UNIVERSIDADE FEDERAL FLUMINENSE

JEFFERSON DE SOUSA SILVA

Obstruções Minimais de Grafos-$(2,1)$

NITERÓI

2013

UNIVERSIDADE FEDERAL FLUMINENSE

JEFFERSON DE SOUSA SILVA

Obstruções Minimais de Grafos-$(2,1)$

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação. Área de concentração: Algoritmos e Otimização.

Orientador:
Loana Tito Nogueira

NITERÓI

2013

Obstruções minimais de grafos-$(2,1)$

Jefferson de Sousa Silva

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Computação.

Aprovada por:

Profa. Dra. Loana Tito Nogueira / IC-UFF (Presidente)

Dra. Raquel de Souza Francisco Bravo

Prof. Dr. Fábio Protti / UFF

Prof. Dr. Luerbio Faria / UERJ

Niterói, 24 de Julho de 2013.

Dedico esse trabalho a Deus por está presente em minha vida me iluminando em todos os momentos, a minha esposa Mônia, aos meus filhos Filipe, Júlia e Enzo pelo amor, apoio constante, compreensão e incentivo, sem os quais não haveria a realização desse sonho e aos meus queridos pais Alvino e Célia por me darem força e estarem sempre ao meu lado..

Agradecimentos

À minha orientadora, Prof.ª Dra. Loana Tito Nogueira, pela orientação atenciosa e segura na qual foi fundamental para a realização deste trabalho. Pela confiança e amizade demonstrada, meus sinceros agradecimentos.

À Prof.ª Dra. Raquel Bravo, pela presteza, amizade, confiança e por transmitir motivação para pesquisa, os meus sinceros agradecimentos.

Aos professores do curso, meus agradecimentos pelos valiosos conhecimentos e por despertar ainda mais o meu entusiasmo pela pesquisa.

Ao meus colegas de Mestrado: minha irmã Jeanne Leite e meu cunhado Gláucio Leite, pelo convívio harmonioso e união demonstrada durante esta fase da vida. Avante trio ternura.

À minha amiga e sogra Fátima Mendes, por cuidar da minha família durante minha ausência.

E a todos que compartilharam o trilhar de mais esse caminho percorrido, contribuindo, direta e indiretamente, para que eu realizasse esta pesquisa, auxiliando-me e dando-me forças nos momentos em que mais precisei.

Resumo

Este trabalho consiste no estudo de um problema de partição de grafos, a saber, um estudo dos grafos-$(2,1)$, i.e., grafos cujo conjunto de vértices pode ser particionado em dois conjuntos independentes e uma clique. Mais precisamente, este trabalho apresenta uma família de subgrafos que, quando presentes em G, impedem que G seja um grafo-$(2,1)$, ou seja, detectamos uma família de subgrafos proibidos dos grafos-$(2,1)$, também chamadas de obstrução-$(2,1)$. Além disso, mostramos que os grafos $(2,1)$ possuem uma família infinita de subgrafos proibidos.

Palavras-chave: obstruções, grafos-(k,l), grafos-$(2,1)$.

Abstract

This work consists of studying a partition problem: we are interested in recognizing $(2,1)$-graphs, i.e., graphs whose set of vertices can be partitioned into two independent sets and one clique. More specifically, this work presents forbidden subgraphs for a graph to be $(2,1)$, that is, we caracterize some forbidden structures in $(2,1)$-graphs. Moreover, we show that $(2,1)$-graphs contain a infinite family of forbidden sugraphs.

Keywords: obstruções, grafos-(k,l), grafos-$(2,1)$.

Palavras-chave

1. Grafos-(k, l)
2. Grafos-$(2, 1)$
3. Obstruções

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Teoria dos Grafos é uma área da Matemática que ao longo dos anos vem ganhando destaque. Surgiu inicialmente como um desafio no problema conhecido como "as pontes de Königsberg". No entanto, com o advento da computação vem ganhando cada vez mais espaço e hoje é tida como uma eficiente ferramenta para problemas de diversas áreas como a indústria, o comércio e as engenharias.

Um problema clássico em Teoria dos Grafos é o problema de partição, que busca particionar o conjunto de vértices de um dado grafo em subconjuntos V_1,V_2,...,V_k que atendam a certas propriedades, que podem ser internas ou externas. Através desse problema, consegue-se modelar um grande número de situações práticas.

Um método de caracterização de classes de grafos que é amplamente empregado é o da definição por proibição. Este método consiste em caracterizar uma classe de grafos por uma família de subgrafos que, quando presentes em G, impedem que G pertença a tal classe.

Por exemplo, um grafo G é bipartido se, e somente se, não contém ciclos de comprimento ímpar.

Nesse trabalho tentamos caracterizar a classe dos grafos-$(2,1)$ por uma família de subgrafos proibidos. Mais especificamente, nosso resultado consiste em apresentar uma família de subgrafos que, se $G \in (2,1)$ então G não contém nenhum dos subgrafos como subgrafo induzido.

A seguir descreveremos como esta dissertação está organizada.

No capítulo 1 estabeleceremos alguns conceitos básicos sobre grafos que serão abordados ao longo deste trabalho. No segundo capítulo realizamos uma revisão bibliográfica

do estado da arte da classe dos grafos-(k, l), em particular a classe $(2, 1)$. No capítulo 3 é descrito o procedimento utilizado para encontrarmos uma família de subgrafos proibidos para grafos-$(2, 1)$. No quarto capítulo apresentamos as obstruções-$(2, 1)$ encontradas. Finalmente, a conclusão deste trabalho é apresentada evidenciando os resultados obtidos em seu desenvolvimento.

No que segue, apresentaremos algumas definições e notações que serão utilizadas no decorrer deste trabalho.

1.1 Grafos - Definições e Notações

Um *grafo simples* é um par ordenado $G = (V, E)$, onde V é um conjunto finito não-vazio de *vértices*, denotado por $V(G)$ e E é um conjunto de pares não-ordenados de vértices distintos, chamados *arestas*, e denotado por $E(G)$. Utilizaremos a notação $n = |V(G)|$ e $m = |E(G)|$ para denotarmos a cardinalidade de $V(G)$ e $E(G)$, respectivamente. Ao decorrer deste trabalho, iremos utilizar a denominação grafos para denotar o que definimos como grafos simples.

Um grafo G é dito *trivial* se $|V(G)| = 1$, isto é G possui um único vértice.

Um vértice u é *adjacente* a outro vértice v em G se a aresta $uv \in E(G)$. Neste caso, dizemos que u e v são *vizinhos* em G, e que a aresta uv é *incidente* a u e a v, ou que tem *extremos* u e v. Denotamos por $N(u)$ o conjunto de vértices adjacentes a u em G e tal conjunto é chamado de *vizinhança* de u, por $\overline{N}(u)$ o conjunto de vértices não-adjacentes a u em G e tal conjunto é chamado de *não vizinhança* de u e por $N[u]$ o conjunto $N(u) \cup \{u\}$ é chamado de *vizinhança fechada* de u.

O *grau* de um vértice $v \in V(G)$, denotado por $d(v)$, é o número de arestas incidentes ao vértice v.

Um grafo H é um *subgrafo* de um grafo G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Dado um conjunto de vértices $Y \subseteq V(G)$, $Y \neq \emptyset$, o *subgrafo induzido* por Y, denotado por $G[Y]$ é o subgrafo H de G tal que $V(H) = Y$ e $E(H)$ é o conjunto das arestas de G que têm ambos os extremos em Y. Neste trabalho assumiremos que todos os termos subgrafo que usarmos refere-se a noção de subgrafo induzido.

Um *passeio* em um grafo G é uma sequência de vértices, cada qual adjacente ao vértice que lhe sucede na sequência. Um *caminho* num grafo G é um passeio $P = v_1, v_2, \ldots, v_k$ onde os $v_i's$ são vértices (dois a dois distintos), e $(v_i, v_{i+1}) \in E(G)$, $1 \leq i \leq k - 1$.

Uma *corda* em P é uma aresta que liga dois vértices não-consecutivos de P. Um *caminho induzido* é um caminho sem cordas. Denotamos por P_k o caminho induzido por k vértices. Dizemos que um grafo é P_k*-free* quando não contém um P_k como subgrafo.

Um passeio $v_1, \ldots, v_k, v_{k+1}$, é denominado *ciclo* quando $v_1, \ldots, v_k$ for um caminho, $k \geq 3$ e $v_1 = v_{k+1}$.

Um grafo G é dito *cíclico* quando G contém um ciclo como subgrafo. Caso contrário, dizemos que G é *acíclico*.

Um conjunto S é *maximal (minimal)* em relação a uma determinada propriedade P se S satisfaz P, e todo conjunto S' que contém propriamente S (que está contido propriamente em S) não satisfaz P.

Um grafo G é *completo* se quaisquer dois vértices distintos de G são adjacentes. Denotamos por K_n o grafo completo com n vértices.

Um conjunto de vértices I de um grafo G é um *conjunto independente* se $G[I]$ é um grafo sem arestas. Definimos por $\alpha(G)$ o tamanho do *conjunto independente máximo*, isto é:

$$\alpha(G) = max\ \{\ |V'|\ /\ V' \subseteq V\ e\ V'\ é\ um\ conjunto\ independente\ de\ G\}.$$

Um conjunto de vértices C de um grafo G é uma *clique* se $G[C]$ é um grafo completo. Denotamos por K_p uma clique de p vértices. Denotamos por $\omega(G)$ o *tamanho da clique máxima*, isto é:

$$\omega(G) = max\ \{\ |V'|\ /\ V' \subseteq V\ e\ V'\ é\ uma\ clique\ de\ G\ \}$$

Um grafo é dito *bipartido* quando seu conjunto de vértices pode ser particionado em dois subconjuntos V_1, V_2, tais que toda aresta de G une um vértice de V_1 a outro de V_2, isto é, podemos particionar $V(G)$ em dois conjuntos independentes. Um grafo é dito *bipartido completo* se é bipartido e possui uma aresta para cada par de vértices v_1, v_2, sendo $v_1 \in V_1$ e $v_2 \in V_2$. Denotamos por $K_{n,m}$ o grafo bipartido completo, onde $|V_1| = n$ e $|V_2| = m$.

Um grafo G é *conexo* se para todo par de vértices distintos v e w de G existe um caminho de v a w. Caso contrário, G é dito *desconexo*. Uma *componente conexa* de G é um subgrafo maximal conexo de G.

Sejam $v, w \in V(G)$. A *distância* entre v e w em G, denotada por $d_G(v, w)$ é o comprimento do menor caminho entre v e w, em G.

Uma das formas de representar um grafo é através de sua matriz de adjacência, por exemplo:

Seja $G = (V, E)$ um grafo com n vértices. A matriz de adjacência para G é um vetor bidimensional $n \times n$, que denotaremos por A, onde $A(i, j)$ é a quantidade de arestas ligando v_i e v_j.

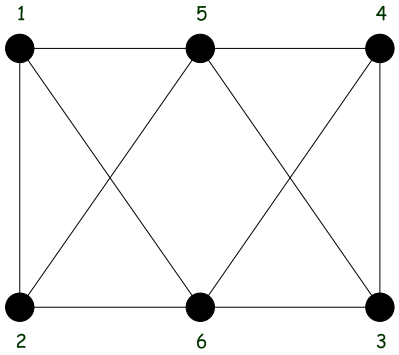


Figura 1.1: Grafo $G = (V, E)$.

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	1	1	0	0
v_2	1	0	0	1	1	1
v_3	1	0	0	1	1	0
v_4	1	1	1	0	1	0
v_5	0	1	1	1	0	1
v_6	0	1	0	0	1	0

Tabela 1.1: Matriz de Adjacência de $G = (V, E)$.

O grau de um vértice v_i em um grafo representado por matriz de adjacência, pode ser obtido pela soma de sua linha ou coluna correspondente.

A matriz de adjacência de um grafo é sempre simétrica.

1.2 Isomorfismo de Grafos

Dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são ditos isomorfos se existe uma função bijetora $f : V_1 \to V_2$ tal que:

- $\forall v, w \in V_1, (v, w) \in E_1 \Leftrightarrow (f(v), f(w)) \in E_2$.

Na Figura 1.2 pode ser visto um exemplo de dois grafos isomorfos, onde é possível encontrar uma função $f : V_1 \rightarrow V_2$, $f = \{f(1,1'),(2,5'),(3,3'),(4,4'),(5,2'),(6,6')\}$ que mantém as adjacências existentes entre os vértices dos grafos.

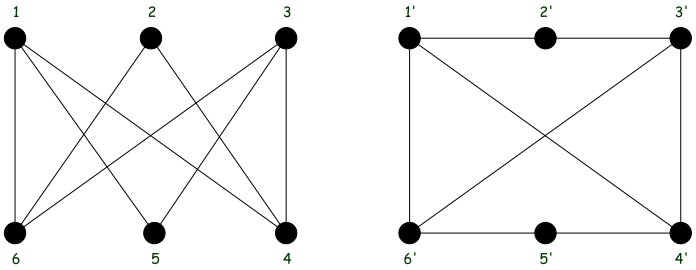


Figura 1.2: Exemplos de grafos isomorfos.

Para garantir o isomorfismo, ao menos as seguintes condições são necessárias:

- Possuir o mesmo número de vértices;
- Possuir o mesmo número de arestas;
- Possuir a mesma sequência de graus dos vértices.

Cada uma destas condições é chamada de *invariante* de um grafo, pois, como o próprio nome sugere, não varia para qualquer grafo isomorfo a ele. Embora necessárias, estas três invariantes não são suficientes para que dois grafos sejam isomorfos. No capítulo 3 abordaremos outras invariantes utilizadas nesse trabalho.

Ainda continua em aberto o conhecimento da relação completa de invariantes de um grafo capaz de caracterizá-lo. Se tal relação fosse verificada, esta resolveria o Problema de Isomorfismo de Grafos que é um problema pertence à classe NP e que não se sabe, porém, se este é polinomialmente solúvel, ou se pertence à família dos problemas de complexidade mais alta desta classe, os NP-completos. É um problema de fundamental importância, seja pelo grande número de aplicações práticas que podem ser modeladas sob este aspecto, seja por se tratar de um problema com características peculiares do ponto de vista da complexidade computacional.

1.3 Complexidade de Algoritmos

Um problema algorítmico π consiste de um conjunto D de todas as possívies entradas para o problema, chamado *conjunto de instâncias*, e de uma *questão* Q sobre estas instâncias. Resolver um problema algorítmico é desenvolver um algoritmo cuja entrada é uma instância do problema e cuja saída é uma resposta à questão do problema.

Um problema é dito de *decisão* quando a questão exige uma resposta do tipo SIM ou NÃO. Como exemplo, seja π o seguinte problema: "Dado um grafo G, reconhecer se G é $(2,1)$". O conjunto de instâncias de π é obviamente o conjunto de todos os grafos. O problema π pode ser assim esquematizado:

Instância genérica de π: um grafo G.

Questão: G é $(2,1)$?

Fica evidente que o problema π acima é um problema de decisão, em particular, um *problema de reconhecimento*. Resolver π significa elaborar um algoritmo de reconhecimento de grafos-$(2,1)$.

Dizemos que um algoritmo é *polinomial* quando sua complexidade de tempo (medida do número de passos que o algoritmo efetua) é uma função polinomial no tamanho da sua entrada. Os problemas de decisão para os quais existem algoritmos polinomiais constituem a classe P. Tais problemas são chamados *polinomiais*.

Um poblema de decisão é *não-determinístico polinomial* quando qualquer instância que produz resposta SIM possui um *certificado* sucinto, isto é, uma comprovação de que a resposta SIM é de fato verificável em tempo polinomail no tamanho da instância. Esta classe de problemas de decisão é a classe *NP*.

A classe *Co-NP* é formada pelos problemas que possuem um certificado sucinto para as instâncias que produzem resposta NÃO.

