Comparação entre uma solução combinatória e um método de planos-de-corte para o problema do emparelhamento de peso máximo

Ander Conselvan de Oliveira

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES.

São Paulo, novembro de 2010

Comparação entre uma solução combinatória e um método de planos-de-corte para o problema do emparelhamento de peso máximo

Este exemplar corresponde a redação final da dissertação devidamente corrigida e defendida por Ander Conselvan de Oliveira e aprovada pela comissão julgadora.

Comissão Julgadora:

- Prof. Dr. Carlos Eduardo Ferreira (orientador) - IME-USP
- Profa. Dra. Yoshiko Wakabayashi - IME-USP
- Prof. Dr. Marcelo Henriques de Carvalho - UFMS

Agradecimentos

Muitas pessoas contribuíram para a execução desse trabalho, seja de forma direta ou indireta. Agradeço aqui a todas elas. Em especial, gostaria de agradecer a minha família por todo afeto, apoio moral e financeiro. A meus pais por todas as razões óbvias e as não tão óbvias. Todo o esforço que eles tiveram para dar uma educação de qualidade para seus filhos. Os computadores e livros que eles compraram, mesmo questionando as suas utilidades. E por me aturarem por tanto tempo.

A meu avô Dante, por todo o afeto, apoio moral e os brindes desejando "sucesso de vocês que estão estudando". E "viva nossas boas qualidades que filha-das-puta nenhum reconhece".

Agradeço meu irmão mais velho, Allan, por me ensinar a programar quando eu tinha pouco mais de 10 anos, por me aturar em sua casa por bastante tempo e por ter esperado até minha defesa para abrir aquela champagne!

A todos os meus irmãos (Allan, Alex e Axel), pela amizade e companherismo com que pude contar até aqui e com que espero contar no futuro.

A meu orientador, Carlinhos, pelos sábios conselhos, tanto os acadêmicos que resultaram em grandes melhorias nessa dissertação e em minha formação, quanto os de cunho pessoal e profissional. Agradeço também sua paciência, visto que levei quase o dobro do tempo esperado para realizar esse trabalho.

Às professoras Yoshiko e Cristina do IME-USP, pelas excelentes aulas em que aprendi muito. Agradeço também os professores do Departamento de Informática da UFPR e, em especial, os professores Jair Donadelli Jr., Marcos Castilho, Fabiano Silva e Luiz Carlos Erpen de Bona, que despertaram meu interesse pelo mundo acadêmico.

Agradeço os colegas do NUMEC, e em especial, o Álvaro, que sempre quebrou os meus galhos quando eu estava longe de São Paulo e sempre me oferecia um lugar para ficar. Agraço também os colegas do C3SL, Tiago, Márcio, Russo, Picussa, Guto, Bina e Ju, e do Tonturinha Futebol e Cachaça por tornarem meus anos na UFPR muito mais divertidos.

Agradeço a meus empregadores, CITS, Mandriva e Userful Corporation, que permitiram de alguma forma que eu continuasse o mestrado, mesmo trabalhando. Agradeço a todos os colegas da Mandriva e especialmente ao Wanderlei Cavassin, pela compreensão e flexibilidade oferecida durante minha permanência sob sua gerência.

A meus amigos do segundo grau, Crus, Molanda, Max e Evelyn e aos amigos/primos/pseudo-primos Bivis, Yuri e Negão por toda diversão que tivemos nesse tempo todo.

Agradeço aos amigos da "pensão do Anderson", Marquinhos, Cris, Vicente, Vera e etc, pelas "merendas", festas, cafés, conversas e outros.

E a todas as pessoas que foram e vieram nesse tempo todo, que, de alguma forma, fizeram parte da minha vida e me influenciaram enquanto eu desenvolvia esse trabalho.

Muito obrigado!

Resumo

Um emparelhamento em um grafo é um conjunto de arestas duas a duas não adjacentes. Dado um grafo G com pesos em suas arestas, o problema do emparelhamento de peso máximo é encontrar um emparelhamento cuja soma dos pesos de suas arestas é máxima. Neste trabalho estudamos diferentes soluções para esse problema.

