UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DAVI GOMES FLORENCIO

ALGORITMOS PARA O PROBLEMA DO NÚMERO DE GRUNDY

QUIXADÁ

2022

DAVI GOMES FLORENCIO

ALGORITMOS PARA O PROBLEMA DO NÚMERO DE GRUNDY

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

Orientador: Prof. Dr. Wladimir Araújo Tavares

QUIXADÁ

2022

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)

F654a Florencio, Davi Gomes.
Algoritmos para o Problema do Número de Grundy / Davi Gomes Florencio. – 2022.
75 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2022.
Orientação: Prof. Dr. Wladimir Araújo Tavares.

1. Teoria dos grafos. 2. Coloração de grafos. 3. Número Grundy. 4. Algoritmos. I. Título.
CDD 004

DAVI GOMES FLORENCIO

ALGORITMOS PARA O PROBLEMA DO NÚMERO DE GRUNDY

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

Aprovada em: ___/___/___

BANCA EXAMINADORA

Prof. Dr. Wladimir Araújo Tavares (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Atílio Gomes Luiz
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio Carlos Sousa Dias
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Henrique Macêdo de Araújo
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Esta vitória também é de vocês.

AGRADECIMENTOS

Aos meus pais Artemir e Mônica, por todo o esforço, sacrifício e apoio em todos esses anos que me permitiram chegar a este momento.

A minha irmã Isadora Gomes por desde sempre compartilhar os momentos de alegrias e dificuldades.

Ao meu Prof. Orientador Dr. Wladimir Araújo pela ótima orientação e suporte no desenvolvimento do trabalho.

Aos meus professores da banca examinadora Atílio Gomes, Fábio Dias e Paulo Henrique pela contribuição com sugestões de melhorias para o trabalho.

A todos os professores da Universidade Federal do Ceará - Campus Quixadá por me proporcionar o conhecimento, pelo tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

A todos os meus amigos que contribuíram de alguma forma para eu concluir essa jornada.

“A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê.”

(Arthur Schopenhauer)

RESUMO

No presente trabalho, estudamos o Problema do Número de Grundy que consiste em determinar a maior quantidades de cores que o Algoritmo Guloso de Coloração consegue atribuir a um grafo. Este algoritmo tem o seguinte princípio geral: receber, um a um, os vértices do grafo a ser colorido, atribuindo sempre a menor cor possível a este vértice. O maior número de cores utilizados pelo Algoritmo Guloso de Coloração é chamado de número de Grundy ou número cromático guloso do grafo. Sabe-se que determinar o número de Grundy de um grafo qualquer pertence à classe NP-completo. O objetivo deste trabalho foi propor novas soluções exatas para o Problema do Número de Grundy. Neste trabalho, foram propostos um limite inferior e um superior, ambos baseados em heurísticas gulosas. Com relação às soluções exatas, foram propostas um algoritmo de Programação Dinâmica, um algoritmo Branch and Bound e três formulações de Programação Linear Inteira. Foram geradas instâncias aleatórias e da classe Stacked Book, no qual foram realizados diversos testes computacionais.

Palavras-chave: Teoria dos grafos. Coloração de grafos. Número Grundy. Algoritmos.

ABSTRACT

In the present work, we study the Grundy Number Problem, which consists of determining the largest amount of colors that the Greedy Coloring Algorithm can assign to a graph. This algorithm has the following general principle: receive, one by one, the vertices of the graph to be colored, always attributing the smallest possible color to this vertex. The largest number of colors used by the Greedy Coloring Algorithm is called the Grundy number or greedy chromatic number of the graph. It is known that determining the Grundy number of any graph belongs to the NP-complete class. The objective of this work was to propose new exact solutions to the Grundy Number Problem. In this work, a lower and an upper bound were proposed, both based on greedy heuristics. Regarding the exact solutions, a Dynamic Programming algorithm, a Branch and Bound algorithm and three Integer Linear Programming formulations were proposed. Random instances and the Stacked Book class were generated, in which several computational tests were performed.

Keywords: Graph theory. Graph coloring. Grundy number. Algorithms.

LISTA DE FIGURAS

Figura 1 – Produto Cartesiano $G\square H$ 19

Figura 2 – Exemplos de grafos Book PB_p 20

Figura 3 – Exemplos de grafos Stacked Book $SB_{p,q}$ 21

Figura 4 – Coloração de Grundy de G com 4 cores 24

Figura 5 – Coloração de G com 3 cores 24

Figura 6 – Coloração própria de G com número de Grundy $\Gamma(G) = 3$ 25

Figura 7 – Representação do modelo geral de Programação Linear. 28

Figura 8 – Representação do modelo de Programação Linear Inteira. 29

Figura 9 – Formulação de Programação Linear Inteira do Problema do Número de Grundy com $C = \{1,\ldots,n\}$ 34

Figura 10 – Um exemplo de sequência de Grundy viável. 41

Figura 11 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1,\ldots,\Delta(G)+1\}$ 56

Figura 12 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1,\ldots,\zeta(G)\}$ 57

Figura 13 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1,\ldots,\Delta(G)\cdot 2^{\Gamma(H)-1}+\Gamma(H)\}$ 58

LISTA DE TABELAS

Tabela 1 – Análise comparativa entre os trabalhos relacionados e o proposto. 35
Tabela 2 – Média de tempo de execução dos algoritmos exatos em instâncias de grafos gerados aleatoriamente comparados pela probabilidade utilizada no algoritmo de geração. 61
Tabela 3 – Número de instâncias com solução ótima encontrada pelos algoritmos exatos em instâncias de grafos da classe Stacked Book. 62
Tabela 4 – Média de tempo de execução dos algoritmos exatos em instâncias de grafos da classe Stacked Book. 63
Tabela 5 – Tempo de execução dos algoritmos em instâncias de grafos gerados aleatoriamente. 70
Tabela 6 – Resultado dos algoritmos em instâncias de grafos gerados aleatoriamente. . 72
Tabela 7 – Tempo de execução dos algoritmos em instâncias de grafos da classe Stacked Book. 74
Tabela 8 – Resultado dos algoritmos em instâncias de grafos da classe Stacked Book. . 75

LISTA DE ALGORITMOS

Algoritmo 1 – COLORACAOGULOSA(G,θ) 23

Algoritmo 2 – BUCKET(G) 38
Algoritmo 3 – DEGENERACAO(G) 39
Algoritmo 4 – LB-DEG(G) 40

Algoritmo 5 – DECOMPOSICAO(G) 43
Algoritmo 6 – STAIRFACTOR(G) 44

Algoritmo 7 – BK(G, C, P, S, CLIQUES) 47
Algoritmo 8 – BKPIVO(G, C, P, S, CLIQUES) 48
Algoritmo 9 – BKDEGENERACAO(G) 49
Algoritmo 10 – PDG(G) 50
Algoritmo 11 – BBCI(G) 52
Algoritmo 12 – BBC(G,C) 53
Algoritmo 13 – BBGI(G, LI, LS) 54
Algoritmo 14 – BBG(G, O, LB, UB, C, K) 55

Algoritmo 15 – RANDOMGRAPH(N, P) 60
Algoritmo 16 – GENERATESTACKEDBOOK(P, Q) 61

SUMÁRIO

1 INTRODUÇÃO 14
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Grafo 17
2.1.1 Produto cartesiano de grafos 19
2.1.2 Grafos Book 20
2.1.3 Grafos Stacked Book 20
2.2 Coloração de vértices 21
2.2.1 Coloração própria de vértices 21
2.2.2 Número Cromático 21
2.2.3 Coloração k-imprópria de vértices 22
2.2.4 Algoritmo Guloso de Coloração 22
2.2.5 Coloração de Grundy 23
2.3 Número de Grundy 23
2.4 Complexidade Parametrizada 25
2.5 Programação dinâmica 26
2.6 Programação Linear 27
2.6.1 Programação Linear Inteira 28
2.6.2 Google OR-Tools 29
2.7 Branch and Bound 29
3 TRABALHOS RELACIONADOS 31
3.1 Complexity of Grundy coloring and its variants 31
3.2 Grundy number and products of graphs 32
3.3 New bounds on the Grundy number of products of graphs 32
3.4 Coloração k-imprópria gulosa 33
3.5 Trabalho Proposto 34
4 LIMITE INFERIOR 36
4.0.1 Degeneração de vértices 36
5 LIMITE SUPERIOR 41
5.1 Stair factor 41
6 ALGORITMOS EXATOS 45

6.1 **Programação Dinâmica** 45
6.2 **Branch-and-Bound** 51
6.3 **Programação Linear Inteira** 56
7 **RESULTADOS COMPUTACIONAIS** 59
7.1 **Instâncias de Teste em grafos aleatórios** 59
7.2 **Instâncias de Teste em grafos da classe Stacked Book** 61
8 **CONCLUSÕES E TRABALHOS FUTUROS** 64
REFERÊNCIAS 66
APÊNDICE A – RESULTADOS DOS TESTES COMPUTACIONAIS . 69

1 INTRODUÇÃO

A coloração é um dos temas mais importantes e conhecidos da área de Teoria dos Grafos, e até hoje, é objeto de estudo para muitos cientistas da área. Este tema começou a ser investigado por Francis Guthrie, no ano de 1852, ao tentar colorir o mapa do condado de Inglaterra de modo que regiões adjacentes receberiam cores distintas, e notou que 4 cores eram suficientes. Logo, ele se questionou se todo mapa político poderia ser colorido com no máximo quatro cores, de forma que regiões adjacentes receberiam cores distintas e propôs a conjectura da Quatro Cores (GUTHRIE, 1880). Esta conjectura foi provada por Appel *et al.* (1977).

A coloração de vértices de um grafo consiste em uma atribuição de cores aos seus vértices. Uma coloração é dita própria se vértices adjacentes em um grafo são coloridos com cores distintas. O problema de coloração de vértices consiste em colorir os vértices de um grafo com o menor número de cores possíveis, além disso, ele possui diversas aplicações práticas. Uma aplicação prática recorrente na literatura é o Problema de Atribuição de Canais de Frequência (HALE, 1980). Uma vez que os sinais de rádio se atenuam com a distância, uma mesma frequência pode ser usada simultaneamente por dois transmissores sem causar interferência se eles estiverem a uma distância suficiente chamada distância de reutilização. Note que esse problema pode ser modelado como um grafo em que os vértices são transmissores e as arestas entre dois vértices indicam que pode haver uma interferência entre os transmissores correspondentes.

O Algoritmo Guloso de Coloração é uma forma de determinar uma coloração própria dos vértices de um grafo. Esse algoritmo considera uma sequência arbitrária dos vértices em uma ordem dada como entrada, e atribui a cada vértice a menor cor que esteja disponível. Uma coloração é chamada de coloração de Grundy se tal coloração pode ser obtida pelo Algoritmo Guloso de Coloração.

A menor quantidade de cores utilizadas para colorir os vértices de um grafo é chamado de número cromático. Determinar o número cromático para grafos em geral é um problema NP-difícil (KARP, 1972), porém, para algumas classes de grafos, o número cromático pode ser determinado em tempo polinomial.

Similar ao número cromático, a maior quantidade de cores que o Algoritmo Guloso de Coloração pode atribuir aos vértices de um grafo é chamado de número de Grundy ou também número cromático guloso e é denotado por $\Gamma(G)$. O número de Grundy determina que existe uma ordem dos vértices do grafo que, quando usada como entrada, o Algoritmo Guloso de Coloração

retorna uma coloração com o máximo de cores possíveis.

Segundo Effantin e Kheddouci (2007), o número de Grundy possui aplicações em campos como escalonamento ou arquiteturas de multiprocessador. Por exemplo, suponha um conjunto de processos tal que um processo P_i poderia ser calculado se os processos $P_1, P_2, \ldots, P_{i-1}$ estiverem calculados. Essa ordem sobre os processos pode ser modelada como uma coloração de Grundy.

De acordo com Effantin e Kheddouci (2007), o número de Grundy já foi determinado para certas classes de grafos, como por exemplo os grafos ciclos, caminhos, completos e bipartidos completos. Para alguma classes de grafos, o número de Grundy pode ser obtido por meio de algoritmos em tempo polinomial. Hedetniemi S. Hedetniemi (1982) forneceram um algoritmo linear para o número de Grundy para a classe de grafos árvores. Telle e Proskurowski (1997) forneceram um algoritmo para o número de Grundy de árvores k-parciais com complexidade de tempo $O(n^{3k^2})$ e limitaram este parâmetro para esses grafos pelo valor $1 + k\lg(n)$, onde n é o número de vértices do grafo.

Neste trabalho, implementamos os seguintes algoritmos exatos para determinar o número de Grundy:

1. Uma variação do algoritmo de programação dinâmica proposto por Bonnet *et al.* (2018) com a complexidade de tempo $O(dn1.4423^d 2^n)$ baseado no algoritmo proposto por Lawler (1976). O número de Grundy é calculado listando todos os conjuntos independentes maximais de G. Para a enumeração dos conjuntos independentes maximais, implementamos o algoritmo de Bron-Kerbosh combinado com a ordem de degeneração do grafo da maneira proposta por Eppstein *et al.* (2010).
2. Um algoritmo de Branch and Bound inspirado no algoritmo proposto por Brélaz (1979) para o problema da coloração. Utilizando a ordem de degeneração proposta por Matula e Beck (1983) como estratégia de ramificação no algoritmo de Branch and Bound e um limite superior baseado no parâmetro *stair factor* proposto por Shi *et al.* (2005) para reduzir o fator de ramificação do algoritmo.
3. Três formulações de Programação Linear Inteira apresentadas em Rodrigues (2020) utilizando, respectivamente, os parâmetros *stair factor*, $\Delta(G) \cdot 2^{\Gamma(H)-1} + \Gamma(H)$ e $\Delta + 1$ para reduzir o número de restrições da formulação.

Os algoritmos implementados acima foram testados na classe de grafos Stacked Book.

Este documento está organizado da seguinte forma. No Capítulo 2, apresentamos conceitos e definições necessários para a compreensão do texto. No Capítulo 3, são mostrados os trabalhos relacionados, com suas semelhanças e diferenças ao trabalho aqui proposto. Já nos Capítulos 4, 5 e 6, são descritas, respectivamente, um limite inferior, um limite superior e os algoritmos exatos propostos neste trabalho. No Capítulo 7, são descritos os resultados obtidos nos testes computacionais realizados. Por fim, no Capítulo 8, são apresentadas as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, apresentamos conceitos e definições que são necessários para a compreensão do texto. Na seção 2.1, são abordados os conceitos principais de grafos. Na seção 2.2, são abordados os conceitos principais de colorações de vértices. Na seção 2.3, são abordados os principais conceitos e resultados do parâmetro do número de Grundy. Na seção 2.4, são apresentados os conceitos principais de Complexidade Parametrizada. Na seção 2.5, são apresentados os conceitos principais do método de Programação Dinâmica. Na seção 2.7, são apresentados os conceitos principais de um Algoritmo Branch and Bound. Por fim, na seção 2.6, são apresentados os conceitos de Programação Linear e Programação Linear Inteira.

2.1 Grafo

Um *grafo simples* G é um par ordenado $G = (V(G), E(G))$, composto por um conjunto finito $V(G) = \{1, 2, \ldots, n\}$, cujos elementos são chamados de *vértices*, e por um conjunto de pares não ordenados $E(G) \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$, cujos elementos são chamados de *arestas*. A partir de agora, vamos representar o conjunto $\{u, v\}$ simplesmente por uv ou vu. Definimos por $n = |V(G)|$ e $m = |E(G)|$. Os vértices $u, v \in V(G)$ são chamados *adjacentes (ou vizinhos)* se $uv \in E(G)$ e *não-adjacentes* se $uv \notin E(G)$. As arestas $e, f \in E(G)$ são ditas adjacentes se $e \cap f \neq \emptyset$ e não adjacentes se $e \cap f = \emptyset$. O conjunto dos *vizinhos* de um vértice v é denotado por $N(v)$.

Um grafo H é dito *subgrafo* do grafo G, denotado por $H \subseteq G$, se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Um *subgrafo induzido* por $V' \subseteq V(G)$, denotado por $G[V']$, é o subgrafo que possui V' como conjunto de vértices e, além disso, para todo $u, v \in V'$, se $uv \in E(G)$ então $uv \in E(G[V'])$.

O *grau* de um vértice v em um grafo G, denotado por $d_G(v)$, corresponde ao número de vizinhos de v no grafo G, isto é $|N(V)|$. O *grau máximo* de um vértice no grafo G é denotado por $\Delta(G)$ e o *grau mínimo* é denotado por $\delta(G)$.

Um teorema bem conhecido em Teoria dos Grafos trata da soma dos graus dos vértices de um grafo. Este teorema foi evidentemente observado pela primeira vez pelo grande matemático suíço Leonhard Euler em um artigo de 1736 que é considerado o primeiro artigo já escrito sobre a teoria dos grafos, embora os grafos nunca tenham sido mencionados no artigo. É frequentemente referido como o primeiro teorema em Teoria dos Grafos e é apresentado logo

abaixo no Teorema 2.1.1.

Teorema 2.1.1 ((EULER, 1736)). *Se G é um grafo, então* $\sum_{v \in V(G)} d_G(v) = 2m$.

Um grafo é dito *trivial* se o conjunto de vértices contém apenas um vértice e *vazio* se o conjunto de arestas é vazio.

Um (v_0, v_k)-*caminho* em G é uma sequência $v_0, v_1, \ldots, v_k$ finita e não vazia de vértices de G, tal que os termos da sequência não se repetem e $v_{i-1}v_i \in E(G)$ para $1 \leq i \leq k$. A *distância* entre dois vértices $u, v \in V(G)$, denotada por $d_G(u,v)$, é o número de arestas em um menor (u,v)-caminho em G. Um grafo G é *conexo* se existe um (u,v)-caminho em G para quaisquer dois vértices $u, v \in V(G)$.

Um *ciclo* é uma sequência $v_0, v_1, \ldots, v_k, v_0$ finita e não vazia de vértices de G, tal que o primeiro e o último termo são iguais e os demais termos da sequência não se repetem e $v_{i-1}v_i \in E(G)$ para $1 \leq i \leq k$ e $v_0v_k \in E(G)$. Uma *árvore* é um grafo conexo *acíclico*, ou seja, que não contém ciclo. Se G é uma árvore então não existe um subgrafo $C \subseteq G$, tal que C é um ciclo.