Sejam $\pi_1(D_1, Q_1)$ e $\pi_2(D_2, Q_2)$ dois problemas de decisão. Uma *tranformação* ou *redução polinomial* de π_1 em π_2 é uma função $f : D_1 \longrightarrow D_2$ tal que as seguintes condições são satisfeitas:

1. f pode ser calculada em tempo polinomial;
2. para toda instância $I \in D_1$, tem-se que I produz resposta SIM para π_1 se e somente se $f(I)$ produz resposta SIM para π_2.

Um problema de decisão π pertence à classe *NP-completo* quando as seguintes condições são satisfeitas:

1. $\pi \in NP$;
2. para todo problema $\pi' \in NP$ existe uma transformação polinomial de π' em π.

Um problema pertencente à classe NP-completo é chamado NP-completo. Para provar que um certo problema π é NP-completo, basta mostrar que $\pi \in NP$ e que existe uma transformação de um problema NP-completo π' em π.

Analogamente, prova-se que um problema de decisão π pertence à classe *Co-NP-completo* (e, neste caso, π é dito Co-NP-completo) quando $\pi \in Co - NP$ e existe um problema π' (Co-)NP-completo tal que:

1. se π' é NP-completo, existe uma função f que pode ser calculada em tempo polinomail tal que para toda instância I' de π', tem-se que I' produz SIM para π' se e somente se $I = f(I')$ produz NÃO para π;
2. se π' é Co-NP-completo, existe uma função f que pode ser calculada em tempo polinomail tal que para toda instância I' de π', tem-se que I' produz NÃO para π' se e somente se $I = f(I')$ produz NÃO para π.

Como fonte de referência para esta seção, indicamos [10, 15].

Capítulo 2

Grafos-(k, l)

O objetivo principal deste capítulo é realizar uma revisão bibliográfica sobre o estado da arte da classe de grafos-(k, l), em particular, o conceito de grafos-$(2, 1)$, para em seguida, abordar o algoritmo de reconhecimento desta classe particular de grafos utilizado nesse trabalho.

Em [2], Brandstädt definiu uma nova classe de grafos: a classe dos grafos-(k, l). Tal classe é uma generalização dos *grafo split* (ou grafos- $(1, 1)$) que são aqueles que podem ter seu conjunto de vértices particionado em um conjunto independente e uma clique [11] e é definida da seguinte forma:

Um grafo G é um grafo-(k, l), ou simplesmente (k, l), se o conjunto de vértices de G pode ser particionado em k conjuntos independentes e l cliques. É importante ressaltar que as cliques desta definição não são necessariamente maximais e, além disso, alguns dos k conjuntos independentes ou das l cliques podem ser vazios.

2.1 Complexidade

O reconhecimento de grafos-(k, l) é em geral NP-completo para $k \geq 3$ ou $l \geq 3$, como foi mostrado por Brandstädt [3], mas os grafos-(k, l) podem ser reconhecidos polinomialmente se restritos à algumas classes de grafos, como por exemplo a classe dos grafos cordais [13], cografos [7], grafos P_4-esparsos [5], dentre outras.

Brandstädt, ainda em [2, 3, 4], apresentou um algoritmo polinomial para reconhecer as classes $(2, 1)$,$(1, 2)$ e $(2, 2)$. Feder *et al.* [8] também apresentaram algoritmos polinomiais para o reconhecimento destas classes que surgiram como sub-produto de algoritmos de partição em subgrafos densos e esparsos.

2.2 Grafos-(2, 1)

Um grafo G é um grafo-$(2,1)$, ou simplesmente $(2,1)$, se pode ter seu conjunto de vértices particionado em dois conjuntos independentes e uma clique, ou equivalentemente, particionado em um grafo bipartido e uma clique. Assim, podemos observar que:

Lema 2.1 *O grafo $G = (V, E)$ é $(2,1)$ se e somente se existe uma clique C que intersecta todo ciclo ímpar de G.*

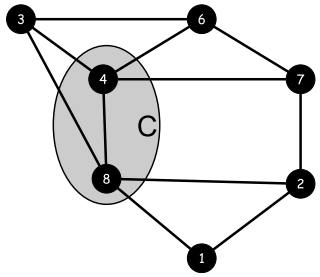


Figura 2.1: Grafo-$(2,1)$.

Prova. Suponhamos que no grafo exista pelo menos um ciclo ímpar, pois, de outro modo, o grafo já seria bipartido.

Vamos supor que em G existe uma clique C que intersecta todo ciclo ímpar. Neste caso, temos um grafo bipartido induzido pelos vértices do grafo que não pertencem a clique.

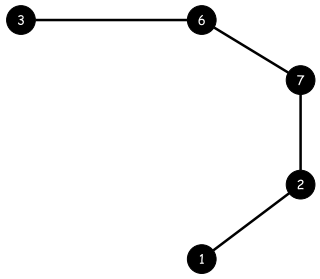


Figura 2.2: grafo-$(2,0)$ induzido.

Logo, $G = (V, E)$ é um grafo-$(2,1)$.

Para mostrar a necessidade do Lema, suponha que $G = (V, E)$ é um grafo- $(2,1)$. Neste caso, $V(G)$ pode ser particionado em um grafo bipartido e uma clique.

É fácil ver que a clique deve intersectar os ciclos ímpares de G, pois, caso contrário, não seria possível obter a bipartição induzida pelos vértices de G que não pertencem à clique. ■

No que segue, apresentamos o algoritmo proposto por Brandstädt para reconhecer grafos-$(2,1)$.

2.3 Algoritmo de Reconhecimento para grafos-$(2,1)$

Em [2], Brandstädt apresentou um algoritmo de reconhecimento dos grafos-$(2,1)$. O príncipio básico utilizado por Brandstädt para reconhecer tais grafos foi a classificação dos vértices de acordo com as seguintes condições de vizinhança:

$N_1 = N(v) \in (1,1)$.

$N_2 = \overline{N}(v) \in (2,0)$.

Podemos observar que quando um grafo é $(2,1)$, ou seja, tem uma $(2,1)$-partição S_1, S_2, C, onde S_1 e S_2 são os conjuntos independentes e C a clique, então, para todo vértice $v \in C$ a não vizinhança de v, denotada por $\overline{N}(v)$, pertence à partição $(2,0)$ (veja Figura 2.3). E, para todo vértice $v \in S_1 \cup S_2$, a vizinhança de $v \in (1,1)$ (veja Figura 3.3).

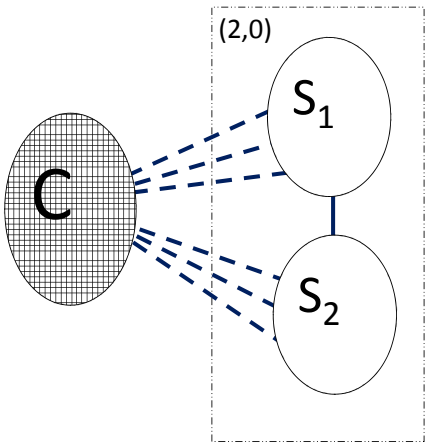


Figura 2.3: $v \in C \rightarrow \overline{N}(v) \in (2,0)$.

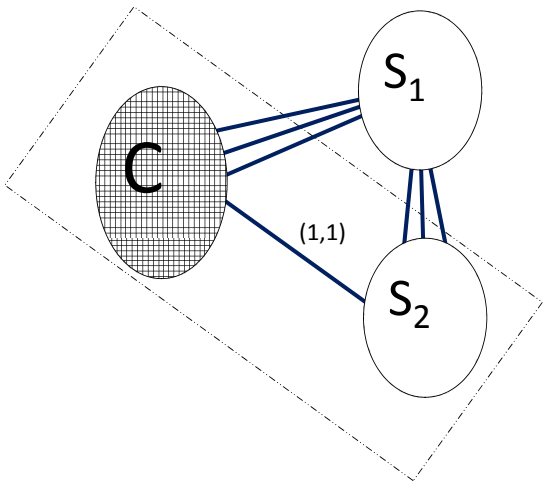


Figura 2.4: $v \in S_1 \cup S_2 \rightarrow N(v) \in (1,1)$.

Para exemplificar, considere como exemplo o grafo da Figura 2.1.

Para todo vértice $v \in C$, $C = \{4, 8\}$ temos:

$\overline{N}(4) = \{1, 2\} \in (2, 0)$.

$\overline{N}(8) = \{6, 7\} \in (2, 0)$.

e para todo vértice $v \in S_1 \cup S_2$, $S_1 \cup S_2 = \{1, 2, 3, 6, 7\}$ temos:

$N(1) = \{2, 8\} \in (1, 1)$.

$N(2) = \{1, 7, 8\} \in (1, 1)$.

$N(3) = \{4, 6, 8\} \in (1, 1)$.

$N(6) = \{3, 4, 7\} \in (1, 1)$.

$N(7) = \{2, 4, 6\} \in (1, 1)$.

Com isso, podemos afirmar que se G tem um vértice v que não atenda às condições N_1 e N_2, ou seja, $N(v) \notin (1, 1)$ e $\overline{N}(v) \notin (2, 0)$ então $G \notin (2, 1)$.

Suponha agora que os vértices de G satisfaçam a pelo menos uma das condições N_1 ou N_2 e G tem uma $(2, 1)$-partição. Se $N(v) \notin (1, 1)$ então v deve pertencer necessariamente a clique C e se $\overline{N}(v) \notin (2, 0)$ então v deve pertencer à bipartição $S_1 \cup S_2$ da $(2, 1)$-partição. Dessa forma conseguimos definir os seguintes conjuntos:

$C^F := \{v : N(v) \notin (1, 1) \text{ e } \overline{N}(v) \in (2, 0)\}$,

$B^F := \{v : N(v) \in (1, 1) \text{ e } \overline{N}(v) \notin (2, 0)\}$.

Neste caso, se C^F não for uma clique ou B^F não for uma bipartição, então G não é (2, 1). No caso contrário, devemos ainda verificar os vértices que satisfazem a N_1 tanto quanto a N_2.

$R := \{v : N(v) \in (1,1) \text{ e } \overline{N}(v) \in (2,0)\}$.

Podemos perceber que se R é um conjunto vazio, então C^F e B^F definem uma $(2,1)$-partição para o grafo G. Mas se existe um vértice $x \in R$, então temos em G uma $(3,1)$-partição definida pela $(1,1)$-partição S_1^x, C^x de $N(x)$ e a $(2,0)$-partição S_2^x,S_3^x de $\overline{N}(x)$. Agora é necessário verificar se podemos reduzir essa $(3,1)$-partição à uma $(2,0)$-partição.

O procedimento $Modify(x)$ será responsável por modificar a $(3,1)$-partição a fim de encontrar uma $(2,1)$-partição, para o caso em que o tamanho da maior clique de R é no mínimo 3, ou seja, um K_3. No caso em que $|R| < 3$, utiliza-se força bruta para verificar se existe uma clique contida em R tal que, $C^F \cup C^R$ é uma clique e $B^F \cup (R \backslash C^R)$ é uma bipartição.

Algorithm 1 Procedimento $Modify(x)$

se $S_1^x \cup S_2^x \cup S_3^x$ é bipartido **então**
    $G \in (2,1)$
senão
    para todo $v \in S_1^x$ **faça**
        verifique se $V \backslash (\{v\} \cup (C^x \cap N(v)))$ é bipartido. ▷ C^x de $N[x]$
    fim para
    se tal vértice v existir **então**
        pare, $G \in (2,1)$
    fim se
fim se
Fim do algoritmo.

Lema 2.2 *Se $\{a,b,c\}$ é um triângulo em R então $G \in (2,1)$ se e somente se para pelo menos um vértice $x \in \{a,b,c\}$ o procedimento $Modify(x)$ nos leva a uma $(2,1)$-partição de G.*

Prova. Temos que mostrar que se $G \in (2,1)$ então o procedimento termina com sucesso. Seja S_1, S_2, C uma $(2,1)$-partição de G e, sem perda de generalidade, seja C uma clique maximal de G, tal que pelo menos um dos vértices do triângulo $\{a,b,c\} \in R$ está em C. Digamos que $a \in C$. Isto implica que $C \subseteq N[a]$. Seja S_1^a, C^a uma $(1,1)$-partição de $N[a]$ e S_2^a, S_3^a uma $(2,0)$-partição de $\overline{N}(a)$. Como $C \subseteq N[a]$ e $|C \cap S_1^a| \leq 1$ sendo C uma clique maximal, temos $C = C^a$ ou $C = \{v\} \cup (C^a \cap N(v))$ para algum $v \in S_1^a$. Todos estes casos são verificados pelo procedimento. ■

Para exemplificar, considere o grafo da Figura 2.5.

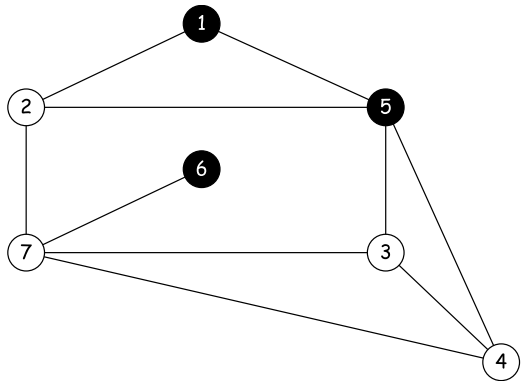


Figura 2.5: Grafo $(2,1)$ com o conjunto $R = [2,3,4,7]$

Seja $K_3 = \{3,4,7\} \in R$ e $C = \{3,4,5\}$ uma clique maximal em G, podemos perceber que os vértices 3 e 4 do K_3 estão em C.

Considerando o vértice 4, temos:

$N[4] = \{3,4,5,7\}$.

$C^4 = \{3,4,5\}$.

$S_1^4 = \{7\}$.

$\overline{N}(4) = \{2,6,1\}$.

$S_2^4 = \{2,6\}$.

$S_3^4 = \{1\}$.

Verificando se $C = C^4$ percebemos que $\{3,4,5\} = \{3,4,5\}$.

Considerando agora o vértice 3, temos:

$N[3] = \{3,4,5,7\}$.

$C^3 = \{3,4,7\}$.

$S_1^3 = \{5\}$.

$\overline{N}(3) = \{2,6,1\}$.

$S_2^3 = \{2,6\}$.

$S_3^3 = \{1\}$.

Ao verificar se $C = C^3$, percebemos que $\{3,4,5\} \neq \{3,4,7\}$.

Analizaremos agora se $C = \{v\} \cup (C^3 \cap N(v))$ para algum $v \in S_1^3$.

Para $v = 5$, temos:

$N(5) = \{1, 2, 3, 4\}$.

$C^3 \cap N(v) = \{3, 4, 7\} \cap \{1, 2, 3, 4\} = \{3, 4\}$.

Podemos afirmar que $C = \{5\} \cup \{3, 4\}$.

Vale mencionar que mantivemos aqui neste trabalho o nome dos procedimentos, conforme apresentados em [3].

No que segue, temos o algoritmo de reconhecimento de grafos-$(2,1)$.

