UNIVERSIDADE FEDERAL DO ABC
Centro de Matemática, Computação e Cognição

Algoritmos para emparelhamentos em grafos

Aluna: Beatriz Favini Chicaroni
Orientadora: Profa. Dra. Carla Negri Lintzmayer

Santo André
Maio de 2023

Resumo

Um grafo é uma estrutura muito utilizada para representar diversas situações do mundo real que envolvem relações entre pares de objetos. Um emparelhamento é um subconjunto de arestas de um grafo escolhidas de maneira que, duas a duas, elas não sejam adjacentes. O problema de encontrar um emparelhamento de tamanho máximo em grafos pode ser aplicado a qualquer situação em que se queira formar o maior número possível de pares de objetos em uma amostra de modo que nenhum objeto pertença a mais de um par ao mesmo tempo. O principal objetivo deste trabalho foi estudar resultados importantes sobre emparelhamentos. Além disso, implementamos o algoritmo de Egerváry, que encontra um emparelhamento máximo em um grafo bipartido em tempo polinomial. Esses resultados são clássicos da computação e não são vistos na matriz curricular do Bacharelado em Ciência da Computação na UFABC, de forma que esse estudo enriqueceu a formação da aluna.

Palavras-Chave: Teoria dos grafos, emparelhamentos em grafos, emparelhamento máximo, algoritmo de Egerváry.

Sumário

1 Introdução

Diversas situações reais que envolvem relações entre pares de objetos podem ser modeladas através de grafos. Por exemplo, cada vértice pode representar um usuário de uma rede social e as arestas podem representar a amizade entre eles, ou cada vértice pode representar uma cidade e as arestas, a existência de ferrovias entre duas cidades. A partir dessa modelagem, é possível fazer perguntas como qual o usuário mais influente na rede, ou qual o menor caminho entre duas cidades.

Um emparelhamento em um grafo é um subconjunto de arestas desse grafo escolhidas de forma que, duas a duas, elas não estejam ligadas a um mesmo vértice. Emparelhamentos são estruturas muito úteis no estudo de grafos que podem ser encontradas em tempo polinomial [4] e, uma vez que o conceito é compreendido, é possível aplicá-lo para resolver diversas situações em que se deseja parear elementos.

Considere, por exemplo, o problema de alocação de docentes a disciplinas durante um período em uma universidade. Nesse caso, cada professor e cada instância de uma disciplina que deve ser ministrada serão representados por vértices; cada docente será associado ao conjunto de disciplinas que ele pode ministrar através das arestas. Queremos que o maior número possível de disciplinas sejam ofertadas e que o maior número possível de docentes ministre uma disciplina. Também queremos que cada docente ministre apenas uma disciplina e sabemos que uma instância de uma disciplina só pode ser ofertada por um único docente. Essa escolha ótima é dada por um emparelhamento máximo no grafo de disciplinas e professores.

Encontrar um emparelhamento máximo é um problema clássico da computação que já foi muito estudado. Este projeto teve como principal objetivo estudar alguns resultados importantes sobre emparelhamentos, em particular o algoritmo de Egerváry e o algoritmo de Edmonds. O objetivo secundário do projeto foi estudar e implementar o algoritmo de Egerváry que, utilizando caminhos aumentadores, resolve o problema de encontrar um emparelhamento de cardinalidade máxima em grafos bipartidos em tempo polinomial.

Esses algoritmos, apesar de serem resultados clássicos na computação, não são estudados na grade curricular do Bacharelado em Ciência da Computação na UFABC. Portanto, o estudo e implementação desses algoritmos e a compreensão dos resultados estruturais que os envolvem foram realizados com o intuito de enriquecer a formação da aluna.

O restante do documento está dividido da seguinte forma: a Seção 2 define alguns conceitos utilizados no desenvolvimento do projeto e apresenta a definição formal do problema a ser estudado; a Seção 3 apresenta dois resultados clássicos sobre emparelhamentos; a Seção 4 é uma revisão bibliográfica sobre o tema; a Seção 5 traz a definição e a análise do Algoritmo de Egerváry; a Seção 6 discute o funcionamento

e corretude do Algoritmo de Edmonds; e, por fim, a implementação do Algoritmo de Egerváry é descrita na Seção 7.

2 Fundamentação Teórica

Nesta seção são apresentados e definidos os principais conceitos que serão utilizados no desenvolvimento do trabalho. Tomamos como fonte principal o livro de Bondy e Murty [4].

2.1 Definições

Um **grafo** (simples) é um par ordenado $G = (X, Y)$ em que X é um conjunto de elementos que chamaremos de **vértices** e Y é um conjunto de elementos que chamaremos de **arestas**, sendo que cada aresta é um par não ordenado de vértices. Seja $Z = (X, Y)$ um grafo, chamaremos $V(Z)$ o conjunto de vértices desse grafo e $E(Z)$ o seu conjunto de arestas. Sendo assim, $V(Z) = X$ e $E(Z) = Y$. Por simplicidade, denotaremos uma aresta $\{u, v\}$ apenas por uv ou vu.

A Figura 1 mostra a representação gráfica de um grafo, em que os pontos representam os vértices e as linhas que os ligam representam as arestas.

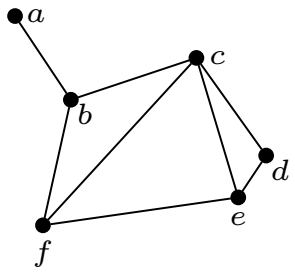


Figura 1: Exemplo de um grafo G com $V(G) = \{a, b, c, d, e, f\}$ e $E(G) = \{ab, bc, bf, cd, ce, cf, de, ef\}$, em que os pontos representam os vértices e as linhas representam as arestas.

Sejam H e G grafos. Se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$ dizemos que H é **subgrafo** de G e denotamos isso por $H \subseteq G$. Se $V(H) = V(G)$, então H é um **subgrafo gerador** de G. Dado $F \subseteq E(G)$, denotamos por $G[F]$ o subgrafo de G **induzido** pelo conjunto F, definido de forma que $E(G[F]) = F$ e $V(G[F]) = \{x : xy \in F\}$. Uma forma de se obter subgrafos é removendo elementos do grafo original. Assim, dados G e H com $H \subseteq G$, o grafo $K = G - H$ é o subgrafo de G tal que $V(K) = V(G) \setminus V(H)$ e $E(K) = E(G) \setminus E(H)$. O grafo $K = G - V(H)$ é o subgrafo de G tal que $V(K) = V(G) \setminus V(H)$ e $E(K) = E(G) \setminus A$, em que $A = \{uv \in E(G) : u \in V(H)\}$. O grafo $K = G - E(H)$ é o grafo K tal que $V(K) = V(G)$ e $E(K) = E(G) \setminus E(H)$. Caso $E(H) = \emptyset$, então $K = G - H$ equivale a $K = G - V(H)$.

Dizemos que a **ordem** de um grafo G é a quantidade de vértices que ele possui e o **tamanho** do grafo G é a quantidade de arestas. No exemplo dado na Figura 1 temos um grafo de ordem 6 e tamanho 7.

Dados u e v vértices quaisquer de G, dizemos que eles são **vizinhos** se $uv \in E(G)$. Nesse caso, u e v também podem ser denominados **adjacentes** ou **extremos** da aresta uv e pode-se dizer que a aresta uv **incide** em u e em v. Dizemos que duas arestas são **adjacentes** se elas incidem em um mesmo vértice. O conjunto dos vizinhos de u é denominado **vizinhança** de u e será denotado por $N_G(u)$. O **grau** de um vértice u é denotado por $d_G(u)$ e é definido como a quantidade de vizinhos desse vértice. Como G é simples, também podemos dizer $d_G(u) = |N_G(u)|$. Se o grafo G estiver claro no contexto, usamos apenas $d(u)$ e $N(u)$.

Um **emparelhamento** em um grafo G é um subconjunto de arestas de G escolhidas de forma que, duas a duas, elas não sejam adjacentes. Na Figura 2 o conjunto das arestas em vermelho é um exemplo de emparelhamento. As arestas que fazem parte de um emparelhamento são chamadas **emparelhadas** e as outras são chamadas de **arestas livres**. Dizemos que os vértices em que as arestas emparelhadas incidem são **vértices emparelhados** e os vértices em que não incidem arestas emparelhadas são **vértices livres**.

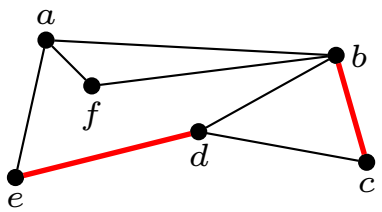


Figura 2: As arestas destacadas em vermelho são um exemplo de emparelhamento no grafo.

Dados um grafo G e um emparelhamento M em G, dizemos que M é um emparelhamento **maximal** caso nenhuma outra aresta do grafo possa ser adicionada a M sem que a propriedade de não-adjacência seja destruída. No caso da Figura 2, por exemplo, ainda é possível adicionar a aresta af ao emparelhamento satisfazendo a propriedade de não-adjacência das arestas, então aquele não se trata de um emparelhamento maximal.

Dado um emparelhamento maximal, ele é chamado de emparelhamento **máximo** se tiver o maior número possível de arestas dentre todos os emparelhamentos do grafo. Um emparelhamento máximo é **perfeito** se todos os vértices do grafo forem emparelhados. No caso da Figura 2, $M = \{af, bc, de\}$ é máximo e $M' = \{ae, bf, cd\}$ também.

Um **passeio** em um grafo é uma sequência de vértices em que vértices consecutivos possuem aresta entre si. Um exemplo de passeio no grafo apresentado

na Figura 1 é $P = (c, f, e, c, b)$. Formalmente, dado um grafo G, dizemos que $P = (v_0, v_1, ..., v_{k-1}, v_k)$ em que $v_i \in V(G)$ para todo $0 \leq i \leq k$ e $v_{i-1}v_i \in E(G)$ para todo $1 \leq i \leq k$ é um passeio. Os vértices v_0 e v_k são os **extremos** de P e os demais vértices são chamados de **vértices internos**. Uma **trilha** é um passeio que não repete arestas e um **caminho** é um passeio que não repete vértices. Se P é um caminho de u até v dizemos que P é um uv-caminho. Nos casos em que os vértices extremos de P são os mesmos, ou seja, $v_0 = v_k$, dizemos que o passeio é **fechado**. Um **ciclo** é um passeio fechado em que todos os vértices internos são diferentes. A quantidade de arestas em um passeio é chamada de **comprimento** do passeio.