Um *grafo completo* é um grafo com $n \geq 3$ vértices, denotado por K_n, no qual, para quaisquer dois vértices $u, v \in V(G)$, temos que $uv \in E(G)$. Em um grafo completo, o número de arestas é igual a $\frac{n(n-1)}{2}$.

Um grafo G é *bipartido* se seu conjunto de vértices pode ser particionado em dois subconjuntos disjuntos não vazios X e Y, de forma que, para toda aresta $uv \in E(G)$, temos que $u \in X$ e $v \in Y$, ou $u \in Y$ e $v \in X$. Um grafo G é *bipartido completo*, quando todo vértice de X é adjacente a todos os vértices de Y e é denotado por $K_{p,q}$, onde $p = |X|$ e $q = |Y|$. O *grafo estrela*, denotado por S_p, é o grafo bipartido completo $K_{1,p}$:

Uma *clique* é um subconjunto de vértices $C \subseteq V(G)$, tal que o grafo induzido $G[C]$ é completo. Uma *clique é maximal* se não pode ser estendida ao se adicionar um ou mais vértices adjacentes. Uma clique máxima é a clique de maior cardinalidade no grafo. O tamanho de uma clique máxima é denotado por $\omega(G)$.

Um *conjunto independente de vértices* é um subconjunto $S \subseteq V(G)$, tal que, para quaisquer dois vértices $u, v \in S$, temos que $uv \neq E(G)$. Um *conjunto independente máximo de vértices* é aquele de maior cardinalidade no grafo G e o seu tamanho é denotado por $\alpha(G)$. Um grafo com n vértices contém no máximo $3^{\frac{n}{3}}$ conjuntos independentes máximos (MOON; MOSER, 1965).

O *complemento* ou *inverso* de G denotado por $\overline{G}$, é um grafo H com $V(H) = V(G)$ tal que, $u, v \in V(\overline{H})$ se e somente se $uv \notin E(G)$. O complemento do grafo vazio é o grafo completo e vice-versa. Qualquer subgrafo induzido do grafo do complemento de G é o complemento do subgrafo induzido correspondente em G.

Um conjunto independente de vértices em G é um clique no grafo do complemento e vice-versa. Este é um caso especial das duas propriedades anteriores, pois um conjunto independente é um subgrafo induzido sem arestas e um clique é um subgrafo induzido completo.

Uma *decomposição de vértices* é uma sequência $D = (v_1, \ldots, v_n)$ de um grafo G calculada removendo repetidamente um vértice v_i de grau máximo de $G_i = G - \{v_1, \ldots, v_{i-1}\}$ (Observe que v_1 é um vértice de grau máximo em G). O grau do vértice v_i na remoção é chamado de *grau residual* e denotamos a sequência de graus residuais por $(d_1, \ldots, d_n)$.

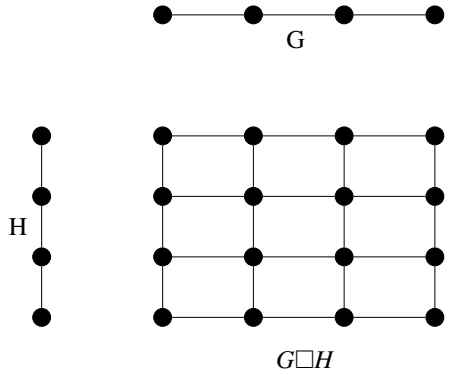
Um grafo G é *k-degenerado* se seus vértices podem ser sucessivamente removidos do grafo de modo que, quando removidos, cada um tem grau no máximo k. Esta ordem de remoção dos vértices é denominada *ordem de degeneração* do grafo G.

A maioria dos conceitos e definições de grafos citados anteriormente são provenientes dos livros (BONDY; MURTY, 1982), (WEST, 2018) e (CHARTRAND; ZHANG, 2008) e são essencialmente necessários para a compreensão deste trabalho.

2.1.1 Produto cartesiano de grafos

Um *produto cartesiano* de dois grafos G e H, denotado por $G \square H$, é o grafo com, conjunto de vértices é $V = V(G) \times V(H)$, e quaisquer dois vértices (u, u') e (v, v') são adjacentes, se e somente se, $u = v$ e $u'v' \in E(H)$ ou $u' = v'$ e $uv \in E(G)$. A Figura 1 exibe um exemplo de produto cartesiano entre dois grafos G e H.

Figura 1 – Produto Cartesiano $G \square H$

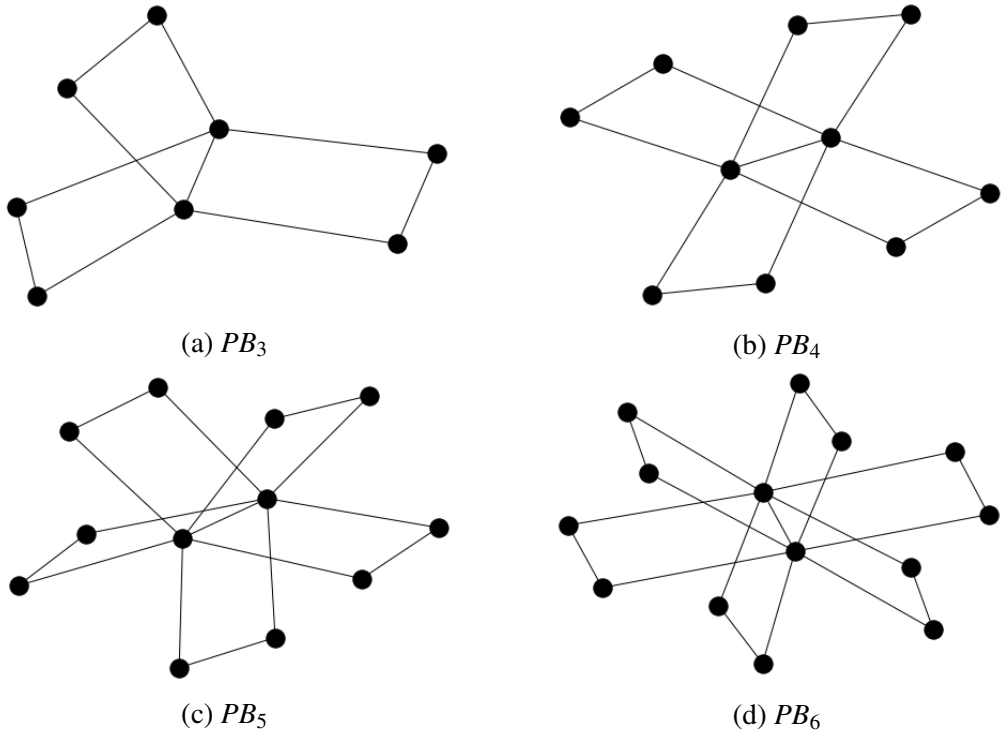


Fonte: Elaborado pelo autor (2022).

2.1.2 Grafos Book

O *grafo* $p-book$ denotado por PB_p é definido como o grafo do produto cartesiano $S_p \square P_2$, onde S_p é um grafo estrela e P_2 é o grafo caminho com dois vértices. A Figura 2 ilustra alguns exemplos de grafos PB_p.

Figura 2 – Exemplos de grafos Book PB_p



(a) PB_3 (b) PB_4

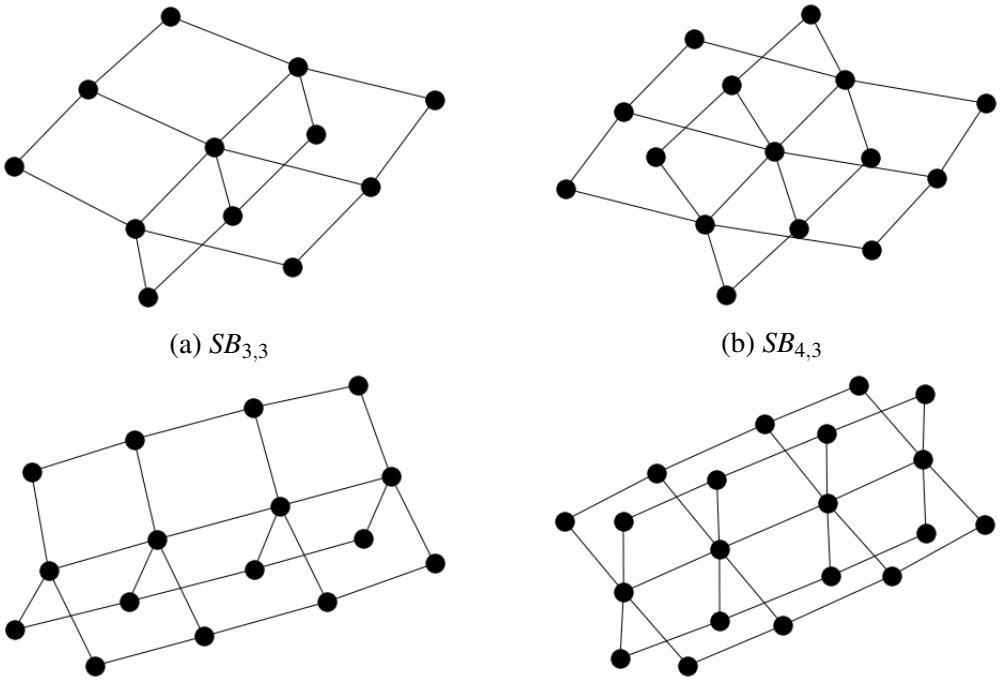
(c) PB_5 (d) PB_6

Fonte: Elaborado pelo autor (2022).

2.1.3 Grafos Stacked Book

O *grafo* $(p,q)-$*Stacked Book* denotado por $SB_{p,q}$ é definido como o grafo do produto cartesiano $S_p \square P_q$, onde S_p é um grafo estrela e P_q é o grafo caminho com q vértices. O grafo $SB_{p,q}$ contém $q \cdot (p+1)$ vértices e $q \cdot (2p+1) - (p+1)$ arestas. Este grafo é correspondente a uma generalização de um grafo $p-$book e sua representação pode ser interpretada como q cópias de um $p-$book empilhadas uma em cima da outra. A Figura 3 ilustra alguns exemplos de grafos $SB_{p,q}$.

Figura 3 – Exemplos de grafos Stacked Book $SB_{p,q}$



(a) $SB_{3,3}$ (b) $SB_{4,3}$

(c) $SB_{3,4}$ (d) $SB_{4,4}$

Fonte: Elaborado pelo autor (2022).

2.2 Coloração de vértices

2.2.1 Coloração própria de vértices

Dado um grafo $G = (V(G), E(G))$, uma coloração de vértices, ou apenas coloração, é uma função $c : V \to \{1, 2, \ldots, k\}$, $k \in \mathbb{N}$, sendo $\mathbb{N}$ o conjunto dos números inteiros positivos. Os valores de $\{1, 2, \ldots, k\}$ são chamados de *cores*. Comumente, utilizamos o conjunto $S = \{1, 2, \ldots, k\}$ e dizemos que $c(u)$ é uma cor atribuída ao vértice u. Dizemos que c é uma k-coloração do grafo G, quando $|S| = k$. Uma coloração de vértices c é dita própria quando, para todo $u, v \in V(G)$, temos que se $uv \in E(G)$, então $c(u) \neq c(v)$.

Uma *k-coloração própria* também é definida de forma equivalente como uma partição do conjunto de vértices de G em subconjuntos independentes $\{S_1, \ldots, S_k\}$, $k \in \mathbb{N}$. Cada conjunto S_i é chamado de classe de cor.

2.2.2 Número Cromático

Uma coloração própria de vértices $c\colon V(G) \to S$ é dita *ótima*, quando $|S|$ é o menor possível. O valor de $|S|$ em uma coloração própria de vértices ótima é chamado de *número cromático* e é denotado por $\chi(G)$.

Teorema 2.2.1 ((WEST, 2018)). *Se G é um grafo, então* $\chi(G) \leq \Delta(G) + 1$.

Teorema 2.2.2 ((BROOKS, 1941)). *Seja G um grafo conexo tal que G não é um grafo completo ou um ciclo ímpar. Então* $\chi(G) \leq \Delta(G)$.

Os Teoremas 2.2.1 e 2.2.2 determinam um limitante superior para o $\chi(G)$. Portanto, $\Delta(G) + 1$ cores são suficientes para colorir todos os vértices de um grafo G arbitrário.

2.2.3 Coloração k-imprópria de vértices

Uma *coloração imprópria* é uma generalização de coloração própria de vértices, em que relaxa-se a restrição de vértices adjacentes receberem cores distintas. Dizemos que G possui uma $(m,k)-$coloração ou uma coloração $k-$imprópria com m cores se seus vértices podem ser coloridos com m cores de forma que cada vértice é adjacente a, no máximo, k vizinhos coloridos com a mesma cor (RODRIGUES, 2020). Uma $(m,k)-$coloração pode ser vista como uma partição de $V(G)$ em m classes de cores na qual cada classe de cor induz um subgrafo onde o grau máximo não excede k.

2.2.4 Algoritmo Guloso de Coloração

A coloração de grafos é um dos problemas de otimização combinatória mais estudados. O problema representa um grande desafio teórico uma vez que o problema pertence a classe dos problemas NP-completo para os quais nenhum Algoritmo polinomial é conhecido para os problemas desta classe (CORMEN *et al.*, 2009). Comumente, neste problema, métodos heurísticos são empregados para encontrar soluções sub-ótimas para o problema. Os métodos heurísticos são Algoritmos que buscam a melhor solução possível de um problema, porém não garantem encontrar a solução ótima. Um dos Algoritmos de coloração de grafos que emprega heurística é o Algoritmo Guloso de Coloração, que determina uma coloração própria de vértices de um grafo.

Algoritmo 1: COLORACAOGULOSA(G,θ)

Entrada: Grafo $G = (V(G), E(G))$, Ordem de vértices $\theta = (v_1, v_2, \ldots, v_n)$ de $V(G)$

```
1 Coloração própria c de G início
2     para cada v_i ∈ θ hacer
3         c(v_i) ← a menor cor não utilizada em N(v_i) ∩ {v_1, ..., v_{i-1}}
4     fin
5     c
6 fim
```

Fonte: Elaborado pelo autor (2022).

O Algoritmo 1 considera um grafo G e uma sequência arbitrária do vértices de G em uma ordem dada como entrada $\theta = (v_1, \ldots, v_n)$ e atribui cores aos vértices de G seguindo a ordem θ de modo que cada vértice recebe a menor cor que esteja disponível. A coloração c retornada pelo Algoritmo 1 é um coloração própria de vértices.

Uma *coloração c é gulosa* se ela pode ser gerada pelo Algoritmo 1. Uma coloração c é chamada $k-$*coloração gulosa* se são utilizadas k cores para colorir um grafo G. Sendo assim, qualquer coloração c gerada para um grafo G pelo Algoritmo 1 que utiliza k cores é um limitante superior para o número cromático $\chi(G)$. Desta forma, existe uma ordem θ dos vértices de um grafo G passada como entrada para o Algoritmo 1 que retorna uma $\chi(G)$-coloração gulosa.

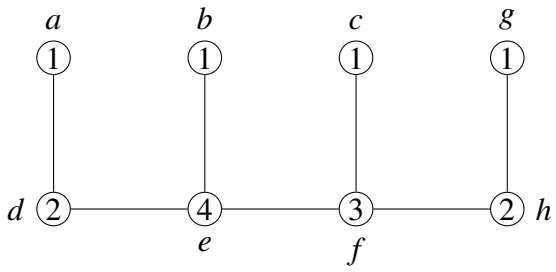
2.2.5 Coloração de Grundy

Em uma coloração de G com k classes de cores $\{S_1, \ldots, S_k\}$, um vértice $v \in S_i$ é um vértice de Grundy, ou vértice guloso, se v é adjacente a pelo menos um vértice na classe de cor S_j , para cada $j < i$ (RODRIGUES, 2020). Uma *coloração de Grundy* ou *Coloração Gulosa* é uma coloração onde cada vértice $v \in V$ é um vértice de Grundy. Todas as colorações geradas pelo Algoritmo 1 são colorações de Grundy. A Figura 4 exibe uma coloração de Grundy com 4 cores e, portanto, também é uma 4-coloração própria. No entanto, a 3-coloração exibida na Figura 5 não é uma coloração de Grundy com 3 cores.

2.3 Número de Grundy

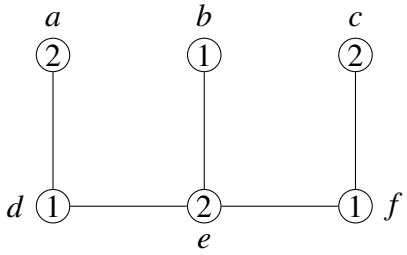
Dado um grafo G, o *número de Grundy* ou *número guloso* de G é o maior k tal que G possui uma k-coloração gulosa dentre todas as possíveis ordenações de $V(G)$. O número de

Figura 4 – Coloração de Grundy de G com 4 cores



Fonte: Elaborado pelo autor (2022).

Figura 5 – Coloração de G com 3 cores



Fonte: Elaborado pelo autor (2022).

Grundy, denotado por $\Gamma(G)$, consiste no maior número de cores que podem ser utilizadas pelo Algoritmo 1. Consequentemente, $\Gamma(G)$ é o maior inteiro k tal que G admite uma coloração de Grundy com k cores. O número de Grundy é o conceito mais importante do nosso trabalho e será fundamental durante todo o desenvolvimento dos Algoritmos.

O número de Grundy determina o pior caso de k-coloração gulosa de G obtida pelo Algoritmo 1. Sendo assim, o parâmetro $\Gamma(G)$ determina um limitante superior para o número cromático $\chi(G)$.

Teorema 2.3.1 ((CHARTRAND; ZHANG, 2008))**.** *Seja G um grafo então* $\chi(G) \leq \Gamma(G)$.

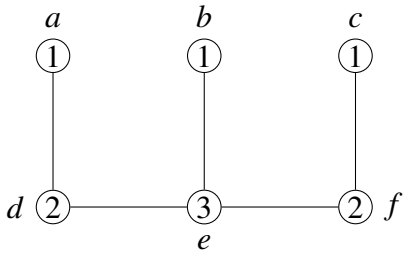
O limitante superior Teorema 2.3.2 é obtido a partir do Algoritmo 1.