Algorithm 2 Algoritmo de reconhecimento de grafos-$(2,1)$

Entrada: Grafo G
Saída: $G \in (2,1)$ ou $G \notin (2,1)$
Início

para todo $v \in V(G)$ **faça**
    verifique se $N(v) \in (1,1)$ e $\overline{N}(v) \in (2,0)$;
fim para
se existir um vértice $v \in V(G)$ com $N(v) \notin (1,1)$ e $\overline{N}(v) \notin (2,0)$ **então**
    pare, $G \notin (2,1)$;
senão
    $C^F := \{v : N(v) \notin (1,1)$ e $\overline{N}(v) \in (2,0)\}$;
    $B^F := \{v : N(v) \in (1,1)$ e $\overline{N}(v) \notin (2,0)\}$;
    se C^F não for clique ou B^F não for bipartido **então**
        pare, $G \notin (2,1)$;
    senão
        $R := \{v : N(v) \in (1,1)$ e $\overline{N}(v) \in (2,0)\}$;
        se $R = \emptyset$ **então**
            pare, $G \in (2,1)$;
        senão
            se $\omega(R) \geq 3$ **então**
                Escolha um K_3 $\{a,b,c\} \subseteq R$;
                para todo $x \in \{a,b,c\}$ **faça**
                    $Modify(x)$;
                fim para
                se para todo $x \in \{a,b,c\}$ $Modify(x)$ falhar **então**
                    pare, $G \notin (2,1)$;
                senão
                    pare, $G \in (2,1)$;
                fim se
            senão
                para todo clique $C^R \subseteq R, 0 \leq |C^R| \leq 2$ **faça**
                    Verifique se $C := C^F \cup C^R$ é uma clique e
                    se $B := B^F \cup (R \backslash C^R)$ é bipartido;
                fim para
                se tal clique C existir **então**
                    pare, $G \in (2,1)$;
                senão
                    pare, $G \notin (2,1)$;
                fim se
            fim se
        fim se
    fim se
fim se

Fim do algoritmo.

2.3.1 Complexidade

Sabemos que, grafos-$(1,1)$ e grafos-$(2,0)$ podem ser reconhecidos linearmente em tempo $O(n+m)$ [12]. Então, o passo (1) do algoritmo pode ser executado em tempo $O(n(n+m))$ e, os passos (2) a (11), não exigem mais tempo. No passo (12), no máximo $n+m$ vértices e arestas devem ser considerados. O passo (13) é executado em $O(n+m)$ e é realizado com no máximo, $n+m$ vértices e arestas. Ao todo, o tempo de execução do algoritmo é de $O((n+m)^2)$.

A seguir, apresentamos a técnica utilizada nesse trabalho para detecção de obstruções para grafos-$(2,1)$.

Capítulo 3

Metodologia utilizada para obtenção das estruturas proibidas

Neste capítulo apresentamos a metodologia utilizada para encontrar obstruções minimais para grafos-$(2,1)$, a ferramenta desenvolvida para gerar os grafos a partir de matrizes de adjacência e o algoritmo utilizado para o refinamento das obstruções obtidas.

É importante lembrar que o objetivo do nosso trabalho é demonstrar uma metodologia para a obtenção de obstruções-$(2,1)$ baseando-se no princípio utilizado por Brandstädt para o reconhecimento de grafos-$(2,1)$ e não a obtenção de todas as suas estruturas proibidas. Portanto, não foi possível considerar nesse trabalho todas as possibilidades para a obtenção dessas obstruções.

Como descrito no capítulo anterior, Brandstädt definiu um algoritmo para reconhecimento de grafos-$(2,1)$, que recebe como entrada um grafo G e fornece como saída a decisão se este grafo possui ou não uma $(2,1)$-partição. A falha desse algoritmo nos remete que $G \notin (2,1)$, ou seja, G é um grafo proibido para classe de grafos-$(2,1)$.

Seguindo o princípio utilizado por Brandstädt, encontramos as obstruções definindo estruturas que forcem as seguintes falhas do algoritmo:

- Falha 1: Nesse caso, o algoritmo de Brandstädt falha se existir pelo menos um vértice v em G, tal que, $N(v) \notin (1,1)$ e $\overline{N}(v) \notin (2,0)$. Se essa condição for verificada então $G \notin (2,1)$, ou seja, um grafo proibido para os grafos-$(2,1)$.

- Falha 2: Essa falha ocorre quando em $C^F := \{v : N(v) \notin (1,1) \text{ e } \overline{N}(v) \in (2,0)\}$ os vértices candidatos a compor esse conjunto não formam uma $(1,1)$-partição. Se essa falha ocorrer, teremos um grafo proibido para os grafos-$(2,1)$.

- Falha 3: Ocorre quando em $B^F := \{v : N(v) \in (1,1) \text{ e } \overline{N}(v) \notin (2,0)\}$ os vértices candidatos a compor esse conjunto não formam uma $(2,0)$-partição. Quando essa condição é verificada, temos um grafo proibido para os grafos-$(2,1)$.

A seguir, mostramos como as obstruções-$(2,1)$ minimais foram obtidas forçando as falhas do algoritmo descritas acima.

3.1 Obstruções-$(2,1)$ obtidas através da Falha 1 do algoritmo de Brandstädt

Como dito na seção anterior, a Falha 1 do algoritmo de Brandstädt ocorre quando existe pelo menos um vértice v em G, tal que, $N(v) \notin (1,1)$ e $\overline{N}(v) \notin (2,0)$. Observamos que para essa falha ocorrer precisamos de um vértice v em G, tal que, atenda simultaneamente as seguintes condições de vizinhança:

$N_1 = N(v) \notin (1,1)$.

$N_2 = \overline{N}(v) \notin (2,0)$.

Teorema 3.1 *[9] Um grafo G é um grafo split se, e somente se, não contém $2K_2$, C_4 ou C_5 como subgrafo induzido.*

Teorema 3.2 *[1] Um grafo G é um grafo bipartido se, e somente se, não contém ciclo ímpar como subgrafo induzido.*

Para atender às condições N_1 e N_2 definimos uma estrutura trivial em que a vizinhança do vértice v induz um subgrafo $2K_2$ e a não vizinhança induz um subgrafo não bipartido respectivamente.

Observe que para que um grafo seja não bipartido, ele deverá conter pelo menos um ciclo ímpar. Como em grafos gerais podemos ter infinitos clicos ímpares, isso impossibilitou a caracterização completa por subgrafos proibidos através da metodologia utilizada.

Na Figura 3.1 temos o grafo obtido a partir das condições de vizinhança definidas para essa Falha do algoritmo.

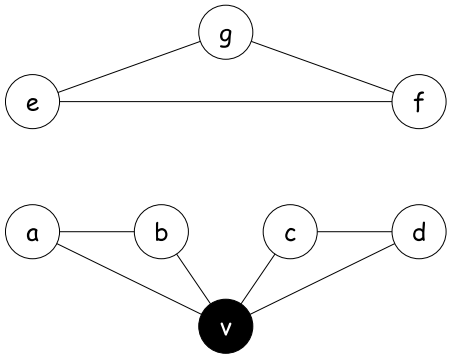


Figura 3.1: Obstrução-$(2,1)$ obtida através das condições de vizinhança definidas para Falha 1 do algoritmo de Brandstädt.

3.1.1 Matriz de adjacência do grafo da Figura 3.1

Nessa seção apresentamos a matriz de adjacência do grafo da Figura 3.1, onde definimos três tipos de arestas, classificadas da seguinte forma:

- Arestas obrigatórias - São as arestas que devem sempre existir para garantir as condições N_1 e N_2. São representadas na matriz de adjacência pelo numeral 1.
- Arestas opcionais - Se existirem ou não é indiferente para preservar as condições N_1 e N_2. Na matriz de adjacência são representadas pelo ponto de interrogação.
- Arestas proibidas - Essas arestas não devem existir pois descaracterizam as condições N_1 e N_2. São representadas na matriz de adjacência pelo numeral 0.

Na tabela abaixo temos a matriz de adjacência do grafo da Figura 3.1 representada com os tipos de arestas supracitados.

	a	b	c	d	e	f	g	v
a	0	1	0	0	?	?	?	1
b		0	0	0	?	?	?	1
c			0	1	?	?	?	1
d				0	?	?	?	1
e					0	1	1	0
f						0	1	0
g							0	0
v								0

Tabela 3.1: Matriz de adjacência do grafo da Figura 3.1.

Para melhorar a identificação dessas arestas no grafo, elaboramos a seguinte convenção.

Representação na Matriz	Descrição	Representação no grafo
1	Aresta obrigatória	———
?	Aresta opcional	. . . . .
0	Aresta proibida	Nenhuma linha

3.1.2 Análise da Matriz de adjacência

A seguir temos a análise da matriz de adjacência do grafo da Figura 3.1 onde classificamos suas arestas de acordo com os tipos acima referenciados.

1. Arestas obrigatórias: $\{ab, av, bv, cd, cv, dv, ef, eg, fg\}$.
2. Arestas proibidas:
   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Cria uma $(2, 0)$-partição induzida por $\overline{N}(v)$: $\{ev, fv, gv\}$.
3. Arestas opcionais: $\{ae, af, ag, be, bf, ag, ce, cf, cg, de, df, dg\}$.

Na Figura 3.2 temos o grafo com as arestas opcionais encontradas na análise.

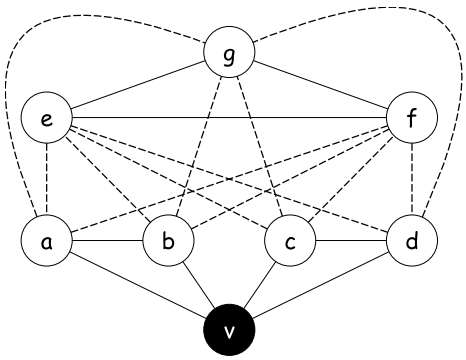


Figura 3.2: Obstrução-$(2, 1)$ obtida através da abordagem da Falha 1 do algoritmo de Brandstädt representado com as arestas opcionais.

3.1.3 Obstruções encontradas

As obstruções obtidas a partir de cada grafo proibido são definidas de acordo com a quantidade de arestas opcionais. Supondo que um grafo proibido tenha apenas uma

aresta opcional podemos gerar no máximo $2^1 = 2$ obstruções (já que desconsideramos isomorfos), ou seja, uma obstrução com a aresta e outra sem.

Como nesse grafo temos 9 (nove) arestas opcionais, geramos $2^9 = 4096$ subgrafos proibidos, alguns deles possivelmente isomorfos entre si. vale mencionar que esses subgrafos proibidos serão analisados a fim de avaliarmos se estes são minimais. Além disso, também iremos desconsiderar grafos isomoformos. Tais análises serão realizadas na seção 3.2.13.

3.2 Obstruções-$(2,1)$ obtidas através da Falha 2 do algoritmo de Brandstädt.

Como dito anteriormente, a Falha 2 do algoritmo de Brandstädt ocorre quando em $C^F := \{v : N(v) \notin (1,1) \text{ e } \overline{N}(v) \in (2,0)\}$ os vértices candidatos a compor esse conjunto não formam uma $(1,1)$-partição. Observamos que para essa falha ocorrer precisamos de pelo menos dois vértices, v e w, não-adjacentes entre si, que atendam simultaneamente às seguintes condições de vizinhança:

$N_1 = N(v) \notin (1,1)$.

$N_2 = \overline{N}(v) \in (2,0)$.

Estes vértices serão selecionados pelo algoritmo para compor o conjunto C^F, porém, por serem não-adjacentes entre si não formam uma clique. Isto implica em $G \notin (2,1)$.

Para atender às condições N_1 e N_2 definimos estruturas em que a vizinhança dos vértices v e w induz subgrafos $2K_2$, C_4 ou C_5 e a não vizinhança induz subgrafos bipartidos.

Observamos que para garantir a condição N_1 entre os vértices v e w podemos ter, a menos de isomorfismo, as seguintes combinações:

$N(v)$	$N(w)$
$2K_2$	$2K_2$
$2K_2$	C_4
$2K_2$	C_5
C_4	C_4
C_4	C_5
C_5	C_5

Consideramos nesse trabalho apenas os seguintes casos:

Estrutura	$N(v)$	$N(w)$
A	$2K_2$	$2K_2$
B	C_4	C_4

No que segue, temos a análise das Estruturas A e B.

3.2.1 Estrutura A

Para garantir a condição $N_1 = N(v) \notin (1,1)$ definimos nesta estrutura que a vizinhança dos vértices v e w induz um subgrafo $2K_2$ e como v e w são não-adjacentes entre si a condição $N_2 = \overline{N}(v) \in (2,0)$ fica preservada.

Na Figura 3.3 temos o grafo obtido a partir das condições de vizinhança definidas para a Estrutura A.

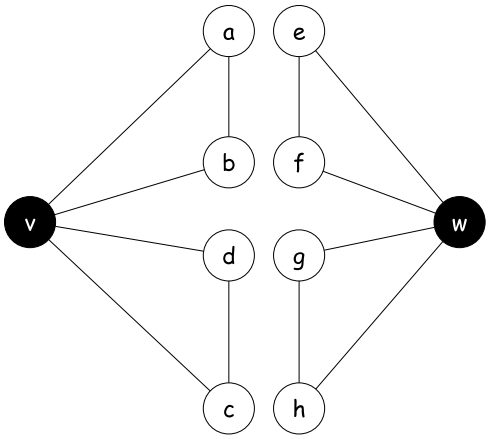


Figura 3.3: Obstrução-$(2,1)$ obtida a partir das condições de vizinhança definidas para a Estrutura A.

Com o objetivo gerar grafos com uma menor quantidade de vértices, observamos que a vizinhança de v e w pode induzir o mesmo $2K_2$. Assim, iremos que gerar subgrafos para a Estrutura A fazendo w adjacente a vizinhança de v conforme a tabela abaixo:

Estrutura	regras de adjacências
A_1	w adjacente a 4 vértices do $2K_2$ induzido por $N(v)$
A_2	w adjacente a 3 vértices do $2K_2$ induzido por $N(v)$
A_3	w adjacente a 2 vértices do $2K_2$ induzido por $N(v)$
A_4	w adjacente a 1 vértice do $2K_2$ induzido por $N(v)$
A_5	w adjacente a 0 vértice do $2K_2$ induzido por $N(v)$

Apresentamos agora a análise de cada uma dessas estruturas.

3.2.2 Estrutura A_1

Nesta estrutura fizemos w ser adjacente aos 4 (quatro) vértices do $2K_2$ induzido pela vizinhança de (v). Como conseqüência, v e w possuem um mesmo $2K_2$ induzido pelos seus vizinhos. Na Figura 3.4 temos o grafo obtido a partir dessas regras de adjacências.

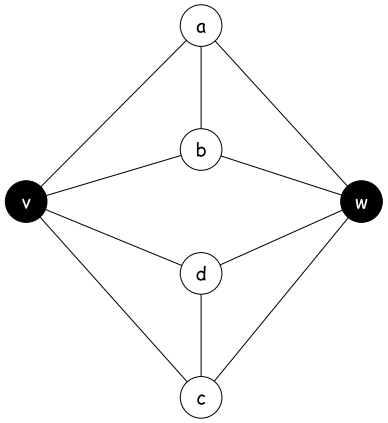


Figura 3.4: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_1.

A seguir apresentamos a matriz de adjacência do grafo da Figura 3.4.

3.2.2.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_1

	a	b	c	d	v	w
a	0	1	0	0	1	1
b		0	0	0	1	1
c			0	1	1	1
d				0	1	1
v					0	0
w						0

Tabela 3.2: Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_1.

3.2.2.2 Análise da Matriz de adjacência

Nesta seção temos a análise da matriz de adjacência do grafo da Figura 3.4 onde classificamos suas arestas de acordo com os tipos já mencionados.

1. Arestas obrigatórias: $\{ab, av, aw, bv, bw, cd, cv, cw, dv, dw\}$.

2. Arestas proibidas:

   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Desfazem o $2K_2$ induzido por $N(w)$: $\{ac, ad, bc, bd\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: Não existem.

3.2.2.3 Obstruções encontradas

Como já mencionado, as obstruções são obtidas de acordo com a quantidade de arestas opcionais e, como nesta estrutura, não temos nenhuma aresta opcional, geramos $2^0 = 1$ subgrafo proibido, ou seja, o próprio grafo obtido a partir das regras de adjacências definidas para a Estrutura A_1.

Em seguida, temos a análise da Estrutura A_2, bem como as obstruções obtidas após essa análise.