Dizemos que um vértice v é **alcançável** por u se existe um caminho de u a v. Um grafo é **conexo** se existe pelo menos um caminho entre quaisquer pares de vértices. A **distância** entre um vértice u e um vértice v é o menor comprimento de um uv-caminho dentre todos os uv-caminhos possíveis. Se v não é alcançável por u, definimos a distância de u a v como ∞.

Dizemos que um grafo T é uma **árvore** se T é conexo e não contém ciclos. Se T é uma árvore, dizemos que T é **enraizada** se ela possui um vértice especial u chamado de **raiz**. Nesse caso, podemos dizer que T é uma **u-árvore**. Para dois vértices u e v quaisquer de T, denotamos por uTv o uv-caminho em T.

Dado um emparelhamento M em um grafo G, um **caminho M-alternante** é um caminho $P = (v_0, v_1, ..., v_{k-1}, v_k)$ cujas arestas alternam entre livres e emparelhadas. Caso v_0 e v_k sejam vértices livres o caminho é denominado **M-aumentador**. Em outras palavras, um **caminho M-aumentador** é um caminho $P = (v_0, v_1, ..., v_{k-1}, v_k)$ M-alternante em que nem a aresta v_0v_1 nem a aresta $v_{k-1}v_k$ estão emparelhadas. Um u-caminho M-aumentador é um caminho M-aumentador que começa no vértice u. Na Figura 2, $P = (a, e, d, b, c)$ é um caminho M-alternante e $P' = (a, e, d, c, b, f)$ é um caminho M-aumentador.

Se G é um grafo, u é um vértice de G, $T \subseteq G$ é uma u-árvore, e M é um emparelhamento de G tal que u não é um vértice M-emparelhado, dizemos que T é uma **u-árvore M-alternante** se para todos os vértices v de T tais que $v \neq u$, o caminho uTv é um caminho M-alternante. Uma u-árvore M-alternante T é **M-coberta** se todos os vértices de T exceto u estão cobertos por $M \cap E(T)$.

Um **grafo bipartido** é um grafo cujos vértices podem ser divididos em dois conjuntos disjuntos X e Y tais que toda aresta do grafo conecta um vértice em X a um vértice em Y. Um grafo G bipartido com partes X e Y pode ser denotado por $G[X, Y]$. A Figura 3 é um exemplo de grafo bipartido. É importante observar que um grafo é bipartido se e somente se não existir um ciclo de comprimento ímpar.

Para dois conjuntos A e B, definimos a **diferença simétrica** entre A e B, denotada $A \triangle B$, como sendo o conjunto tal que $A \triangle B = (A \cup B) \setminus (A \cap B)$.

Por fim, sendo G um grafo e F um subgrafo de G, definimos a **contração**

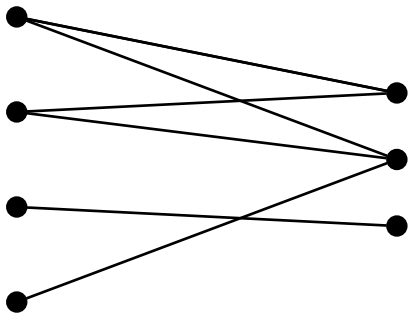
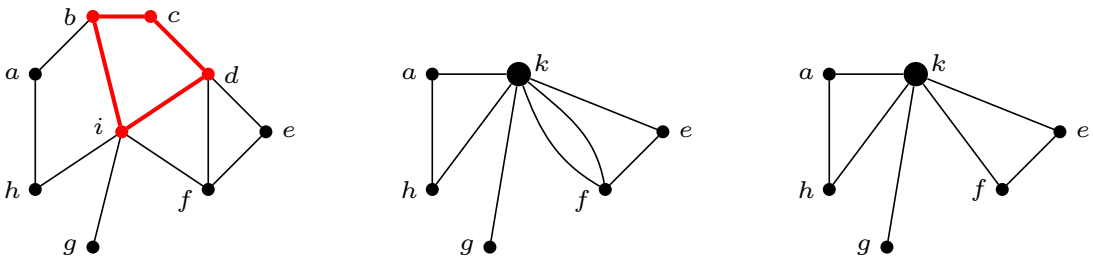


Figura 3: Exemplo de um grafo bipartido.

de $V(F)$ em G como a operação de remover o conjunto $V(F)$ de G substituindo-os por um único pseudovértice q que os represente no novo grafo, de forma que todas as arestas que têm ambas as extremidades em $V(F)$ não existem no novo grafo e as arestas que têm somente uma extremidade em $V(F)$ passam a ter essa extremidade ligada a q; caso surjam duas ou mais arestas que ligam o mesmo par de vértices no processo elas devem ser removidas. A contração é denotada por G/F e é exemplificada na Figura 4. Dado um emparelhamento M em G, podemos obter o emparelhamento M/F em G/F ao fazer $M \setminus E(F)$ e, caso existam arestas que possuem uma extremidade em $V(F)$ e uma extremidade fora, elas devem ser substituídas por arestas que liguem as extremidades que não pertencem a F ao pseudovértice que representa o conjunto $V(F)$ contraído. Dado um caminho P em G, podemos obter o passeio P/F em G/F ao remover o subcaminho $P' = P \cap E(F)$ de P e substituí-lo pelo pseudovértice q. Se $P \cap F$ for conexo, então P/F será um caminho. Na Figura 4, seja $P = (h, a, b, c, d, f, e)$; temos que $P/F = (h, a, k, f, e)$, em que k é o pseudovértice que substitui o subcaminho $P' = (b, c, d)$. Dado um conjunto $K \subseteq V(G)$ o conjunto K/F é o conjunto $K \setminus V(F) \cup \{q\}$, em que q é o pseudovértice que representa o conjunto $V(F)$ contraído.



(a) Grafo original G e subgrafo K destacado em vermelho.

(b) Grafo resultante da contração de $V(K)$ representada pelo pseudovértice k antes da remoção das arestas que ligam o mesmo par de vértices.

(c) Grafo resultante da contração de $V(K)$ representada pelo pseudovértice k após a remoção das arestas que ligam o mesmo par de vértices

Figura 4: Representação da operação de contração de um conjunto de vértices.

2.2 Problemas de Interesse

Emparelhamentos em grafos podem ser aplicados em situações em que se deseja formar o maior número de pares. Nesses casos, deseja-se encontrar um emparelhamento de cardinalidade máxima. Em alguns casos, além do grafo, temos uma função que associa um peso não-negativo a cada uma das arestas. Nessas situações, é possível que o problema de interesse seja encontrar um emparelhamento cuja soma dos pesos de suas arestas seja máximo. Vejamos alguns exemplos a seguir.

Em um problema conhecido como Problema de Distribuição de Pessoal [24], o gerente de uma empresa tem um determinado número de tarefas para atribuir à sua equipe, sabendo que cada funcionário tem conhecimento para executar apenas determinadas atividades. Nesse caso, podemos usar um grafo bipartido para representar os funcionários e as tarefas com os vértices e a relação entre eles com as arestas. Por exemplo, na Figura 3 os vértices do conjunto da esquerda representam os funcionários, o conjunto da direita representa as tarefas e cada aresta representa a habilidade de um funcionário realizar aquela atividade. O objetivo desse gerente é atribuir o maior número possível de atividades aos funcionários, sem que um funcionário precise executar mais de uma atividade e sem que uma tarefa seja executada por mais de um funcionário. Em outras palavras, o gerente deseja escolher o maior número possível de arestas do grafo de funcionários e atividades. Esse conjunto é, no caso, um emparelhamento máximo no grafo bipartido que representa a entrada.

Poderíamos, também, associar um peso a cada aresta que representaria a habilidade que aquele funcionário tem de realizar aquela tarefa. Dessa forma, o gerente teria interesse em escolher as arestas do emparelhamento de forma que a soma dos pesos fosse a maior possível. Nesse caso, a entrada se tornou uma instância do problema de encontrar um emparelhamento de peso máximo em grafos bipartidos.

No problema conhecido como Problema de Amostragem [6], por sua vez, um vendedor de brinquedos precisa carregar uma amostra de blocos com formatos e cores diferentes. Por exemplo, o vendedor pode ter blocos em quatro formatos diferentes e em quatro cores diferentes; para carregar exemplares de todas as cores e formatos ele levaria dezesseis blocos. Apesar disso, é possível escolher a amostra de forma que o comerciante carregue menos blocos e ainda tenha exemplares de todas as cores e formatos. No caso, o objetivo é formar o maior número possível de pares entre formatos e cores diferentes para que o vendedor possa carregar um exemplo de cada formato e cada cor disponível levando a menor quantidade de objetos que ele puder. Podemos modelar esse problema representando as cores e formatos através dos vértices de um grafo bipartido. As arestas relacionam os formatos às cores. O maior número possível de pares é o emparelhamento máximo desse grafo.

Figueiredo e Szwarcfiter [6] propõem uma divisão em quatro problemas relacio-

nados, definidos abaixo em ordem de dificuldade.

Problema 2.1. ***Emparelhamentos em grafos bipartidos.*** *Dado um grafo G bipartido, encontrar um emparelhamento de cardinalidade máxima.*

Problema 2.2. ***Emparelhamentos em grafos.*** *Dado um grafo G qualquer, encontrar um emparelhamento de cardinalidade máxima.*

Problema 2.3. ***Emparelhamentos em grafos bipartidos com peso.*** *Dado um grafo G bipartido e uma função $w\colon E(G) \to \mathbb{R}^+$, encontrar um emparelhamento de peso máximo.*

Problema 2.4. ***Emparelhamentos em grafos com peso.*** *Dado um grafo G qualquer e uma função $w\colon E(G) \to \mathbb{R}^+$, encontrar um emparelhamento de peso máximo.*

Existe um problema chamado Cobertura por Arestas em que se deseja selecionar um conjunto de arestas com o menor tamanho possível que cubra todos os vértices do grafo. Um dos algoritmos que conhecemos para resolver esse problema inicialmente encontra um emparelhamento máximo no grafo de entrada e em seguida estende-o gulosamente, selecionando novas arestas para cobrir os vértices que ficaram desemparelhados, caso o grafo não tenha um emparelhamento perfeito [21].

Além dos problemas de encontrar o emparelhamento máximo apresentados, existem outros problemas envolvendo emparelhamentos. Dentre eles temos o problema em que se deseja encontrar um emparelhamento perfeito de peso mínimo [5] e o problema conhecido como b-emparelhamento [1], definido no Problema 2.5.