Estudamos algoritmos combinatórios que resolvem o problema no caso em que G é bipartido e no caso geral. O algoritmo de Edmonds [Edm65a] é um algoritmo polinomial cuja complexidade de tempo é $O(n^4)$, onde n é o número de vértices do grafo G. Discutimos nesse trabalho nossa implementação desse algoritmo.

Num trabalho de 1985, Grötschel e Holland [GH85] propuseram o uso de ferramentas de programação linear para resolver o mesmo problema. O método chamado de planos-de-corte baseia-se em um resultado de Padberg e Rao [PR82] de que o problema da separação associado ao poliedro dos emparelhamentos pode ser resolvido em tempo polinomial.

Neste trabalho fizemos implementações dos dois métodos e os utilizamos para resolver diversos tipos de instâncias do problema. Nossa conclusão é que o método poliédrico, apesar de utilizar ferramentas genéricas, é bastante eficiente na prática.
Palavras-chave: emparelhamento, plano-de-corte, otimização combinatória.

Abstract

A matching in a graph G is a set of pairwise disjoint edges of G. Given a graph G with edge weights, we define the maximum weight matching problem as that of finding a matching which maximizes the sum of its weights. In this thesis we study different solutions to this problem.

We studied combinatorial algorithms that solve this problem in the case where G is bipartite and also in the general case. Edmonds' algorithm [Edm65a] is a polynomial time algorithm with complexity $O(n^4)$, where n is the number of vertices in the graph G. We discuss in this document our implementation of this algorithm.

In a paper from 1985, Grötschel & Holland [GH85] discussed the use of linear programming tools for solving the maximum weight matching problem. This so called cut-plane method relies on a result by Padberg & Rao [PR82] that proves that the separation problem associated with matching polyhedron is solvable in polinomial time.

In this work we implemented both methods and used then to solve different instances of the problem. Our conclusion is that the polyhedral method, although using generical tools is very efficient in practice.
Keywords: matching, cut-planes, combinatorial optimization.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Um emparelhamento num grafo é um conjunto de arestas duas a duas não adjacentes. O *Problema do Emparelhamento Máximo* é definido como segue: dado um grafo $G = (V, E)$ e uma função $w : E \to \mathbb{R}$ que atribui pesos às arestas de G, encontrar um emparelhamento M que maximiza

$$\sum_{e \in M} w(e).$$

Claramente, um emparelhamento nas condições acima não contém arestas de peso negativo. Portanto, assumiremos pesos não negativos quando conveniente.

Neste trabalho exploramos o problema do emparelhamento de peso máximo sob duas óticas diferentes. A primeira, aborda o problema do ponto de vista combinatório, mostrando soluções propostas na literatura e visitando resultados clássicos de teoria dos grafos (Parte I). Em seguida, mudaremos nosso ponto de vista para a combinatória poliédrica, explorando a relação entre o problema e certos objetos num espaço n dimensional (Parte II). Por fim, discutimos resultados computacionais de implementações de soluções computacionais para esse problema baseados nos dois pontos de vista (Parte III).

Uma das motivações para tal abordagem é a relevância de um resultado devido a Edmonds para a combinatória poliédrica. Esse resultado foi a primeira caracterização de um poliedro para um problema de otimização combinatória e contribuiu muito para o desenvolvimento da área.

No capítulo 2, discutimos algoritmos combinatórios para esse problema no caso de grafos bipartidos e no caso genérico. No capítulo 3 apresentamos em detalhes nossa implementação do algoritmo de Edmonds, incluindo nossa adaptação das idéias de Lovász e Plummer [LP86] para evitar operações de contração de vértices.

No capítulo 4 discutimos um problema que, apesar de combinatório, tem forte relação com a abordagem poliédrica que faremos na parte II. No capítulo seguinte (capítulo 5), discutimos o conceito de métodos de planos-de-corte e apresentamos um método de planos-de-corte proposto por Grötschel e Holland [GH85].