3.2.3 Estrutura A_2

Nesta estrutura temos w adjacente a apenas 3 (três) dos 4 (quatro) vértices do $2K_2$ induzido pela vizinhança de (v). Com isso houve a necessidade de adicionarmos mais 1 (um) vértice à Estrutura A_1. Abaixo temos o grafo obtido a partir dessas regras de adjacências.

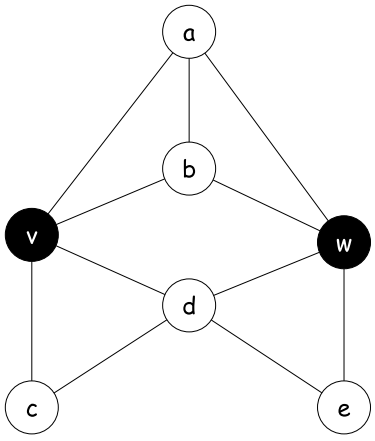


Figura 3.5: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_2.

A seguir temos a matriz de adjacência do grafo da Figura 3.5.

3.2.3.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_2

	a	b	c	d	e	v	w
a	0	1	0	0	0	1	1
b		0	0	0	0	1	1
c			0	1	?	1	0
d				0	1	1	1
e					0	0	1
v						0	0
w							0

Tabela 3.3: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura A_2.

Análise da Matriz de adjacência

Abaixo apresentamos a análise da matriz de adjacência do grafo da Figura 3.5 com a classificação de suas arestas de acordo com os tipos supracitados.

1. Arestas obrigatórias: $\{ab, av, aw, bv, bw, cd, cv, de, dv, dw, ew\}$.
2. Arestas proibidas:
   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Desfazem o $2K_2$ induzido por $N(w)$: $\{bd, be, ad, ae\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.
3. Arestas opcionais: $\{ce\}$.

Na Figura 3.6 temos o grafo com a aresta opcional encontrada na análise.

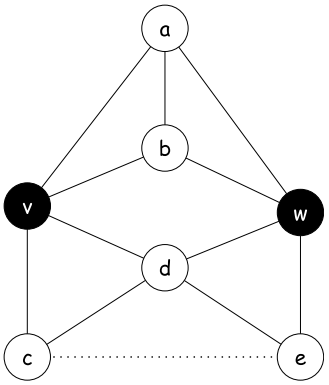


Figura 3.6: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_2 representado com as arestas opcionais.

3.2.3.2 Obstruções encontradas

No resultado da análise da matriz de adjacência do grafo obtido a partir dessa estrutura encontramos 1 (uma) aresta opcional que nos gerou $2^1 = 2$ subgrafos proibidos.

No que segue, apresentamos a análise da Estrutura A_3, assim como as obstruções obtidas a partir dessa análise.

3.2.4 Estrutura A_3

Nesta estrutura temos w adjacente a apenas 2 (dois) dos 4 (quatro) vértices do $2K_2$ induzido pela vizinhança de (v). Para isso tivemos que adicionar mais 2 (dois) vértices à Estrutura A_1. Na Figura 3.7 temos o grafo obtido a partir dessas regras de adjacências.

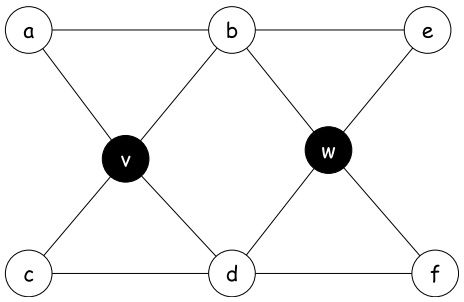


Figura 3.7: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_3.

A seguir temos a matriz de adjacência do grafo da Figura 3.7.

3.2.4.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_3

	a	b	c	d	e	f	v	w
a	0	1	0	0	?	?	1	0
b		0	0	0	1	0	1	1
c			0	1	?	?	1	0
d				0	0	1	1	1
e					0	0	0	1
f						0	0	1
v							0	0
w								0

Tabela 3.4: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura A_3.

3.2.4.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.7 com a classificação de suas arestas de acordo com os tipos supracitados.

1. Arestas obrigatórias: $\{ab, be, bv, bw, cd, cv, df, dv, dw, ew, fw\}$.
2. Arestas proibidas:
   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Desfazem o $2K_2$ induzido por $N(w)$: $\{bd, bf, ed, ef\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.
3. Arestas opcionais: $\{ae, af, ce, cf\}$.

Na Figura 3.8 temos o grafo com as arestas opcionais encontradas na análise.

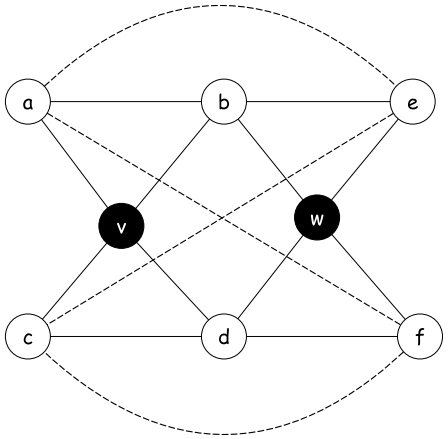


Figura 3.8: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_3 representado com as arestas opcionais.

3.2.4.3 Obstruções encontradas

O resultado dessa análise nos mostrou 4 (quatro) arestas opcionais que nos gerou $2^4 = 16$ subgrafos proibidos.

Apresentamos em seguida, a análise da Estrutura A_4 e as obstruções obtidas após essa análise.

3.2.5 Estrutura A_4

Na Estrutura A_4 temos w adjacente a apenas 1 (um) dos 4 (quatro) vértices do $2K_2$ induzido pela vizinhança de (v). Como conseqüência, adicionamos mais 3 (três) vértices à Estrutura A_1. Na Figura 3.9 temos o grafo obtido a partir dessas regras de adjacências.

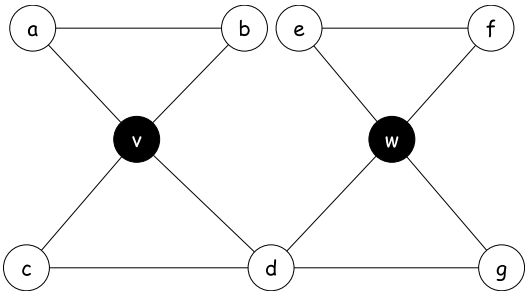


Figura 3.9: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_4.

Observando a estrutura desse grafo, verificamos que ele não atende à condição de vizinhança $N_2 = \overline{N}(v) \in (2,0)$ necessária para a Falha 2 do algoritmo de Brandstädt, pois os vértices w, e, f formam um K_3 em $\overline{N}(v)$, portanto, ciclo ímpar, o mesmo ocorre com a não vizinhança de w.

Para solucionar este problema, poderíamos adicionar as arestas ve ou vf, e as arestas wa ou wb. Entretanto observamos que as arestas ve e vf, assim como wa e wb não podem ocorrer simultaneamente, pois teríamos w adjacente a 3 vértices do $2K_2$ induzido por $N(v)$, situação que já foi abordada na seção 3.2.3. Portanto, para este caso, consideramos apenas as arestas ve e wa para a geração dos demais subgrafos proibidos.

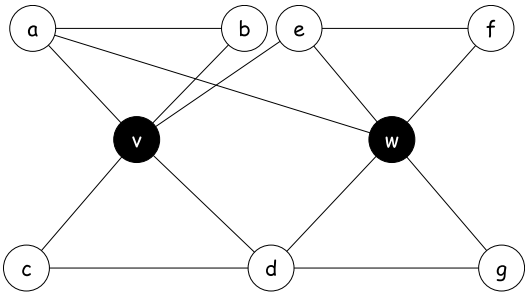


Figura 3.10: Nova Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_4.

A seguir, temos a matriz de adjacência do grafo da Figura 3.10.

3.2.5.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_4

	a	b	c	d	e	f	g	v	w
a	0	1	0	0	0	0	?	1	1
b		0	0	0	0	?	?	1	0
c			0	1	?	?	?	1	0
d				0	0	0	1	1	1
e					0	1	0	1	1
f						0	0	0	1
g							0	0	1
v								0	0
w									0

Tabela 3.5: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura A_4.

3.2.5.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.10 com a classificação de suas arestas de acordo com os descritos anteriormente.

1. Arestas obrigatórias: $\{ab, av, aw, bv, cd, cv, dg, dv, dw, ef, ev, ew, fw, gw\}$.

2. Arestas proibidas:

   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Desfazem o $2K_2$ induzido por $N(w)$: $\{ed, eg, fg, fd\}$.
   - Levam à situação da seção 3.2.4 já abordada, onde w é adjacente a 2 (dois) dos 4 (quatro) vértices do $2K_2$ induzido por $N(v)$: $\{ae, af, be, cw\}$.
   - Levam à situação da seção 3.2.3 já abordada, onde w é adjacente a 3(três) dos 4 (quatro) vértices do $2K_2$ induzido por $N(v)$: $\{fv, bw, gv\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: $\{ag, bf, bg, ce, ce, cg\}$.

Na Figura 3.11 temos o grafo com as arestas opcionais encontradas na análise.

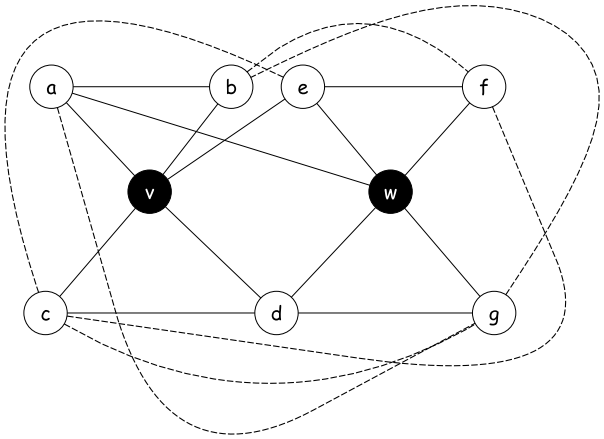


Figura 3.11: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_4 representado com as arestas opcionais.

3.2.5.3 Obstruções encontradas

O resultado dessa análise nos mostrou 6 (seis) arestas opcionais que nos gerou $2^6 = 64$ subgrafos proibidos.

Apresentamos a seguir a análise da Estrutura A_5, bem como as obstruções obtidas com essa análise.

3.2.6 Estrutura A_5

Nesta estrutura, consideramos o caso em que as vizinhanças tanto de (v) quanto de (w) induzem um $2K_2$, sendo estes disjuntos e totalmente isolados(i.e, sem arestas entre eles). Como conseqüência, obtivemos uma estrutura com 10 (dez) vértices. A seguir, temos o grafo obtido a partir dessas regras de adjacências.

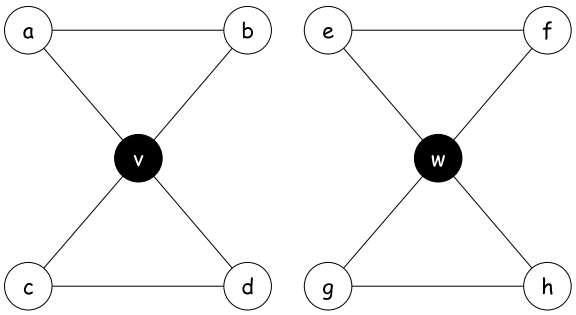


Figura 3.12: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_5.

Estudando a estrutura desse grafo observamos que como na seção anterior essa estrutura não atende à condição de vizinhança $N_2 = \overline{N}(v) \in (2,0)$ necessária para a Falha 2 do algoritmo de Brandstädt, pois os vértices $\{w, e, f\}$ ou $\{w, g, h\}$ formam um ciclo ímpar K_3 em $\overline{N}(v)$, o mesmo ocorre com a não vizinhança de w.

Solucionamos este problema da mesma forma utilizada na Estrutura A_4, adicionando arestas que não nos levam a situações já abordadas. Para isso, foram acrescentadas as arestas ve, vg, wb e wd para a geração dos demais subgrafos proibidos.

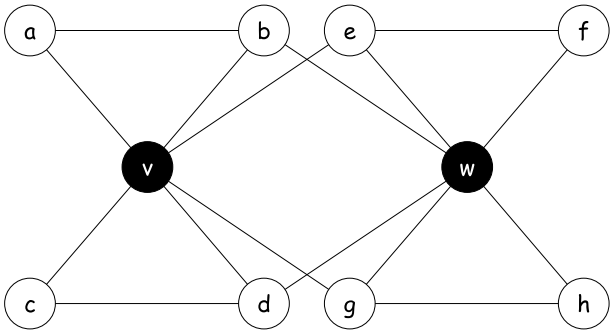


Figura 3.13: Nova Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_5.

A seguir temos a matriz de adjacência do grafo da Figura 3.13.

3.2.6.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura A_5

	a	b	c	d	e	f	g	h	v	w
a	0	1	0	0	0	?	0	?	1	0
b		0	0	0	0	0	0	0	1	1
c			0	1	0	?	0	?	1	0
d				0	0	0	0	0	1	1
e					0	1	0	0	1	1
f						0	0	0	0	1
g							0	1	1	1
h								0	0	1
v									0	0
w										0

Tabela 3.6: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura A_5.

3.2.6.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.13 com a classificação de suas arestas de acordo com os descritos na seção 3.1.2.

1. Arestas obrigatórias: $\{ab, av, bv, cd, cv, dv, dw, ef, ev, ew, fw, gh, gv, gw, hw\}$.

2. Arestas proibidas:

   - Desfazem o $2K_2$ induzido por $N(v)$: $\{ac, ad, bc, bd\}$.
   - Desfazem o $2K_2$ induzido por $N(w)$: $\{eg, eh, fg, fh\}$.
   - Levam à situação da seção 3.2.4 já abordada, onde w é adjacente a 2 (dois) dos 4 (quatro) vértices do $2K_2$ induzido por $N(v)$: $\{vf, vh, wa, wc\}$.
   - Levam à situação da seção 3.2.5 já abordada, onde w é adjacente a 1 (um) dos 4 (quatro) vértices do $2K_2$ induzido por $N(v)$: $\{ae, ag, be, bf, bg, cg, dg, dh, eg\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: $\{af, ah, cf, ch\}$.

Na Figura 3.14 temos o grafo com as arestas opcionais encontradas na análise.

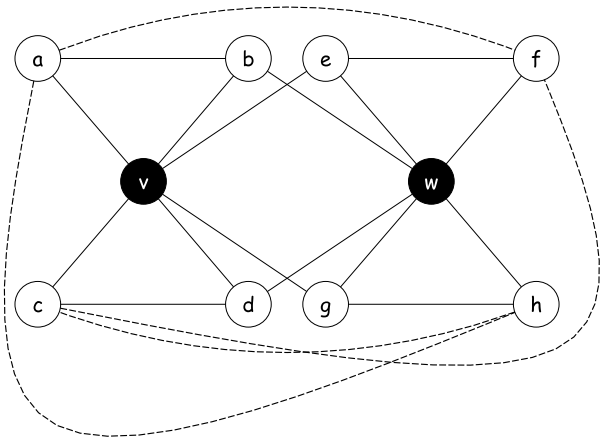


Figura 3.14: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura A_5 representado com as arestas opcionais.

3.2.6.3 Obstruções encontradas

Como resultado dessa análise obtivemos 4 (quatro) arestas opcionais que nos gerou $2^4 = 16$ subgrafos proibidos.

No que segue, apresentamos a análise da Estrutura B e suas variantes.