Problema 2.5. ***b-emparelhamentos.*** *Seja um grafo G qualquer, uma função $w\colon E(G) \to \mathbb{R}^+$ que atribui pesos às arestas e uma função $b\colon V(G) \to \mathbb{R}^+$ que representa capacidades dos vértices. Um b-emparelhamento é uma função binária $x\colon E(G) \to \{0,1\}$ tal que, para cada vértice v vale que*

$$b(v) \geq \sum_{u \in N(v)} w(uv)x(uv)\,.$$

O objetivo do problema é encontrar x que maximize $\sum_{e \in E(G)} w(e)x(e)$.

Observe que um b-emparelhamento com $b = 1$ é um emparelhamento comum. Nosso foco, neste projeto, são os Problemas 2.1 e 2.2.

3 Teoremas Importantes

A seguir serão apresentados dois resultados clássicos que fornecem características estruturais importantes de emparelhamentos em grafos. As demonstrações foram estudadas e adaptadas a partir do livro de Bondy e Murty [4].

O Teorema 3.1, conhecido como Teorema de Berge, relaciona a existência de caminhos aumentadores com a identificação de emparelhamentos máximos em um grafo.

Teorema 3.1. *Um emparelhamento M em um grafo G é um emparelhamento máximo se e somente se G não tem nenhum caminho M-aumentador.*

Demonstração. Vamos dividir o teorema em duas proposições e provar ambas:

1. Se um emparelhamento M em um grafo G é um emparelhamento máximo, então G não tem nenhum caminho M-aumentador.

2. Se M é um emparelhamento em um grafo G e G não tem nenhum caminho M-aumentador, então M é um emparelhamento máximo.

Inicialmente, vamos provar a Proposição 1 através da sua contrapositiva: se G tem um caminho M-aumentador, então M não é um emparelhamento máximo. Veja a Figura 5 para um exemplo da discussão a seguir.

Então, seja M um emparelhamento em G. Vamos supor que G contém um caminho M-aumentador e vamos chamar esse caminho de P. Vamos definir M' tal que M' é formado por $(M \cup E(P)) \setminus (M \cap E(P))$, isto é, $M' = M \triangle E(P)$. Dessa forma, sabemos que $|M'| = |M| + 1$. Nesse caso, como $|M'| > |M|$, então M não é um emparelhamento máximo.

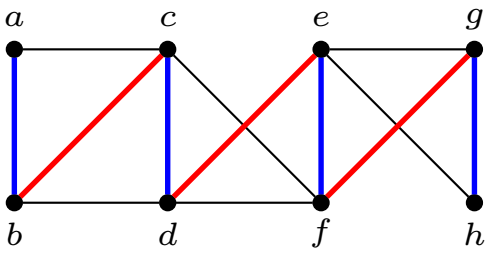
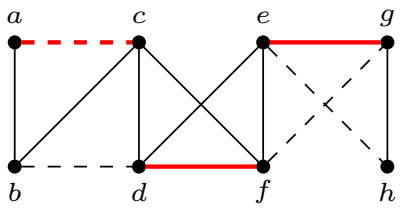


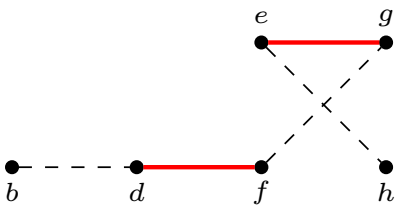
Figura 5: Exemplo de grafo em que as arestas do emparelhamento não máximo M estão representadas em vermelho e as arestas do emparelhamento máximo M' estão representadas em azul. O caminho M-aumentador é (a, b, c, d, e, f, g, h).

Agora vamos provar a Proposição 2 também através da sua contrapositiva: se o emparelhamento M não é um emparelhamento máximo, então G possui algum caminho M-aumentador. Veja a Figura 6 para um exemplo da discussão a seguir.

Para isso, vamos supor que M é um emparelhamento não máximo de G e M^* é um emparelhamento máximo de G. Então, $|M^*| > |M|$. Vamos definir $H := G[M \triangle M^*]$. Assim, cada vértice de H tem grau 1 ou 2 em H, pois cada vértice pode ser extremo de, no máximo uma aresta de M e uma aresta de M^*. Então, cada componente de H pode ser um ciclo ou caminho com arestas alternadas em M e em M^*. Além disso, note que uma componente ciclo tem comprimento necessariamente par.



(a) Exemplo de grafo G em que as arestas do emparelhamento não máximo M estão representadas em vermelho e as arestas do emparelhamento máximo M^* estão representadas com linhas tracejadas.



(b) Grafo $H := G[M \triangle M^*]$.

Figura 6: Exemplo da construção de um grafo H a partir de dois emparelhamentos em um grafo G.

Como $|M^*| > |M|$, o subgrafo H contém mais arestas de M^* que de M, então algum componente de H que é um caminho tem que, obrigatoriamente, começar e terminar com arestas de M^*. Pela definição de H, se uma aresta está em M^* ela não está em M, então tal caminho é M-aumentador. □

O Teorema 3.2, conhecido como Teorema de Hall, fornece condições necessárias e suficientes para que, em um grafo bipartido, exista um emparelhamento que cobre todos os vértices em uma das partes da bipartição do grafo.

Teorema 3.2. *Um grafo bipartido $G := G[X, Y]$ tem um emparelhamento que cobre todo vértice em X se e somente se $|N(S)| \geq |S|$ para todo $S \subseteq X$.*

Demonstração. Vamos dividir o teorema em duas proposições e provar ambas:

1. Se um grafo bipartido $G := G[X, Y]$ tem um emparelhamento que cobre todo vértice em X, então $|N(S)| \geq |S|$ para todo $S \subseteq X$.

2. Se $G := G[X, Y]$ é um grafo bipartido com $|N(S)| \geq |S|$ para todo $S \subseteq X$, então G tem um emparelhamento que cobre todo vértice em X.

Primeiro, vamos provar a Proposição 1 através da sua prova direta. Veja a Figura 7 para um exemplo da discussão a seguir. Seja $G := G[X, Y]$ um grafo bipartido que tem um emparelhamento M cobrindo todo vértice em X. Considere um subconjunto S de X (para ilustrar, podemos considerar que, na Figura 7, $S = \{b, c\}$ e $N(S) = \{e, f, g\}$). Os vértices em S estão emparelhados por M com $|S|$ vértices distintos em Y, todos em $N(S)$. Então, $|N(S)| \geq |S|$.

Vamos provar a Proposição 2 através da sua contrapositiva: se G não tem um emparelhamento que cobre todo vértice em X, então existe pelo menos um $S \subseteq X$ tal que $|N(S)| < |S|$. Veja a Figura 8 para um exemplo da discussão a seguir.

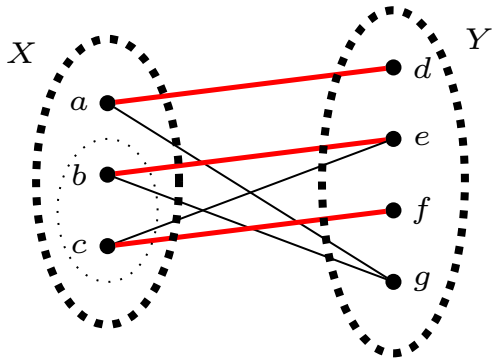


Figura 7: Exemplo de grafo bipartido $G := G[X, Y]$ em que as arestas de um emparelhamento M estão representadas em vermelho. O subconjunto S de X, $S = \{b, c\}$ está destacado e é possível observar que $|N(S)| \geq |S|$.

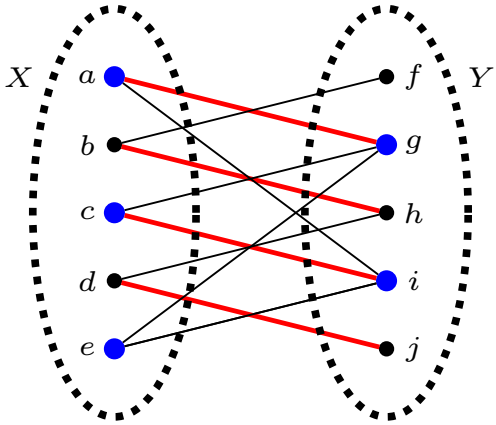


Figura 8: Exemplo de grafo bipartido $G := G[X, Y]$ em que as arestas de um emparelhamento M^* estão representadas em vermelho. O subconjunto Z de vértices alcançáveis por e por caminhos M^*-alternantes está destacado em azul.

Seja $G := G[X, Y]$ um grafo bipartido que não tem um emparelhamento cobrindo todo vértice em X. Seja M^* um emparelhamento máximo em G e u um vértice em X não coberto por M^* (na Figura 8 as arestas do emparelhamento M^* estão destacadas em vermelho e o vértice e não está emparelhado). Denote por Z o conjunto de todos os vértices alcançáveis por u por caminhos M^*-alternantes (no exemplo, o conjunto Z está destacado em azul e é formado pelos vértices $Z = \{a, c, e, g, i\}$). Como M^* é um emparelhamento máximo, segue do Teorema 3.1 que u é o único vértice em Z não coberto por M^*, pois caso contrário teríamos um caminho M^*-aumentador.

Vamos agora definir $R := X \cap Z$ e $B := Y \cap Z$. Dessa definição, temos que os vértices de $R \setminus \{u\}$ estão emparelhados por M^* com os vértices de B. Então $|B| = |R| - 1$. Além disso, podemos mostrar que $N(R) = B$. Para isso, suponha por contradição que existe $x \in N(R)$ com $x \notin B$. Seja $y \in R$ um vizinho de x. Por definição, existe um uy-caminho M^*-alternante e y está coberto pelo emparelhamento M^*, então sabemos que $yx \notin M^*$. Consequentemente o caminho de u a x é M^*-alternante e $x \in B$, que é uma contradição. Assim, podemos afirmar que $|N(R)| = |B| = |R| - 1$. Então $|N(R)| < |R|$ e a contrapositiva vale para o conjunto $S := R$. □

4 Revisão Bibliográfica

Diversos estudos apresentaram algoritmos para resolver os problemas apresentados na Seção 2.2. Na discussão a seguir, n é o número de vértices do grafo e m é o número de arestas. Em 1931, Egerváry [11, 19] propôs um algoritmo de complexidade $O(nm)$ para resolver o Problema 2.1, de emparelhamentos em grafos bipartidos. O Algoritmo de Egerváry será analisado de forma detalhada na Seção 5. Posteriormente, em 1973, foi proposto o Algoritmo de Hopcroft-Karp [16], que resolve o mesmo problema em tempo $O(n^{5/2})$. Em 1955, Kuhn [20] propôs um algoritmo que resolve o Problema 2.3, de emparelhamentos em grafos bipartidos com peso, em tempo $O(n^2m)$. Os três algoritmos [11, 16, 19, 20] utilizam técnicas baseadas em caminhos aumentadores para resolver o problema.