Por fim, discutiremos a implementação que fizemos de dois algoritmos diferentes para esse problema. Cada uma alinhada com um dos "pontos de vista" diferentes que mencionamos. No final deste texto, apresentamos resultados computacionais comparativos entre os dois.

A próxima seção apresenta alguns resultados que serão usados ao longo do texto.

1.1 Notação, definições e alguns resultados

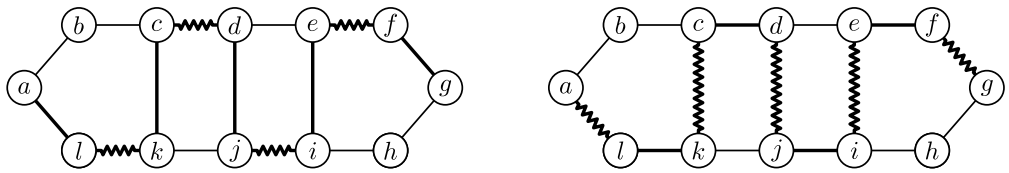
Sejam $G = (V, E)$ um grafo e $v \in V$ um vértice de G. Denotaremos por $N(v)$ o conjunto $\{u \in V : u \text{ é vizinho de } v\}$ e por $\delta(v)$ o conjunto $\{e \in E : e \text{ incide em } v\}$. Para $S \subseteq V$, $N(S) = \bigcup_{v \in S} N(v)$ e $\delta(S) = \bigcup_{v \in S} \delta(v)$.

Num grafo, uma sequência de vértices sem repetições $P = \langle v_0, v_1, \ldots, v_k \rangle$ é dita um *caminho* se v_i, v_{i+1} são adjacentes para $i = 0, 1, \ldots k-1$. Denotamos por P^{-1} o caminho obtido pela inversão de P, ou seja, $P^{-1} = \langle v_k, \ldots, v_0 \rangle$. Dados dois caminhos $Q = \langle q_0, q_1, \ldots, q_s \rangle$ e $R = \langle r_0 = q_s, r_1, \ldots, r_t \rangle$, denotamos por $Q \cdot R$ a sequência $\langle q_0, q_1, \ldots, q_s = r_0, r_1, \ldots, r_t \rangle$. Para que essa sequência seja um caminho, Q e R devem ter apenas $q_s = r_0$ como vértice comum. Poderemos também denotar o caminho P pela sua sequência de arestas $\langle v_0v_1, v_1v_2, \ldots, v_{k-1}v_k \rangle$.

Se $M \subseteq E$ é um emparelhamento em G, dizemos que um vértice $v \in V$ é *coberto* por M se existe uma aresta $e \in M$ que incide em v. Caso contrário, dizemos que v é um vértice *livre*.

Um caminho P em G dado pela sequência de arestas $\langle e_1, e_2, \ldots, e_k \rangle$ é dito *M-alternante* se $e_i \in M$ implica que $e_{i+1} \notin M$ e $e_i \notin M$ implica que $e_{i+1} \in M$, para $i = 1, \ldots, k-1$. Se os extremos de P são livres, dizemos que P é *M-aumentante*. Onde ficar claro pelo contexto, diremos apenas que P é alternante ou aumentante.

Dado um emparelhamento M e um caminho M-aumentante P, podemos obter um emparelhamento M' com uma aresta a mais fazendo $M' = M \Delta E(P) = (M \cup E(P)) \setminus (M \cap E(P)$. Usaremos essa *expansão* várias vezes no decorrer do texto. Um exemplo de expansão é mostrado na figura abaixo. Um dos resultados mais importantes sobre emparelhamentos é o seguinte teorema.



(a) Caminho aumentante $P = \langle a, l, k, c, d, j, i, e, f, g \rangle$.

(b) Emparelhamento obtido pela expansão $M' = M \Delta E(P)$.