3.2.7 Estrutura B

Como dito anteriormente, para atender às condições $N_1 = N(v) \notin (1,1)$ e $N_2 = \overline{N}(v) \in (2,0)$ que força a Falha 2 do algoritmo de Brandstädt, definimos estruturas em que a vizinhança dos vértices v e w induz subgrafos $2K_2$, C_4 ou C_5 e a não vizinhança induz subgrafos bipartidos. Para garantir a condição N_1 definimos nesta estrutura que a vizinhança dos vértices v e w induz um subgrafo C_4 e como v e w são não-adjacentes entre si continuamos preservando a condição $N_2 = \overline{N}(v) \in (2,0)$. Na Figura 3.15 temos o grafo obtido a partir das condições de vizinhança definidas para a Estrutura B.

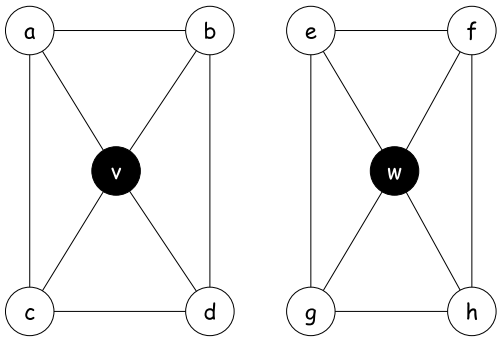


Figura 3.15: Obstrução-$(2,1)$ obtida a partir das condições de vizinhança definidas para a Estrutura B.

Assim como na Estrutura A com relação ao $2K_2$, fizemos também com que a vizinhança de v e w induzem o mesmo C_4. Dessa forma, iremos gerar subgrafos para a Estrutura B fazendo w adjacente a vizinhança de v conforme a tabela abaixo:

Estrutura	regras de adjacências
B_1	w adjacente a 4 vértices do C_4 induzido por $N(v)$
B_2	w adjacente a 3 vértices do C_4 induzido por $N(v)$
B_3	w adjacente a 2 vértices do C_4 induzido por $N(v)$
B_4	w adjacente a 1 vértice do C_4 induzido por $N(v)$
B_5	w adjacente a 0 vértice do C_4 induzido por $N(v)$

A seguir, temos a análise de cada uma dessas estruturas.

3.2.8 Estrutura B_1

A Estrutura B_1 tem w adjacente aos 4 (quatro) vértices do C_4 induzido pela vizinhança de (v). Com isso v e w têm o mesmo C_4 induzido pelas suas vizinhanças. O grafo obtido a partir dessas regras de adjacências é apresentado na Figura 3.16.

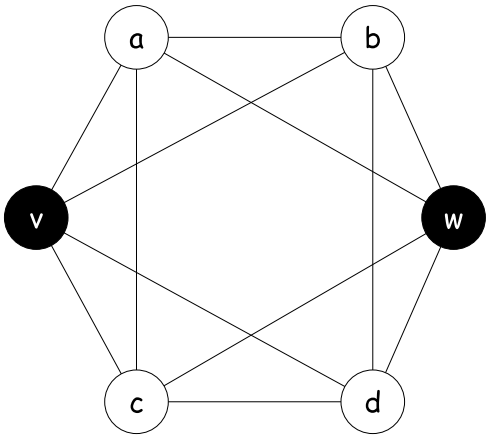


Figura 3.16: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_1.

A seguir temos a matriz de adjacência do grafo da Figura 3.16.

	a	b	c	d	v	w
a	0	1	1	0	1	1
b		0	0	1	1	1
c			0	1	1	1
d				0	1	1
v					0	0
w						0

Tabela 3.7: Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura B_1.

3.2.8.1 Análise da Matriz de adjacência

Abaixo temos a análise da matriz de adjacência do grafo da Figura 3.16, onde classificamos suas arestas de acordo com os tipos mencionados na seção 3.1.2.

1. Arestas obrigatórias: $\{ab, ac, av, aw, bd, bv, bw, cd, cv, cw, dv, dw\}$.
2. Arestas proibidas:
   - Desfazem o C_4 induzido por $N(v)$ ou $N(w)$: $\{ad, bc\}$.
   - Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.
3. Arestas opcionais: Não existem.

3.2.8.2 Obstruções encontradas

Como nesta estrutura também não temos nenhuma aresta opcional, geramos $2^0 = 1$ subgrafo proibido, ou seja, o próprio grafo obtido a partir das regras de adjacências definidas para essa Estrutura.

No que segue, apresentamos a análise da Estrutura B_2, assim como as obstruções obtidas a partir dessa análise.

3.2.9 Estrutura B_2

Nesta estrutura fizemos w ser adjacente aos 3 (trê) vértices do C_4 induzido pela vizinhança de (v). O grafo obtido é mostrado na Figura 3.17.

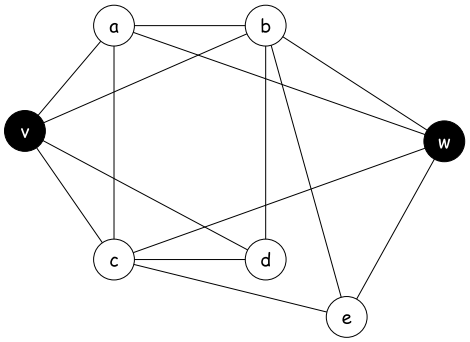


Figura 3.17: Obstrução-(2, 1) obtida a partir das regras de adjacências definidas para a Estrutura B_2.

A seguir apresentamos a matriz de adjacência do grafo da Figura 3.17.

3.2.9.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura B_2

	a	b	c	d	e	v	w
a	0	1	1	0	0	1	1
b		0	0	1	1	1	1
c			0	1	1	1	1
d				0	?	1	0
e					0	0	1
v						0	0
w							0

Tabela 3.8: Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura B_2.

3.2.9.2 Análise da Matriz de adjacência

Nesta seção temos a análise da matriz de adjacência do grafo da Figura 3.17 onde classificamos suas arestas de acordo com os tipos mencionados anteriormente.

1. Arestas obrigatórias: $\{ab, ac, av, aw, bd, be, bv, bw, cd, ce, cv, cw, dv, dw, ew\}$.

2. Arestas proibidas:

   - Desfazem o C_4 induzido por $N(v) : \{ad, bc\}$.

- Desfazem o C_4 induzido por $N(w) : \{ae\}$.
- Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: $\{de\}$

Na Figura 3.18 temos o grafo com a aresta opcional encontrada na análise.

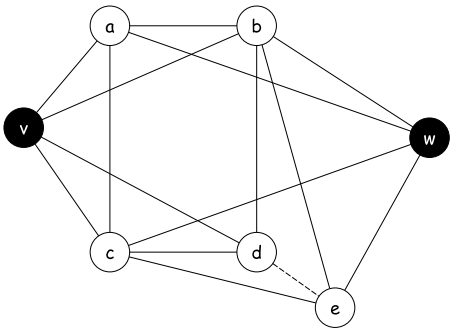


Figura 3.18: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_2 representado com as arestas opcionais.

3.2.9.3 Obstruções encontradas

No resultado da análise da matriz de adjacência do grafo obtido a partir dessa estrutura, encontramos 1 (uma) aresta opcional que nos gerou $2^1 = 2$ subgrafos proibidos.

A seguir, temos a análise da Estrutura B_3.

3.2.10 Estrutura B_3

Nesta estrutura temos w adjacente a apenas 2 (dois) dos 4 (quatro) vértices do C_4 induzido pela vizinhança de (v). Para isso tivemos que adicionar mais 2 (dois) vértices à Estrutura B_1. Na figura a seguir temos o grafo obtido a partir dessas regras de adjacências.

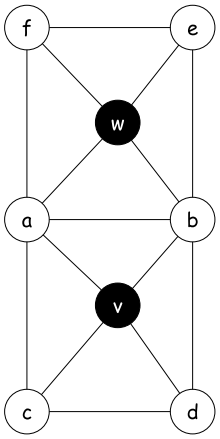


Figura 3.19: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_3.

A seguir temos a matriz de adjacência do grafo da Figura 3.19.

3.2.10.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura B_3

	a	b	c	d	e	f	v	w
a	0	1	1	0	0	1	1	1
b		0	0	1	1	0	1	1
c			0	1	?	?	1	1
d				0	?	?	1	0
e					0	1	0	1
f						0	0	1
v							0	0
w								0

Tabela 3.9: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura B_3.

3.2.10.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.19 com a classificação de suas arestas de acordo com os tipos supracitados.

1. Arestas obrigatórias: $\{ab, ac, af, av, aw, bd, be, bv, bw, cd, cv, cw, dv, dw, ew, fw\}$.

2. Arestas proibidas:

   - Desfazem o C_4 induzido por $N(v) : \{ad, bc\}$.
   - Desfazem o C_4 induzido por $N(w) : \{ae, bf\}$.

- Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: $\{ce, cf, de, df\}$.

Na Figura 3.20 temos o grafo com as arestas opcionais encontradas na análise.

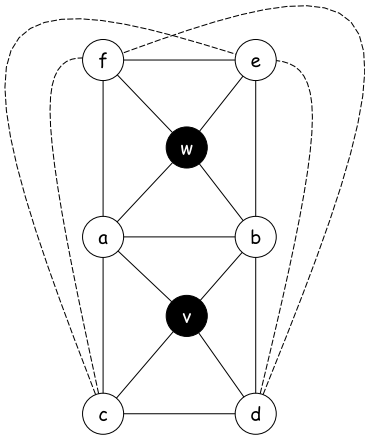


Figura 3.20: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_3 representado com as arestas opcionais.

3.2.10.3 Obstruções encontradas

O resultado dessa análise nos mostrou 4 (quatro) arestas opcionais que nos gerou $2^4 = 16$ subgrafos proibidos.

No que segue, apresentamos a análise da Estrutura B_4, assim como as obstruções obtidas a partir dessa análise.

3.2.11 Estrutura B_4

Nesta estrutura temos w adjacente a apenas 1 (um) dos 4 (quatro) vértices do C_4 induzido pela vizinhança de (v). Na Figura 3.21 temos o grafo definido a partir dessas regras de adjacências.

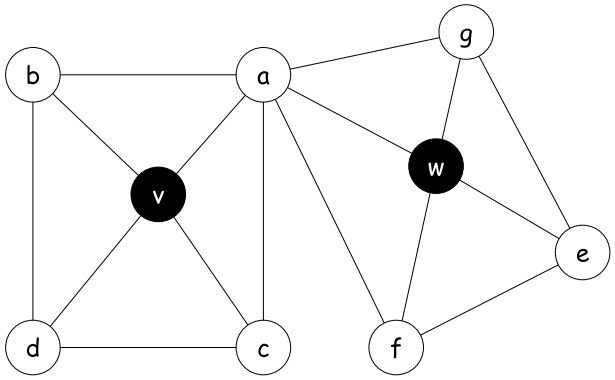


Figura 3.21: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_4.

A seguir temos a matriz de adjacência do grafo da Figura 3.21.

3.2.11.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas para a Estrutura B_4

	a	b	c	d	e	f	g	v	w
a	0	1	1	0	0	1	1	1	1
b		0	0	1	?	?	?	1	0
c			0	1	?	?	?	1	0
d				0	?	?	?	1	0
e					0	1	1	0	1
f						0	0	0	1
g							0	0	1
v								0	0
w									0

Tabela 3.10: Matriz de adjacência do grafo gerado a partir das regras de adjacências definidas para a Estrutura B_4.

3.2.11.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.21 com a classificação de suas arestas de acordo com os tipos supracitados.

1. Arestas obrigatórias: $\{ab, ac, af, ag, av, aw, bd, bv, cd, cv, dv, ef, eg, ew, fw, gw\}$.

2. Arestas proibidas:

   - Desfazem o C_4 induzido por $N(v)$: $\{ad, bc\}$.

- Desfazem o C_4 induzido por $N(w) : \{ae, gf\}$.
- Desfazem a condição de v e w não-adjacentes entre si: $\{vw\}$.

3. Arestas opcionais: $\{be, bf, bg, ce, cf, cg, de, df, dg\}$.

Na Figura 3.22 temos o grafo com as arestas opcionais encontradas na análise.

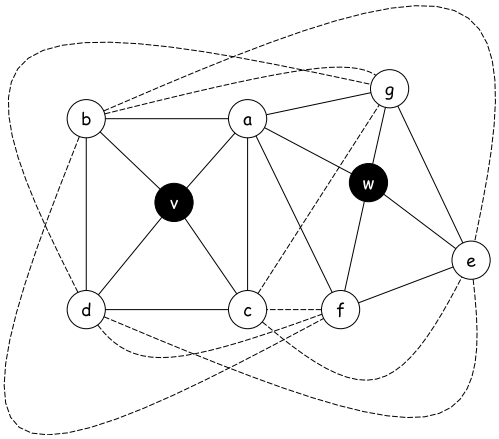


Figura 3.22: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas para a Estrutura B_4 representado com as arestas opcionais.

3.2.11.3 Obstruções encontradas

O resultado dessa análise nos mostrou 9 (nove) arestas opcionais que nos gerou $2^9 = 512$ subgrafos proibidos.

Na sequência, temos as obstruções obtidas através da Falha 3 do algoritmo de Brandstädt.

3.2.12 Obstruções-$(2,1)$ obtidas através da Falha 3 do algoritmo de Brandstädt.

Como descrito no início desse capítulo, a Falha 3 do algoritmo de Brandstädt ocorre quando em $B^F := \{v : N(v) \in (1,1)$ e $\overline{N}(v) \notin (2,0)\}$ os vértices candidatos a compor esse conjunto não formam uma $(2,0)$-partição. Observamos que, para essa falha ocorrer precisamos de pelo menos três vértices (v, w e v) adjacentes entre si, que atendam simultaneamente às seguintes condições de vizinhança:

$N_1 = N(v) \in (1,1)$.

$N_2 = \overline{N}(v) \notin (2,0)$.

Estes vértices serão selecionados pelo algoritmo para compor o conjunto B^F e, por não formarem um grafo bipartido, a resposta do algoritmo será que $G \notin (2,1)$.

Para atender às condições N_1 e N_2, definimos estruturas em que a vizinhança dos vértices (v, w e v) induz um subgrafo *split* e a não vizinhança induz um subgrafo não bipartido.

Na Figura 3.23 apresentamos o grafo obtido a partir das condições N_1 e N_2 supracitadas.

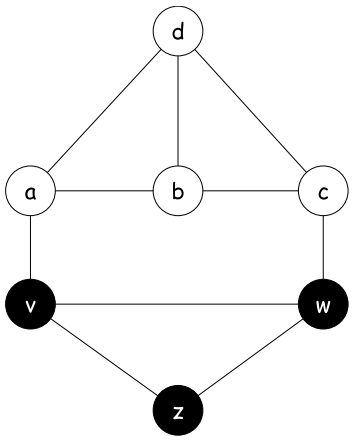


Figura 3.23: Grafo obtido a partir das regras de adjacências definidas pela Falha 3 do algoritmo de Brandstädt.

A seguir apresentamos a matriz de adjacência do grafo da Figura 3.23.

3.2.12.1 Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas pela Falha 3 do algoritmo de Brandstädt

	a	b	c	d	v	w	z
a	0	1	?	1	1	0	?
b		0	1	1	0	0	0
c			0	1	0	1	?
d				0	0	0	0
v					0	1	1
w						0	1
z							0

Tabela 3.11: Matriz de adjacência do grafo obtido a partir das regras de adjacências definidas pela Falha 3 do algoritmo de Brandstädt.

3.2.12.2 Análise da Matriz de adjacência

Apresentamos agora a análise da matriz de adjacência do grafo da Figura 3.23 onde classificamos suas arestas de acordo com os tipos já mencionados.

1. Arestas obrigatórias: $\{ab, ad, av, bc, bd, cd, cw, vw, vz, wz\}$.
2. Arestas proibidas:
   - Desfazem a condição $\overline{N}(v) \notin (2,0)$: $\{aw, bv, bw, bz, cv, cw, cz\}$.
3. Arestas opcionais: $\{ac, az, cz\}$.

Na Figura 3.24 temos o grafo com as arestas opcionais encontradas na análise.