Além disso, é possível resolver o problema de encontrar um emparelhamento máximo em um grafo bipartido utilizando um algoritmo que resolve o problema de encontrar o fluxo máximo em redes realizando pequenas alterações no grafo de entrada [14].

Um **grafo direcionado** é um grafo em que as arestas são pares ordenados de vértices. Para diferenciar, as arestas passam a ser chamadas de arcos. A discussão e definição do problema de encontrar o fluxo máximo em redes apresentadas a seguir foram baseadas principalmente no livro de Kleinberg e Tardos [18].

O problema de fluxo aparece em grafos que representam redes de transporte. Pode ser aplicado a uma rede hidráulica, em que se deseja transportar água pelo encanamento; à internet, em que se deseja enviar dados de um ponto a outro; ou a uma malha ferroviária, entre outras situações semelhantes. Chamaremos de **fluxo** a massa que se deseja transportar de um ponto a outro, seja água, dados, trens, ou qualquer outra coisa que a rede transporte. A seguir definimos o problema do fluxo máximo em redes.

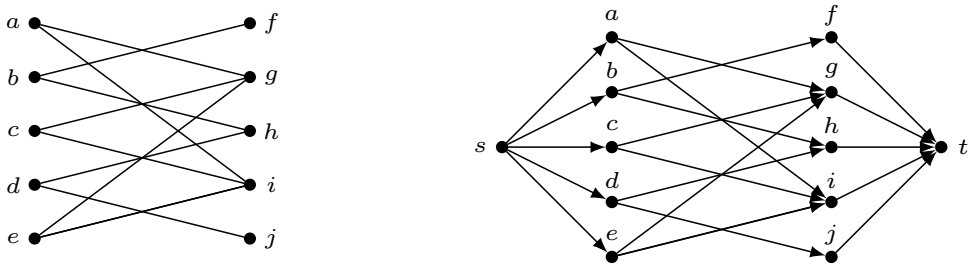
Problema 4.1. ***Fluxo máximo em redes.*** *Seja G um grafo direcionado, c uma função que atribui pesos aos arcos representando sua capacidade, e $s, t \in V(G)$ dois vértices especiais denominados, respectivamente, origem e destino.*

A origem s é uma fonte geradora de fluxo f, que é uma função que atribui valores aos arcos. O destino t é capaz de absorver o fluxo que recebe. O fluxo f enviado por s tem que ser necessariamente igual ao fluxo recebido por t. Todos os outros vértices $v \in V(G), v \neq s, t$, são denominados vértices internos da rede. O fluxo f que entra em um vértice interno tem que ser necessariamente igual ao fluxo que sai dele. O fluxo f que passa por um arco deve ser menor ou igual à sua capacidade c.

Deseja-se enviar o maior fluxo possível de s a t respeitando-se a capacidade máxima dos arcos.

Para transformar o grafo $G := G[X, Y]$ de entrada do Problema 2.1 em uma

entrada do Problema 4.1, seguimos as seguintes etapas: atribuir o mesmo sentido a todas as arestas de G, de X para Y; adicionar um vértice extra s para ser a origem e um vértice extra t para ser o destino; adicionar arcos ligando s a todos os vértices de X; adicionar arcos ligando todos os vértices de Y a t; atribuir capacidade 1 para todos os arcos. O processo está exemplificado na Figura 9.



(a) Exemplo de grafo bipartido $G := G[X, Y]$.

(b) Rede construída a partir do grafo bipartido G.

Figura 9: Representação da transformação de uma entrada do Problema 2.1 em uma entrada do Problema 4.1.

Existe um resultado denominado Teorema da Integralidade [14] que garante que como as capacidades dos arcos são inteiras, então existe uma solução ótima para o Problema 4.1 com fluxo inteiro em cada arco. Como cada vértice do conjunto X recebe fluxo no máximo 1, só será escolhido um arco saindo dele, e como de cada vértice do conjunto Y sai fluxo no máximo 1, também só será escolhido um arco que entra nele. Veja que os arcos com fluxo 1 entre X e Y irão formar o emparelhamento. Na Figura 9, por exemplo, uma possibilidade seria escolher o conjunto $\{ag, bh, ci, dj\}$ e seria impossível incluir os arcos eg ou ei também porque entraria fluxo 2 em g ou i e só sai fluxo 1 de cada um deles.

Dessa forma, o maior fluxo é necessariamente o maior número de arestas de um emparelhamento e, por isso, algoritmos que resolvem o Problema 4.1 são importantes para o estudo de emparelhamentos em grafos.

Em 1962, Ford e Fulkerson [14], em seu livro *Flows in Networks*, apresentaram um algoritmo para o Problema 4.1, de fluxo máximo em redes, de complexidade $O(mf^*)$ em que f^* é a quantidade de caminhos que podem ser encontrados entre s e t. Entretanto, para grafos cujo número de arestas é $\theta(n^2)$ a eficiência desse algoritmo fica muito ruim. Em 1967, Balinski [2] propôs outro algoritmo para resolver o mesmo problema, cuja complexidade é $O(n^3)$, e depende apenas da quantidade de vértices do grafo. Dinic [7], em 1970, propôs mais um algoritmo que resolve o problema no mesmo tempo $O(n^3)$. Dois anos depois, Edmonds e Karp [10] conseguiram melhorar esse tempo apresentando um algoritmo que executa em tempo $O(nm^2)$. Para isso, eles utilizaram uma busca em largura para otimizar o algoritmo de Ford e Fulkerson, que buscava caminhos arbitrariamente. Três anos mais tarde, em 1975, Even e

Tarjan [13] observaram que o algoritmo de Dinic executa em tempo $O(m\sqrt{n})$ nos grafos em que todos os arcos têm capacidade 1.

Sobre o problema de encontrar um emparelhamento máximo em grafos gerais (Problema 2.2), Edmonds [9] propôs, em 1965, um algoritmo de complexidade $O(n^2m)$ que será descrito na Seção 6. Em 1974, Kameda e Munro [17] melhoraram esse tempo para $O(nm)$. O algoritmo de Even e Kariv [12], de 1975, executa em tempo $O(n^{2.5})$. Gabow [15] e Lawler [21], no ano seguinte, apresentaram algoritmos diferentes que resolvem o problema em tempo $O(n^3)$. Em 1980, Micali e Vazirani [22] criaram um algoritmo de complexidade $O(m\sqrt{n})$. Em 1990, dez anos mais tarde, Blum [3] propôs um novo algoritmo que resolve o problema no mesmo tempo, $O(m\sqrt{n})$. Esse tempo de $O(m\sqrt{n})$ é o melhor que conhecemos até hoje [8].

Edmonds também foi o primeiro a apresentar um algoritmo que resolve o Problema 2.4, de encontrar um emparelhamento de peso máximo em grafos gerais [23]. A solução apresentada utiliza conceitos de programação linear e tem como subrotina o algoritmo que ele mesmo desenvolveu para o caso de grafos gerais sem peso (Problema 2.2).

5 Algoritmo de Egerváry

Nesta seção discutiremos o algoritmo proposto por Egerváry para resolver o problema de encontrar um emparelhamento de cardinalidade máxima em um grafo bipartido (Problema 2.1). Nesta seção e na Seção 6 nos baseamos principalmente na apresentação e análise realizadas por Bondy e Murty [4].

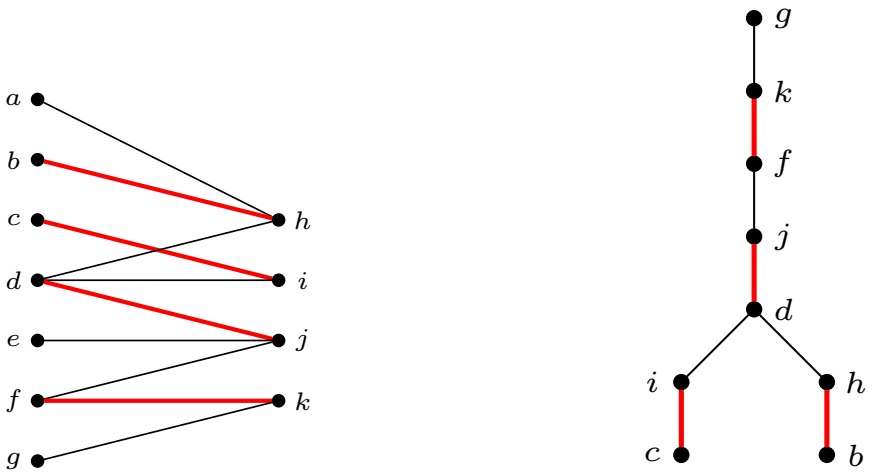
5.1 Augmenting Path Search

Os Teoremas 3.1 e 3.2 serão utilizados na análise do algoritmo denominado *Augmenting Path Search*, ou *APS* [4]. Trata-se de um algoritmo que recebe um grafo bipartido G, um emparelhamento M e um vértice u não coberto por M e ou encontra um u-caminho M-aumentador em G ou garante que esse caminho não existe. O pseudocódigo está representado no Algoritmo 1 e seu funcionamento é explicado a seguir.

A ideia do algoritmo é receber um emparelhamento M e um vértice descoberto u e crescer uma u-árvore M-alternante T porque se um caminho M-aumentador existir, certamente estará nessa árvore.

Seja G um grafo bipartido, u um vértice de G, M um emparelhamento de G e T uma árvore de G com raiz em u coberta por M. Se T é uma u-árvore M-coberta maximal, dizemos que T é uma **árvore APS**. A Figura 10b mostra um exemplo de uma árvore APS construída a partir do grafo bipartido da Figura 10a com raiz no

vértice g.



(a) Exemplo de grafo bipartido $G := G[X, Y]$ e um emparelhamento qualquer.

(b) Árvore APS construída a partir do grafo bipartido G com raiz no vértice g.

Figura 10: Representação da construção de uma APS em um grafo bipartido. Arestas emparelhadas estão em vermelho.

Chamaremos de $R(T)$ o conjunto de vértices com distância par de u em T, visualizados como vértices vermelhos (*red*), e de $B(T)$ o conjunto de vértices com distância ímpar de u em T, visualizados como vértices azuis (*blue*). Dessa forma, temos uma bipartição de T, com $u \in R(T)$. Note ainda que, se T é uma árvore M-alternante e queremos aumentá-la, só faz sentido se a aresta escolhida tiver uma extremidade em $R(T)$.

