado quando a enumeração chega em uma folha da árvore, uma vez que elas representam uma possível solução.

Para a realização dos testes, o CPLEX é utilizado, mas a melhor opção talvez seja uma outra enumeração para lidar com a distribuição de cores.

Para a criação deste modelo, foram utilizadas como base as ideias encontradas no

trabalho de Campêlo, Corrêa e Frota (2004), que demonstra um modelo de representantes para o problema da b-coloração.

O modelo (5.1) até (5.6) utilizado neste trabalho faz uso de uma variável binaria x_{vu}, que indica que a cor do vértice u é representada pela cor do vértice v. A mesma tem seu valor igual a 1 se, o vértice u é representado por v, e 0 caso contrário.

A função objetivo (5.1) faz com que o modelo maximize o número de b-vértices, as restrições (5.2) garantem que algum vértice da anti vizinhança de v seja capaz de representá-lo. As restrições (5.3) garantem que todo b-vértice seja capaz de representar outros vértices. As restrições (5.4) garantem que o vértice v não possa representar dois vértices adjacentes. As restrições (5.5) garantem que existe um vértice com a cor de v na vizinhança de u. E por fim as restrições (5.6) fazem com que o vértice só possa representar alguém se ele fizer parte dos b-vértices.

$$\max \sum_{v \in V} x_{vv} \tag{5.1}$$

$$x_{vv} + \sum_{u \in \overline{N}(v)} x_{uv} = 1 \qquad \forall\, v \in V \tag{5.2}$$

$$x_{vv} = 1 \qquad \forall\, v \in B \tag{5.3}$$

$$x_{vu} + x_{vw} \leq 1 \qquad \forall uw \in E \tag{5.4}$$

$$\sum_{w \in \overline{N}(v) \cap N(v)} x_{uw} \geq x_{vv} + x_{uu} - 1 \qquad \forall\, v,u \in V \tag{5.5}$$

$$x_{vu} \leq x_{vv} \qquad \forall\, v \in V, u \in N(v) \tag{5.6}$$

6 RESULTADOS

Os testes a seguir foram realizados em um computador com processador Intel Core i7 8550U (1.8GHz), 12GB de memória RAM, 8MB de memória cache, sistema operacional Windows 10. Como era necessário utilizar o sistema operacional Linux para a execução do CPLEX, os testes foram realizados com a ajuda de uma máquina virtual (*Oracle VM VirtualBox* versão 6.0) criada com 4GB de RAM e o sistema operacional Ubuntu.

6.1 Instâncias

O autor utilizou um programa, onde é escolhido o número de vértices e qual a densidade do grafo, a densidade do grafo é a razão de suas arestas, pela quantidade de arestas de uma clique de mesma quantidade de vértices. A saída do algoritmo são os arquivos no padrão DIMACS, contendo as informações sobre os grafos gerados.

Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), possui competições voltadas para desenvolvimento teórico e aplicações praticas de matemática discreta e ciência da computação. O padrão estabelecido pela DIMACS foi utilizado para que as instâncias seguissem um padrão bem estabelecido.

6.2 Full Enumeration

Pode-se observar na tabela 1, que o modelo apresentado na seção 5.4 foi capaz de listar todas as soluções viáveis de grafos de tamanho pequeno em um tempo considerável. O modelo tende a ter uma execução mais demorada com grafos que possuem uma densidade mediana e menor tempo de execução com densidades muito altas.

6.3 SolveMax

Com a utilização dos cortes ocorreu uma grande queda em relação ao tempo necessário para que as mesmas instâncias que foram utilizadas nos testes do FullEnum pudessem ser resolvidas. As instâncias que demoraram mais para serem resolvidas ainda continuam sendo as com densidade mediana e mediana-alta, mas no caso das medianas os tempos foram reduzidos significativamente.