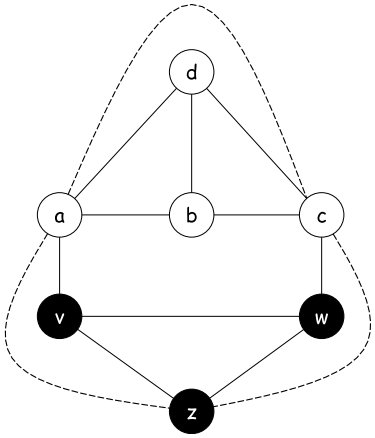


Figura 3.24: Obstrução-$(2,1)$ obtida a partir das regras de adjacências definidas pela Falha 3 do algoritmo de Brandstädt representado com as arestas opcionais.

3.2.12.3 Obstruções encontradas

Nesta estrutura encontramos 3 (três) arestas opcionais no qual geramos $2^3 = 8$ subgrafos proibidos.

3.2.13 Implementação para o refinamento das obstruções

As matrizes de adjacência de todas as estruturas descritas na sessão anterior foram salvas em arquivos do tipo texto para servirem de entrada para a implementação criada com o objetivo de automatizar o trabalho de refinamento das obstruções.

Desenvolvemos esse programa utilizando a Linguagem Java e implementamos as seguintes funcionalidades:

- Lê todos os arquivos contendo as matrizes de adjacência e cria um grafo para cada matriz.
- Encontra a versão minimal de todos os grafos gerados.
- Utilizando invariantes de isomorfismo, realiza o refinamento das obstruções deixando apenas um grafo de cada grupo de isomorfos.

Abaixo descrevemos o layout do arquivo que contêm as matrizes de adjacência das arestas opcionais do grafo da Figura 3.7.

```
0     1     0     0     ?     ?     1     0
0     0     0     0     1     0     1     1
0     0     0     1     ?     ?     1     0
0     0     0     0     0     1     1     1
0     0     0     0     0     0     0     1
0     0     0     0     0     0     0     1
0     0     0     0     0     0     0     0
0     0     0     0     0     0     0     0

0 0
0 0

0 0
0 1

0 0
1 0

...

1 1
1 0

1 1
1 1
```

Figura 3.25: Aquivo texto contendo matrizes de adjacência.

A primeira parte do arquivo descreve a matriz de adjacência do grafo. Em seguida, temos as combinações de arestas opcionais de acordo com a análise realizada. Observe que nesse exemplo temos quatro pontos de interrogação na matriz principal, que representam as arestas opcionais, ou seja, podem ou não existir, nesse caso, podemos gerar $2^4 = 16$ grafos. O programa substitui os quatro pontos de interrogação pelos valores de cada uma das 16 matrizes. Nesse exemplo, teremos ao final 16 grafos gerados a partir desse aquivo.

Na Figura 3.26 apresentamos o diagrama de classe da implementação com o nome das classes e alguns de seus atributos e métodos.

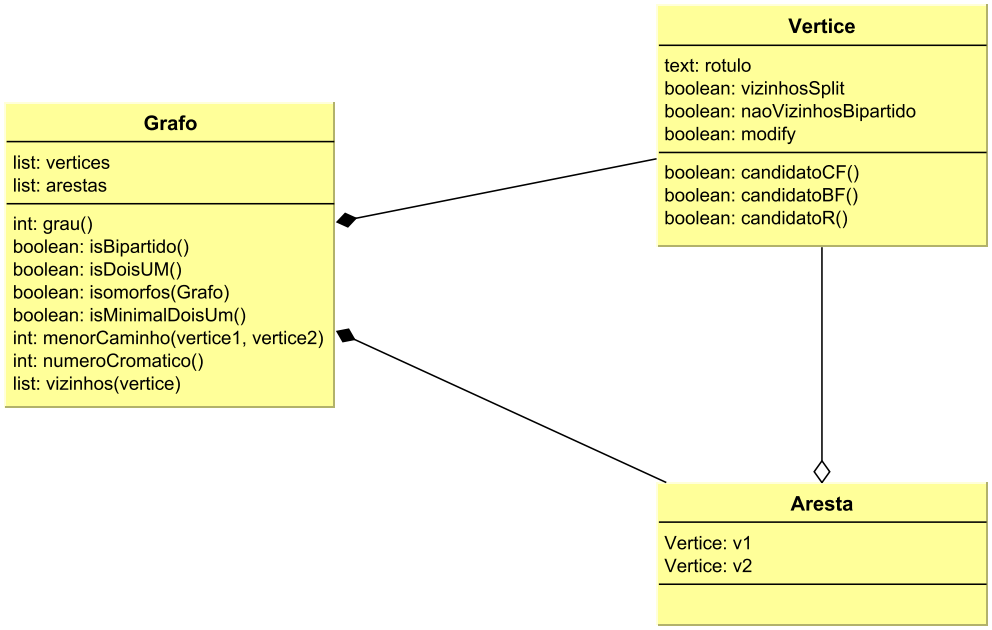


Figura 3.26: Digrama de Classe do Projeto.

3.2.14 Método para encontrar os minimais

Esse método é responsável por "extrair"o minimal de cada grafo gerado pelo programa. O algoritmo monta uma lista de vértices candidatos a serem removidos, ou seja, aqueles vértices que mesmo sendo removidos continuamos com $G \notin (2,1)$. Em seguida, é feita uma iteração nessa lista removendo cada vértice e recursivamente executando novamente o procedimento em $G[V \backslash v]$. O ponto de parada é quando a lista de candidatos não for mais preenchida, nesse caso, todo vértice removido torna $G \in (2,1)$. A seguir os algoritmos utilizados pelo método.

Algorithm 3 Método Minimal(G)

para todo $v \in V$ **faça**
  $G' \leftarrow G((V \backslash v), E)$
  se $G' \notin (2,1)$ **então** ▷ $(2,1)$ por Brandstädt.
    $listaCandidatos[i] \leftarrow v$
    $i \leftarrow i + 1$
  fim se
fim para
para todo $v \in listaCandidatos$ **faça**
  $G' \leftarrow G((V \backslash v), E)$
  $minimal(G')$
fim para
Retorne G'
Fim do algoritmo.

Algorithm 4 Método EncontraMinimal

1: **para todo** $G \in listaGrafos$ **faça**
2: $G \leftarrow minimal(G)$ ▷ $listaGrafos$ = todos os grafos gerados
3: **fim para**
Fim do algoritmo.

3.2.15 Refinamento das obstruções

O principio básico desse processo foi a utilização de algumas invariantes de isomorfismo para a classificação dos grafos em grupos definidos pelos valores dessas invariantes. A seguir listamos as invariantes utilizadas e como esses grupos foram criados.

Invariantes

1. Quantidade de vértices
   Dois grafos G e H são isomorfos se possuírem a mesma quantidade de vértices, pois deve existir uma correspondência um a um entre os vértices.

2. Quantidade de arestas
   Dois grafos simples G e H são isomorfos se possuírem a mesma quantidade de arestas.

3. Seqüência ordenada de graus
   O isomorfismo preserva as adjacências entre os vértices, conseqüentemente se G e H são isomorfos devem possuir a mesma seqüência de graus.

4. Quantidade de cliques maximais

   Como o isomorfismo preserva as características estruturais, dois grafos isomorfos devem possuir a mesma quantidade de cliques maximais.

5. Tamanho da clique maximal

   O tamanho da clique maximal é igual para dois grafos isomorfos.

6. Número cromático

   Denomina-se número cromático de um grafo G o menor número de cores k, para o qual existe uma k-coloração de G, ou seja, o número cromático de um grafo representa o menor número de cores necessárias para colorir os vértices de um grafo sem que vértices adjacentes tenham a mesma cor. O número cromático é o mesmo para dois grafos isomorfos.

7. Distância Múltipla

   Em [14] Remie mostra que somente a seqüência ordenada de graus não é suficiente para garantir o isomorfismo entre dois grafos e definiu uma nova invariante chamada de Distância Múltipla.

   Seja G um grafo conexo com n vértices. Define-se distância entre dois vértices de um grafo como o caminho mais curto de um vértice a outro. A distância máxima é chamada de diâmetro. Seja $g = [[g(1,1),...,g(1,k_1)],...,[g(n,1),...,g(n,k_n)]]$ onde $g(i,j)$ é a quantidade de vértices a uma distância j do vértice i em G. Define-se o conjunto $dmp(G,j) = \{\{g(_{i,j})|i \in \{1,...,n\}\}\}$ e d como o diâmetro de G. A função $f_i : G \to dmp(G,j)$ é invariante para todo $j \in \{1,...,d\}$. Um exemplo é apresentado a seguir.

   Seja:

   Calculando o conjunto dmp de G para as distâncias $\{1,\ 2,\ 3\}$ entre os vértices, temos:

	1	2	3	4	5	6	$dmp(G,1)$	$dmp(G,2)$	$dmp(G,3)$
1	0	1	2	3	2	1	2	2	1
2	1	0	1	2	1	2	3	2	0
3	2	1	0	1	2	3	2	2	1
4	3	2	1	0	1	2	2	2	1
5	2	1	2	1	0	1	3	2	0
6	1	2	3	2	1	0	2	2	1

Tabela 3.12: Distância Múltipla de G.

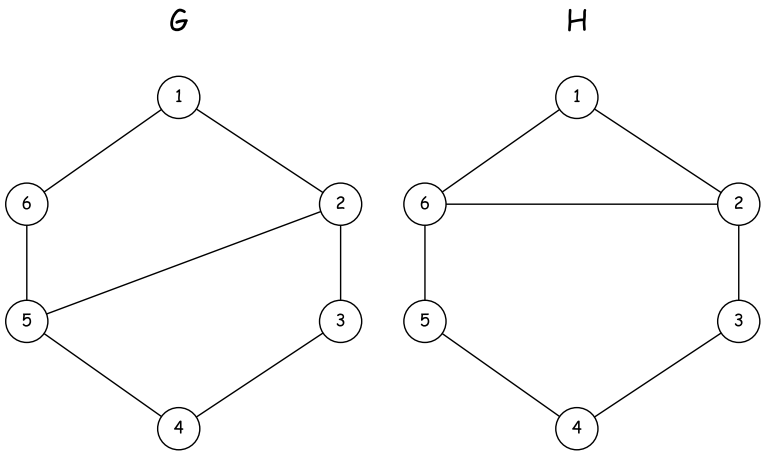


Figura 3.27: Grafos G e H isomorfos.

- $dmp(G,1)$ entre os vértices, G tem a seguinte seqüência: 232232. Ordenando, temos: 222233.
- $dmp(G,2)$ entre os vértices, G tem a seguinte seqüência: 222222.
- $dmp(G,3)$ entre os vértices, G tem a seguinte seqüência: 101101. Ordenando, temos: 001111.

Unindo as seqüências dmp de G temos: 222233222222001111

Calculando o conjunto dmp de H para as distâncias {1, 2, 3} entre os vértices, temos:

	1	2	3	4	5	6	$dmp(H,1)$	$dmp(H,2)$	$dmp(H,3)$
1	0	1	2	3	2	1	2	2	1
2	1	0	1	2	2	1	3	2	0
3	2	1	0	1	2	2	2	3	0
4	3	2	1	0	1	2	2	2	1
5	2	2	2	1	0	1	2	3	0
6	1	1	2	2	1	0	3	2	0

Tabela 3.13: Distância Múltipla de H.

- $dmp(H,1)$ entre os vértices, H tem a seguinte seqüência: 232223. Ordenando, temos: 222223.
- $dmp(H,2)$ entre os vértices, H tem a seguinte seqüência: 223232. Ordenando, temos: 222223.
- $dmp(H,3)$ entre os vértices, H tem a seguinte seqüência: 101101. Ordenando, temos: 001111.

Unindo as seqüências dmp de H temos: 222223222223001111

Com isso verificamos que, de acordo com esta invariante, G e H não são isomorfos, pois $dmp(G) \neq dmp(H)$.

3.2.15.1 Refinamento das obstruções encontradas

Cada grafo da nossa implementação possui uma identificação que é definida pela junção dos valores das invariantes de isomorfismo descritas na sessão anterior. Por exemplo:

Seja:

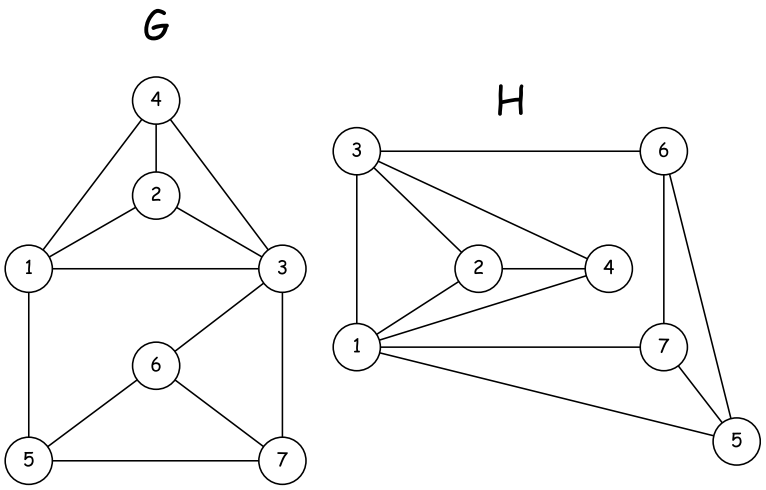


Figura 3.28: Grafos isomorfos G e H.

Valores das Invariantes

1. Quantidade de vértices: 7
2. Quantidade de arestas: 12
3. Seqüência ordenada de graus: 3333345
4. Quantidade de cliques maximais: 4
5. Tamanho da Clique Maximal: 4
6. Número cromático: 4
7. Distância Múltipla: 333334512333330000000

A identificação (id) de G e H gerada pelo programa corresponde a:

$id(G) = 7 - 12 - 3333345 - 4 - 4 - 4 - 333334512333330000000.$

$id(H) = 7 - 12 - 3333345 - 4 - 4 - 4 - 333334512333330000000.$

Isso significa que, baseado nessas invariantes, G e H são candidatos a serem isomorfos. Para termos certeza teremos que validar o isomorfismo entre eles.

Podemos facilmente observar que teremos muitos grafos com a mesma id, esses grafos foram agrupados por essa identificação. Primeiramente criamos uma lista de ids selecionando distintamente todas as identificações com a seguinte implementação:

Algorithm 5 Método SelecionaGrupos.

```
1: para todo G ∈ listaGrafos faça
2:     se id(G) ∉ listaId então    ▷ listaId = lista de todas as identificações distintas.
3:         listaId[i] ← id(G)
4:         i ← i + 1
5:     fim se
6: fim para
```

Fim do algoritmo.

Abaixo temos o método para agrupar os grafos por identificação.

Algorithm 6 Método AgrupaGrafos.

```
1: para todo id ∈ listaId faça
2:     imprima id
3:     para todo G ∈ listaGrafos faça
4:         se id(G) = id então
5:             imprima G
6:         fim se
7:     fim para
8: fim para
```

Fim do algoritmo.

3.2.15.2 Validação do Isomorfismo

Para verificar se os grafos contidos em cada grupo são realmente isomorfos, utilizamos a função ***GraphIsomorphismInspector*** da biblioteca $JGraphT$ que é uma biblioteca livre para Teoria dos Grafos mantida pela SourceForge e está disponível em http://jgrapht.org/

Algumas classes disponíveis na bliblioteca:

- KerboschCliqueFinder
  Proposto por [6], o algoritmo de Bron-Kerbosch é caracterizado pelo *backtraking*, que procura por todas as cliques maximais em um dado grafo G.

- ChromaticNumber
  Calcula o número cromático de um grafo.