Teorema 2.3.2 ((CHARTRAND; ZHANG, 2008))**.** *Seja G um grafo então* $\Gamma(G) \leq \Delta(G) + 1$.

Na Figura 6, é exibida uma coloração de Grundy que usa o máximo possível de cores e essa é a mesma quantidade máxima de cores que o Algoritmo 1 consegue utilizar para colorir o grafo da Figura 6.

Segundo Bonnet *et al.* (2018), o conceito do número de Grundy foi estudado pela primeira vez por Grundy em 1939 no contexto de grafos direcionados e jogos (GRUNDY, 1939), e formalmente introduzido 40 anos mais tarde por Christen e Selkow (1979). Goyal e Vishwanathan (1997) provaram que o problema de determinar o número de Grundy é NP-completo para grafos em geral.

Figura 6 – Coloração própria de G com número de Grundy $\Gamma(G) = 3$



Fonte: Elaborado pelo autor (2022).

Segundo Garey e Johnson (1979), determinar o número de Grundy em grafos direcionados é um problema NP-completo. Mesmo para grafos não-direcionados o problema ainda permanece NP-difícil em grafos bipartidos (HAVET; SAMPAIO, 2013) e seus complementos (ZAKER, 2005), grafos cordais (SAMPAIO, 2012) e grafos linha (HAVET *et al.*, 2015). Segundo Effantin e Kheddouci (2007), para algumas classes de grafo, como por exemplo, grafos ciclos, caminhos, completos e bipartido completos o valor do número de Grundy é determinado, respectivamente, pelo Teoremas 2.3.3, 2.3.4, 2.3.5 e 2.3.6 .

Teorema 2.3.3 ((EFFANTIN; KHEDDOUCI, 2007))**.** *Se um grafo G é um grafo ciclo C_n então:*

$$\Gamma(C_n) = \begin{cases} 2, \textit{ se n=4} \\ 3, \textit{ caso contrário.} \end{cases}$$

Teorema 2.3.4 ((EFFANTIN; KHEDDOUCI, 2007))**.** *Se um grafo G é um grafo caminho P_n então:*

$$\Gamma(P_n) = \begin{cases} 2, \textit{ se n=2 ou n=3} \\ 3, \textit{ caso contrário.} \end{cases}$$

Teorema 2.3.5 ((EFFANTIN; KHEDDOUCI, 2007))**.** *Se um grafo G é um grafo completo K_n então $\Gamma(K_n) = n$.*

Teorema 2.3.6 ((EFFANTIN; KHEDDOUCI, 2007))**.** *Se um grafo G é um grafo bipartido completo $K_{n,m}$ então $\Gamma(K_{n,m}) = 2$.*

2.4 Complexidade Parametrizada

Em ciência da computação, a Teoria da Complexidade Parametrizada é um ramo da teoria da complexidade computacional que foca em classificar problemas computacionais de acordo com sua dificuldade inerente com respeito a múltiplos parâmetros da entrada. Esta Teoria surgiu como uma alternativa promissora para classificar problemas considerados NP-completos.

Sob a hipótese de que $P \neq NP$, existem vários problemas naturais que requerem tempo de execução exponencial quando a complexidade é medida apenas baseada no tamanho do parâmetro de entrada. Entretanto, alguns problemas podem ser resolvidos por Algoritmos que são polinomiais apenas no tamanho de um parâmetro fixo, enquanto são exponenciais no tamanho do parâmetro de entrada. Problemas em que algum parâmetro k é fixado, são chamados de problemas tratáveis em parâmetro fixos

Segundo Downey e Fellows (2013), a classe FPT é uma classe de problemas que contém os problemas tratáveis em parâmetro fixo, que são aqueles que podem ser resolvidos em tempo $f(k) \cdot |x|^{O(1)}$ para alguma função computável f. Tipicamente, essa função é vista como unicamente exponencial, como por exemplo $2^{O(k)}$ mas a definição permite funções que crescem ainda mais rápido. Se um Algoritmo parametrizado tem tempo de execução $f(k) \cdot |x|^c$, escreveremos que o Algoritmo está rodando no tempo $O^*(f(k))$, ou seja, ignorando a parte polinomial e se concentrando na parte exponencial.

No trabalho de Aboulker *et al.* (), foi provada a complexidade parametrizada da Coloração de Grundy. Na verdade, provaram que não há função computável f tal que Coloração de Grundy seja solucionável em tempo $f(k)n^{O(2^{k-logk})}$. Este resultado responde de forma negativa uma questão postada no trabalho de Sampaio (2012), se existe um Algoritmo no tempo $n^{k^{O(1)}}$. Também foi provado que a Coloração de Grundy pertence à classe FPT para o parâmetro k (números de cores) em uma classe específica de grafos. Além disso, foi estabelecido que a Coloração de Grundy é um problema tratável de parâmetro fixo em uma subclasse dos grafos $K_{t,t}$, mesmo no parâmetro $k+t$, assumindo que t não é uma constante fixa.

2.5 Programação dinâmica

A programação dinâmica é um método para a construção de Algoritmos que resolvem problemas combinando as soluções para os subproblemas. Este método foi criado por Richard Bellman (BELLMAN, 1966) e diversos de seus trabalhos realçam a sua importância e suas numerosas aplicações. A programação dinâmica é aplicada quando os subproblemas se sobrepõem, isto é, quando os subproblemas compartilham os mesmos subproblemas.

Um Algoritmo de programação dinâmica resolve cada subproblema apenas uma vez e, em seguida, salva sua resposta em uma tabela, evitando assim o trabalho de recalcular a resposta toda vez que resolver cada subproblema.

Normalmente aplicamos a programação dinâmica na resolução de problemas com-

putacionais , em especial os de otimização combinatória. Tais problemas podem ter muitas soluções possíveis e como cada solução tem um valor, queremos encontrar uma solução com o valor ideal (mínimo ou máximo).

Problemas de programação dinâmica podem ser abordados de duas formas: ***top-down*** ou ***bottom-up***. A primeira abordagem é o método *top-down* com memoização. Nesta abordagem, escrevemos o procedimento recursivamente de maneira natural, mas modificado para salvar o resultado de cada subproblema (geralmente em uma matriz ou tabela hash). O procedimento agora verifica primeiro se esse subproblema já foi resolvido. Em caso afirmativo, ele retorna o valor salvo, salvando cálculos adicionais neste nível; caso contrário, o procedimento calcula o valor da maneira usual.

A segunda abordagem é o método *bottom-up*. Essa abordagem normalmente depende de alguma noção natural do "tamanho"de um subproblema, de modo que a solução de qualquer subproblema particular dependa apenas da solução de subproblemas "menores". Inicialmente, classificamos os subproblemas por tamanho e os resolvemos em ordem de tamanho, o menor primeiro. Ao resolver um subproblema específico, já resolvemos todos os subproblemas menores dos quais sua solução depende e salvamos suas soluções. Resolvemos cada subproblema apenas uma vez e, quando o vemos pela primeira vez, já resolvemos todos os seus subproblemas de pré-requisito.

2.6 Programação Linear

A Programação Linear (PL) ou otimização linear pode ser definida como o problema de maximizar ou minimizar uma função linear que está sujeita a restrições lineares. As restrições podem ser igualdades ou desigualdades. As restrições determinam uma região a qual se denomina de conjunto viável, a melhor das soluções viáveis (soluções que pertencem ao conjunto viável), ou seja, aquela que maximiza ou minimiza a função objetivo, denomina-se solução ótima (MARINS, 2011).

Em outras palavras, a Programação Linear é considerada como um método de otimização para maximizar ou minimizar a função objetivo de um determinado modelo matemático com o conjunto de alguns requisitos que estão representados na relação linear com o principal objetivo de encontrar a solução ótima.

Os componentes básicos de um modelo de PL são os seguintes: Variáveis de decisão, restrições, matriz de coeficientes e a função objetivo. Um Problema de Programação Linear

(PPL) pode ser expresso de forma geral conforme o modelo matemático abaixo:

Figura 7 – Representação do modelo geral de Programação Linear.

$$\max \quad c^T x \tag{2.1}$$
$$s.a \qquad Ax \leq b \tag{2.2}$$
$$x \geq 0 \tag{2.3}$$

Fonte: Elaborado pelo autor (2022).

Os termos do modelo da Figura 7 representam: $c \in \mathbb{R}^n$ e $b \in \mathbb{R}^m$ vetores de constantes, $x \in \mathbb{R}^n$ o vetor das variáveis de decisão e $A \in \mathbb{R}^{m \times n}$ a matriz de coeficientes das restrições. Neste modelo todas as restrições da expressão (2.2) são de desigualdade e a função objetivo é de maximização.

Para a resolução de um PPL, dois passos são necessários. O primeiro é a modelagem do problema, seguindo-se o método de resolução do modelo. No caso de um PPL, o método mais utilizado para a resolução é o método Simplex (MARINS, 2011).

Em 1963, George Dantzig (DANTZIG, 2016) introduziu o método Simplex, que é um procedimento iterativo que fornece a solução de qualquer modelo de PL em um número finito de iterações. Além disso, este método determina também se o modelo tem solução ilimitada, se não tem solução, ou se possui infinitas soluções. Este método é exponencial no pior caso. Porém, na prática, seu desempenho costuma ser melhor que Algoritmos polinomiais como o Algoritmo de pontos interiores.

2.6.1 Programação Linear Inteira

Um Problema ou Programação Linear Inteiro (PLI) é um método de otimização em que o vetor de variáveis de decisão x são restritas a números inteiros não negativos. Outro caso específico do PLI, é a Programação Linear Inteira Binária (PLIB), onde o vetor de variáveis de decisão x são restritas ao conjuntos de números inteiros $\{0, 1\}$. Na Figura 8 abaixo é exibida a representação do modelo matemático de Programação Linear Inteira. Neste modelo, todas as restrições da expressão (2.5) são de desigualdade e a função objetivo é de minimização.

Resolver problemas de PLI é uma tarefa difícil em geral. Uma abordagem comumente usada em matemática computacional para facilitar a resolução do PLI é utilizar técnicas de relaxação lineares. Uma relaxação linear é uma técnica que elimina as restrições de integralidade

Figura 8 – Representação do modelo de Programação Linear Inteira.

$$\min \quad c^T x \tag{2.4}$$
$$s.a \quad Ax \leq b \tag{2.5}$$
$$x \in \mathbb{Z}_+^n \tag{2.6}$$

Fonte: Elaborado pelo autor (2022).

do modelo, transformando em um PPL. Como essas restrições são retiradas, o resultado do modelo relaxado é um limite inferior para o problema de minimização, caso o problema seja de maximação a relaxação nos fornece um limite superior para o modelo PLI. Geralmente, são utilizados solvers para a resolução de modelos de PL.

2.6.2 Google OR-Tools

Neste trabalho, foi utilizada a ferramenta OR-Tools para a resolução do modelos de PLI. Esta ferramenta é um pacote de software gratuito e de código aberto desenvolvido pelo Google para resolver problemas de Programação Linear (PL), Programação Inteira Mista (MIP), Programação Linear por Restrição (CP), roteamento de veículos (VRP) e problemas de otimização relacionados.

As ferramentas do OR-Tools suportam uma variedade de linguagens de programação, incluindo: C++, .NET, Java e Python. Além disso, ela contém diversos tipos de solvers, como Gurobi ou CPLEX, ou solucionadores de código aberto, como SCIP, GLPK ou Google's GLOP e CP-SAT.

2.7 Branch and Bound

A estrutura Branch and Bound $(B\&B)$ é uma metodologia fundamental e amplamente utilizada para a produção de soluções exatas para problemas de otimização NP-difíceis (MORRISON *et al.*, 2016).

A técnica *B&B*, foi proposta pela primeira vez por Land e Doig (1960), e é frequentemente referida como um Algoritmo, no entanto, talvez seja mais apropriado dizer que o *B&B* encapsula uma família de Algoritmos que compartilham um procedimento de solução central comum.

Este técnica enumera todas as soluções possíveis para o problema, armazenando

soluções parciais chamadas subproblemas em uma estrutura de árvore. Nós inexplorados na árvore geram filhos, particionando o espaço de solução em regiões menores, que podem ser resolvidas ramificando recursivamente usando regras para diminuir regiões do espaço de busca. Uma vez que toda a árvore foi explorada, a melhor solução encontrada na pesquisa é retornada. Uma visão geral inicial do Algoritmo principal de *B&B* foi fornecida por Lawler e Wood (1966).

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados alguns trabalhos relacionados com o trabalho proposto mostrando os principais resultados e algumas questões em aberto. Por fim, na seção 3.5 é feita uma comparação entre este trabalho e os trabalhos descritos neste capítulo.

3.1 Complexity of Grundy coloring and its variants

No trabalho de Bonnet *et al.* (2018), é feito o estudo do cálculo da complexidade computacional da Coloração de Grundy e do problema de determinar se um determinado grafo possui $\Gamma(G) = k$. Também estudaram as variantes da Coloração de Grundy: A Coloração de Grundy Fraca e a Coloração Grundy conectada.

Bonnet *et al.* (2018) provaram a complexidade da Coloração de Grundy e Coloração de Grundy Fraca para grafos de ordem n, como apresentamos nos Teoremas 3.1.1 e 3.1.2. Embora a Coloração de Grundy e a Coloração de Grundy Fraca sejam conhecidas por serem solucionáveis no tempo $O^*(2^{O(wk)})$ para árvores de largura w (onde k é o número de cores), provaram que não podem ser resolvidos com complexidade parametrizada de tempo $O^*(2^{O(w \lg w)})$.

Teorema 3.1.1 ((BONNET *et al.*, 2018))**.** *A coloração de Grundy pode ser resolvida no tempo* $O(nm2.4423^n)$.

Teorema 3.1.2 ((BONNET *et al.*, 2018))**.** *A coloração de Grundy Fraca pode ser resolvida no tempo* $O(nm2.7159^n)$.

Nesse mesmo trabalho, Bonnet *et al.* (2018) descreveram um algoritmo com complexidade de tempo $O^*(2^{2^{O(k)}})$ para Coloração de Grundy Fraca. Além disso, provaram que esse tempo de execução é essencialmente ótimo (este limite inferior também é válido para a Coloração de Grundy). Além disso, descreveram um algoritmo com complexidade de tempo quase polinomial para Coloração de Grundy e Coloração de Grundy Fraca em uma determinada classe de grafos. Em forte contraste com os outros dois problemas, mostraram que o Coloração de Grundy Conectada já está $NP-$completo para $k = 7$ cores.

Como trabalhos futuros, os autores recomendaram que seria interessante determinar a complexidade (clássica) da Coloração de Grundy em grafos de intervalo e grafos bipartidos cordais ou estudar a Coloração de Grundy Conectada em classes de grafos restritas. Além

disso, observaram um algoritmo na literatura que talvez possua complexidade de tempo quase polinomial para grafos planares, assim tornando improvável que seja $NP-$completo nesta classe.

Por fim, deixaram as seguintes questões em aberto: A complexidade exata da aproximação de tempo polinomial da Coloração de Grundy, complexidade exata da Coloração de Grundy Fraca, complexidade de aproximação de Coloração de Grundy Conectada e o status de complexidade para $k = 4,5,6$ da Coloração de Grundy.

3.2 Grundy number and products of graphs

No trabalho de Asté *et al.* (2010), é estudado o número de Grundy em produtos lexicográficos e cartesiano de dois grafos G e H em termos dos seus números de Grundy. Mais precisamente, são fornecido limitantes inferiores e superiores para os produtos lexicográficos.

Para o produto cartesiano de dois grafos, foi mostrado o Teorema 3.2.1 e este limite inferior foi aumentado em alguns casos particulares. Especificamente, foi mostrado que, para o bipartido completo $K_{p,p}$, temos que $\Gamma(K_{p,p}) = 2$ mas, $\Gamma(K_{p,p}\square K_{p,p}) \geq p+1$.

Teorema 3.2.1 ((ASTÉ *et al.*, 2010))**.** *Como G e H são subgrafos induzidos de $G\square H$ então* $\Gamma(G\square H) \geq \max\{\Gamma(G),\Gamma(H)\}$.

Neste trabalho, também mostraram que o produto cartesiano, em contraste com o produto lexicográfico, não possui limite superior para o $\Gamma(G\square H)$ em função de $\Gamma(G)$ e $\Gamma(H)$. No entanto, eles mostraram dois limitantes superiores para o $\Gamma(G\square H)$ que é limitado em função de $\Delta(G)$ e $\Gamma(H)$ no Teorema 3.2.2 e na Conjectura 3.2.1.

Teorema 3.2.2 ((ASTÉ *et al.*, 2010))**.** *Para quaisquer dois grafos G e H, temos que* $\Gamma(G\square H) \leq \Delta(G)\cdot 2^{\Gamma(H)-1} + \Gamma(H)$.

Conjectura 3.2.1 ((ASTÉ *et al.*, 2010))**.** *Para quaisquer dois grafos G e H temos que,* $\Gamma(G\square H) \leq (\Delta(G)+1)\cdot\Gamma(H)$.

3.3 New bounds on the Grundy number of products of graphs

O trabalho de Campos *et al.* (2012), apresenta diversos resultados dos autores relacionado ao número de Grundy na classe de produtos de grafos não-direcionados. Especificamente, é fornecido limitantes para o produto direto, produto lexicográfico, produto cartesiano e o produto forte de dois grafos G e H, que generaliza para valores exatos para algumas classes especiais.

Havet, Kaiser e Stehlik provaram a Conjectura 3.2.1 do trabalho de Asté *et al.* (2010) no caso em que G é um grafo arbitrário e H é uma árvore no Teorema 3.3.1.

Teorema 3.3.1 ((HAVET *et al.*, 2008)). *Para qualquer grafo G e uma árvore H temos que,* $\Gamma(G\square H) \leq (\Delta(G)+1)\cdot\Gamma(H)$.

Neste mesmo trabalho, foi obtido o Teorema 3.3.2, que valida a Conjectura 3.2.1 caso $\Gamma(H) = 2$.

Teorema 3.3.2 ((CAMPOS *et al.*, 2012)). *Para quaisquer dois grafos G e H temos que, se* $\Gamma(H) = 2$ *então* $\Gamma(G\square H) \leq 2\cdot(\Delta(G)+1)$.