O algoritmo mantém a invariante de que T é M-coberta no início de cada iteração. Antes do laço começar, ele coloca o vértice u na árvore T e no conjunto $R(T)$. A cada passo, procura por uma aresta xy tal que x esteja no conjunto $R(T)$, conforme comentado no parágrafo anterior, e y esteja no grafo mas ainda não seja da árvore e acrescenta o vértice y à arvore T (linha 5). Note que nesse momento a árvore T deixa de ser M-coberta e passa a ser M-alternante. Então, verifica se o vértice y é coberto por M. Se não for, significa que o algoritmo encontrou um caminho M-aumentador e a função termina (linhas 9 a 11). Mas, se o vértice y for coberto pelo emparelhamento M, então ele obrigatoriamente é ligado a um vértice z também coberto por M, que não está em T, pois T é M-alternante. Nesse caso, o vértice z é acrescentado à árvore T, que volta a ser M-coberta, e ao conjunto $R(T)$ (linhas 14 e 15).

O processo descrito acima se repete procurando uma nova aresta para incluir na árvore até que seja encontrado um caminho aumentador ou não exista mais nenhuma aresta no grafo com extremidade em $R(T)$ que não esteja em T. Nesse caso, o algoritmo devolve a 5-upla $(T, u(T), R(T), B(T), M(T))$ em que T é uma

Algoritmo 1 *Augmenting Path Search*

```
1: Função APS(G, M, u)
2:     V(T) = {u}
3:     R(T) = {u}
4:     E(T) = B(T) = M(T) = ∅
5:     Enquanto existe uma aresta xy com x ∈ R(T) e y ∈ V(G) \ V(T) faça
6:         V(T) = V(T) ∪ {y}
7:         B(T) = B(T) ∪ {y}
8:         E(T) = E(T) ∪ {xy}
9:         Se y não está coberto por M então
10:            M = M△E(P), em que P = uTy
11:            Devolve M
12:        Senão
13:            Seja z tal que yz ∈ M
14:            V(T) = V(T) ∪ {z}
15:            R(T) = R(T) ∪ {z}
16:            M(T) = M(T) ∪ {yz}
17:            E(T) = E(T) ∪ {yz}
18:    T = (V(T), E(T))
19:    u(T) = u
20:    Devolve (T, u(T), R(T), B(T), M(T))
```

árvore APS, $u(T)$ é o vértice escolhido como raiz de T, $R(T)$ é o conjunto de vértices vermelhos, que possui os vértices com distância par de u em T, $B(T)$ é o conjunto de vértices azuis, que possui os vértices com distância ímpar de u em T, e $M(T)$ é o emparelhamento máximo em T.

O teorema a seguir formaliza a corretude do algoritmo.

Teorema 5.1. *Dado um grafo bipartido G, um emparelhamento M de G e um vértice $u \in V(G)$ não coberto por M, o algoritmo APS encontra um u-caminho M-aumentador se e somente se ele existir.*

Demonstração. Primeiro, note que a afirmação "se o algoritmo *APS* encontra um u-caminho M-aumentador então ele existe" é claramente verdadeira.

Vamos, então, provar que se existe um u-caminho M-aumentador então o Algoritmo *APS* devolve esse caminho. Faremos isso através da contrapositiva: se o Algoritmo *APS* não devolve um u-caminho M-aumentador então esse caminho não existe. Devido ao funcionamento da Função *APS* sabemos que essa contrapositiva é equivalente a dizer que se o algoritmo devolver a 5-upla $(T, u(T), R(T), B(T), M(T))$, então não existe um u-caminho M-aumentador. Sendo assim, vamos provar essa última afirmação.

Seja G um grafo bipartido, u um vértice de G e M um emparelhamento de G. Seja T a árvore construída pelo algoritmo. Conforme a discussão anterior, no início de cada iteração ela é M-coberta. Note que T é uma árvore APS.

Então, temos:

$$|B(T)| = |R(T)| - 1 \tag{1}$$

e

$$B(T) \subseteq N_G(R(T)) \tag{2}$$

pois, como T é M-coberta, os vértices de $R(T) \setminus u(T)$ estão necessariamente emparelhados com os vértices de $B(T)$. Também sabemos que

$$N_G(R(T)) \subseteq R(T) \cup B(T) \tag{3}$$

porque T ser uma árvore maximal garante que nenhum vértice de $R(T)$ é adjacente em G a nenhum vértice de $V(G) \setminus V(T)$.

Como G é bipartido, sabemos que é impossível dois vértices vermelhos de T serem adjacentes em G. Em outras palavras, G bipartido garante que

$$N_G(R(T)) \cap R(T) = \emptyset\,. \tag{4}$$

Com a Equação (4), podemos reduzir a Equação (3) a

$$N_G(R(T)) \subseteq B(T)\,. \tag{5}$$

Finalmente, podemos concluir das Equações (2) e (5) que $N_G(R(T)) = B(T)$. Com isso, sabemos que é impossível existir um u-caminho M-aumentador em G, já que todos os vizinhos de $R(T)$ estão garantidamente na árvore APS T. □

Para finalizar a análise do algoritmo *APS*, resta realizar a análise do tempo de execução do algoritmo. Sejam n a quantidade de vértices e m a quantidade de arestas do grafo de entrada. Considere que um grafo está representado por uma lista de adjacências; um emparelhamento, por um vetor de inteiros de tamanho n em que cada posição representa um vértice e, caso o vértice esteja coberto, o conteúdo armazenado naquela posição representa o rótulo do vértice na outra extremidade da aresta emparelhada; e a representação de um caminho é feita com um vetor de tamanho n em que cada posição armazena o vértice anterior àquele no caminho. Primeiro observe que, como o grafo é conexo, $m \geq n - 1$. Agora, note que todas as linhas dentro do laço da linha 5, exceto a linha 10, executam em tempo constante. Como o algoritmo vai entrar no laço da linha 5 para adicionar arestas na árvore e, no pior caso, uma árvore tem $n - 1$ arestas, então esse laço vai executar $O(n)$ vezes, de forma que essas linhas levam tempo $O(n)$ para executar. A linha 10 executa no máximo uma vez e leva tempo $O(n)$ pois, no pior caso, todos os vértices de G estarão no caminho P e precisaremos percorrer todos os vértices de G. Por fim,

repare que a linha 5 por si só leva um tempo para executar que deve ser considerado no cálculo. Para facilitar a compreensão do tempo de execução dessa linha, podemos lê-la como "para todo $x \in R(T)$, procure $y \in N(x)$ com $y \notin V(T)$". A primeira parte dessa frase (para todo $x \in R(T)$) executa em tempo $O(n)$ pois, no pior caso, passa por todos os vértices de G. A segunda parte dessa frase (procure $y \in N(x)$ com $y \notin V(T)$), executa em tempo $O(m)$ pois, no pior caso, passa por todos os vizinhos dos possíveis vértices x, o que em uma lista de adjacências resulta em um tempo equivalente à soma dos graus do vértices. Sendo assim, a linha 5 executa em tempo $O(n + m)$. Note que a implementação deve ser feita de forma que a vizinhança de cada vértice seja percorrida uma única vez na execução da função. Finalmente, o tempo do algoritmo é composto pelo tempo de execução da linha 5 somado com os tempos de execução da linha 10 e das linhas dentro do laço. Ou seja, $T(APS) = O(n + m) + O(n) + O(n) = O(n + m)$ e, como $m \geq n - 1$, então $T(APS) = O(m)$.

5.2 Algoritmo de Egerváry

O Algoritmo 1 foi utilizado por Egerváry, um matemático húngaro, para desenvolver a função que será descrita a seguir. Quando a função APS devolve a 5-upla sabemos que $M(T)$ é o emparelhamento máximo no subgrafo T, então podemos remover essa árvore do grafo original e continuar procurando pelo emparelhamento máximo no grafo restante. O Algoritmo de Egerváry recebe um grafo bipartido G e encontra um emparelhamento máximo nesse grafo. A ideia é executar a função APS repetidas vezes e a cada vez que não for possível aumentar o emparelhamento inicial, remover a árvore APS devolvida do grafo e iniciar a busca novamente escolhendo um novo vértice como raiz até que não sobre nenhum vértice descoberto no grafo. O pseudocódigo está representado no Algoritmo 2 e seu funcionamento em detalhes é explicado a seguir.

O algoritmo parte de um grafo bipartido $G[X, Y]$ e inicia a busca por um emparelhamento máximo com um emparelhamento arbitrário M do grafo, que pode ser, por exemplo, o emparelhamento vazio. Então, o algoritmo escolhe arbitrariamente um vértice u que não esteja coberto pelo emparelhamento M e executa a função APS, descrita no Algoritmo 1, que pode devolver um novo emparelhamento M' maior que o emparelhamento M inicial, ou uma árvore T com raiz em u.

A cada iteração o algoritmo escolhe um vértice u descoberto e executa $APS(G, M, u)$. Caso a função APS devolva um emparelhamento M', o algoritmo de Egerváry simplesmente executa a função APS novamente passando como parâmetro o mesmo grafo G, um novo vértice u e o novo emparelhamento M' (linha 10).

Caso a função APS não consiga aumentar o emparelhamento passado como pa-

Algoritmo 2 *Algoritmo de Egerváry*

1: **Função** AlgoritmoDeEgervary(G, M)
2: $\tau = \emptyset$ ▷ Conjunto de árvores APS
3: $R = \emptyset$ ▷ Conjunto de vértices vermelhos
4: $B = \emptyset$ ▷ Conjunto de vértices azuis
5: $U = \emptyset$ ▷ Conjunto de vértices não cobertos por M^*
6: $M^* = \emptyset$ ▷ Emparelhamento máximo em G
7: $F = G$
8: **Enquanto** existe um vértice não coberto por M em F **faça**
9: Seja u um vértice não coberto por M
10: $aps = \text{APS}(G, M, u)$
11: **Se** aps não é um emparelhamento **então**
12: Seja $(T, u(T), R(T), B(T), M(T)) = aps$
13: $\tau = \tau \cup \{T\}$
14: $R = R \cup R(T)$
15: $B = B \cup B(T)$
16: $U = U \cup u(T)$
17: $M^* = M^* \cup M(T)$
18: $M = M \setminus E(T)$
19: $F = F - T$
20: **Senão**
21: $M = aps$
22: $M^* = M^* \cup M$
23: **Devolve** (τ, R, B, M^*, U, F)

râmetro e devolva uma árvore T com raiz em u, é possível restringir a busca pelo emparelhamento máximo de G ao subgrafo $G' = G - V(T)$ conforme discussão anterior. O algoritmo então guarda o conjunto $M(T) = M \cap E(T)$, substitui o emparelhamento M pelo emparelhamento $M \setminus E(T)$, o grafo G pelo subgrafo G', e aplica a função APS novamente, partindo de um vértice descoberto de G', se existir algum, até que a função não consiga encontrar nenhum u-caminho M-aumentador. Esse processo se repete até que não exista nenhum vértice descoberto no grafo atual.