- ***GraphIsomorphismInspector***
  Essa classe utiliza uma busca exaustiva, realizando todas as permutações possíveis, para verificar se dois grafos são isomorfos. O procedimento tem complexidade em $O(n!)$. É importante mencionar que, algumas invariantes são checadas antes de realizar as permutações e que apenas os vértices com o mesmo grau são comparados.

  Em nossa pesquisa, trabalhamos com grafos com no máximo nove vértices, o que não comprometeu a aplicação devido à ineficiência desse algoritmo.

- DijkstraShortestPath
  Uma implementação do algoritmo de Caminho de Custo Mínimo de Dijkstra.

- EulerianCircuit
  Este algoritmo verifica se um grafo é Euleriano (contém um circuito Euleriano). Além disso, se o grafo for Euleriano, pode obter uma lista dos vértices que compõem o circuito.

- HamiltonianCycle
  Esta classe lida com a descoberta do caminho ótimo ou aproximadamente ótimo (Problema do Caixeiro Viajante). Sabe-se que este problema é NP-completo e portanto, por vezes, não teremos a solução precisa, mas aproximada.

- NeighborIndex
  Mantém uma lista dos vizinhos de cada vértice.

Com essa metodologia, geramos 83 (oitenta e três) grupos, cada um com seus respectivos grafos, lembrando que não abordamos todas as combinações descritas na seção 3.2, trabalho que poderá ser realizado futuramente, possivelmente com outra técnica.

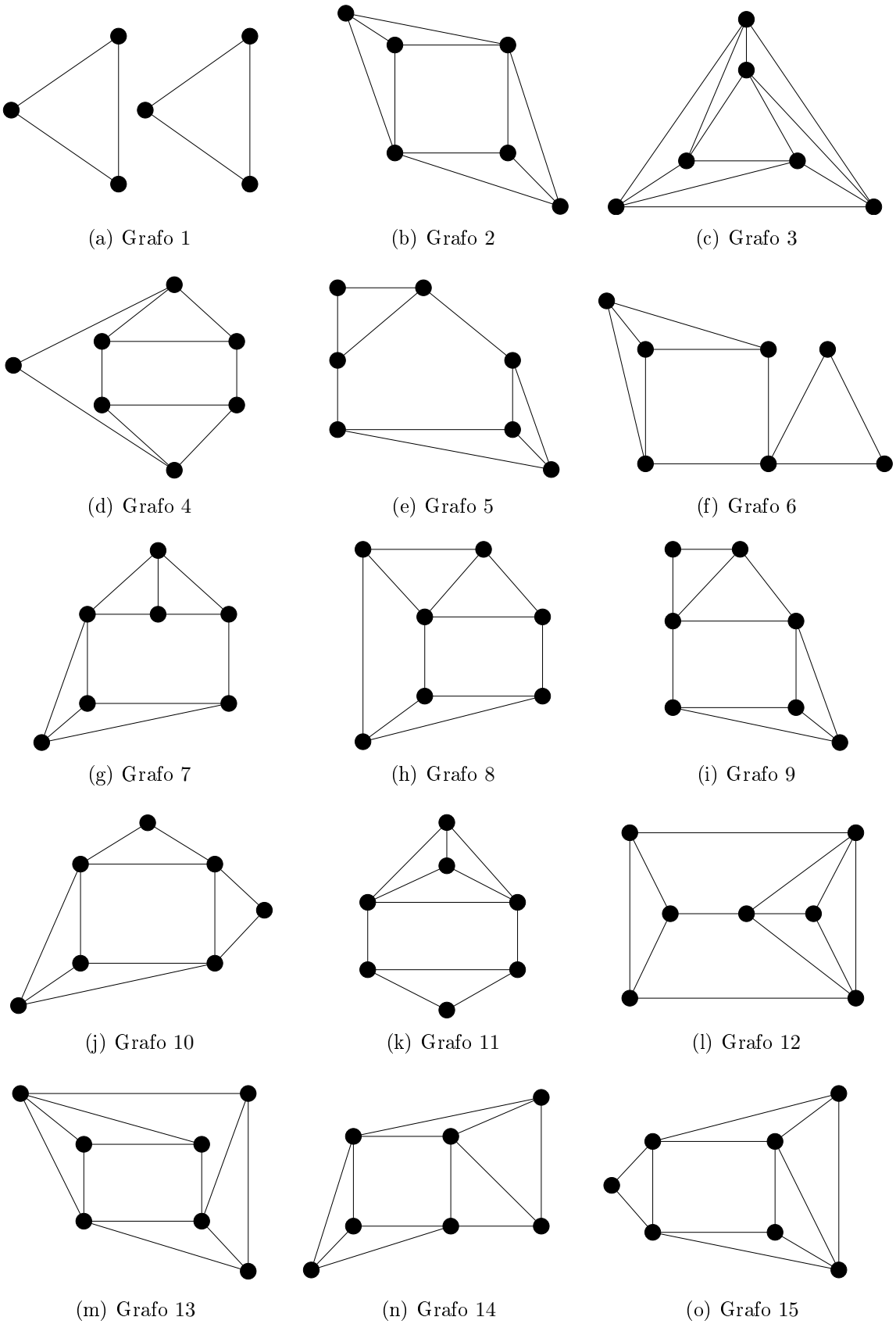
Na seqüência, iremos ilustrar as obstruções-$(2,1)$ obtidas no nosso trabalho.

Capítulo 4

Obstruções Minimais dos Grafos-$(2,1)$

Neste capítulo, apresentamos de forma geral, todas as obstruções-$(2,1)$ minimais encontradas nesse trabalho. Para gerar a imagem de cada obstrução, utilizamos a ferramenta "yEd Graph Editor", que está disponível gratuitamente em http://www.yworks.com e roda nas principais plataformas: Windows, Unix / Linux e Mac OS X.

Vale ressaltar que, tentamos representar todas as obstruções de forma planar.



(a) Grafo 1 (b) Grafo 2 (c) Grafo 3

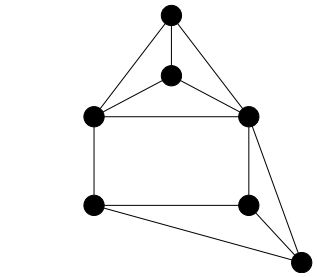
(d) Grafo 4 (e) Grafo 5 (f) Grafo 6

(g) Grafo 7 (h) Grafo 8 (i) Grafo 9

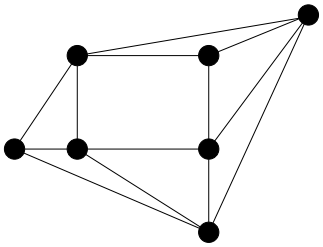
(j) Grafo 10 (k) Grafo 11 (l) Grafo 12

(m) Grafo 13 (n) Grafo 14 (o) Grafo 15

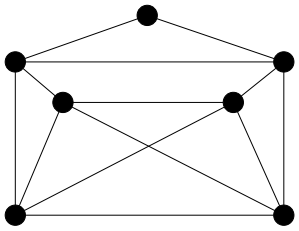
Figura 4.1: Obstruções-$(2,1)$ encontradas - 1 a 15