Por fim, deixaram as seguintes questões em aberto: O estudo de limitantes para o $\Gamma(G)$ em produto direto de dois grafos G e H onde $\Gamma(H) \leq 4$. Além disso, comentaram que as questões mais interessantes neste domínio são sobre o produto cartesiano, já que produto lexicográfico de quaisquer dois grafos G e H pode ser visto como um caso particular do número de Grundy do produto cartesiano de dois grafos.

3.4 Coloração k-imprópria gulosa

No trabalho de Rodrigues (2020) foi estudada a heurística de coloração gulosa para o *Problema de Coloração $k-$imprópria* de grafos, que consiste em determinar se um grafo possui uma $(m,k)-$coloração gulosa com o menor número de cores possíveis. Além disso, foi introduzido o conceito de número de Grundy $k-$impróprio, que é o maior inteiro l, tal que, um grafo G admite uma coloração $k-$imprópria gulosa com l cores e é denotado por $\Gamma^k(G)$.

Em seu trabalho, Rodrigues (2020) determinou o valor exato do número de Grundy $1-$impróprio para a classe de grafos árvores binomiais. Além disso, foi apresentada uma formulação de Programação Linear Inteira para o Problema do Número de Grundy, preenchendo um vácuo nesse estudo. A formulação de Programação Linear Inteira pode ser observada na Figura 9.

Figura 9 – Formulação de Programação Linear Inteira do Problema do Número de Grundy com $C = \{1, \ldots, n\}$

$$\max \sum_{c \in C} z_c \tag{3.1}$$

$$s.a. \quad \sum_{c \in C} x_v^c = 1 \qquad \forall v \in V(G) \tag{3.2}$$

$$z_c \leq \sum_{v \in V(G)} x_v^c \qquad \forall c \in C \tag{3.3}$$

$$x_v^c + x_u^c \leq z_c \qquad \forall \{u,v\} \in E(G), \forall c \in C \tag{3.4}$$

$$x_v^c \geq 1 - \sum_{d=1}^{c-1} x_v^d - \sum_{u \in N(v)} x_u^c \qquad \forall v \in V(G), \forall c \in C \tag{3.5}$$

$$x_v^c \in \{0,1\} \qquad \forall v \in V(G), \forall c \in C \tag{3.6}$$

$$z_c \in \{0,1\} \qquad \forall c \in C \tag{3.7}$$

Fonte: Adaptado de Rodrigues (2020).

As variáveis x_v^c indicam se o vértice v é colorido com a cor c. Cada variável z_c indica se pelo menos um vértice é colorido com a cor c. A função objetivo (3.1) indica que o número de cores deve ser maximizado. As restrições (3.2) definem que cada vértice deve ser colorido com apenas uma cor. As restrições (3.3) expressam que $z_c = 0$ caso nenhum vértice seja colorido com a cor c. As restrições (3.4) proíbem que vértices adjacentes sejam coloridos com a mesma cor. As restrições (3.5) definem que se $x_v^c = 1$ então v é um vértice guloso. As duas últimas restrições (3.6) e (3.7) garantem o domínio das variáveis utilizadas no modelo.

No mesmo trabalho, foi sugerido como trabalhos futuros o estudo da formulação de programação inteira do problema do número de Grundy. Além disso, investigar resultados computacionais de instâncias de alguma aplicação do problema.

Por fim, deixaram as seguintes questões em aberto: estudo de outras heurísticas e complexidade parametrizada para o problema de coloração $k-$imprópria. Além disso, foi sugerido a determinação do parâmetro do número de Grundy $k-$impróprio, para $k > 1$, das árvores binomiais ou outra classe de grafos.

3.5 Trabalho Proposto

Este trabalho, assim como (BONNET *et al.*, 2018; ASTÉ *et al.*, 2010; CAMPOS *et al.*, 2012; RODRIGUES, 2020), estuda e apresenta novos resultados para o Problema do Número de Grundy. Inicialmente, foi implementada uma variação do algoritmo de programação

dinâmica proposto por (BONNET *et al.*, 2018). Além disso, um algoritmo Branch and Bound foi proposto para o problema, visando encontrar soluções mais eficientes do que o proposto por (BONNET *et al.*, 2018). Os limitantes superiores dos Trabalhos de (ASTÉ *et al.*, 2010) e (CAMPOS *et al.*, 2012) são utilizados nos algoritmos elaborados neste documento. Foram propostas também, novas formulações de PLI, visando comparar e obter a melhor formulação com o Trabalho de (RODRIGUES, 2020). Além disso, foram propostos testes computacionais com algoritmos desenvolvidos, buscando determinar o parâmetro $\Gamma(G)$ em grafos aleatórios e da classe Stacked Book com o objetivo de comparar seus resultados.

A Tabela 1 apresenta de forma resumida a comparação deste trabalho com os trabalhos apresentados neste Capítulo.

Tabela 1 – Análise comparativa entre os trabalhos relacionados e o proposto.

	Estudo do número de Grundy e variações	**Elaboração de novos algoritmos para o problema do número de Grundy**	**Determinação do número de Grundy para alguma classe de grafo por meio de algoritmos**	**Implementação e testes computacionais com modelos de Programação Linear Inteira para o Problema do Número de Grundy**
(BONNET *et al.*, 2018)	X	X	X	
(ASTÉ *et al.*, 2010)	X			
(CAMPOS *et al.*, 2012)	X			
(RODRIGUES, 2020)	X			
Este trabalho	X	X	X	X

Fonte: Elaborado pelo autor (2022).

4 LIMITE INFERIOR

Neste capítulo é descrita uma heurística gulosa proposta neste trabalho. O objetivo desta heurística é fornecer um limitante inferior viável para os algoritmos desenvolvidos.

4.0.1 Degeneração de vértices

Esta heurística proposta consiste em uma coloração gulosa em uma ordem de degeneração de vértices D de um grafo G. No trabalho de Matula e Beck (1983), é descrito que é possível encontrar uma ordenação de vértices por degeneração em tempo linear, selecionando repetidamente o vértice de grau mínimo entre os vértices restantes utilizando a estrutura de dados bucket queue inventado por Dial (1969).

Os Algoritmos 2 e 3 apresentam o pseudocódigo do algoritmo que encontra uma ordem de degeneração de um grafo G. Especificamente, no Algoritmo 2, inicializamos algumas variáveis locais (Linha 02). Em seguida, calculamos o grau para cada vértice v de G armazenando no vetor *deg* e também o grau máximo de G é salvo na variável $delta$. Depois disso, classificamos os vértices em ordem crescente de seus graus usando o algoritmo de ordenação *bucket sort*, que funciona dividindo os vértices em um número finito de compartimentos numerados de 0 a $maxGrau$.

Primeiramente, contamos (Linhas $08-09$) quantos vértices haverá em cada compartimento (o compartimento consiste em vértices com o mesmo grau). A partir dos tamanhos dos compartimentos, podemos determinar (Linhas $11-16$) as posições iniciais dos compartimentos no vetor $vert$. O compartimento 0 começa na posição 1, enquanto os outros compartimentos começam na posição, igual à soma da posição inicial e tamanho do compartimento anterior. Para evitar o uso de um vetor adicional, usamos o mesmo vetor *bin* para armazenar as posições iniciais dos compartimentos.

Agora podemos colocar (Linhas $17-21$) os vértices de G no vetor $vert$. Para cada vértice, sabemos a qual compartimento ele pertence e qual é a sua posição inicial. Dessa forma, podemos colocar o vértice no lugar adequado e aumentar a posição inicial do compartimento que usamos, visto que, sabemos sua posição no vetor *pos*.

Agora temos os vértices classificados por seus graus, falta apenas recuperar as posições iniciais dos compartimentos (Linhas $22-24$). Vale acrescentar que nós incrementamos várias vezes na etapa anterior, quando colocamos os vértices nos compartimentos correspondentes.

É óbvio que a posição inicial alterada é a posição inicial original do próximo compartimento. Para restaurar as posições iniciais corretas, temos que mover os valores no compartimento da vetor para uma posição à direita. Também temos que redefinir a posição inicial do compartimento 0 para o valor 1 (Linha 25).

No Algoritmo 3, inicializamos uma lista vazia D (Linha 02) usada para armazenar a ordem de degeneração de G. Além disso, o resultado do Algoritmo 2 é armazenado nas variáveis *bin*, *vert*, *pos* e *graus* (Linha 03). O loop principal (Linhas $04-23$) é percorrido todos os vértices de G na ordem, determinada pelo vetor *vert*. O vértice atual v é um vértice de menor grau entre os vértices restantes. Para cada vizinho u de v com grau mais alto, temos que diminuir seu grau e movê-lo para um compartimento à esquerda. Mover o vértice u de um compartimento para a esquerda é feita em tempo constante $O(1)$. Primeiro temos que trocar o vértice u e o primeiro vértice no mesmo compartimento. No vetor *pos*, também temos que trocar suas posições. Finalmente, aumentamos a posição inicial do compartimento (aumentamos o compartimento anterior e reduzimos o compartimento atual para um elemento) e adicionamos v na lista D.

Durante a execução do algoritmo, o número total de movimentos é, no máximo, a soma dos graus dos vértices de G, logo pelo Teorema 2.1.1 o algoritmo tem complexidade $O(m)$.

Algoritmo 2: BUCKET(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1  bin, vert, pos, graus início
2      bin ← pos ← vert ← {}
3      deg ← calcular o grau de cada vértice v ∈ V(G)
4      delta ← grau máximo de G
5      para cada d de 0 até delta hacer
6          bin[d] ← 0
7      fin
8      para cada v em V hacer
9          bin[deg[v]] ← bin[deg[v]] + 1
10     fin
11     start ← 1
12     para cada d de 0 até delta hacer
13         num ← bin[d]
14         bin[d] ← start
15         start ← start + num
16     fin
17     para cada v em V hacer
18         pos[v] ← bin[deg[v]]
19         vert[pos[v]]← v
20         bin[deg[v]] ← bin[deg[v]] + 1
21     fin
22     para cada d de delta até 0 hacer
23         bin[d] ← bin[d-1]
24     fin
25     bin[0] ← 1
26     bin, vert, pos, deg
27 fim
```

Fonte: Elaborado pelo autor(2022).

Algoritmo 3: DEGENERACAO(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1  D início
2      D ← {}
3      bin, vert, pos, deg ← BUCKET(G)
4      para cada i de 1 até n hacer
5          v ← vert[i]
6          para cada u em N(v) hacer
7              deg[u] > deg[v] du ← deg[u]
8              pu ← pos[u]
9              pw ← bin[du]
10             w ← vert[pw]
11             u = w pos[u] ← pw
12             vert[pu] ← w
13             pos[w] ← pu
14             vert[pw] ← u  bin[du] ← bin[du] + 1
15             deg[u] ← deg[u] + 1
16         fin
17         adiciona v em D
18     fin
19     D
20 fim
```

Fonte: Adaptado de (MATULA; BECK, 1983).

O Algoritmo 4, denominado LB-DEG, é baseado na ordem de degeneração D e na observação de que vértices com menor grau são mais flexíveis na escolha de cores se comparado a vértices com maior grau. Esse algoritmo considera um grafo G e uma sequência $D = (d_1, \ldots, d_n)$ dada como entrada para o Algoritmo Guloso de Coloração e atribui cores aos vértices de G seguindo a ordem D de modo que cada vértice recebe a menor cor que esteja disponível e no final retorna o número de cores utilizadas nesta coloração gulosa.

Algoritmo 4: LB-DEG(G)

Entrada: Grafo $G = (V(G), E(G))$

número de cores utilizadas em C **início**

    $D \leftarrow$ DEGENERACAO(G)

    $C \leftarrow COLORACAOGULOSA(G, D)$

    número de cores utilizadas em C

fim

Fonte: Elaborado pelo autor (2022).

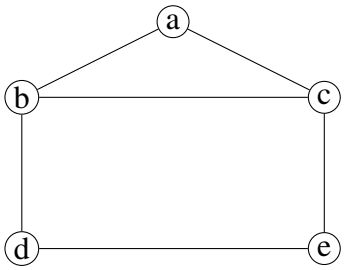
5 LIMITE SUPERIOR

Neste capítulo, é descrito um limite superior usado neste trabalho para os algoritmos desenvolvidos.

5.1 Stair factor

Shi *et al.* (2005) definiram um limite superior para o número de Grundy em termos de uma "sequência de Grundy viável". Uma sequência S de r vértices distintos $(g_1, \ldots, g_r)$ de um grafo G é uma *sequência de Grundy viável* se para $1 \leq i \leq r$ o grau de g_i em $G - \{g_{i+1}, \ldots, g_r\}$ é pelo menos $i-1$. Por exemplo, no grafo da Figura 10, a sequência (e, d, c, b) é uma sequência de Grundy viável.

Figura 10 – Um exemplo de sequência de Grundy viável.



Fonte: Elaborado pelo autor (2022).

O *stair factor* de G é a cardinalidade máxima de uma sequência de Grundy viável e é denotado por $\zeta(G)$.

Shi *et al.* (2005) mostraram no Teorema 5.1.1 que, o stair factor é um limitante superior do *número de Grundy parcial*, que é o maior inteiro k, tal que, existe uma coloração de Grundy parcial com k cores de G e é denotado por $\partial\Gamma(G)$. Uma coloração é chamada de *Coloração de Grundy parcial* se para cada classe de cor tem pelo menos um vértice de Grundy.

Teorema 5.1.1 ((SHI *et al.*, 2005))**.** *Seja G um grafo, então:*

$$\Gamma(G) \leq \partial\Gamma(G) \leq \zeta(G) \leq \Delta(G) + 1 \tag{5.1}$$

O Teorema 5.1.1 se baseia no resultado apresentado em (ERDÖS *et al.*, 2003), que mostrou que o número de Grundy parcial é um limitante superior para o número de Grundy.

O *stair factor* pode ser calculado por meio de uma sequência de decomposição de vértices de G pelo Teorema 5.1.2.

Teorema 5.1.2 ((SHI *et al.*, 2005)). *Seja D uma sequência de decomposição de vértices de um grafo G com graus residuais* $(d_1, ..., d_n)$. *Então o stair factor pode ser calculado da seguinte forma:*

$$\zeta(G) = \min_i(d_i + i)$$

No Algoritmo 5, é apresentado o pseudocódigo para determinar uma sequência de decomposição de vértices D de um grafo G. Neste Algoritmo, procedemos da seguinte forma: Usamos o Algoritmo 2 para distribuir os n vértices em um vetor de $\Delta(G)+1$ compartimentos correspondendo ao grau residuais 0 a $\Delta(G)$. O loop principal (Linhas $04-23$) é similar ao do Algoritmo 3, entretanto a ordem percorrida é a ordem inversa de todos os vértices de G, determinada pelo vetor *vert*. O vértice atual v nesta ordem inversa é o vértice de maior grau entre os vértices restantes.

Durante a execução do Algoritmo, o número total de movimentos é, no máximo, a soma dos graus dos vértices de G, logo pelo Teorema 2.1.1 o Algoritmo tem complexidade $O(m)$.

Algoritmo 5: DECOMPOSICAO(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1  D início
2      D ← {}
3      bin, vert, pos, deg ← BUCKET(G)
4      para cada i de n até 1 hacer
5          v ← vert[i]
6          para cada u em N(v) hacer
7              deg[u] > deg[v] du ← deg[u]
8              pu ← pos[u]
9              pw ← bin[du]
10             w ← vert[pw]
11             u = w pos[u] ← pw
12             vert[pu] ← w
13             pos[w] ← pu
14             vert[pw] ← u  bin[du] ← bin[du] + 1
15             deg[u] ← deg[u] + 1
16         fin
17         adiciona v em D
18     fin
19     D
20 fim
```

Fonte: Elaborado pelo autor(2022).

Em (SHI *et al.*, 2005) é descrito que o parâmetro stair factor pode ser calculado em tempo linear. No Algoritmo 6, denominado STAIRFACTOR, é apresentado o pseudocódigo do calculo deste parâmetro em tempo linear $O(m)$.

Algoritmo 6: STAIRFACTOR(G)

Entrada: Grafo $G = (V(G), E(G))$

1 $\zeta(G)$ **início**

2 $D \leftarrow$ DECOMPOSICAO(G)

3 $\zeta \leftarrow \min_i(D_i + i)$

4 ζ

5 **fim**

Fonte: Elaborado pelo autor(2022).

6 ALGORITMOS EXATOS

Neste capítulo, são descritos os algoritmos exatos desenvolvidos neste trabalho para o Problema do Número de Grundy.

6.1 Programação Dinâmica

Neste trabalho, foi implementada uma variação do algoritmo proposto por (BONNET *et al.*, 2018). O algoritmo é obtido por programação dinâmica nos subconjuntos de vértices de um grafo G e usa um algoritmo que enumera todos os conjuntos independentes máximos. Primeiramente, é preenchida recursivamente uma tabela $\Gamma(G[S])$ com todos os subconjuntos de $S \subseteq V(G)$ de maneira ascendente começando de $S = \emptyset$. O caso base da recursão é $\Gamma(G[S]) = 0$.

No algoritmo proposto por Bonnet *et al.* (2018), para todo $S \subseteq V(G)$, o número de Grundy de $G[S]$ é dado pela recorrência 6.1 expressa no Teorema 6.1.1.

Teorema 6.1.1 ((BONNET *et al.*, 2018)). *Para todo $S \subseteq V(G)$, o número de Grundy de $G[S]$ é dado pela recorrência:*

$$\Gamma(G[S]) = \begin{cases} 0 & \text{, se } S = \emptyset \\ 1 + \max(\Gamma(G[S \setminus X])) \text{ para todo conjunto independente } X \subseteq G[S] & \text{, se } S \neq \emptyset. \end{cases} \tag{6.1}$$

Seja S um subconjunto de vértices de $V(G)$ de modo que $V(G) = \{v_1, \ldots, v_n\}$. O subconjunto S está representado como um vetor binário de n bits. Cada um dos n bits desta representação indica se o vértice v_i para $i \in \{0, \ldots, n-1\}$ está contido no subconjunto ou não.

Por exemplo, seja o conjunto de vértices $\{v_1, v_2, v_3\}$. Assim, a representação binária de $S \subseteq V(G)$ é um vetor de tamanho 3, e a associação de cada S ao seu respectivo valor binário é dada por:

$$\begin{array}{ll} 000 \rightarrow \emptyset & 100 \rightarrow \{v_3\} \\ 001 \rightarrow \{v_1\} & 101 \rightarrow \{v_1, v_3\} \\ 010 \rightarrow \{v_2\} & 110 \rightarrow \{v_2, v_3\} \\ 011 \rightarrow \{v_1, v_2\} & 111 \rightarrow \{v_1, v_2, v_3\} \end{array}$$

Encontrar o conjunto de conjuntos independentes maximais de vértices de um grafo, pode também ser interpretado como encontrar o conjunto de cliques maximais no complemento

de G, pois um conjunto independente de vértices em G é um clique no grafo complemento e vice-versa.

Toda as cliques maximais podem ser enumeradas com o algoritmo de enumeração de Bron-Kerbosch, projetado pelos cientistas holandeses Coenraad Bron e Joep Kerbosch, que publicaram sua descrição em 1973. Esse algoritmo encontra todas as cliques maximais de um grafo e ao mesmo tempo utiliza a técnica $B\&B$ para cortar os ramos que não levam a uma solução factível (BRON; KERBOSCH, 1973).

A base desse algoritmo é a busca exaustiva, que procura todas as cliques máximas em um grafo baseado em três estruturas que são definidas a seguir:

- C : conjunto de vértices já definidos como parte da clique;
- P : vértices adjacentes a todos os vértices de C; esse é o conjunto de candidatos a entrar na clique;
- S : vértices que já foram analisados e não levam a uma extensão do conjunto de candidatos P , cujo objetivo é evitar comparação excessiva.

A execução do algoritmo é feita inicialmente com C e S vazios e P contendo todos os vértices do grafo. Em cada chamada do algoritmo, se o conjunto P está vazio, o algoritmo encontra uma clique maximal (se S também estiver vazio), caso contrário realiza busca exaustiva. P e S são conjuntos disjuntos cuja união consiste naqueles vértices que formam cliques quando adicionados a C. Em outros palavras, $P \cup S$ é o conjunto de vértices que são unidos a cada elemento de C. Quando P e S estão vazios, não há mais elementos que possam ser adicionados a C, então C é um clique máximo. Se C é um clique máximo então ele é adicionado em um vetor das cliques maximais $CLIQUES$. A versão básica do algoritmo de Bron-Kerbosch é apresentada no Algoritmo 7.

Algoritmo 7: BK(G, C, P, S, CLIQUES)

Entrada: Grafo $G = (V(G), E(G))$, Conjunto de vértices C, Conjunto de vértices P, Conjunto de vértices S, vetor CLIQUES