O Algoritmo de Egerváry devolve a 6-upla (τ, R, B, M^*, U, F) em que τ é um conjunto de árvores APS, $R = \bigcup_{T \in \tau} R(T)$ é o conjunto de vértices vermelhos, $B = \bigcup_{T \in \tau} B(T)$ é o conjunto de vértices azuis, M^* é um emparelhamento máximo de G, $U = \{u(T) : T \in \tau\}$ é o conjunto de vértices não cobertos por M^*, e F é o subgrafo $G - (R \cup B)$ que tem um emparelhamento perfeito e não está contido em nenhuma das árvores APS de τ. Esse emparelhamento perfeito de F é o emparelhamento M que chegou na linha 22.

O Teorema 5.2 a seguir formaliza a corretude do Algoritmo de Egerváry.

Teorema 5.2. *Dado um grafo bipartido G e um emparelhamento arbitrário M, o emparelhamento M^* devolvido pelo Algoritmo de Egerváry é um emparelhamento máximo em G.*

Demonstração. Sejam τ, R, B e U respectivamente os conjuntos de árvores, vértices vermelhos, vértices azuis e vértices descobertos devolvidos pelo Algoritmo Egerváry. Temos que

$$|\tau| = |U| \tag{6}$$

pois cada árvore T de τ contém um único vértice descoberto, que é sua raiz.

Além disso, sabemos, da Equação (1), que para cada árvore T de τ

$$|B(T)| = |R(T)| - 1\,. \tag{7}$$

Então, para concluir que

$$|B| = |R| - |\tau| \tag{8}$$

basta somar a Equação (7) para toda $T \in \tau$. Ao substituir a Equação (6) na Equação (8), concluímos que

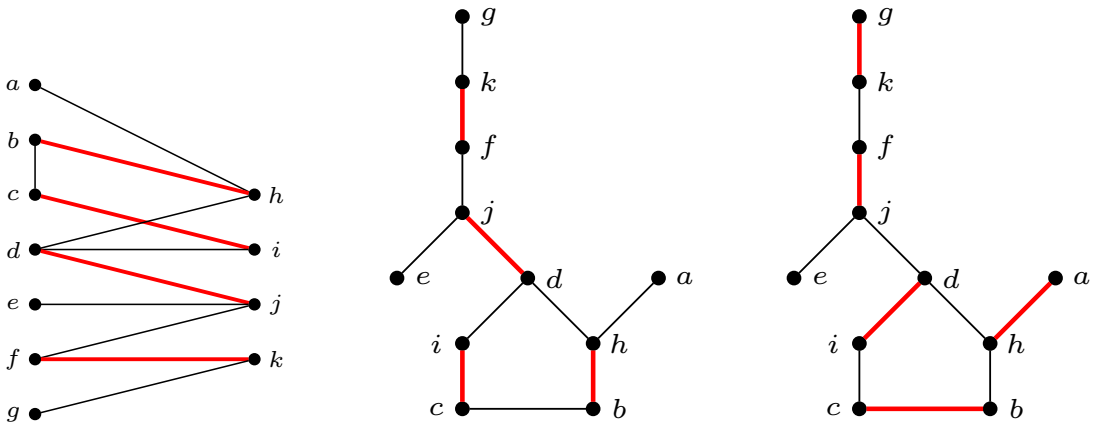
$$|U| = |R| - |B|\,. \tag{9}$$

Note que os vértices vermelhos só podem ser emparelhados com vértices azuis, pois os vértices vermelhos em G são adjacentes somente a vértices azuis já que G é bipartido. Pela definição de B todos os vértices do conjunto B estão cobertos por arestas de M. Com isso, podemos observar que existem pelo menos $|R|-|B|$ vértices vermelhos não cobertos por M. Então, pela Equação (9), existem pelo menos $|U|$ vértices vermelhos não cobertos por M. Como existe exatamente esse número de vértices não cobertos por M^*, é possível afirmar que M^* é um emparelhamento máximo. □

Com a demonstração de que o algoritmo funciona, precisamos analisar o seu tempo de execução. Sejam n a quantidade de vértices e m a quantidade de arestas. Podemos utilizar o pseudocódigo apresentado no Algoritmo 2 para observar que todas as linhas, exceto as linhas 8, 10 e 13 a 19, executam em tempo constante. A linha 8, no pior caso, precisa checar todos os vértices de G, então leva tempo $O(n)$. Como, no pior caso, o laço vai executar para todos os vértices do grafo G, então temos que o laço executa no máximo n vezes. Cada linha entre as linhas 13 a 19 leva tempo $O(n)$ para executar e a função APS leva tempo $O(m)$ para executar. Como cada uma dessas linhas é chamada a cada execução do laço (Linha 8) podemos concluir que o laço leva tempo $O(n(n+m)) = O(n^2) + O(nm)$ mas, como o grafo é conexo sabemos que $m > n - 1$, então o laço leva tempo $O(nm)$. O Algoritmo de Egerváry leva o tempo de execução da linha 8 mais o tempo de execução do laço, portanto leva tempo $O(n) + O(nm) = O(nm)$ para executar.

6 Algoritmo de Edmonds

Conforme discutido na Seção 5, a função *APS* garante encontrar um caminho aumentador se e somente se ele existir apenas quando o grafo de entrada é bipartido, pois conta com o fato de não existirem arestas entre vértices da mesma cor. Se a aresta for entre dois vértices azuis, a função *APS* ainda vai funcionar, porque em um caminho da raiz até um vértice azul, a última aresta é necessariamente desemparelhada. Portanto, a existência de uma aresta entre dois vértices azuis nunca irá gerar um novo caminho aumentador. No entanto, é necessário analisar o caso em que existe uma aresta entre dois vértices vermelhos.



(a) Exemplo de um grafo G' e um emparelhamento qualquer.

(b) Grafo G' redesenhado.

(c) Emparelhamento ótimo no grafo G'.

Figura 11: Representação da construção de uma APS em um grafo qualquer. Arestas emparelhadas estão em vermelho.

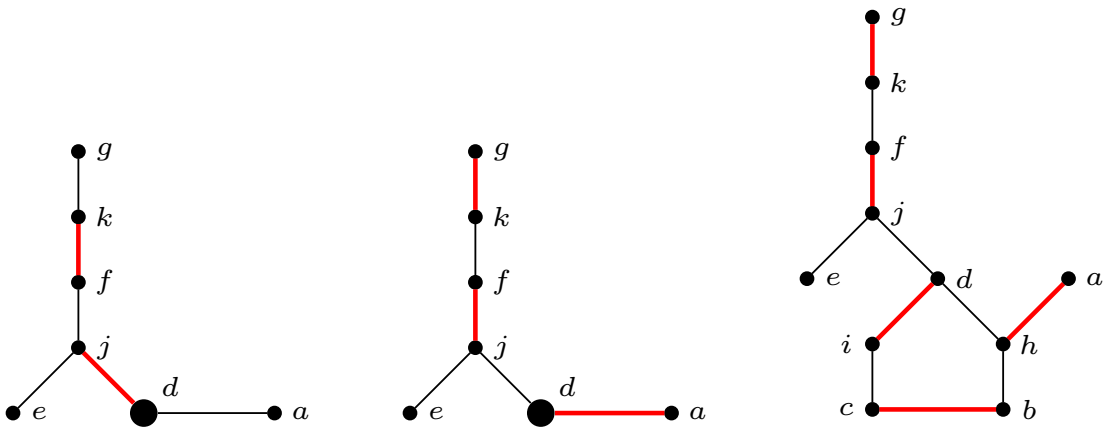
Considere o grafo bipartido da Figura 10a e a árvore APS correspondente construída com raiz em g representada na Figura 10b. Em seguida, considere acrescentar uma aresta entre os vértices b e c do grafo G original, como na Figura 11a. Isso faz com que o grafo G deixe de ser bipartido e com que exista no grafo G uma aresta entre vértices vermelhos da árvore APS construída originalmente. A Figura 11b representa o grafo da Figura 11a redesenhado. Nessa versão do grafo a árvore APS construída com o algoritmo utilizado até o momento seria a mesma da Figura 10b, mas ela não possibilitaria encontrar o caminho aumentador $(g, k, f, j, d, i, c, b, h, a)$, que passa a existir quando o grafo deixa de ser bipartido. O algoritmo *APS+* possibilita encontrar o emparelhamento M^* exibido na Figura 11c.

6.1 *APS+*

Para entender o funcionamento do algoritmo *APS+* é necessário antes definir o conceito de **botão** (***blossom***). Note que a única diferença entre um grafo bipartido

e um grafo geral é a possibilidade de existirem ciclos ímpares, e que foi a presença de um ciclo ímpar no exemplo da Figura 11 que impediu que o caminho aumentador fosse encontrado através da função *APS*. Dessa forma, justifica-se dar um tratamento especial a esses ciclos. Portanto, seja T uma árvore APS de G, e sejam a e b dois vértices vermelhos de T adjacentes em G. Então o ciclo formado em $T + ab$ sempre vai ser ímpar e vai conter um número par de vértices emparelhados e um vértice vermelho adicional, que será denominado **base** (***root***) do botão. Em outras palavras, um botão é um ciclo de tamanho $2k+1$ em que k é a quantidade de arestas emparelhadas. Na Figura 11b o ciclo (d, i, c, b, h, d) é um botão e d é a sua base.

A ideia do algoritmo *APS+* é contrair os botões em suas bases e, caso existam caminhos aumentadores no novo grafo formado, então existirá um caminho aumentador no grafo original. Contrair um botão F de G significa criar um novo grafo H a partir de G tal que $H = G/F$. O pseudovértice que representará o botão após a contração terá o mesmo rótulo da base de F. Na Figura 12a, o vértice d é a contração do botão D que é o ciclo (d, i, c, b, h, d) no grafo da Figura 11b. Na Figura 12b está representado o novo emparelhamento criado a partir do caminho aumentador (g, k, f, j, d, a) da Figura 12a. A Figura 12c mostra o emparelhamento ótimo encontrado depois da operação de descontração do botão D.