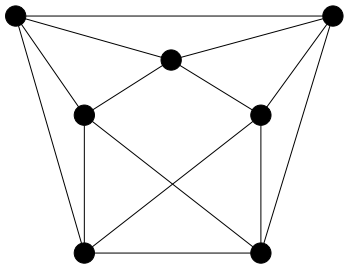
(a) Grafo 16



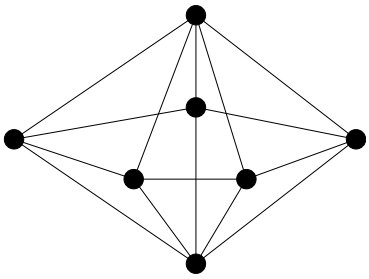
(b) Grafo 17



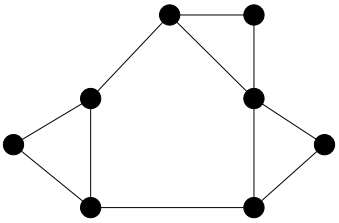
(c) Grafo 18



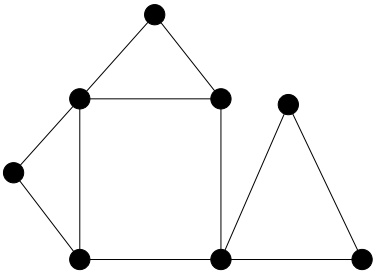
(d) Grafo 19



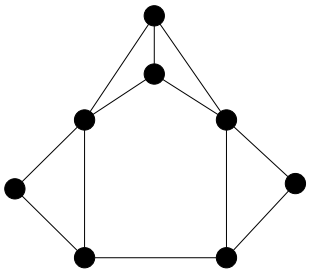
(e) Grafo 20



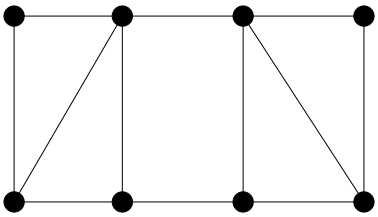
(f) Grafo 21



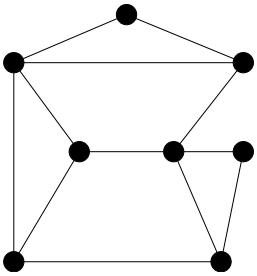
(g) Grafo 22



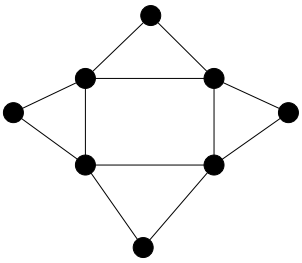
(h) Grafo 23



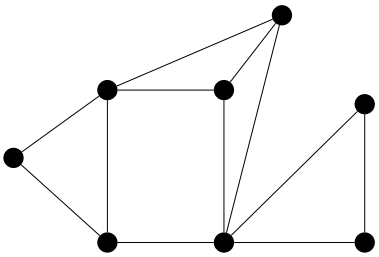
(i) Grafo 24



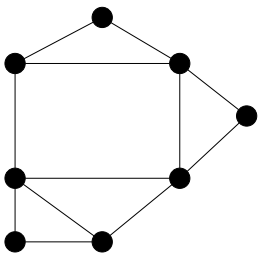
(j) Grafo 25



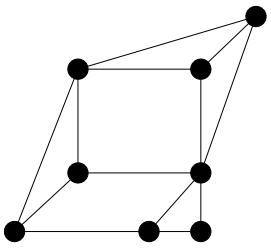
(k) Grafo 26



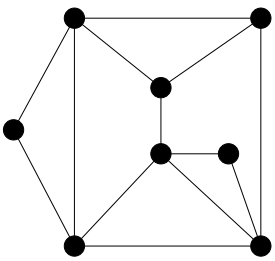
(l) Grafo 27



(m) Grafo 28

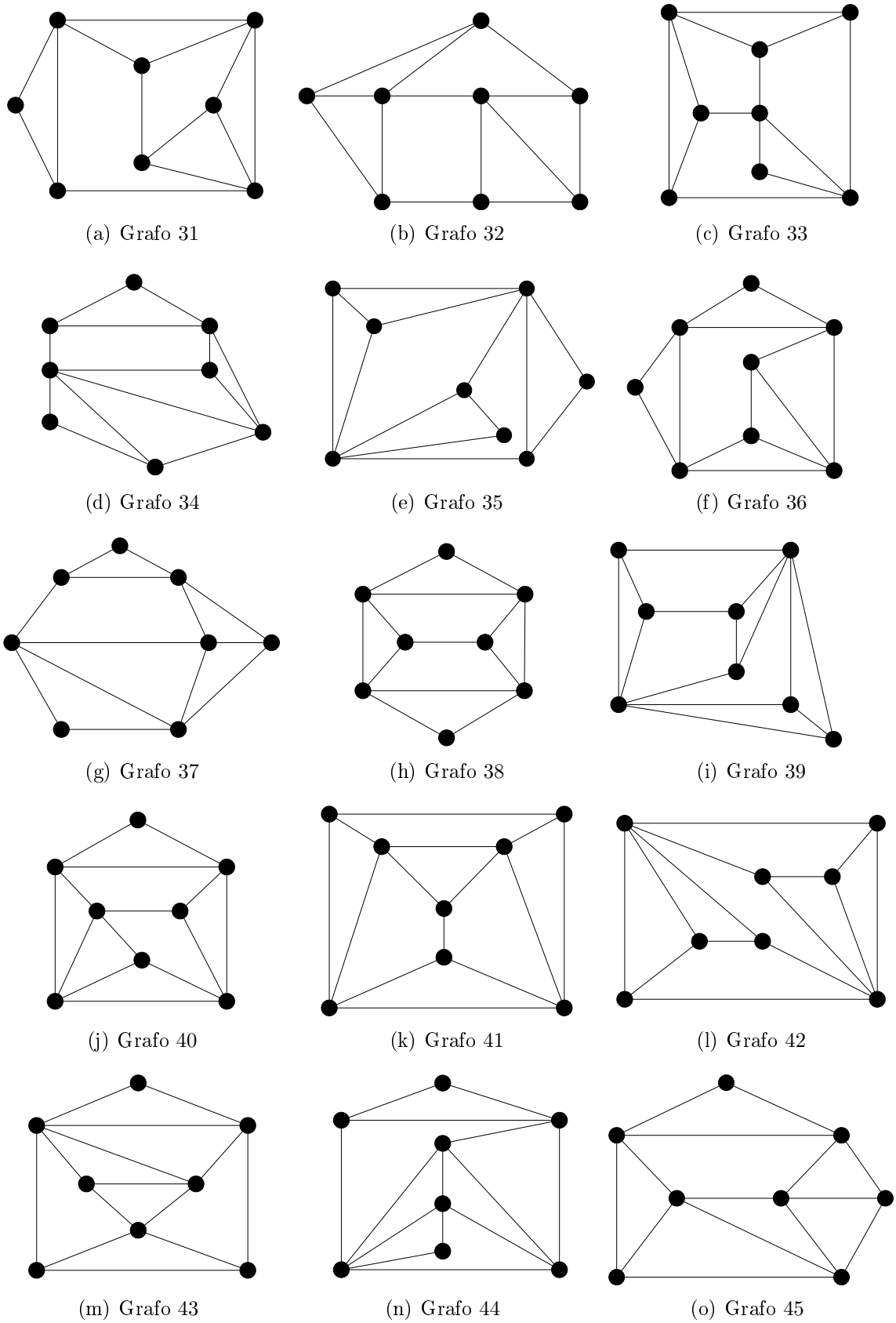


(n) Grafo 29



(o) Grafo 30

Figura 4.2: Obstruções-$(2,1)$ encontradas - 16 a 30



(a) Grafo 31 (b) Grafo 32 (c) Grafo 33

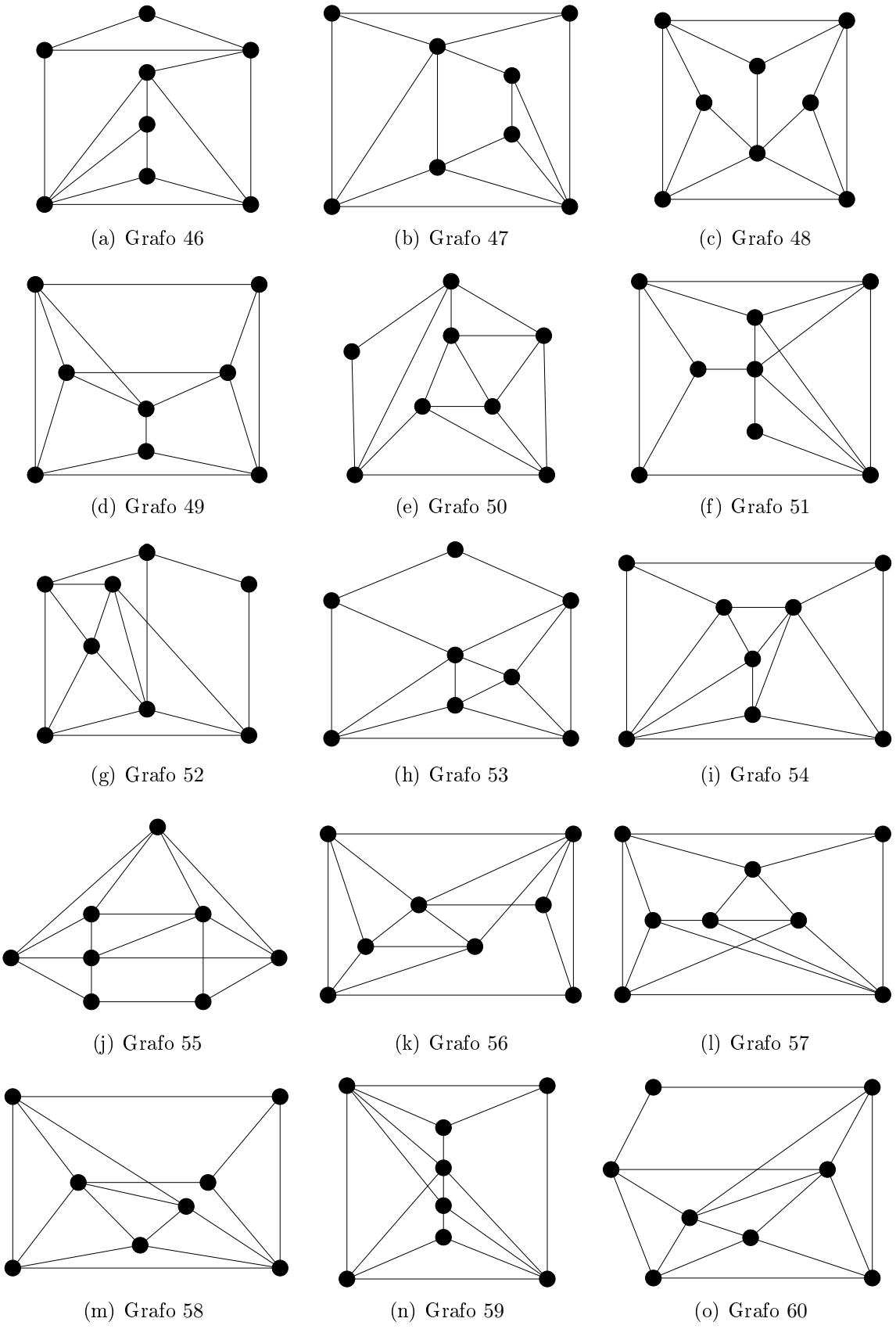
(d) Grafo 34 (e) Grafo 35 (f) Grafo 36

(g) Grafo 37 (h) Grafo 38 (i) Grafo 39

(j) Grafo 40 (k) Grafo 41 (l) Grafo 42

(m) Grafo 43 (n) Grafo 44 (o) Grafo 45

Figura 4.3: Obstruções-$(2,1)$ encontradas - 31 a 45



(a) Grafo 46 (b) Grafo 47 (c) Grafo 48

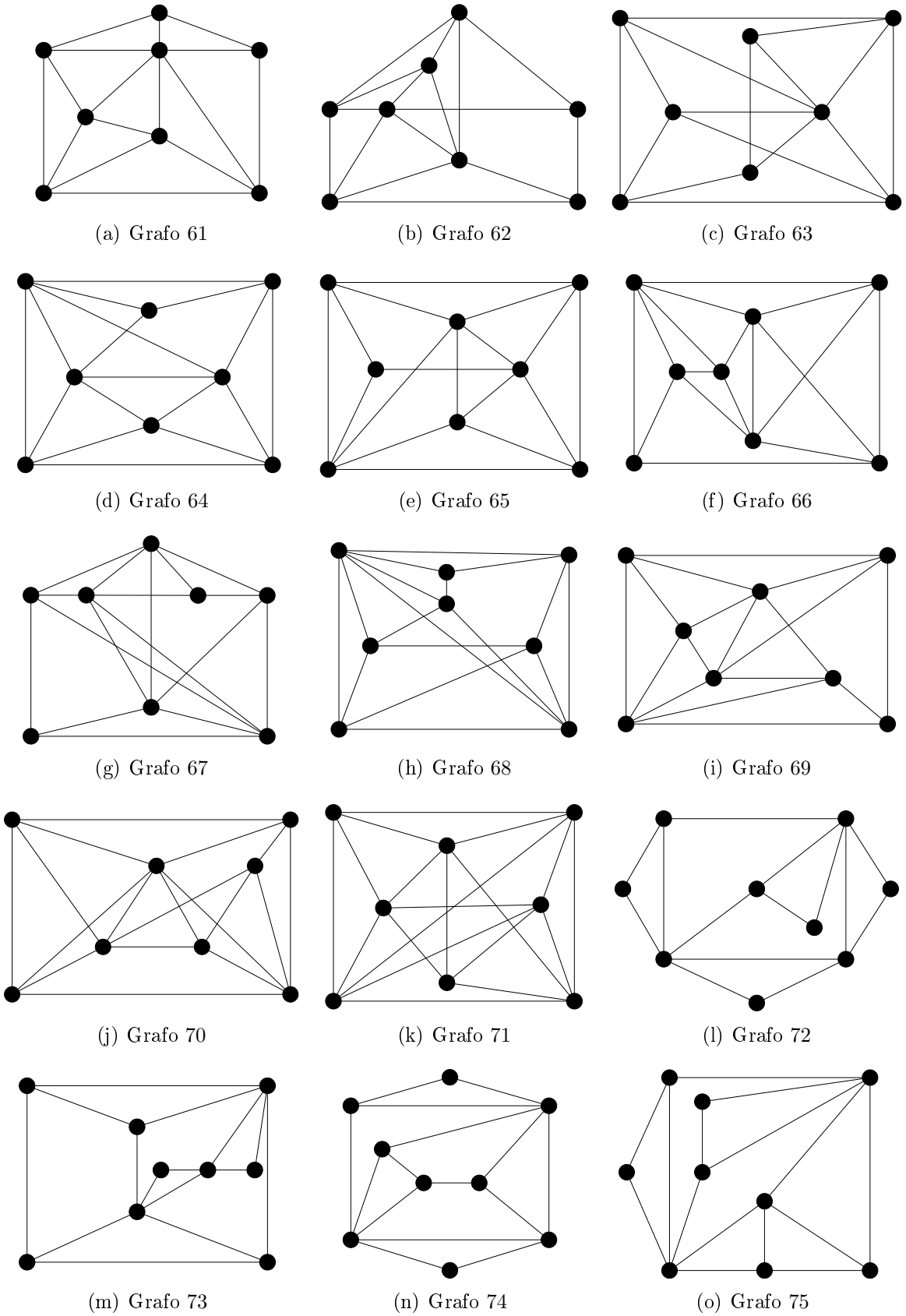
(d) Grafo 49 (e) Grafo 50 (f) Grafo 51

(g) Grafo 52 (h) Grafo 53 (i) Grafo 54

(j) Grafo 55 (k) Grafo 56 (l) Grafo 57

(m) Grafo 58 (n) Grafo 59 (o) Grafo 60

Figura 4.4: Obstruções-$(2,1)$ encontradas - 46 a 60



(a) Grafo 61 (b) Grafo 62 (c) Grafo 63

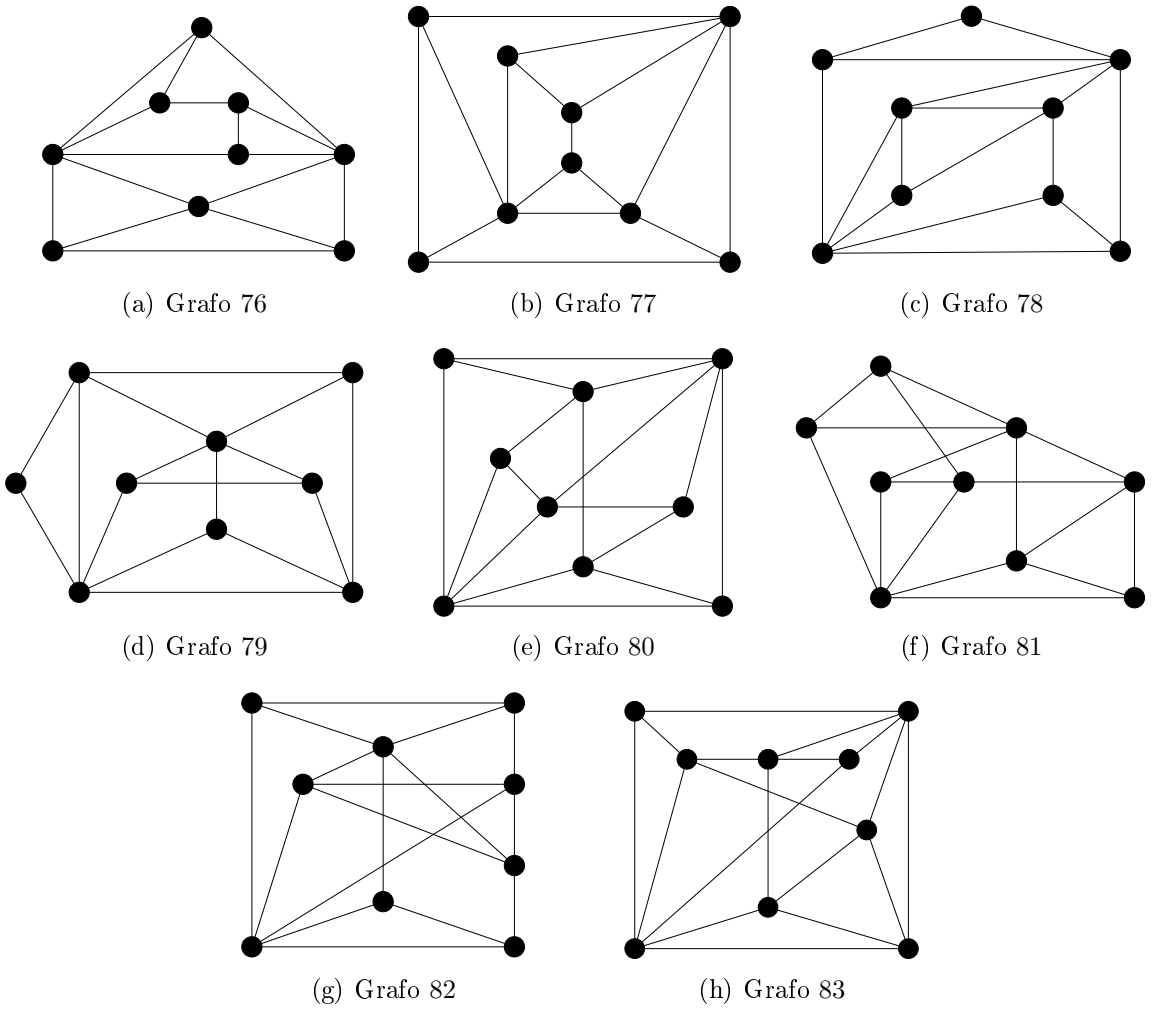
(d) Grafo 64 (e) Grafo 65 (f) Grafo 66

(g) Grafo 67 (h) Grafo 68 (i) Grafo 69

(j) Grafo 70 (k) Grafo 71 (l) Grafo 72

(m) Grafo 73 (n) Grafo 74 (o) Grafo 75

Figura 4.5: Obstruções-$(2,1)$ encontradas - 61 a 75



(a) Grafo 76 (b) Grafo 77 (c) Grafo 78

(d) Grafo 79 (e) Grafo 80 (f) Grafo 81

(g) Grafo 82 (h) Grafo 83

Figura 4.6: Obstruções-$(2,1)$ encontradas - 76 a 83

Capítulo 5

Conclusão

Este trabalho detectou uma família de subgrafos proibidos para grafos-$(2,1)$, também chamados de obstrução-$(2,1)$ que, quando presentes em G, impedem que o mesmo seja um grafo-$(2,1)$. Para tal descoberta, realizamos um estudo sobre a classe dos grafos-(k,l), mais precisamente dos grafos-$(2,1)$, onde analisamos o algoritmo proposto por Brandstädt para o reconhecimento dessa classe de grafo. Foi através desse algoritmo que criamos a técnica utilizada nesse trabalho para determinação das obstruções.

Apresentamos também a ferramenta e a metodologia desenvolvida para realizar o refinamento das obstruções obtidas, ou seja, eliminar os grafos isomorfos.

Vale mencionar que, no decorrer desse trabalho, tentamos encontrar o que havia em comum nas obstruções que surgiam, com o intuito de descobrir a caracterização dos grafos-$(2,1)$. E também, utilizando a linguagem de programação Java, implementamos o algoritmo de Brandstädt e testamos todas as obstruções encontradas, tendo como resposta o esperado: $G \notin (2,1)$.

Com a implementação desenvolvida nesse trabalho, podemos aplicar algoritmos computacionais, aos grafos carregados pela aplicação a partir do arquivo contendo suas matrizes de adjacência.

Como contribuição, apresentamos uma técnica para detecção de obstruções para grafos através do seu algoritmo de reconhecimento, além de mostrarmos que os grafos-$(2,1)$ possuem uma família infinita de subgrafos proibidos.

5.1 Trabalhos Futuros

Como propostas para trabalhos futuros, pretendemos realizar os seguintes estudos:

1. Utilizar essa técnica a fim de encontrar obstruções para outras classes de grafos.

2. Verificar se surgem novas obstruções, caso a técnica utilizada na Falha 2 do algoritmo de Brandstädt seja implementada com todas as combinações descritas na seção 3.2.

3. Detectar um padrão existente entre as estruturas das obstruções encontradas, a fim de realizar uma caracterização formal dos grafos-$(2, 1)$.

Referências

[1] Béla Bollobás. *Modern Graph Theory*. Springer-Verlag, New York, NY, USA, 1998.

[2] Andreas Brandstädt. Partitions of graphs into one or two independent sets and cliques. *Discrete Mathematics*, 152(1-3):47–54, 1996.

[3] Andreas Brandstädt. Corrigendum. *Discrete Mathematics*, 186(1-3):295, 1998.

[4] Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. The complexity of some problems related to graph 3-colorability. *Discrete Applied Mathematics*, 89(1-3):59–73, 1998.

[5] Raquel S. F. Bravo, Sulamita Klein, Loana Tito Nogueira, and Fábio Protti. Characterization and recognition of p4-sparse graphs partitionable into k independent sets and l cliques. *Discrete Appl. Math.*, 159(4):165–173, February 2011.

[6] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[7] Raquel de Souza Francisco, Sulamita Klein, and Loana Tito Nogueira. Characterizing (k, l)-partitionable cographs. *Electronic Notes in Discrete Mathematics*, 22:277–280, 2005.

[8] Tomas Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. Complexity of graph partition problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 464–472, New York, NY, USA, 1999. ACM.

[9] Stéphane Foldes and Peter L. Hammer. Split graphs. In *Proceedings of the Eighth Southeastern Conference on Combinatorics, Graph Theory and Computing (Louisiana State Univ., Baton Rouge, La., 1977)*, pages 311–315. Congressus Numerantium, No. XIX, Winnipeg, Man., 1977. Utilitas Math.

[10] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[12] Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[13] Pavol Hell, Sulamita Klein, Loana Tito Nogueira, and Fáibio Protti. Partitioning chordal graphs into independent sets and cliques. *Discrete Appl. Math.*, 141(1-3):185–194, May 2004.

[14] Vincent Remie. Graph isomorphism problem. Technical University of Eindhoven, 2003. Undergraduate Thesis.

[15] J. L. Szwarcfiter. *Grafos e algoritmos computacionais*. Editora Campus, 1986.