```
1 Todos os cliques maximais em G início
2     P=∅ e S=∅ Adiciona a Clique C em CLIQUES
3     para cada v ∈ P hacer
4         BK(G, C ∪ {v}, P ∩ N(v), S ∩ N(v), CLIQUES)
5         P ← P \ { v }
6         S ← S ∪ { v }
7     fin
8 fim
```

Fonte: Elaborado pelo autor (2022).

A forma básica do algoritmo é ineficiente no caso de grafos com muitas cliques não maximais, pois é realizada uma chamada recursiva em cada clique maximal ou não. Para que o algoritmo termine mais rápido nos ramos da busca que não contêm cliques maximais, Bron e Kerbosch introduziram uma variante do algoritmo que envolve um "vértice pivô"u, escolhidos de P (ou de $P \cup S$) (BRON; KERBOSCH, 1973).

Qualquer clique máxima deve incluir u ou um de seus não vizinhos, caso contrário, o clique poderia ser aumentado adicionando-se u a ele. Portanto, apenas u e seus não vizinhos precisam ser testados como escolhas para o vértice v que é adicionado a C em cada chamada recursiva do algoritmo. A versão com pivoteamento do algoritmo de Bron-Kerbosch é apresentada no Algoritmo 8.

Algoritmo 8: BKPIVO(G, C, P, S, CLIQUES)

Entrada: Grafo $G = (V(G), E(G))$, Conjunto de vértices C, Conjunto de vértices P, Conjunto de vértices S, vetor "CLIQUES"

```
1 Todos os cliques maximais em G início
2     P=∅ e S=∅ Adiciona a Clique C em CLIQUES  Escolha um vértice pivô u de P ∪ S
3     para cada v ∈ P \ N(u) hacer
4         BKPIVO(G, C ∪ {v}, P ∩ N(v), S ∩ N(v), CLIQUES)
5         P ← P \ { v }
6         S ← S ∪ { v }
7     fin
8 fim
```

Fonte: Elaborado pelo autor (2022).

Por um resultado de (MOON; MOSER, 1965), qualquer grafo com n vértices tem no máximo $3^{\frac{n}{3}}$ cliques máximos e o pior caso de tempo de execução do algoritmo de Bron-Kerbosch (com uma estratégia de pivô que minimiza o número de chamadas recursivas feitas em cada etapa) é $O(3^{\frac{n}{3}})$, correspondendo a esse limite (TOMITA *et al.*, 2006).

Um método alternativo para melhorar a forma básica do algoritmo de Bron-Kerbosch envolve desconsiderar o pivoteamento no nível mais externo de recursão e, em vez disso, escolher a ordenação das chamadas recursivas com cuidado para minimizar o tamanho do conjunto P de vértices candidatos, dentro de cada chamada recursiva. O método alternativo é utilizar a degeneração de um grafo. A versão com a degeneração de um grafo do algoritmo de Bron-Kerbosch é apresentada no Algoritmo 9.

Algoritmo 9: BKDEGENERACAO(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1  Todos os cliques maximais em G início
2      P ← V
3      R ← X ← ∅
4      D ← DEGENERACAO(G)
5      CLIQUES ← { }
6      para cada v ∈ D hacer
7          BKPIVO( G, {v}, P ∩ N(v), S ∩ N(v), CLIQUES )
8          P ← P \ { v }
9          S ← S ∪ { v }
10     fin
11     CLIQUES
12 fim
```

Fonte: Elaborado pelo autor (2022).

Em particular, a versão com a degeneração de vértices do algoritmo de Bron-Kerbosch pode ser executada no tempo $O(dn3^{\frac{d}{3}})$, onde d é a degeneração do grafo (EPPSTEIN *et al.*, 2010).

Para o algoritmo de programação dinâmica proposto para o Problema do Número de Grundy, a versão com a degeneração do algoritmo de Bron-Kerbosch é utilizada para encontrar os cliques maximais do grafo complemento. No Algoritmo 10, é apresentado o pseudocódigo deste algoritmo.

Algoritmo 10: PDG(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1  Número de Grundy de G início
2      X ← vetor indexado de 0 até 2^n − 1
3      X[0] = 0
4      para cada S de 1 até 2^n − 1 hacer
5          X[S] ← 0
6          Z = G[S]‾
7          para cada I ∈ BKDEGENERACAO( Z ) hacer
8              X[S] ← max(X[S], X[S \ I] + 1)
9          fin
10     fin
11     X[2^n − 1]
12 fim
```

Fonte: Elaborado pelo autor (2022).

O Algoritmo 10, denominado PDG, percorre todos os 2^n subgrafos induzidos de G, de maneira que os subconjuntos de S sejam processados antes de S. Para cada S, computam-se o $\Gamma[S]$ e os seus conjuntos independentes maximais no grafo complemento de G com o Algoritmo 9.

Teorema 6.1.2. *O Algoritmo PDG para o Problema do Número de Grundy executa em tempo* $O(dn1.4423^d 2^n)$ *e espaço* $\theta(2^n)$*, para* $n = |V(G)|$*,* $m = |E(G)|$ *e a degeneração d de G.*

Demonstração. Seja G um grafo com n vértices e m arestas. Para cada $S \subseteq V(G)$, seja $T(S)$ o tempo gasto na execução do loop interno no Algoritmo 10 usando o Algoritmo 9. Além disso, o Algoritmo usado para a geração dos conjuntos independentes maximais executa em tempo $O(dn3^{\frac{d}{3}})$, onde d é a degeneração de G.

Sabemos que, existem $c > 0$ e $n_0 \in \mathbb{N}$ tal que, sempre que $|S| \geq n_0$:

$$T(S) \leq cdn3^{\frac{d}{3}}$$

Somando-se todos $S \subseteq V(G)$:

$$\begin{aligned}
\sum_{S\subseteq V(G)} T(S) &\leq \sum_{S\subseteq V(G)} cdn3^{\frac{d}{3}} \\
&= cdn3^{\frac{d}{3}} \sum_{S\subseteq V(G)} 1 \\
&= cdn3^{\frac{d}{3}} \sum_{i=0}^{n} \sum_{S\in\binom{V(G)}{i}} 1 \\
&= cdn3^{\frac{d}{3}} \sum_{i=0}^{n} \binom{n}{i} 1 \\
&= cdn3^{\frac{1}{3}^d} (1+1)^n \\
&= cdn1.4423^d 2^n
\end{aligned}$$

Portanto, podemos concluir que o tempo de execução do Algoritmo 10 com G como entrada é $O(dn1.4423^d 2^n)$. □

6.2 Branch-and-Bound

Nesta seção, apresentamos um algoritmo de Branch and Bound para o Problema do Número de Grundy. O algoritmo desenvolvido é baseado em DSATUR (BRÉLAZ, 1979), que é um algoritmo heurístico guloso onde cada vértice $v \in V(G)$ é colorido iterativamente com uma cor que ainda não foi usada na vizinhança de v. Dada uma coloração parcial C e um vértice $v \in V(G)$, o grau de saturação $DSAT(v,C)$ corresponde ao número de cores diferentes em $N(v)$. A cada iteração de DSATUR, o vértice com o valor $DSAT$ mais alto é colorido até que uma coloração viável de todo o grafo G seja obtida, assim o número de cores usadas é então um limite superior válido para $\chi(G)$.

Um algoritmo Branch and Bound exato pode ser derivado do DSATUR. Dada uma coloração parcial e um vértice sem cor $v \in V$, em vez de fixar sua cor de forma gulosa, uma árvore ramificada é criada colorindo v com todas as cores possíveis já usadas na coloração parcial mais uma nova. Em cada nó dessa árvore ramificada, recebemos uma coloração parcial C com cores $\widetilde{k}$, um limite superior (UB) e um limite inferior (LB) para $\Gamma(G)$. Trivialmente, $\widetilde{k}$ pode ser usado como um limite inferior para $\Gamma(G)$ de C, ou seja, o número de Grundy de G parcialmente colorido por C. Outro LB pode ser encontrado no nó raiz do algoritmo Branch and Bound determinando um clique em G. Este LB normalmente nunca é atualizado na implementação

clássica de DSATUR.

No Algoritmo 11 e no Algoritmo 12, é apresentado o pseudocódigo do algoritmo de DSATUR. Precisamente, o Algoritmo 11 recebe como entrada um grafo G para colorir, em seguida calcula-se o limite inferior para inicializar LB e produz na saída uma coloração ótima de G. No Algoritmo 12, o mecanismo para criar os nós filhos funciona de seguinte forma, após selecionado um vértice v_i descolorido, até $\widetilde{k}+1$ nós filhos são criados colorindo o vértice selecionado com todas as cores viáveis se $\max(\widetilde{k}, LB) < UB$. No caso de todos os vértices serem coloridos, se $\widetilde{k} < UB$ então $UB \leftarrow \widetilde{k}$ e $C^* \leftarrow C$.

Algoritmo 11: BBCI(G)

Entrada: Grafo $G = (V(G), E(G))$

```
1 Coloração ótima de G início
2     LB ← MaxClique(G)
3     UB ← n
4     C ← Coloração Vazia
5     BBC(G, V, LB, UB, C)
6     C*
7 fim
```

Fonte: Adaptado de (BRÉLAZ, 1979).

Algoritmo 12: BBC(G,C)

Entrada: Grafo $G = (V(G), E(G)$,Vértices ainda não coloridos V, Limite Inferior LB, Limite Superior UB, Coloração C,cores usadas k

início
  todos os vértices estão coloridos em C $\widetilde{k} < UB$ $UB \leftarrow \widetilde{k}$
  $C^* \leftarrow C$ Seleciona um vértice $v_i \in V(G)$ que ainda não foi colorido
  $cores \leftarrow CoresDisponives(C, v_i, \widetilde{k})$
  para cada *cor em cores* **hacer**
    $\widetilde{C} \leftarrow C$
    $\widetilde{C}[v_i] \leftarrow cor$
    $\max(\widetilde{k}, LB) < UB$ $\widetilde{V} \leftarrow V$
    remove v_i de $\widetilde{V}$
    BBC(G,$\widetilde{V}$,LB,UB,$\widetilde{C}$)
  fin
fim

Fonte: Adaptado de (BRÉLAZ, 1979).

A regra de seleção de vértices básica, proposta por Brélaz (1979) para o Algoritmo 12, consiste em colorir o vértice com o valor máximo de DSAT, minimizando assim o número de nós filhos. Entretanto, durante a execução de DSATUR, muitas vezes acontece que muitos vértices diferentes compartilham o mesmo valor DSAT máximo, ou seja, criando possíveis laços.

No Algoritmo 13 e no Algoritmo 14, é apresentado o pseudocódigo do algoritmo de Branch and Bound derivado de DSATUR para o Problema do Número de Grundy. Especificamente, o Algoritmo 13 denominado BBGI, recebe como entrada um grafo G para colorir, em seguida é atribuído o limite inferior LI para inicializar LB e o limite superior LS para inicializar UB. Em seguida, as variáveis inicializadas são passadas como parâmetro no Algoritmo 14. Por fim, é retornado na saída o $\Gamma(G)$. A regra de seleção de vértices é baseada na sequencia de degeneração de G.

No Algoritmo 14 denominado BBG, o procedimento para criar os nós filhos executa da seguinte maneira: Primeiramente seleciona-se o vértice v_1 da sequência D, em seguida, até $\min(|N(v_1)|, UB)$ nós filhos são criados colorindo o vértice selecionado com todas as cores viáveis apenas se, o número de cores utilizadas em $\widetilde{C} \geq \min(K, LB)$, ou seja, o nó é expandido se o número de cores utilizadas em $\widetilde{C}$ é maior ou igual que o mínimo entre o número de cores

utilizadas K do nó e o limite inferior LB passado como parâmetro. Para cada nó criado o vértice v_1 é removido da sequência $\widetilde{D}$. Na condição de que todos os vértices estão coloridos, se C é uma coloração de Grundy então a quantidade de cores utilizadas na coloração do nó é atualizada para $K \leftarrow \max(K$, número de cores utilizadas em $C)$.

Algoritmo 13: BBGI(G, LI, LS)

Entrada: Grafo $G = (V(G), E(G))$, Limite Inferior LI, Limite Superior LS