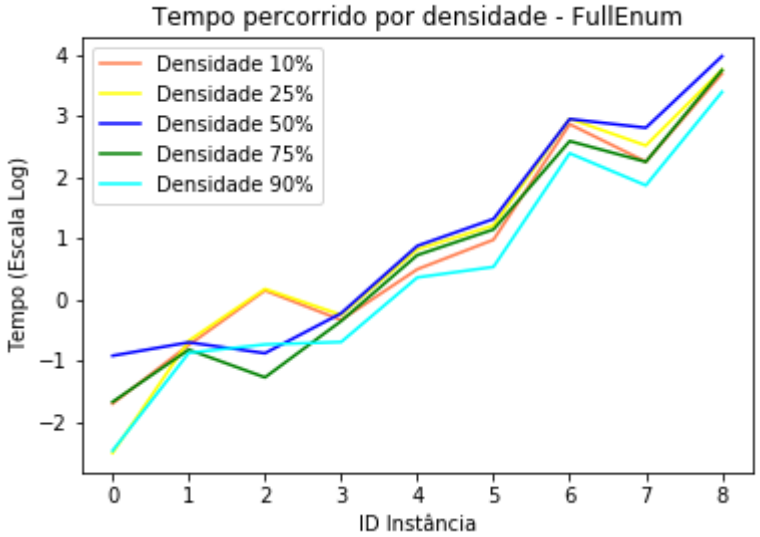

Figura 9 – Gráfico contendo o tempo percorrido para cada instância do FullEnum, agrupados por densidade.

Fonte: O autor

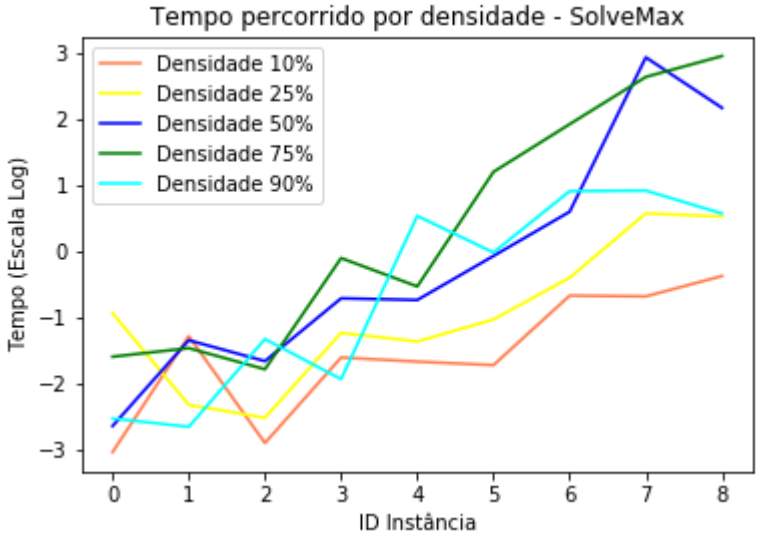

Figura 10 – Gráfico contendo o tempo percorrido para cada instância do SolveMax, agrupados por densidade.

Fonte: O autor

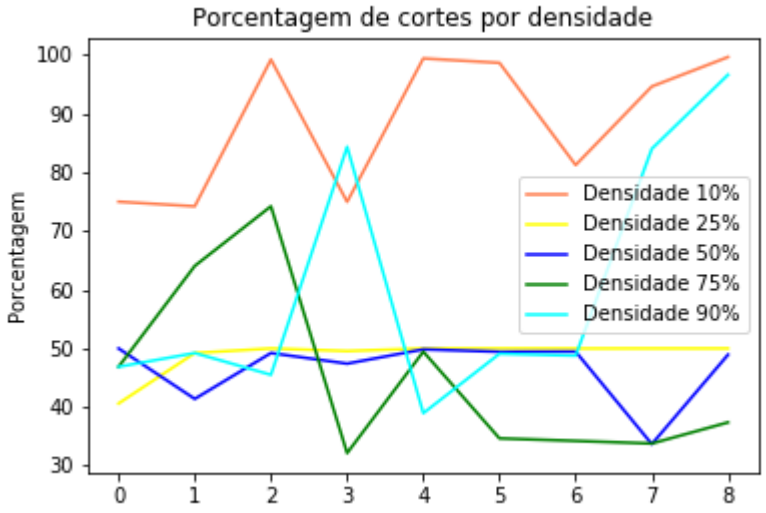

Figura 11 – Porcentagem de cortes realizados em cada instância durante a execução do SolveMax, agrupados por densidade.