(a) Grafo H formado pelo grafo G' com o botão $D = (d, i, c, b, h)$ contraído e um emparelhamento qualquer.

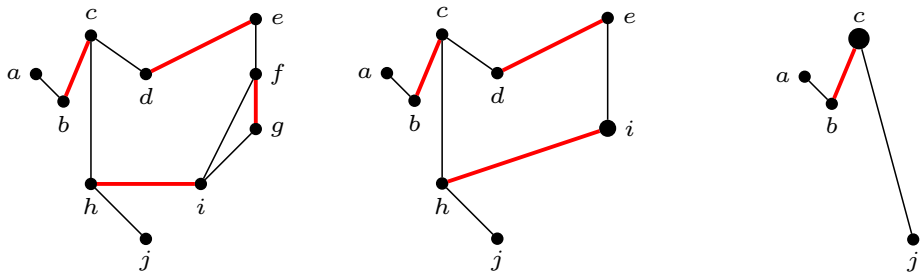
(b) Grafo H e um emparelhamento ótimo.

(c) Grafo original G' e um emparelhamento ótimo.

Figura 12: Representação da operação de contrair e descontrair um botão para encontrar caminhos aumentadores. Arestas emparelhadas estão em vermelho.

O subgrafo de G que dá origem ao pseudovértice é denominado **flor** (***flower***) de G. Uma flor é composta de um ou mais botões do grafo original. Na Figura 13a o subgrafo induzido pelo conjunto de vértices $\{c, d, e, f, g, h, i\}$ é uma flor de G composta por dois botões que dá origem ao pseudovértice c na Figura 13c.

O pseudocódigo da função *APS+* descrita acima é apresentado no Algoritmo 3.



(a) Exemplo de grafo G com um emparelhamento qualquer.

(b) Grafo G' com botão $I = (i, f, g)$ do grafo G contraído.

(c) Grafo G'' com botão $C = (c, d, e, i, h)$ do grafo G' contraído.

Figura 13: Exemplo de flor composta de dois botões (I e C) do grafo. Arestas emparelhadas estão em vermelho.

A função $APS+$ é muito semelhante à função APS apresentada no Algoritmo 1. A principal diferença entre elas é a operação de contração e descontração do botão que foi acrescentada entre as linhas 15 e 30. Como essa operação utiliza uma chamada recursiva da função na linha 18, também há uma pequena diferença entre os parâmetros recebidos pela função APS e pela função $APS+$, pois no Algoritmo 1 a árvore APS é inicializada apenas com o vértice u entre as linhas 2 e 4 e não queremos reinicializar a árvore que está sendo construída a cada chamada recursiva. Assim, a função $APS+$ recebe a árvore e os conjuntos de vértices vermelhos e azuis como parâmetros. A última diferença está no retorno da função quando é encontrado um caminho aumentador. A função $APS+$ devolve, além do novo emparelhamento, o caminho aumentador encontrado, para que caso haja um botão contraído seja possível realizar a operação de descontração e encontrar o novo emparelhamento no grafo anterior à contração desse botão. Dado um grafo G qualquer, um emparelhamento arbitrário M em G e u um vértice de G não M-emparelhado, criamos $T = (\{u\}, \emptyset)$, uma árvore que contém apenas o vértice u, e a primeira chamada deve ser $APS + (G, M, u, T, \emptyset, \emptyset, \emptyset)$.

Vamos, agora, provar que sempre que existir um caminho aumentador no subgrafo sem o botão, existirá um caminho aumentador no grafo original.

Teorema 6.1. *Seja G um grafo, M um emparelhamento em G e $u \in V(G)$ um vértice não coberto por M. Seja F um botão em G, $H = G/F$, $M' = M/F$. Note que $u \in V(G)$ e $u \in V(H)$. Existe um u-caminho M'-aumentador em H se e somente se existe um u-caminho M-aumentador em G.*

Demonstração. Primeiro, vamos provar que se existe um u-caminho M'-aumentador em H, então existe um u-caminho M-aumentador em G.

Suponha que existe um u-caminho M'-aumentador P em H. Seja u_F o vértice que representa o botão F contraído. Se P não passa por u_F, então P é um caminho aumentador em G sem necessidade de qualquer alteração. Se P passa por u_F,

Algoritmo 3 *Augmenting Path Search +*

Função APS+(G, M, u, T, $R(T)$, $B(T)$, $M(T)$)
  Enquanto existe uma aresta xy com $x \in R(T)$ e $y \in V(G) \setminus V(T)$ **faça**
    $V(T) = V(T) \cup \{y\}$
    $B(T) = B(T) \cup \{y\}$
    $E(T) = E(T) \cup \{xy\}$
    Se y não está coberto por M **então**
      $M = M \triangle E(P)$, em que $P = uTy$
      Devolve (M, P)
    Senão
      Seja z tal que $yz \in M$
      $V(T) = V(T) \cup \{z\}$
      $R(T) = R(T) \cup \{z\}$
      $M(T) = M(T) \cup \{yz\}$
      $E(T) = E(T) \cup \{yz\}$
      Se existe $h \in N(z) \cap R(T)$ **então** ▷ Encontrou botão em G
        Seja F o botão encontrado em G
        Seja w a base do botão ▷ Note que não necessariamente $h = w$
        aps = APS+$(G/F, M/F, u, T/F, R(T)/F, B(T) \setminus V(F), M(T)/F)$
        Se aps é um par (M', P') **então**
          Seja M' o emparelhamento encontrado em G/F
          Seja P' o caminho aumentador encontrado em G/F
          Se P' passa por w **então**
            Seja $P' = (u, \ldots, v_1, w, v_2, \ldots, t)$
            Seja $k \in N(v_2)$ tal que $k \in F$
            Seja $k' \in F$ tal que $kk' \in M$
            Faça $P = (u, \ldots, v_1, w, \ldots, k', k, v_2, \ldots, t)$
          Senão
            Faça $P = P'$
          $M = M' \cup [M \triangle (E(P) \cap E(F))]$
          Devolve (M, P)
  $u(T) = u$
  Devolve $(T, u(T), R(T), B(T), M(T))$

podemos escrever P como $P = (u, P', v, u_F, v', P'')$, em que P' e P'' são subcaminhos. Como P é aumentador, ou vu_F é emparelhada ou $u_F v'$ é emparelhada. Vamos supor que $vu_F \in M'$ e, portanto, $u_F v' \notin M'$. Seja w_0 a base do botão F e seja $w_1, w_2, ..., w_{k-1}, w_k$ os demais vértices de F. Considere que P''' é o caminho em F de w_0 até algum w_j que termina com uma aresta emparelhada. Sendo assim, o caminho Q em G pode ser escrito como $Q = (u, P', v, w_0, P''', w_j, v', P'')$, que é M-aumentador pela forma como foi construído.

Agora, vamos provar que se existe um u-caminho M-aumentador em G, então existe um u-caminho M'-aumentador em H.

Primeiro, note que qualquer botão é um ciclo ímpar de tamanho $2k+1$ em que k é o número de arestas emparelhadas e que todos os vértices de F exceto a base estão emparelhados por arestas de F. Note, também, que qualquer ciclo ímpar dividido em dois caminhos terá um caminho de comprimento par e um caminho com comprimento ímpar. Observe agora que, em um caminho aumentador, qualquer subcaminho de comprimento par é alternante e começa e termina com arestas de tipos diferentes, ou seja, se R é um caminho aumentador, qualquer subcaminho de R de comprimento par que começa com uma aresta emparelhada termina com uma aresta desemparelhada e vice-versa. Além disso, sabemos que ao remover um subcaminho de comprimento par de um caminho aumentador, substituindo o subcaminho por um pseudovértice, o caminho resultante será necessariamente aumentador. Por fim, repare que as arestas do botão ligadas à base nunca estarão M-emparelhadas.

Agora, assuma que existe um u-caminho M-aumentador P em G e que G tem um botão F. Se P não passa pelos vértices de F, claramente P é M'-aumentador em H. Se P passa pelos vértices de F, é necessário analisar a interseção de P e F. Queremos garantir que $P' = P/F$ seja aumentador. Sabemos que P é aumentador, então todos os seus subcaminhos pares são alternantes. Então resta provar que $P \cap F$ é um caminho par, para conseguir mostrar que P' é aumentador. Note que o botão pode ser dividido em duas partes, uma pela qual passa o caminho P e a outra não. Ao dividir um ciclo ímpar em duas partes, sempre existirá uma delas que é par e uma ímpar. Se $P \cap F$ fosse um caminho ímpar, em outras palavras, se a parte do botão F pela qual passa P fosse a parte ímpar, então $P \cap F$ iria começar em uma aresta que não pertence a M, pois as arestas do botão ligadas a base nunca estarão M-emparelhadas, e terminaria em uma aresta desemparelhada também, já que P é alternante e $P \cap F$ é um subcaminho ímpar. Sabemos, pela definição, que todos os vértices do botão F, exceto a base, são emparelhados por arestas de F. Então a aresta de P que toca o último vértice de $P \cap F$ e um vértice que não pertence a F não pode ser emparelhada, porque o último vértice de $P \cap F$ está emparelhado por uma aresta do botão F. Mas, se a última aresta do subcaminho $P \cap F$ é desemparelhada e a aresta que toca o último vértice de $P \cap F$ e um vértice que

não pertence a F também é desemparelhada, então P não pode ser alternante, o que gera uma contradição. Assim, temos certeza que $P \cap F$ é um caminho par. Dessa forma, podemos garantir que existirá um caminho M'-alternante em H, pois ao retirar de um caminho aumentador um subcaminho par alternante temos certeza que o caminho resultante será aumentador também. □

A corretude do *APS+* é formalizada pelo teorema abaixo.

Teorema 6.2. *Dado um grafo G qualquer, um emparelhamento M em G e um vértice $u \in V(G)$ não coberto por M, o algoritmo APS+ encontra um u-caminho M-aumentador se e somente se ele existir.*

Demonstração. A prova se dá por indução no número x de botões no grafo.

Se $x = 0$, o algoritmo se comporta da mesma forma que a função APS original, que já provamos, no Teorema 5.1, que encontra um u-caminho M-aumentador se e somente se ele existir.