```
1 Γ(G) início
2     LB ← LI
3     UB ← LS
4     C ← Coloração Vazia
5     K ← 0
6     D ← DEGENERACAO(G)
7     BBG(G, D, LB, UB, C, K)
8     K
9 fim
```

Fonte: Elaborado pelo autor (2022).

Algoritmo 14: BBG(G, O, LB, UB, C, K)

Entrada: Grafo $G = (V(G), E(G))$, Sequência de degeneração D, Limite Inferior LB, Limite Superior UB, Coloração C, cores usadas K

1 **início**
2 todos os vértices estão coloridos em C C é uma coloracao de Grundy $K \leftarrow \max(K,$ número de cores utilizadas em $C)$
3 Seleciona o vértice v_1 da sequência D
4 $cores \leftarrow CoresDisponives(C, v_1, \min(N(v_1), UB))$
5 **para cada** *cor em cores* **hacer**
6 $\widetilde{C} \leftarrow C$
7 $\widetilde{C}[v_i] \leftarrow cor$
8 número de cores utilizadas em $\widetilde{C} \geq \min(K, LB)$ $K \leftarrow$ número de cores utilizadas em $\widetilde{C}$
9 $\widetilde{D} \leftarrow D$
10 remove v_1 de $\widetilde{D}$
11 BBG(G,$\widetilde{D}$,LB,UB,$\widetilde{C}$,K)
12 **fin**
13 **fim**

Fonte: Elaborado pelo autor (2022).

6.3 Programação Linear Inteira

Neste capítulo, apresentamos formulações de Programação Linear Inteira para o Problema do Número de Grundy. A formulações propostas aqui, são variações da formulação apresentada na Figura 9 do Trabalho de (RODRIGUES, 2020), descrita na Seção 3.4 deste trabalho.

Assim como na formulação do Trabalho de (RODRIGUES, 2020), todas as variáveis do modelo são binárias. As variáveis x_v^c indicam se o vértice v é colorido com a cor c. Cada variável z_c indica se pelo menos um vértice é colorido com a cor c. Dada a definição das variáveis do modelo, a formulação, denominada *F_DELTA*, é apresentada na Figura 11.

Figura 11 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1, \ldots, \Delta(G)+1\}$

$$\max \quad \sum_{c \in C} z_c \tag{6.2}$$

$$s.a. \quad \sum_{c \in C} x_v^c = 1 \qquad \forall v \in V(G) \tag{6.3}$$

$$z_c \leq \sum_{v \in V} x_v^c \qquad \forall c \in C \tag{6.4}$$

$$x_v^c + x_u^c \leq z_c \qquad \forall \{u,v\} \in E(G), \forall c \in C \tag{6.5}$$

$$x_v^c \geq 1 - \sum_{d=1}^{c-1} x_v^d - \sum_{u \in N(v)} x_u^c \qquad \forall v \in V(G), \forall c \in C \tag{6.6}$$

$$x_v^c \in \{0,1\} \qquad \forall v \in V(G), \forall c \in C \tag{6.7}$$

$$z_c \in \{0,1\} \qquad \forall c \in C \tag{6.8}$$

Fonte: Elaborado pelo autor (2022).

Na formulação da Figura 9, o conjunto de cores que podem ser usadas para colorir um grafo G é $C = \{1, \ldots, n\}$. Entretanto, grande parte dessas n cores no conjunto C podem não vir a serem utilizadas. Logo, este parâmetro é um limitante superior ruim. O Teorema 2.3.2 determina um limitante superior bem melhor para o número de Grundy em comparação ao número de vértices de um grafo.

Assim como na formulação da Figura 9, buscamos maximizar a função objetivo (6.2). O conjunto de cores que podem ser usadas para colorir um grafo G neste modelo é $C = \{1, \ldots, \Delta(G)+1\}$. As restrições (6.3) definem que cada vértice deve ser colorido com apenas uma cor. As restrições (6.4) expressam que $z_c = 0$ caso nenhum vértice seja colorido

com a cor c. As restrições (6.5) proíbe que vértices adjacentes sejam coloridos com a mesma cor. As restrições (6.6) definem que se $x_v^c = 1$ então v é um vértice guloso. As duas últimas restrições (6.7) e (6.8) definem que as variáveis do modelo são binárias.

Similar a formulação *F_DELTA*, a formulação da Figura 12, denominada *F_STF*, e a formulação da Figura 13, denominada *F_PROD*, possuem algumas semelhanças entre si, como a função objetivo e as restrições.

Figura 12 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1, \ldots, \zeta(G)\}$

$$\max \quad \sum_{c \in C} z_c \tag{6.9}$$

$$s.a. \quad \sum_{c \in C} x_v^c = 1 \qquad \forall v \in V(G) \tag{6.10}$$

$$z_c \leq \sum_{v \in V} x_v^c \qquad \forall c \in C \tag{6.11}$$

$$x_v^c + x_u^c \leq z_c \qquad \forall \{u,v\} \in E(G), \forall c \in C \tag{6.12}$$

$$x_v^c \geq 1 - \sum_{d=1}^{c-1} x_v^d - \sum_{u \in N(v)} x_u^c \qquad \forall v \in V(G), \forall c \in C \tag{6.13}$$

$$x_v^c \in \{0,1\} \qquad \forall v \in V(G), \forall c \in C \tag{6.14}$$

$$z_c \in \{0,1\} \qquad \forall c \in C \tag{6.15}$$

Fonte: Elaborado pelo autor (2022).

O conjunto de cores que podem ser usadas para colorir um grafo G nas formulações *F_STF* e *F_PROD* são, respectivamente, $C = \{1, \ldots, \zeta(G)\}$ e $C = \{1, \ldots, \Delta(G) \cdot 2^{\Gamma(H)-1} + \Gamma(H)\}$. Estes conjuntos de cores são baseados, respectivamente, no parâmetro stair factor e no limitante superior do Teorema 3.2.2. Vale destacar que, a formulação *F_PROD* é válida apenas para as classes de produto cartesiano de grafos. Estas formulações foram elaboradas com o objetivo de reduzir o número de restrições da formulação do Trabalho de (RODRIGUES, 2020).

Figura 13 – Formulação de Programação Linear Inteira para o Problema do Número de Grundy com $C = \{1, \ldots, \Delta(G) \cdot 2^{\Gamma(H)-1} + \Gamma(H)\}$

$$\max \quad \sum_{c \in C} z_c \tag{6.16}$$

$$s.a. \quad \sum_{c \in C} x_v^c = 1 \qquad \forall v \in V(G) \tag{6.17}$$

$$z_c \leq \sum_{v \in V} x_v^c \qquad \forall c \in C \tag{6.18}$$

$$x_v^c + x_u^c \leq z_c \qquad \forall \{u,v\} \in E(G), \forall c \in C \tag{6.19}$$

$$x_v^c \geq 1 - \sum_{d=1}^{c-1} x_v^d - \sum_{u \in N(v)} x_u^c \qquad \forall v \in V(G), \forall c \in C \tag{6.20}$$

$$x_v^c \in \{0,1\} \qquad \forall v \in V(G), \forall c \in C \tag{6.21}$$

$$z_c \in \{0,1\} \qquad \forall c \in C \tag{6.22}$$

Fonte: Elaborado pelo autor (2022).

7 RESULTADOS COMPUTACIONAIS

Neste capítulo são apresentados e discutidos os resultados obtidos neste trabalho. Na Seção 7.1 e 7.2 são apresentados os resultados dos experimentos feitos com os algoritmos desenvolvidos no Capítulo 6.

Todos os experimentos reportados aqui foram implementados na linguagem de programação Python e executados em um computador com um processador AMD Ryzen 5 3500U, 10GB de memória RAM e sistema operacional Ubuntu 20.04.3 LTS 64 bits. Todos os algoritmos foram executado com limitação de tempo de execução de 900 segundos (15 minutos) para testar o tempo que cada algoritmo leva para encontrar o valor de $\Gamma(G)$. Se a instância possuir um tempo de execução maior que 900 segundos isso significa que a execução do algoritmo foi interrompida antes de conseguir encontrar o número de Grundy e o melhor resultado obtido nas sub-soluções é retornado.

Nos algoritmos desenvolvidos, utilizamos o pacote networkX, que é um pacote Python utilizado para manipulação de grafos e redes complexas. A documentação da biblioteca, além de exemplos mais aprofundados, estão disponíveis em: https://networkx.org/.

7.1 Instâncias de Teste em grafos aleatórios

Nesta seção, os experimentos realizados se baseiam em grafos gerados aleatoriamente pelo Algoritmo de geração de grafos aleatórios apresentado no trabalho de Gilbert (1959). Neste algoritmo, um grafo aleatório é obtido para cada combinação n (número de vértices) e p (probabilidade de existência de cada aresta) considerada. Neste algoritmo, cada aresta é incluída no grafo com probabilidade p, independentemente de todas as outras arestas. Logo, todos os grafos com n vértices e m arestas têm a mesma probabilidade de serem gerados, que é definida por:

$$p^m(1-p)^{\frac{n(n-1)}{2}-m} \tag{7.1}$$

No Algoritmo 15, é apresentado o pseudocódigo do trabalho de Gilbert (1959). Além disso, considere que rand() é uma função que retorna um valor aleatório uniforme dentro do intervalo $[0.0, 1.0]$ a cada chamada.

Algoritmo 15: RANDOMGRAPH(N, P)

Entrada: Número de vértices N, probabilidade P

```
1  Grafo Aleatório G início
2      V(G) ← {1,...,N}
3      E(G) ← ∅
4      para i = 1 → N − 1 faça
5          para j = i + 1 → N faça
6              rand() ≤ P E(G) ← E(G) ∪ {(i, j)}
7          fim
8      fim
9      G
10 fim
```

Fonte: Adaptado de (GILBERT, 1959).

Em nosso experimento, foram gerados grafos aleatórios, utilizando probabilidade de geração de arestas, de 10, 30, 50, 70 e 90 por cento, onde para cada probabilidade foram gerados 5 grafos com 10, 15, 20 e 25 vértices, totalizando 100 grafos.

Para o experimento, foram utilizados os algoritmos exatos propostos, PDG, BBG e as formulações F_DELTA e F_STF. Para o algoritmo BBG, foi utilizado no parâmetro LI o resultado do algoritmo 4 e no parâmetro LS foram utilizados dois limitantes superiores distintos: $\Delta(G)+1$ e $\zeta(G)$.

A Tabela 5 no Apêndice A apresenta o tempo de execução, em segundos, que cada algoritmo levou pra encontrar o valor de $\Gamma(G)$. Podemos verificar nesta tabela que, as formulações F_DELTA e F_STF em quase todos os testes, conseguiram gerar uma resposta em menos de 900 segundos. Entretanto, os algoritmos PDG e BBG em grande parte dos testes não conseguiu determinar a solução exata, pois o tempo de execução ultrapassou o tempo limite de 900 segundos.

Na Tabela 2, é apresentado o tempo de execução médio, em segundos, dos algoritmos utilizados no teste comparados pela probabilidade utilizada no algoritmo de geração.

Analisando as Tabelas 2 e 6, pode-se notar que as formulações F_DELTA e F_STF conseguiram soluções de mesma qualidade nas instâncias de densidade baixa e média. Na instâncias de densidade média, a formulação F_DELTA apresentou a melhor média de tempo de execução. Na instâncias de densidade alta, a formulação F_STF se mostrou superior aos demais

Tabela 2 – Média de tempo de execução dos algoritmos exatos em instâncias de grafos gerados aleatoriamente comparados pela probabilidade utilizada no algoritmo de geração.

Probabilidade	*PDG*	$BBG\|\Delta+1$	$BBG\|\zeta$	*F_DELTA*	*F_STF*
0.1	253.0470	236.0048	235.9144	0.5191	0.4952
0.3	454.7198	560.7978	541.2434	155.1419	189.5339
0.5	467.9902	675.5241	675.0097	431.5029	408.3918
0.7	453.6546	679.8622	675.8909	496.6668	496.3798
0.9	453.2332	683.8917	676.6191	616.8374	528.6375

Fonte: Elaborado pelo autor (2022).

em uma grande quantidade instâncias, principalmente em relação à quantidade de instâncias onde o algoritmo encontrou a melhor solução.

Comparando os resultados obtidos e apresentados nas Tabelas 2, 5 e 6, pode-se notar que, em termos gerais, a formulação F_STF se mostra mais eficiente que os demais algoritmos exatos em instâncias de grafos gerados aleatoriamente.

7.2 Instâncias de Teste em grafos da classe Stacked Book

Nesta seção, os experimentos realizados se baseiam em grafos gerados pelo Algoritmo de criação de grafos da classe Stacked Book apresentado neste trabalho. No Algoritmo 16 é apresentado o pseudocódigo deste algoritmo. Neste algoritmo, um grafo desta classe é obtido para cada combinação P (grafo estrela S_P com $P+1$ vértices) e Q (grafo caminho P_Q com Q vértices) considerada. Além disso, considere que $nx.star\_graph(P)$, $nx.path\_graph(Q)$ e $nx.cartesian\_product(G,H)$ são funções do pacote networkX que retornam, respectivamente, o grafo S_P, grafo P_Q e o produto cartesiano dos grafos G e H.

Algoritmo 16: GENERATESTACKEDBOOK(P, Q)

Entrada: Número de vértices P do grafo estrela, Número de vértices Q do grafo caminho