Fonte: O autor

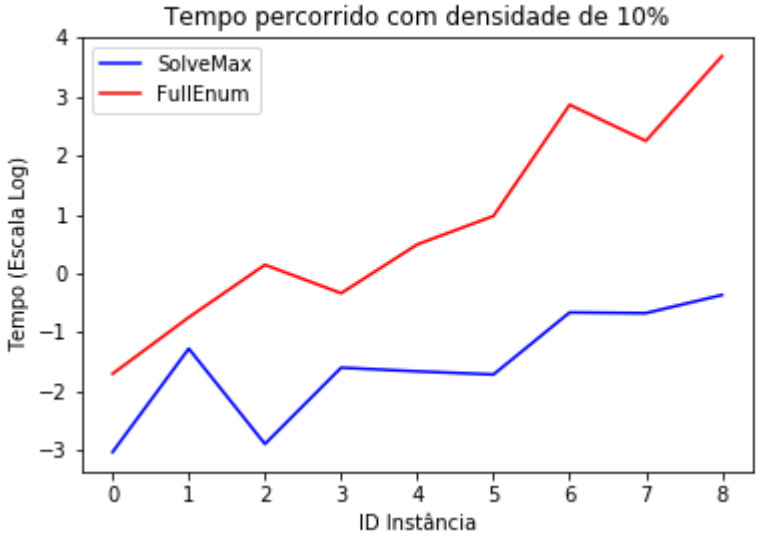

Figura 12 – Comparação do tempo percorrido para grafos com densidade 10%

Fonte: O autor

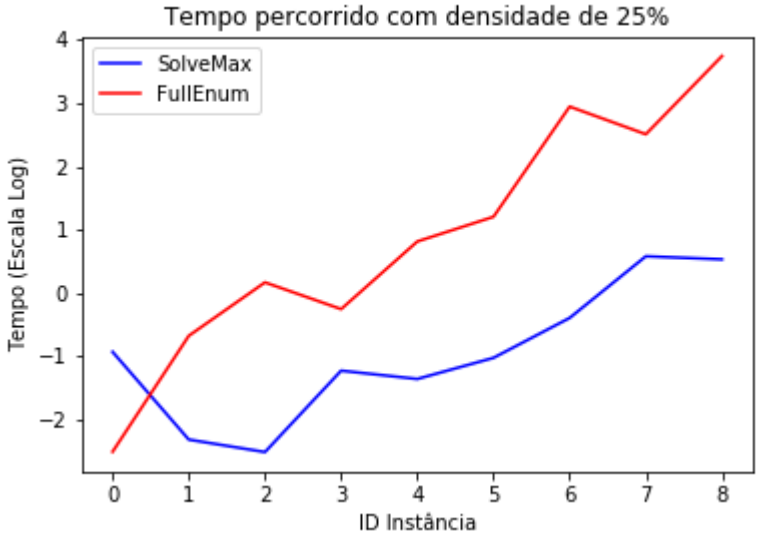

Figura 13 – Comparação do tempo percorrido para grafos com densidade 25%
Fonte: O autor

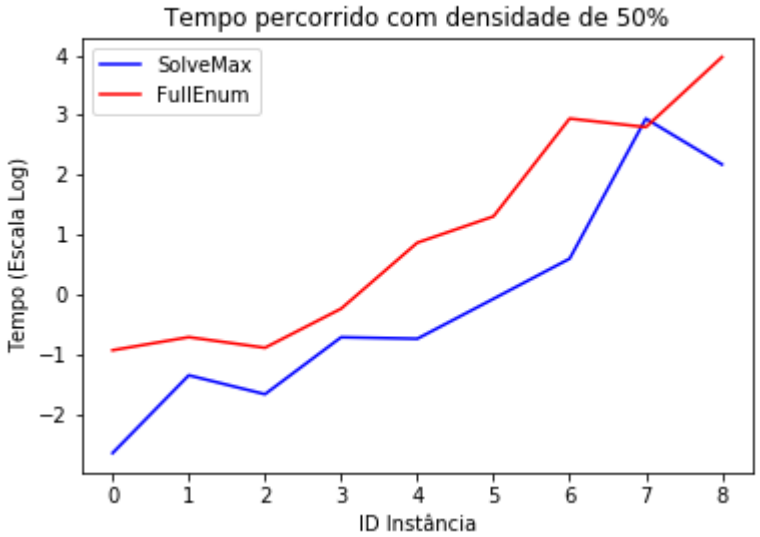

Figura 14 – Comparação do tempo percorrido para grafos com densidade 50%.
Fonte: O autor