Seja então $x \geq 1$ e assuma que o *APS+* funciona quando há menos de x botões. Na linha 18, a chamada recursiva é feita sobre G/F, que é um grafo com $x - 1$ botões. Então, por hipótese de indução, o *APS+* encontra um u-caminho M-aumentador em G/F se e somente se tal caminho existir. Entre as linhas 22 a 28 acontece a construção do caminho a partir da operação de descontração do botão. Pelo Teorema 6.1 um u-caminho M-aumentador existe em G se e somente se existe um u-caminho M/F-aumentador em G/F. Dessa forma, podemos garantir que o algoritmo *APS+* encontra um u-caminho M-aumentador se e somente se ele existir. □

Finalmente, vamos realizar a análise da complexidade do tempo de execução da função *APS+*. Sejam n a quantidade de vértices e m a quantidade de arestas. Considere que um grafo está representado por uma lista de adjacências; um emparelhamento está representado por um vetor de inteiros de tamanho n em que cada posição representa o rótulo de um vértice e o conteúdo armazenado, caso o vértice esteja coberto, representa o rótulo do vértice na outra extremidade da aresta emparelhada; e a representação de um caminho é feita com um vetor de tamanho n em que cada posição armazena o vértice anterior àquele no caminho.

Note que a cada execução da função ocorre uma única chamada recursiva. Assim, a árvore de recursão da função tem o formato de um caminho. Além disso, note que apesar de ser uma função recursiva, apenas uma árvore APS será construída ao longo de todas as chamadas. Então o tempo total de execução da função é o tempo de construção de uma árvore APS somado ao tempo necessário para identificar, contrair e descontrair todos os botões do grafo de entrada.

A identificação dos botões acontece nas linhas 15 e 16, que levam tempo $O(n)$ pois o vértice pode ser vizinho de todos os outros do grafo e o ciclo pode conter todos os vértices do grafo também. O menor botão possível é um ciclo de comprimento 3. Portanto, a maior quantidade possível de botões que um grafo pode ter é $n/3$. A operação de contração leva tempo $O(m+n)$ se for necessário criar um grafo novo. A operação de descontração acontece entre as linhas 19 e 30. As linhas 23 a 26 levam tempo $O(n)$ pois é necessário percorrer o ciclo e o caminho. A linha 29 custa tempo $O(n)$, porque precisa percorrer o vetor que representa o caminho para encontrar a interseção entre as arestas do caminho construído e as arestas do botão, depois precisa percorrer um vetor de tamanho n para calcular a diferença simétrica e, por fim, percorrer outro vetor de tamanho n para encontrar a união entre os dois emparelhamentos. Todas as outras linhas entre as linhas 19 e 30 executam em tempo constante. A descontração, então, executa em tempo $O(n) + O(n) = O(n)$. Encontrar, contrair e descontrair um botão, portanto, leva tempo $O(n)+O(m+n)+O(n) = O(m+n)$. Como isso pode acontecer $n/3 = O(n)$ vezes, então temos que o tempo para contrair e descontrair todos os botões do grafo é $O(n) \cdot O(m+n) = O(mn+n^2)$.

O tempo de construção de uma árvore APS foi discutido na análise do algoritmo *APS* e, no pior caso, é $O(m)$. Portanto, o tempo total de execução da função, juntando a construção da árvore com a contração e descontração dos botões resulta em $O(m) + O(mn+n^2) = O(m+mn+n^2)$. Como o grafo é conexo, temos que $m \geq n-1$. Então o tempo $O(m+mn+n^2)$ da função *APS+* pode ser escrito como $O(mn)$.

6.2 Algoritmo de Edmonds

O Algoritmo conhecido como Algoritmo de Edmonds utiliza a ideia do algoritmo de Egerváry, cujo pseudocódigo é apresentado no Algoritmo 2, trocando apenas a função *APS* da linha 10 pela função *APS+*. Sendo assim, sua complexidade é $O(n) \cdot O(mn) = O(n^2m)$ e seu funcionamento é garantido pela corretude da função *APS+*.

7 Implementação

Para complementar o estudo teórico realizado, implementamos o Algoritmo de Egerváry, descrito na Seção 5. A implementação, realizada na linguagem C, está disponível em `https://github.com/bfchicaroni/Hungarian_algorithm`. A estrutura escolhida para representar o grafo foi uma lista de adjacências para garantir a complexidade de tempo mencionada. O controle do laço da linha 5 do Algoritmo 1 foi realizado utilizando uma fila de vértices, adicionando-os à fila sempre que fossem

adicionados à árvore como um vértice vermelho e removendo-os da fila sempre que não houvessem mais vizinhos a serem explorados. Isso faz com que os vértices sejam explorados em uma sequência semelhante à que aconteceria em uma busca em largura. Também foi implementada uma função auxiliar que recebe um grafo e gera um emparelhamento arbitrário gulosamente, para que não fosse necessário passar o emparelhamento vazio como parâmetro para o Algoritmo de Egerváry.

Para testar o funcionamento do algoritmo foram gerados 36 grafos aleatórios com a ferramenta *SageMath*[1], com a qual também é possível obter um emparelhamento máximo para cada grafo gerado. Dado $G := G[X, Y]$ um grafo bipartido, a ferramenta permite escolher a quantidade de vértices em X e em Y e a probabilidade de existência das arestas. Foram gerados 30 grafos combinando dois a dois os valores do conjunto $N = \{10, 25, 50, 100, 200, 500\}$ para determinar a quantidade de vértices em X e em Y e cada combinação foi testada com probabilidades 0,1 e 0,5. Os outros 6 grafos foram gerados com probabilidade 0,1 e quantidades iguais de vértices em X e em Y com os 6 valores do conjunto N. A Tabela 1 mostra as combinações utilizadas, o tamanho do emparelhamento máximo naquela instância, e o nome do arquivo que está disponível no diretório *instances* do repositório.

Note que, para implementar o Algoritmo de Edmonds, que foi descrito na Seção 6, seria necessário apenas implementar a operação de contração e descontração do botão. Por uma questão de tempo, decidimos não implementar esse algoritmo visto que o objetivo do trabalho, de reforçar o conhecimento da aluna sobre resultados clássicos da computação, já foi atingido.

[1]Veja mais em https://www.sagemath.org/.

Tabela 1: Combinações de entradas utilizadas para gerar grafos aleatórios com a ferramenta SageMath e o nome do arquivo gerado por cada combinação.

Probabilidade de uma aresta existir	$\|X\|$	$\|Y\|$	$\|M^*\|$	**Nome do arquivo de teste**
0,5	10	25	10	instance_10_25_5.txt
0,5	10	50	10	instance_10_50_5.txt
0,5	10	100	10	instance_10_100_5.txt
0,5	10	200	10	instance_10_200_5.txt
0,5	10	500	10	instance_10_500_5.txt
0,5	25	50	25	instance_25_50_5.txt
0,5	25	100	25	instance_25_100_5.txt
0,5	25	200	25	instance_25_200_5.txt
0,5	25	500	25	instance_25_500_5.txt
0,5	50	100	50	instance_50_100_5.txt
0,5	50	200	50	instance_50_200_5.txt
0,5	50	500	50	instance_50_500_5.txt
0,5	100	200	100	instance_100_200_5.txt
0,5	100	500	100	instance_100_500_5.txt
0,5	200	500	200	instance_200_500_5.txt
0,1	10	10	7	instance_10_10_1.txt
0,1	25	10	8	instance_25_10_1.txt
0,1	25	25	22	instance_25_25_1.txt
0,1	50	10	10	instance_50_10_1.txt
0,1	50	25	25	instance_50_25_1.txt
0,1	50	50	49	instance_50_50_1.txt
0,1	100	10	10	instance_100_10_1.txt
0,1	100	25	25	instance_100_25_1.txt
0,1	100	50	50	instance_100_50_1.txt
0,1	100	100	100	instance_100_100_1.txt
0,1	200	10	10	instance_200_10_1.txt
0,1	200	25	25	instance_200_25_1.txt
0,1	200	50	50	instance_200_50_1.txt
0,1	200	100	100	instance_200_100_1.txt
0,1	200	200	200	instance_200_200_1.txt
0,1	500	10	10	instance_500_10_1.txt
0,1	500	25	25	instance_500_25_1.txt
0,1	500	50	50	instance_500_50_1.txt
0,1	500	100	100	instance_500_100_1.txt
0,1	500	200	200	instance_500_200_1.txt
0,1	500	500	500	instance_500_500_1.txt

Referências

[1] R. P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

[2] M. L. Balinski. Labelling to obtain a maximum matching. In *Combinatorial Mathematics and Its Applications (Proceedings Conference Chapel Hill*, pages 585–602, 1967.

[3] N. Blum. A new approach to maximum matching in general graphs. In *Automata, Languages and Programming: 17th International Colloquium Warwick University*, pages 586–597. Springer, 1990.

[4] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008. ISBN 9781846289699.

[5] W. Cook and A. Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2):138–148, 1999.

[6] C. M. H. de Figueiredo and J. L. Szwarcfiter. Emparelhamento em grafos, algoritmos e complexidade. In *Anais da XVII Jornada de Atualização em Informática*, 1999.

[7] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Mathematics Doklady*, 11(5):1277–1280, 1970. English translation by RF. Rinehart.

[8] G. Ducoffe. Maximum matching in almost linear time on graphs of bounded clique-width. *Algorithmica*, 84(11):3489–3520, 2022.

[9] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17: 449–467, 1965.

[10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. ISSN 0004-5411. doi: 10.1145/321694.321699.

[11] E. Egerváry. Matrixok kombinatorius tulajdonságairol. *Matematikai és Fizikai Lapok*, 38:16–28, 1931. Tradução em inglês por H. W. Kuhn.

[12] S. Even and O. Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *16th Annual Symposium on Foundations of Computer Science (SFCS 1975)*, pages 100–112. IEEE, 1975.

[13] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

[14] L. R. Ford Jr and D. R. Fulkerson. *Flows in networks*, volume 56. Princeton university press, 2015.

[15] H. N. Gabow. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM (JACM)*, 23(2):221–234, 1976.

[16] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[17] T. Kameda and I. Munro. A $O(|V| \cdot |E|)$ algorithm for maximum matching of graphs. *Computing*, 12(1):91–98, 1974.

[18] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.

[19] H. W. Kuhn. On combinatorial properties of matrices. *George Washington University Logistics Papers*, 11:1–11, 1955.

[20] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[21] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[22] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (SFCS 1980)*, pages 17–27. IEEE, 1980.

[23] M. D. Plummer and L. Lovász. *Matching theory*. Elsevier, 1986.

[24] S. S. Skiena. *The Algorithm Design Manual*. Texts in Computer Science. Springer, third edition, 2020. ISBN 9783030542559.