```
Grafo Stacked Book C início
    G = nx.star_graph(P)
    H = nx.path_graph(Q)
    C = nx.cartesian_product(G,H)
    C
fim
```

Fonte: Elaborado pelo autor (2022).

Para o experimento, foram gerados 56 grafos da classe Stacked Book com o algo-

ritmo 16, utilizados nos algoritmos exatos propostos: PDG, BBG e todas as formulações de Programação Linear Inteira. No Algoritmo BBG, foi utilizado no parâmetro LI o resultado do Algoritmo 4 e no parâmetro LS foram utilizados três limitantes superiores distintos: $\Delta(G)+1$, $\zeta(G)$ e 6. O limitante 6 é obtido a partir dos Teoremas 3.2.2 e 3.3.1 e determina que com 6 cores é possível colorir qualquer grafo da classe Stacked Book.

A Tabela 7 no Apêndice A, apresenta o tempo de execução, em segundos, que cada algoritmo levou pra encontrar o valor de $\Gamma(G)$. Como podemos verificar na Tabela 7 no Apêndice A, as formulações *F_DELTA* e *F_STF* em quase todos os testes, conseguiram gerar uma resposta em menos de 900 segundos. Já a formulação *F_PROD* conseguiu determinar o número de Grundy em todos os testes. Entretanto, os algoritmos PDG e BBG em grande parte dos testes não conseguiu determinar a solução exata, pois o tempo de execução ultrapassou o tempo limite de 900 segundos.

A Tabela 3, apresenta a quantidade de instâncias que cada algoritmo conseguiu determinar a solução ótima. O resultado da solução ótima das instâncias se baseiam no resultado da formulação *F_PROD* na Tabela 8 no Apêndice A, já que esta formulação conseguiu determinar o resultado exato em todas as instâncias.

Tabela 3 – Número de instâncias com solução ótima encontrada pelos algoritmos exatos em instâncias de grafos da classe Stacked Book.

PDG	*BBG*$\|\Delta+1$	*BBG*$\|\zeta$	*BBG*$\|6$	*F_DELTA*	*F_STF*	*F_PROD*
28	28	28	28	52	53	56

Fonte: Elaborado pelo autor (2022).

Podemos observar na Tabela 3, que a formulação *F_PROD* consegue encontrar a solução ótima em um maior número de instâncias $(100,00\%)$, comparado aos demais algoritmos exatos. Já a formulação *F_STF*, se mostra mais eficiente que a formulação *F_DELTA*. Já os algoritmos PDG E BGG encontraram a mesma quantidade de soluções ótimas.

Visando analisar melhor o comportamento dos algoritmos exatos, foi realizada uma comparação de tempo de execução nas instâncias, onde os algoritmos exatos conseguiram encontrar e garantir a solução ótima. Nesta nova comparação, apenas 28 instâncias foram consideradas. A Tabela 4, apresenta o tempo de execução médio, em segundos, dos algoritmos utilizados no teste.

Comparando os resultados obtidos e apresentados nas Tabelas 3, 4, 7 e 8, pode-se notar que, em termos gerais, a formulação *F_PROD* se mostra mais eficiente que os demais algoritmos exatos em instâncias de grafos da classe Stacked Book.

Tabela 4 – Média de tempo de execução dos algoritmos exatos em instâncias de grafos da classe Stacked Book.

PDG	*BBG*$\|\Delta+1$	*BBG*$\|\zeta$	*BBG*$\|6$	*F_DELTA*	*F_STF*	*F_PROD*
650.8053	652.4946	565.4679	622.9077	10.3721	6.6302	8.6833

Fonte: Elaborado pelo autor (2022).

Analisando a Tabela 8 no Apêndice A, as soluções propostas permitiram definir a Conjectura 7.2.1 que define o número de Grundy para a classe Stacked Book.

Conjectura 7.2.1. *Seja $SB_{p,q}$ um grafo da classe Stacked Book. Então, nós temos que:*

- *Seja $p \geq 3$ um inteiro. Então $\Gamma(SB_{p,2}) = 4$ e $\Gamma(SB_{p,3}) = 4$.*
- *Seja $p \geq 3$ um inteiro. Então $\Gamma(SB_{p,4}) = 5$ e $\Gamma(SB_{p,5}) = 5$.*
- *Seja $p \geq 3$ e $q \geq 6$, ambos inteiros. Então $\Gamma(SB_{p,q}) = 6$.*

Acreditamos que a Conjectura 7.2.1 seja verdadeira, em consequência dos resultados dos trabalhos de (ASTÉ *et al.*, 2010) e (CAMPOS *et al.*, 2012).

8 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foram propostas soluções algorítmicas exatas para o Problema do Número de Grundy. Especificamente, foram propostos dois limitantes e cinco soluções exatas. Em relação aos limitantes, foram propostos um limite inferior e um superior, ambos baseados em heurísticas gulosas. Com relação às soluções exatas, foram propostas um algoritmo de Programação Dinâmica, um algoritmo Branch and Bound e três formulações de Programação Linear Inteira.

Inicialmente, foram geradas instâncias de testes aleatórias, com base na quantidade de vértices do grafo e na probabilidade p. Após isso, foram realizados experimentos computacionais com os algoritmos exatos propostos. Estes experimentos evidenciaram que o algoritmo proposto BBG supera o algoritmo PBG em relação à qualidade das soluções retornadas. Vale destacar, que a formulação F_STF se mostrou mais eficiente em qualidade de solução que as demais soluções exatas. Em relação ao tempo de execução, apenas a formulação F_STF se mostrou mais eficiente que as demais soluções exatas em grande parte dos casos. Em instâncias de densidade média, a formulação *F_DELTA* conseguiu encontrar o resultado mais rápido que os demais algoritmos, enquanto que nas classes de densidade alta, a formulação F_STF se mostrou mais eficiente.

Também foram realizados testes computacionais com os métodos exatos propostos na classe de grafos Stacked Book. Em termos gerais, os métodos exatos propostos, a formulação *F_PROD* conseguiu determinar o $\Gamma(G)$ em todas as instâncias e se mostrou mais eficiente que os demais algoritmos propostos. Além disso, com os resultados das soluções propostas, nós conjecturamos o valor exato do número de Grundy nesta classe na Conjectura 7.2.1, que acreditamos que seja verdadeira em consequência dos resultados existentes na literatura.

Os algoritmos apresentados neste trabalho abrem várias direções de pesquisas promissoras. Como trabalhos futuros, dentre os fatores que podem ser melhorados ou testados nos algoritmos, alguns são:

1. Estudo sobre outras heurísticas para o Problema do Número de Grundy;
2. Estudar técnicas de ordenação de vértices baseada em coloração ou em outros critérios, no qual tais técnicas podem resultar redução do tempo de execução e na obtenção de resultados mais exatos;
3. Propor uma versão do algoritmo Branch and Bound, onde fixamos cores em alguns vértices do grafo;

4. Avaliar o uso do parâmetro revised stair factor do Trabalho de Panda e Verma (2019) como limite superior nos algoritmos propostos neste trabalho;
5. Avaliar o uso de funções callback que permitem modificar o comportamento padrão nos modelos de PLI;
6. A determinação do parâmetro do número de Grundy em outras classes de produto cartesiano de grafos.

REFERÊNCIAS

ABOULKER, P.; BONNET, É.; KIM, E. J.; SIKORA, F. Grundy Coloring & Friends, Half-Graphs, Bicliques. In: PAUL, C.; BLÄSER, M. (Ed.). **37th International Symposium on Theoretical Aspects of Computer Science** (STACS 2020). Schloss Dagstuhl–Leibniz-Zentrum für Informatik: (Leibniz International Proceedings in Informatics (LIPIcs)), [S.l.]. v. 154, p. 18–58, 2020.

APPEL, K.; HAKEN, W.; KOCH, J. Every planar map is four colorable. part ii: Reducibility. **Illinois Journal of Mathematics**, Duke University Press, v. 21, n. 3, p. 491–567, 1977.

ASTÉ, M.; HAVET, F.; LINHARES-SALES, C. Grundy number and products of graphs. **Discrete Mathematics**, Elsevier, v. 310, n. 9, p. 1482–1490, 2010.

BELLMAN, R. Dynamic programming. **Science**, [S.l.], v. 153, n. 3731, p. 34–37, 1966.

BONDY, A. J.; MURTY, U. S. R. **Graph Theory with Applications**. [S.l.]: Elsevier Science Publishing, 1982. v. 5.

BONNET, É.; FOUCAUD, F.; KIM, E. J.; SIKORA, F. Complexity of grundy coloring and its variants. **Discrete Applied Mathematics**, Elsevier, v. 243, p. 99–114, 2018.

BRÉLAZ, D. New methods to color the vertices of a graph. **Communications of the ACM**, ACM New York, NY, USA, v. 22, n. 4, p. 251–256, 1979.

BRON, C.; KERBOSCH, J. Algorithm 457: finding all cliques of an undirected graph. **Communications of the ACM**, ACM New York, NY, USA, v. 16, n. 9, p. 575–577, 1973.

BROOKS, R. L. On colouring the nodes of a network. **Math. Proc. Cambridge Philos. Soc.**, [S.l.], v. 37, p. 194–197, 1941.

CAMPOS, V.; GYÁRFÁS, A.; HAVET, F.; SALES, C. L.; MAFFRAY, F. New bounds on the grundy number of products of graphs. **Journal of Graph Theory**, Wiley Online Library, v. 71, n. 1, p. 78–88, 2012.

CHARTRAND, G.; ZHANG, P. **Chromatic Graph Theory**. [S.l.]: Chapman & Hall/CRC, 2008. ISBN 1584888008.

CHRISTEN, C. A.; SELKOW, S. M. Some perfect coloring properties of graphs. **Journal of Combinatorial Theory, Series B**, Elsevier, v. 27, n. 1, p. 49–59, 1979.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

DANTZIG, G. **Linear Programming and Extensions**. [S.l.]: Princeton University Press, 2016. ISBN 9781400884179. Disponível em: https://doi.org/10.1515/9781400884179. Acesso em: 14 fev. 2022.

DIAL, R. B. Algorithm 360: Shortest-path forest with topological ordering [h]. **Communications of the ACM**, ACM New York, NY, USA, v. 12, n. 11, p. 632–633, 1969.

DOWNEY, R. G.; FELLOWS, M. R. **Fundamentals of parameterized complexity**. [S.l.]: Springer, 2013. v. 4.

EFFANTIN, B.; KHEDDOUCI, H. Grundy number of graphs. **Discussiones Mathematicae Graph Theory**, De Gruyter Open, v. 27, n. 1, p. 5–18, 2007.

EPPSTEIN, D.; LÖFFLER, M.; STRASH, D. Listing all maximal cliques in sparse graphs in near-optimal time. In: SPRINGER. **International Symposium on Algorithms and Computation**. [S.l.], 2010. p. 403–414.

ERDÖS, P.; HEDETNIEMI, S. T.; LASKAR, R. C.; PRINS, G. C. On the equality of the partial grundy and upper ochromatic numbers of graphs. **Discrete Mathematics**, Elsevier, v. 272, n. 1, p. 53–64, 2003.

EULER, L. Solutio problematis ad geometriam situs pertinensis. **Comm. Acad. Sci. Imper. Petropol.**, [S.l.], v. 8, p. 128–140, 1736.

GAREY, M. R.; JOHNSON, D. S. A guide to the theory of np-completeness. **Computers and intractability**, WH freeman & Co, 1979.

GILBERT, E. N. Random graphs. **The Annals of Mathematical Statistics**, JSTOR, v. 30, n. 4, p. 1141–1144, 1959.

GOYAL, N.; VISHWANATHAN, S. Np-completeness of undirected grundy numbering and related problems. **Manuscript, Bombay**, [S.l.], 1997.

GRUNDY, P. M. Mathematics and games. **Eureka**, [S.l.], v. 2, p. 6–9, 1939.

GUTHRIE, F. 9. note on the colouring of maps. **Proceedings of the Royal Society of Edinburgh**, Royal Society of Edinburgh Scotland Foundation, v. 10, p. 727–728, 1880.

HALE, W. K. Frequency assignment: Theory and applications. **Proceedings of the IEEE**, [S.l.], v. 68, p. 1497–1514, 1980.

HAVET, F.; KAISER, T.; STEHLIK, M. **Grundy number of the Cartesian product of a tree and a graph**. [S.l.: s.n.], 2008.

HAVET, F.; MAIA, A. K.; YU, M.-L. Complexity of greedy edge-colouring. **Journal of the Brazilian Computer Society**, SpringerOpen, v. 21, n. 1, p. 1–7, 2015.

HAVET, F.; SAMPAIO, L. On the grundy and b-chromatic numbers of a graph. **Algorithmica**, Springer, v. 65, n. 4, p. 885–899, 2013.

HEDETNIEMI S. HEDETNIEMI, T. B. S. A linear algorithm for the grundy (coloring) number of a tree. **Congressus Numeration**, [S.l.], v. 36, p. 351–363, 1982.

KARP, R. M. Reducibility among combinatorial problems. In: **Complexity of computer computations**. [S.l.]: Springer, 1972. p. 85–103.

LAND, A.; DOIG, A. An automatic method of solving discrete programming problems. **Econometrica**, Citeseer, v. 28, n. 3, p. 497–520, 1960.

LAWLER, E. A note on the complexity of the chromatic number problem. **Information Processing Letters**, v. 5, n. 3, p. 66–67, 1976.

LAWLER, E. L.; WOOD, D. E. Branch-and-bound methods: A survey. **Operations research**, INFORMS, v. 14, n. 4, p. 699–719, 1966.

MARINS, F. A. S. **Introdução à pesquisa operacional**. São Paulo: Cultura Acadêmica: Universidade Estadual Paulista, 2011. ISBN 9788579831676.

MATULA, D. W.; BECK, L. L. Smallest-last ordering and clustering and graph coloring algorithms. **Journal of the ACM (JACM)**, ACM New York, NY, USA, v. 30, n. 3, p. 417–427, 1983.

MOON, J. W.; MOSER, L. On cliques in graphs. **Israel journal of Mathematics**, Springer, v. 3, n. 1, p. 23–28, 1965.

MORRISON, D. R.; JACOBSON, S. H.; SAUPPE, J. J.; SEWELL, E. C. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. **Discrete Optimization**, [S.l.], v. 19, p. 79–102, 2016.

PANDA, B.; VERMA, S. On partial grundy coloring of bipartite graphs and chordal graphs. **Discrete Applied Mathematics**, [S.l.], v. 271, p. 171–183, 2019. ISSN 0166-218X. Disponível em: https://www.sciencedirect.com/science/Article/pii/S0166218X19303531. Acesso em: 14 fev. 2022.

RODRIGUES, E. N. H. D. **Coloração k-imprópria gulosa**. [S.l.], 2020. Disponível em: http://www.repositorio.ufc.br/handle/riufc/50955. Acesso em: 14 fev. 2022.

SAMPAIO, L. **Algorithmic aspects of graph colourings heuristics**. Tese (Doutorado) — Université Nice Sophia Antipolis, 2012.

SHI, Z.; GODDARD, W.; HEDETNIEMI, S. T.; KENNEDY, K.; LASKAR, R.; MCRAE, A. An algorithm for partial grundy number on trees. **Discrete Mathematics**, Elsevier, v. 304, n. 1-3, p. 108–116, 2005.

TELLE, J. A.; PROSKUROWSKI, A. Algorithms for vertex partitioning problems on partial k-trees. **SIAM Journal on Discrete Mathematics**, SIAM, v. 10, n. 4, p. 529–550, 1997.

TOMITA, E.; TANAKA, A.; TAKAHASHI, H. The worst-case time complexity for generating all maximal cliques and computational experiments. **Theoretical computer science**, Elsevier, v. 363, n. 1, p. 28–42, 2006.

WEST, D. B. **Introduction to Graph Theory**. [S.l.]: Pearson Modern Classic, 2018.

ZAKER, M. Grundy chromatic number of the complement of bipartite graphs. **Australas. J Comb.**, Citeseer, v. 31, p. 325–330, 2005.

APÊNDICE A – RESULTADOS DOS TESTES COMPUTACIONAIS

Nas tabelas abaixo, são exibidos os resultados obtidos com os testes feitos com os algoritmos descritos neste trabalho.

Na Tabela 5, é exibido o tempo de execução dos testes computacionais em instâncias de grafos gerados aleatoriamente. A primeira coluna exibe a quantidade de vértices do grafo testado, na segunda coluna é exibido o número de arestas do grafo testado. A terceira coluna exibe a probabilidade utilizada no algoritmo de geração de grafos aleatórios, na quarta coluna é mostrada a *densidade* do grafo, que é a razão entre a quantidade de arestas do grafo e a quantidade de arestas do grafo completo com o mesma quantidade de vértices. Entre a quinta e nona coluna é exibido o tempo em segundos que o algoritmos propostos no teste levaram para encontrar o número de Grundy do grafo.

Na Tabela 6, é apresentado o resultado retornado pelos algoritmos nos testes computacionais em instâncias de grafos gerados aleatoriamente. As quatros primeiras colunas exibem o mesmos dados da Tabela 5 do grafo testado. Entre a quinta e nona coluna é exibido o resultado retornado pelo algoritmos propostos no teste.

Na Tabela 7, é exibido o tempo de execução dos testes computacionais em instâncias de grafos da classe Stacked Book. A primeira coluna exibe a quantidade de vértices do grafo testado, na segunda coluna é exibido o número de arestas do grafo testado. Entre a terceira e nona coluna é exibido o tempo em segundos que o algoritmos propostos no teste levaram para encontrar o número de Grundy do grafo.