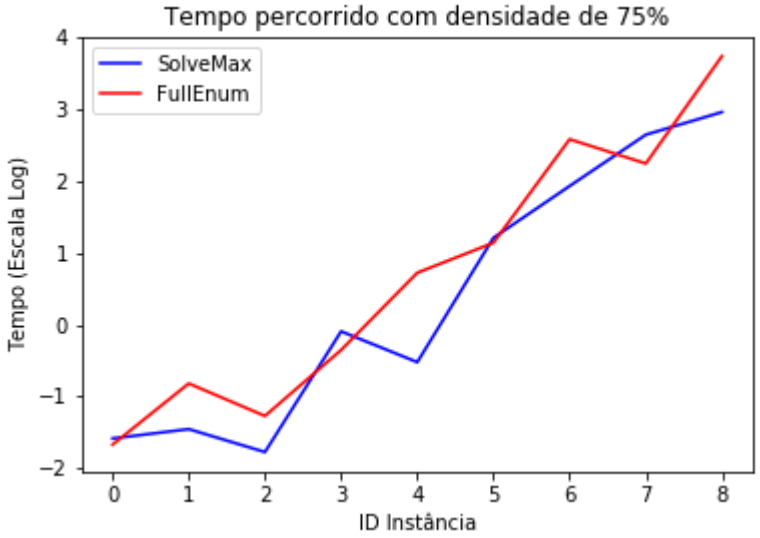

Figura 15 – Comparação do tempo percorrido para grafos com densidade 75%.
Fonte: O autor

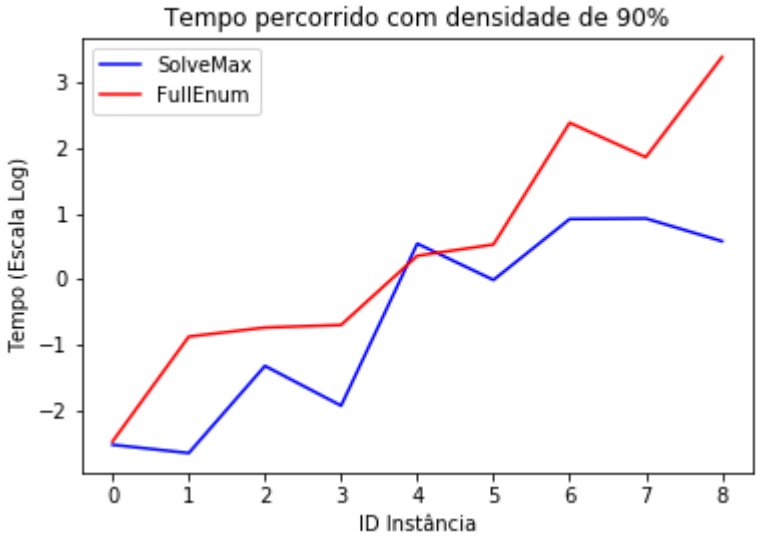

Figura 16 – Comparação do tempo percorrido para grafos com densidade 90%.
Fonte: O autor

7 CONCLUSÕES

Apesar de que o FullEnum possui um tempo de execução bastante elevado, o mesmo pode ser tratado para que se torne viável a sua utilização. Os cortes de poda demonstrados neste trabalho se demonstram eficazes, e conseguem reduzir consideravelmente o espaço de busca para que o SolveMax possa ser resolvido de maneira mais eficaz. Da mesma maneira que o SolveMax utiliza cortes para encontrar o número b-cromático, alguns deles podem ser utilizados durante o FullEnum, para que o mesmo tenha seu tempo de execução e espaço de busca reduzidos e obtendo os mesmos resultados. Como os dois algoritmos FullEnum e SolveMax, não dependem do testador que será usado, em trabalhos futuros pretende-se efetuar a substituição do CPLEX por um método combinatório capaz de efetuar a b-coloração.

REFERÊNCIAS

ARORA, S.; BARAK, B. **Computational complexity: a modern approach**. [S.l.]: Cambridge University Press, 2009.

CAMPÊLO, M.; CORRÊA, R.; FROTA, Y. Cliques, holes and the vertex coloring polytope. **Information Processing Letters**, Elsevier, v. 89, n. 4, p. 159–164, 2004.

DEKAR, L.; KHEDDOUCI, H. A graph b-coloring based method for composition-oriented web services classification. In: SPRINGER. **International Symposium on Methodologies for Intelligent Systems**. [S.l.], 2008. p. 599–604.

ELGHAZEL, H. *et al.* A new clustering approach for symbolic data and its validation: Application to the healthcare data. In: SPRINGER. **International Symposium on Methodologies for Intelligent Systems**. [S.l.], 2006. p. 473–482.

GACEB, D. *et al.* Improvement of postal mail sorting system. **International Journal of Document Analysis and Recognition (IJDAR)**, Springer, v. 11, n. 2, p. 67–80, 2008.

IRVING, R. W.; MANLOVE, D. F. The b-chromatic number of a graph. **Discrete Applied Mathematics**, Elsevier, v. 91, n. 1-3, p. 127–141, 1999.

KOCH, I.; MARENCO, J. An integer programming approach to b-coloring. **Discrete Optimization**, Elsevier, 2018.

KORSHUNOV, A. Coefficient of internal stability of graphs. **Cybernetics and Systems Analysis**, Springer, v. 10, n. 1, p. 19–33, 1974.

KRATOCHVíL, J.; TUZA, Z.; VOIGT, M. On the b-chromatic number of a graph. **Lecture Notes in Comput. Sci.**, v. 2573, n. 0, p. 310–320, 2002, traduzido pelo autor.

LIMA, A. M. de; CARMO, R. Exact algorithms for the graph coloring problem. **Revista de Informática Teórica e Aplicada**, v. 25, n. 4, p. 57–73, 2018.

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

ÖSTERGÅRD, P. R. A fast algorithm for the maximum clique problem. **Discrete Applied Mathematics**, Elsevier, v. 120, n. 1-3, p. 197–207, 2002.

APÊNDICE A – TABELAS DE RESULTADOS

FullEnum	N Vértices	N Arestas	FO	Tempo Percorrido	Soluções Encontradas
Inst 05_10	5	2	2	0.019897	1
Inst 05_25	5	6	2	0.003123	2
Inst 05_50	5	12	3	0.120529	2
Inst 05_75	5	16	4	0.021256	1
Inst 05_90	5	16	4	0.003398	1
Inst_07_10	7	8	3	0.180261	4
Inst_07_25	7	14	3	0.211872	9
Inst_07_50	7	22	4	0.199191	5
Inst_07_75	7	28	4	0.151244	2
Inst_07_90	7	38	6	0.134157	1
Inst 10_10	10	4	2	1.400376	2
Inst 10_25	10	22	3	1.481568	12
Inst 10_50	10	48	5	0.132364	38
Inst 10_75	10	74	7	0.053329	6
Inst 10_90	10	80	7	0.18409	4
Inst_12_10	12	16	3	0.462182	14
Inst_12_25	12	40	4	0.560114	59
Inst_12_50	12	62	5	0.594275	93
Inst_12_75	12	98	7	0.443554	129
Inst_12_90	12	122	9	0.201826	2
Inst 15_10	15	12	3	3.106914	7
Inst 15_25	15	44	4	6.561939	110
Inst 15_50	15	86	6	7.485786	227
Inst 15_75	15	144	9	5.289624	414
Inst 15_90	15	182	9	2.28556	72
inst_16_10	16	28	4	9.409044	16
inst_16_25	16	68	6	16.115704	298
inst_16_50	16	116	7	20.542768	882
inst_16_75	16	176	9	13.855399	1566
inst_16_90	16	222	12	3.392622	19
Inst_18_10	18	36	4	724.690857	55
Inst_18_25	18	86	6	888.625061	1282
Inst_18_50	18	152	8	880.034119	5494
Inst_18_75	18	234	10	383.739014	1577
Inst_18_90	18	286	13	242.676743	12
Inst 20_10	20	34	4	176.703949	43
Inst 20_25	20	84	6	324.949982	830
Inst 20_50	20	180	8	633.085999	18693
Inst 20_75	20	288	11	176.148849	1755
Inst 20_90	20	340	13	72.667458	234
Inst_21_10	21	44	4	4856.410645	118
Inst_21_25	21	116	7	5592.432617	2371
Inst_21_50	21	200	9	9335.09668	13614
Inst_21_75	21	324	12	5546.230957	8994
Inst_21_90	21	392	12	2436.87793	128