Na Tabela 8, é apresentado o resultado retornado pelos algoritmos nos testes computacionais em instâncias de grafos da classe Stacked Book. A primeira coluna exibe a quantidade de vértices do grafo testado, na segunda coluna é exibido o número de arestas do grafo testado. Entre a terceira e nona coluna é exibido o resultado retornado pelo algoritmos propostos no teste.

Tabela 5 – Tempo de execução dos algoritmos em instâncias de grafos gerados aleatoriamente.

$\|V(G)\|$	$\|E(G)\|$	**Probabilidade**	**Densidade**	*PDG*	$BBG\|\Delta+1$	$BBG\|\zeta$	*F_DELTA*	*F_STF*
10	1	0.1	0.0222	0.2515	0.0002	0.0001	0.0119	0.0157
10	1	0.1	0.0222	0.2598	0.0005	0.0001	0.0108	0.0129
10	4	0.1	0.0889	0.2582	0.0015	0.0012	0.0284	0.0208
10	4	0.1	0.0889	0.2613	0.0015	0.0004	0.0177	0.0099
10	6	0.1	0.1333	246	0.0029	0.0016	0.0292	0.0185
10	8	0.3	0.1778	0.2566	0.0023	0.1071	0.0311	0.0963
10	13	0.3	0.2889	0.2249	0.0449	1.5538	0.0795	0.0725
10	12	0.3	0.2667	0.2468	0.0412	0.2997	0.0606	0.0637
10	12	0.3	0.2667	0.2581	0.0716	0.0771	0.0762	0.0663
10	17	0.3	0.3778	0.2447	0.3545	1.7141	0.1449	0.0345
10	17	0.5	0.3778	0.2471	0.359	0.0061	0.1322	0.0562
10	25	0.5	0.5556	0.2229	0.4108	0.0404	0.7168	1.0744
10	23	0.5	0.5111	277	3.3716	0.0365	0.6689	0.2125
10	19	0.5	0.4222	0.2574	0.8377	0.0271	0.5618	0.0383
10	26	0.5	0.5778	0.2619	5.5033	0.0849	1.1794	1.2967
10	28	0.7	0.6222	0.2111	1.0349	1.3922	0.9007	1.2815
10	32	0.7	0.7111	0.2123	17.0138	5.1758	0.8713	1.3593
10	33	0.7	0.7333	0.2174	16.5634	4.2902	1.9226	1.4801
10	32	0.7	0.7111	0.2074	14.8062	0.4409	0.8967	0.0527
10	35	0.7	0.7778	0.2288	47.8265	6.52	3.1379	1.1978
10	40	0.9	0.8889	0.1726	47.585	13.5252	1.9686	1.5233
10	42	0.9	0.9333	0.1758	29.0042	4.5154	1.403	1.0559
10	38	0.9	0.8444	0.1826	34.3237	1.15	1.1742	0.0495
10	42	0.9	0.9333	0.166	46.4523	10.8979	1469	1.1816
10	42	0.9	0.9333	0.1585	20.4704	2.2952	0.0928	0.0646
15	16	0.1	0.1524	18.7623	0.8387	0.8766	0.0879	0.0772
15	11	0.1	0.1048	18.9249	0.1508	0.0282	0.06	0.0469
15	10	0.1	0.0952	19.414	0.0516	0.041	0.0462	0.0505
15	10	0.1	0.0952	19.1046	0.0514	0.0437	0.048	0.0561
15	15	0.1	0.1429	19.4142	0.968	0.1733	0.0739	0.0471
15	27	0.3	0.2571	18.8191	153.3028	144.9361	1.6404	1.591
15	33	0.3	0.3143	18.5926	900.0	655.5087	2.0773	1.938
15	25	0.3	0.2381	19.3349	118.0314	24.7585	0.5147	1.98
15	47	0.3	0.4476	17.2387	900.0	900.0	11.1161	7.9937
15	26	0.3	0.2476	19.1799	144.1089	95.9129	0.6713	0.6332
15	59	0.5	0.5619	16.6505	900.0	900.0	30.2045	19.3987
15	53	0.5	0.5048	16.7148	900.0	900.0	19.1252	20.506
15	56	0.5	0.5333	16.5284	900.0	900.0	22.1073	11.7819
15	63	0.5	0.6	15.9486	900.0	900.0	41.3577	28.1442
15	54	0.5	0.5143	15.9727	900.0	900.0	20.8988	20.6971
15	71	0.7	0.6762	14.7705	900.0	900.0	59.3668	57.1034
15	79	0.7	0.7524	14.1843	900.0	900.0	236.3678	257.9647
15	79	0.7	0.7524	14.2263	900.0	900.0	368.3202	310.7801
15	75	0.7	0.7143	14.5852	900.0	900.0	202.5697	239.7873
15	77	0.7	0.7333	14.2499	900.0	900.0	56.8919	55.1051
15	97	0.9	0.9238	12.6786	900.0	900.0	255.2446	197.306
15	96	0.9	0.9143	12.3081	900.0	900.0	407.1343	331.0792
15	94	0.9	0.8952	13.1204	900.0	900.0	900.1575	773.1803
15	96	0.9	0.9143	12.8046	900.0	900.0	165.6825	131.0978
15	95	0.9	0.9048	12.8969	900.0	900.0	132.2415	134.0237

$\|V(G)\|$	$\|E(G)\|$	**Probabilidade**	**Densidade**	*PDG*	$BBG\|\Delta+1$	$BBG\|\zeta$	*F_DELTA*	*F_STF*
20	18	0.1	0.0947	22.9227	22.9489	22.9227	0.2117	0.0872
20	22	0.1	0.1158	58.0643	31.0353	58.0643	0.0948	0.0783
20	23	0.1	0.1211	70.2427	189.0121	70.2427	1.0873	0.071
20	19	0.1	0.1	60.3039	30.2956	60.3039	0.1217	0.1124
20	16	0.1	0.0842	6.7558	3.5331	6.7558	0.0583	0.0586
20	60	0.3	0.3158	900.0	900.0	900.0	55.0297	54.4168
20	48	0.3	0.2526	900.0	900.0	900.0	22.8978	22.2265
20	52	0.3	0.2737	900.0	900.0	900.0	42.8496	43.3564
20	65	0.3	0.3421	900.0	900.0	900.0	82.6814	95.7179
20	55	0.3	0.2895	900.0	900.0	900.0	32.9144	48.9408
20	103	0.5	0.5421	900.0	900.0	900.0	900.1386	900.1529
20	95	0.5	0.5	900.0	900.0	900.0	900.0994	900.097
20	106	0.5	0.5579	900.0	900.0	900.0	900.1658	900.097
20	84	0.5	0.4421	900.0	900.0	900.0	422.388	192.0769
20	93	0.5	0.4895	900.0	900.0	900.0	869.3215	671.4922
20	138	0.7	0.7263	900.0	900.0	900.0	900.153	900.1128
20	131	0.7	0.6895	900.0	900.0	900.0	900.1816	900.1406
20	131	0.7	0.6895	900.0	900.0	900.0	900.1664	900.1506
20	126	0.7	0.6632	900.0	900.0	900.0	900.236	900.1006
20	142	0.7	0.7474	900.0	900.0	900.0	900.2769	900.1489
20	174	0.9	0.9158	900.0	900.0	900.0	900.2701	900.1445
20	170	0.9	0.8947	900.0	900.0	900.0	900.2477	900.1718
20	176	0.9	0.9263	900.0	900.0	900.0	900.1885	900.2173
20	177	0.9	0.9316	900.0	900.0	900.0	900.2152	900.2156
20	173	0.9	0.9105	900.0	900.0	900.0	900.228	900.2024
25	24	0.1	0.08	900.0	900.0	900.0	0.5229	0.4831
25	24	0.1	0.08	900.0	900.0	900.0	0.1715	0.1365
25	26	0.1	0.0867	900.0	841.2043	900.0	0.1849	0.1633
25	42	0.1	0.14	900.0	900.0	900.0	7.7589	8.5693
25	33	0.1	0.11	900.0	900.0	900.0	0.1703	0.1446
25	92	0.3	0.3067	900.0	900.0	900.0	390.7195	900.0
25	109	0.3	0.3633	900.0	900.0	900.0	900.1336	900.0
25	86	0.3	0.2867	900.0	900.0	900.0	327.2523	333.3051
25	87	0.3	0.29	900.0	900.0	900.0	541.5691	900.0
25	87	0.3	0.29	900.0	900.0	900.0	690.3803	492.7151
25	152	0.5	0.5067	900.0	900.0	900.0	900.2666	900.1233
25	150	0.5	0.5	900.0	900.0	900.0	900.2006	900.1283
25	151	0.5	0.5033	900.0	900.0	900.0	900.1998	900.1512
25	170	0.5	0.5667	900.0	900.0	900.0	900.1692	900.1564
25	142	0.5	0.4733	900.0	900.0	900.0	900.1573	900.1562
25	213	0.7	0.71	900.0	900.0	900.0	900.2199	900.1791
25	213	0.7	0.71	900.0	900.0	900.0	900.2241	900.1762
25	208	0.7	0.6933	900.0	900.0	900.0	900.1804	900.1928
25	199	0.7	0.6633	900.0	900.0	900.0	900.244	900.1347
25	201	0.7	0.67	900.0	900.0	900.0	900.2087	900.1477
25	270	0.9	0.9	900.0	900.0	900.0	900.3176	900.2648
25	266	0.9	0.8867	900.0	900.0	900.0	900.2874	900.2133
25	272	0.9	0.9067	900.0	900.0	900.0	900.2522	900.2233
25	274	0.9	0.9133	900.0	900.0	900.0	900.2752	900.2903
25	276	0.9	0.92	900.0	900.0	900.0	900.3682	900.2459

Tabela 6 – Resultado dos algoritmos em instâncias de grafos gerados aleatoriamente.

$\|V(G)\|$	$\|E(G)\|$	**Probabilidade**	**Densidade**	*PDG*	$BBG\|\Delta+1$	$BBG\|\zeta$	*F_DELTA*	*F_STF*
10	1	0.1	0.0222	2.0	2.0	2.0	2.0	2.0
10	1	0.1	0.0222	2.0	2.0	2.0	2.0	2.0
10	4	0.1	0.0889	3.0	3.0	3.0	3.0	3.0
10	4	0.1	0.0889	2.0	2.0	2.0	2.0	2.0
10	6	0.1	0.1333	3.0	3.0	3.0	3.0	3.0
10	8	0.3	0.1778	4.0	4.0	4.0	4.0	4.0
10	13	0.3	0.2889	4.0	4.0	4.0	4.0	4.0
10	12	0.3	0.2667	4.0	4.0	4.0	4.0	4.0
10	12	0.3	0.2667	4.0	4.0	4.0	4.0	4.0
10	17	0.3	0.3778	5.0	5.0	5.0	5.0	5.0
10	17	0.5	0.3778	5.0	5.0	5.0	5.0	5.0
10	25	0.5	0.5556	6.0	6.0	6.0	6.0	6.0
10	23	0.5	0.5111	5.0	5.0	5.0	5.0	5.0
10	19	0.5	0.4222	5.0	5.0	5.0	5.0	5.0
10	26	0.5	0.5778	6.0	6.0	6.0	6.0	6.0
10	28	0.7	0.6222	6.0	6.0	6.0	6.0	6.0
10	32	0.7	0.7111	7.0	7.0	7.0	7.0	7.0
10	33	0.7	0.7333	7.0	7.0	7.0	7.0	7.0
10	32	0.7	0.7111	7.0	7.0	7.0	7.0	7.0
10	35	0.7	0.7778	6.0	6.0	6.0	6.0	6.0
10	40	0.9	0.8889	7.0	7.0	7.0	7.0	7.0
10	42	0.9	0.9333	8.0	8.0	8.0	8.0	8.0
10	38	0.9	0.8444	8.0	8.0	8.0	8.0	8.0
10	42	0.9	0.9333	7.0	7.0	7.0	7.0	7.0
10	42	0.9	0.9333	9.0	9.0	9.0	9.0	9.0
15	16	0.1	0.1524	4.0	4.0	4.0	4.0	4.0
15	11	0.1	0.1048	4.0	4.0	4.0	4.0	4.0
15	10	0.1	0.0952	3.0	3.0	3.0	3.0	3.0
15	10	0.1	0.0952	3.0	3.0	3.0	3.0	3.0
15	15	0.1	0.1429	4.0	4.0	4.0	4.0	4.0
15	27	0.3	0.2571	5.0	5.0	5.0	5.0	5.0
15	33	0.3	0.3143	6.0	6.0	6.0	6.0	6.0
15	25	0.3	0.2381	6.0	6.0	6.0	6.0	6.0
15	47	0.3	0.4476	7.0	6.0	6.0	7.0	7.0
15	26	0.3	0.2476	6.0	6.0	6.0	6.0	6.0
15	59	0.5	0.5619	9.0	4.0	4.0	9.0	9.0
15	53	0.5	0.5048	8.0	4.0	4.0	8.0	8.0
15	56	0.5	0.5333	9.0	4.0	4.0	9.0	9.0
15	63	0.5	0.6	9.0	4.0	4.0	9.0	9.0
15	54	0.5	0.5143	8.0	4.0	4.0	8.0	8.0
15	71	0.7	0.6762	8.0	6.0	6.0	8.0	8.0
15	79	0.7	0.7524	10.0	6.0	6.0	10.0	10.0
15	79	0.7	0.7524	9.0	6.0	6.0	9.0	9.0
15	75	0.7	0.7143	9.0	6.0	6.0	9.0	9.0
15	77	0.7	0.7333	10.0	6.0	6.0	10.0	10.0
15	97	0.9	0.9238	11.0	6.0	6.0	11.0	11.0
15	96	0.9	0.9143	10.0	6.0	6.0	10.0	10.0
15	94	0.9	0.8952	10.0	6.0	6.0	10.0	10.0
15	96	0.9	0.9143	11.0	6.0	6.0	11.0	11.0
15	95	0.9	0.9048	11.0	6.0	6.0	11.0	11.0

$\|V(G)\|$	$\|E(G)\|$	**Probabilidade**	**Densidade**	PDG	$BBG\|\Delta+1$	$BBG\|\zeta$	F_DELTA	F_STF
20	18	0.1	0.0947	4.0	4.0	4.0	4.0	4.0
20	22	0.1	0.1158	5.0	5.0	5.0	5.0	5.0
20	23	0.1	0.1211	5.0	5.0	5.0	5.0	5.0
20	19	0.1	0.1	4.0	4.0	4.0	4.0	4.0
20	16	0.1	0.0842	4.0	4.0	4.0	4.0	4.0
20	60	0.3	0.3158	0.0	5.0	5.0	8.0	8.0
20	48	0.3	0.2526	0.0	5.0	5.0	7.0	7.0
20	52	0.3	0.2737	0.0	5.0	5.0	7.0	7.0
20	65	0.3	0.3421	0.0	5.0	5.0	8.0	8.0
20	55	0.3	0.2895	0.0	5.0	5.0	7.0	7.0
20	103	0.5	0.5421	0.0	5.0	5.0	10.0	10.0
20	95	0.5	0.5	0.0	5.0	5.0	10.0	10.0
20	106	0.5	0.5579	0.0	5.0	5.0	10.0	11.0
20	84	0.5	0.4421	0.0	5.0	5.0	9.0	9.0
20	93	0.5	0.4895	0.0	5.0	5.0	10.0	10.0
20	138	0.7	0.7263	0.0	5.0	5.0	12.0	13.0
20	131	0.7	0.6895	0.0	5.0	5.0	12.0	12.0
20	131	0.7	0.6895	0.0	5.0	5.0	12.0	12.0
20	126	0.7	0.6632	0.0	5.0	5.0	11.0	11.0
20	142	0.7	0.7474	0.0	5.0	5.0	13.0	13.0
20	174	0.9	0.9158	0.0	5.0	5.0	16.0	16.0
20	170	0.9	0.8947	0.0	5.0	5.0	14.0	14.0
20	176	0.9	0.9263	0.0	5.0	5.0	15.0	15.0
20	177	0.9	0.9316	0.0	5.0	5.0	13.0	13.0
20	173	0.9	0.9105	0.0	5.0	5.0	14.0	14.0
25	24	0.1	0.08	0.0	5.0	4.0	4.0	4.0
25	24	0.1	0.08	0.0	5.0	4.0	4.0	4.0
25	26	0.1	0.0867	0.0	5.0	5.0	5.0	5.0
25	42	0.1	0.14	0.0	0.0	5.0	6.0	6.0
25	33	0.1	0.11	0.0	0.0	5.0	5.0	5.0
25	92	0.3	0.3067	0.0	0.0	5.0	9.0	9.0
25	109	0.3	0.3633	0.0	0.0	5.0	10.0	11.0
25	86	0.3	0.2867	0.0	0.0	5.0	9.0	9.0
25	87	0.3	0.29	0.0	0.0	5.0	9.0	9.0
25	87	0.3	0.29	0.0	0.0	5.0	9.0	9.0
25	152	0.5	0.5067	0.0	0.0	5.0	12.0	12.0
25	150	0.5	0.5	0.0	0.0	5.0	10.0	12.0
25	151	0.5	0.5033	0.0	0.0	5.0	11.0	12.0
25	170	0.5	0.5667	0.0	0.0	5.0	12.0	12.0
25	142	0.5	0.4733	0.0	0.0	5.0	10.0	11.0
25	213	0.7	0.71	0.0	0.0	0.0	14.0	14.0
25	213	0.7	0.71	0.0	0.0	0.0	14.0	14.0
25	208	0.7	0.6933	0.0	0.0	0.0	14.0	14.0
25	199	0.7	0.6633	0.0	0.0	0.0	13.0	14.0
25	201	0.7	0.67	0.0	0.0	0.0	14.0	15.0
25	270	0.9	0.9	0.0	0.0	0.0	17.0	17.0
25	266	0.9	0.8867	0.0	0.0	0.0	17.0	17.0
25	272	0.9	0.9067	0.0	0.0	0.0	17.0	17.0
25	274	0.9	0.9133	0.0	0.0	0.0	18.0	18.0
25	276	0.9	0.92	0.0	0.0	0.0	16.0	16.0

Tabela 7 – Tempo de execução dos algoritmos em instâncias de grafos da classe Stacked Book.

$\|V(G)\|$	$\|E(G)\|$	*PDG*	$BBG\|\Delta+1$	$BBG\|\zeta$	$BBG\|6$	*F_DELTA*	*F_STF*	*F_PROD*
8	10	0.0762	0.0161	0.0027	0.0293	0.0394	0.082	0.0491
10	13	0.3778	0.1651	0.0116	0.2872	0.0606	0.0435	0.0629
12	16	1.2062	1.4246	0.0445	0.4883	0.068	0.0376	0.0572
14	19	11.4254	11.1389	0.1894	3.009	0.0721	0.0375	0.0727
16	22	108.3477	116.5312	0.8395	16.4833	0.116	0.0523	0.0875
18	25	900.0	900.0	2.9018	100.3524	0.1887	0.0494	0.0874
20	28	900.0	900.0	11.6255	900.0	0.2177	0.0551	0.0868
12	17	2.1991	1.8346	0.4093	3.2807	0.4721	0.1091	0.4377
15	22	32.7218	72.1091	23.9064	75.0785	1.0623	0.6602	0.7215
18	27	900.0	900.0	426.9011	900.0	1.4386	0.6507	0.6518
21	32	900.0	900.0	900.0	900.0	2.4435	0.7305	0.8026
24	37	900.0	900.0	900.0	900.0	6.9443	1.1645	1.2642
27	42	900.0	900.0	900.0	900.0	6.5457	2.46	2.7442
30	47	900.0	900.0	900.0	900.0	12.2451	1.5074	1.6489
16	24	66.1942	66.6297	66.2718	142.4074	1.159	1.0894	1.1959
20	31	900.0	900.0	900.0	900.0	2.3749	2.2848	1.8785
24	38	900.0	900.0	900.0	900.0	5.162	3.2007	3.5301
28	45	900.0	900.0	900.0	900.0	8.1062	5.1624	0.8722
32	52	900.0	900.0	900.0	900.0	14.626	6.182	4.7127
36	59	900.0	900.0	900.0	900.0	20.5432	6.5794	5.2777
40	66	900.0	900.0	900.0	900.0	22.8508	16.7327	23.828
20	31	900.0	900.0	900.0	900.0	2.9415	2.6892	3.2066
25	40	900.0	900.0	900.0	900.0	8.786	8.4706	6.8728
30	49	900.0	900.0	900.0	900.0	12.9326	12.3889	8.9107
35	58	900.0	900.0	900.0	900.0	17.2346	20.3157	13.6159
40	67	900.0	900.0	900.0	900.0	37.8896	22.2911	32.295
45	76	900.0	900.0	900.0	900.0	51.6786	34.523	49.3161
50	85	900.0	900.0	900.0	900.0	52.2218	36.0979	78.8473
24	38	900.0	900.0	900.0	900.0	9.0053	8.4999	9.5815
30	49	900.0	900.0	900.0	900.0	36.2336	31.9167	8.7341
36	60	900.0	900.0	900.0	900.0	67.1495	63.4895	15.2737
42	71	900.0	900.0	900.0	900.0	48.8592	45.0427	7.5533
48	82	900.0	900.0	900.0	900.0	66.5362	135.5291	64.4436
54	93	900.0	900.0	900.0	900.0	77.5418	157.839	1.0228
60	104	900.0	900.0	900.0	900.0	87.5413	115.2027	209.3208
28	45	900.0	900.0	900.0	900.0	0.8058	0.7864	0.8313
35	58	900.0	900.0	900.0	900.0	206.4198	190.081	2.0438
42	71	900.0	900.0	900.0	900.0	124.7866	117.3388	7.5533
49	84	900.0	900.0	900.0	900.0	236.5694	216.6919	72.3437
56	97	900.0	900.0	900.0	900.0	110.8571	96.575	13.8502
63	110	900.0	900.0	900.0	900.0	111.9609	102.8806	12.9716
70	123	900.0	900.0	900.0	900.0	900.0	127.9469	55.372
32	52	900.0	900.0	900.0	900.0	3.289	2.6586	3.1894
40	67	900.0	900.0	900.0	900.0	351.0871	309.2882	0.9798
48	82	900.0	900.0	900.0	900.0	158.4098	144.8124	17.3972
56	97	900.0	900.0	900.0	900.0	118.7356	109.5733	95.2578
64	112	900.0	900.0	900.0	900.0	169.1582	161.7867	7.7991
72	127	900.0	900.0	900.0	900.0	179.3899	215.138	622.954
80	142	900.0	900.0	900.0	900.0	245.0823	900.0	0.9805
36	59	900.0	900.0	900.0	900.0	12.3682	20.6295	14.4236
45	76	900.0	900.0	900.0	900.0	900.0	900.0	1.6039
54	93	900.0	900.0	900.0	900.0	900.0	900.0	28.9685
63	110	900.0	900.0	900.0	900.0	329.3761	576.5879	20.1874
72	127	900.0	900.0	900.0	900.0	563.4221	900.0	756.8502
81	144	900.0	900.0	900.0	900.0	900.0	900.0	10.4491
90	161	900.0	900.0	900.0	900.0	332.0421	365.5683	46.2144

Fonte: Elaborado pelo autor (2022).

Tabela 8 – Resultado dos algoritmos em instâncias de grafos da classe Stacked Book.

$\|V(G)\|$	$\|E(G)\|$	PDG	$BBG\|\Delta+1$	$BBG\|\zeta$	$BBG\|6$	F_DELTA	F_STF	F_PROD
8	10	4.0	4.0	4.0	4.0	4.0	4.0	4.0
10	13	4.0	4.0	4.0	4.0	4.0	4.0	4.0
12	16	4.0	4.0	4.0	4.0	4.0	4.0	4.0
14	19	4.0	4.0	4.0	4.0	4.0	4.0	4.0
16	22	4.0	4.0	4.0	4.0	4.0	4.0	4.0
18	25	4.0	4.0	4.0	4.0	4.0	4.0	4.0
20	28	4.0	4.0	4.0	4.0	4.0	4.0	4.0
12	17	4.0	4.0	4.0	4.0	4.0	4.0	4.0
15	22	4.0	4.0	4.0	4.0	4.0	4.0	4.0
18	27	4.0	4.0	4.0	4.0	4.0	4.0	4.0
21	32	4.0	4.0	4.0	4.0	4.0	4.0	4.0
24	37	4.0	4.0	4.0	4.0	4.0	4.0	4.0
27	42	4.0	4.0	4.0	4.0	4.0	4.0	4.0
30	47	4.0	4.0	4.0	4.0	4.0	4.0	4.0
16	24	5.0	5.0	5.0	5.0	5.0	5.0	5.0
20	31	5.0	5.0	5.0	5.0	5.0	5.0	5.0
24	38	5.0	5.0	5.0	5.0	5.0	5.0	5.0
28	45	5.0	5.0	5.0	5.0	5.0	5.0	5.0
32	52	5.0	5.0	5.0	5.0	5.0	5.0	5.0
36	59	5.0	5.0	5.0	5.0	5.0	5.0	5.0
40	66	5.0	5.0	5.0	5.0	5.0	5.0	5.0
20	31	5.0	5.0	5.0	5.0	5.0	5.0	5.0
25	40	5.0	5.0	5.0	5.0	5.0	5.0	5.0
30	49	5.0	5.0	5.0	5.0	5.0	5.0	5.0
35	58	5.0	5.0	5.0	5.0	5.0	5.0	5.0
40	67	5.0	5.0	5.0	5.0	5.0	5.0	5.0
45	76	5.0	5.0	5.0	5.0	5.0	5.0	5.0
50	85	5.0	5.0	5.0	5.0	5.0	5.0	5.0
24	38	5.0	5.0	5.0	5.0	6.0	6.0	6.0
30	49	5.0	5.0	5.0	5.0	6.0	6.0	6.0
36	60	5.0	5.0	5.0	5.0	6.0	6.0	6.0
42	71	5.0	5.0	5.0	5.0	6.0	6.0	6.0
48	82	5.0	5.0	5.0	5.0	6.0	6.0	6.0
54	93	5.0	5.0	5.0	5.0	6.0	6.0	6.0
60	104	5.0	5.0	5.0	5.0	6.0	6.0	6.0
28	45	5.0	5.0	5.0	5.0	6.0	6.0	6.0
35	58	5.0	5.0	5.0	5.0	6.0	6.0	6.0
42	71	5.0	5.0	5.0	5.0	6.0	6.0	6.0
49	84	5.0	5.0	5.0	5.0	6.0	6.0	6.0
56	97	5.0	5.0	5.0	5.0	6.0	6.0	6.0
63	110	5.0	5.0	5.0	5.0	6.0	6.0	6.0
70	123	5.0	5.0	5.0	5.0	5.0	6.0	6.0
32	52	5.0	5.0	5.0	5.0	6.0	6.0	6.0
40	67	5.0	5.0	5.0	5.0	6.0	6.0	6.0
48	82	5.0	5.0	5.0	5.0	6.0	6.0	6.0
56	97	5.0	5.0	5.0	5.0	6.0	6.0	6.0
64	112	5.0	5.0	5.0	5.0	6.0	6.0	6.0
72	127	5.0	5.0	5.0	5.0	6.0	6.0	6.0
80	142	5.0	5.0	5.0	5.0	6.0	5.0	6.0
36	59	5.0	5.0	5.0	5.0	6.0	6.0	6.0
45	76	5.0	5.0	5.0	5.0	5.0	5.0	6.0
54	93	5.0	5.0	5.0	5.0	5.0	5.0	6.0
63	110	5.0	5.0	5.0	5.0	6.0	6.0	6.0
72	127	5.0	5.0	5.0	5.0	6.0	6.0	6.0
81	144	5.0	5.0	5.0	5.0	5.0	6.0	6.0
90	161	5.0	5.0	5.0	5.0	6.0	6.0	6.0

Fonte: Elaborado pelo autor (2022).