Tabela 1 – Tabela com os resultados do método FullEnum em grafos gerados aleatoriamente.

SolveMax	N Vértices	N Arestas	F.O.	Tempo Percorrido	Soluções Encontradas	% de cortes
Inst 05_10	5	2	2	0.000936	2	75
Inst 05_25	5	6	2	0.118187	1	40.625
Inst 05_50	5	12	3	0.002323	1	50
Inst 05_75	5	16	4	0.025934	1	46.875
Inst 05_90	5	16	4	0.003019	1	46.875
Inst_07_10	7	8	3	0.053115	1	74.219
Inst_07_25	7	14	3	0.004857	1	49.219
Inst_07_50	7	22	4	0.046013	2	41.406
Inst_07_75	7	28	0	0.03498	0	64.062
Inst_07_90	7	38	6	0.002256	1	49.219
Inst 10_10	10	4	2	0.001284	1	99.219
Inst 10_25	10	22	3	0.003085	1	50
Inst 10_50	10	48	5	0.022256	1	49.219
Inst 10_75	10	74	7	0.016752	1	74.219
Inst 10_90	10	80	7	0.047968	1	45.508
Inst_12_10	12	16	3	0.025247	2	74.976
Inst_12_25	12	40	4	0.059611	2	49.561
Inst_12_50	12	62	5	0.198324	1	47.412
Inst_12_75	12	98	7	0.805938	2	32.178
Inst_12_90	12	122	9	0.01189	1	84.351
Inst 15_10	15	12	3	0.021853	1	99.408
Inst 15_25	15	44	4	0.044083	2	49.997
Inst 15_50	15	86	6	0.187238	2	49.844
Inst 15_75	15	144	9	0.300035	2	49.454
Inst 15_90	15	182	9	3.503145	1	38.971
inst_16_10	16	28	4	0.019344	2	98.654
inst_16_25	16	68	6	0.095057	3	49.992
inst_16_50	16	116	7	0.868106	1	49.435
inst_16_75	16	176	9	16.238522	2	34.679
inst_16_90	16	222	12	0.97833	1	49.097
Inst_18_10	18	36	4	0.217977	2	81.242
Inst_18_25	18	86	6	0.406286	3	49.987
Inst_18_50	18	152	8	4.055124	1	49.462
Inst_18_75	18	234	10	85.601212	1	34.25
Inst_18_90	18	286	13	8.298862	1	48.773
Inst 20_10	20	34	4	0.212138	2	94.629
Inst 20_25	20	84	6	3.829465	3	49.997
Inst 20_50	20	180	8	877.319458	1	33.705
Inst 20_75	20	288	11	444.226227	1	33.806
Inst 20_90	20	340	13	8.458164	1	84.013
Inst_21_10	21	44	4	0.431105	2	99.644
Inst_21_25	21	116	7	3.425662	4	49.995
Inst_21_50	21	200	9	150.984695	3	48.932
Inst_21_75	21	324	12	918.585266	2	37.403
Inst_21_90	21	392	12	3.809371	1	96.625

Tabela 2 – Tabela com os resultados do método SolveMax em grafos gerados aleatoriamente.