Figura 1.1: *Exemplo de expansão de emparelhamento através de caminho aumentante.*

Teorema 1 (Berge [BM76])**.** *Um emparelhamento M num grafo G tem cardinalidade máxima se e somente se não existe caminho M-aumentante em G.*

Demonstração. Mostramos acima que dado um emparelhamento M e um caminho M-aumentante podemos obter um emparelhamento M' tal que $|M'| > |M|$, ou seja, provamos que se M é máximo então não existe caminho M-aumentante. Resta provar que se não existe um caminho M-aumentante, então M tem cardinalidade máxima. Isso é equivalente a provar que se M não tem cardinalidade máxima então existe um caminho M-aumentante.

Suponha que $|M|$ não é máximo e considere um emparelhamento máximo M^*. Seja H o grafo induzido pelo conjunto de arestas $M \Delta M^*$. Nos vértices de H incidem no máximo uma aresta de M e uma de M^*. Portanto, as componentes de H são circuitos pares com arestas se alternando entre M e M^* ou um caminho com arestas alternadamente em M e M^*. Como $|M| < |M^*|$, H contém pelo menos uma componente P formada por um caminho que contém mais arestas de M^*

do que M. Portanto, os extremos de P são incidentes a M^* e não a M. Ou seja, P é um caminho M-aumentante, uma contradição. □

Seja $r \in V$ um vértice livre. Diremos que uma árvore $T \subseteq G$ é uma árvore M-alternante com raiz r se o caminho de r até qualquer outro vértice de T é alternante. Diremos que uma floresta é alternante se as árvores que a compõem são alternantes.

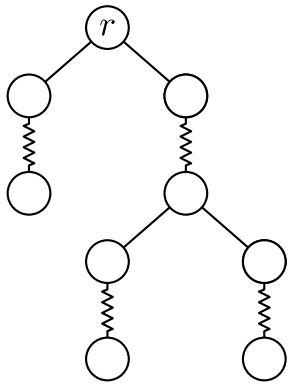


Figura 1.2: *Exemplo de árvore M-alternante com raiz r. As arestas serrilhadas pertencem ao emparelhamento M.*

No próximo capítulo, utilizaremos o seguinte resultado sobre emparelhamentos em grafos bipartidos.

Teorema 2 (Hall [BM76])**.** *Seja G um grafo bipartido com bipartição (X, Y). Existe em G um emparelhamento que cobre X se e somente se*

$$|N(S)| \geq |S| \quad \textit{para todo } S \subseteq X.$$

Demonstração. Seja M um emparelhamento que cobre X. Seja S um subconjunto de X. Como todo vértice de S está coberto, sabemos que S tem pelo menos $|S|$ vizinhos, ou seja, $|S| \leq |N(S)|$.

Vamos provar o outro lado da implicação por contradição. Seja G um grafo bipartido tal que $|S| \leq |N(S)|$ para todo $S \subseteq X$ e suponha que não existe um emparelhamento em G que cobre X. Seja M um emparelhamento de cardinalidade máxima em G e seja $u \in X$ um vértice livre. Defina o conjunto Z como o conjunto de todos os vértices atingíveis por caminhos M-alternantes a partir de u. Claramente, u é o único vértice livre de Z, pois o caminho a qualquer outro vértice livre seria aumentante, contradizendo o fato de M ser máximo. Tome $S = Z \cap X$ e $T = Z \cap Y$.

Como cada vértice de $S \setminus \{u\}$ está emparelhado a um vértice de T, temos que $|T| = |S| - 1$ e $T \subseteq N(S)$. Claramente, qualquer aresta $uv \notin M$ é um caminho alternante. Além disso, se $x \in X$ é um vértice atingível a partir de u por um caminho alternante, então qualquer vértice $y \in N(x)$ é atingível a partir de u por um caminho alternante. Isso implica que $N(S) \subseteq T$. Logo, temo que

$$|N(S)| = |T| = |S| - 1 < |S|,$$

o que é uma contradição. □

Dizemos que M é um emparelhamento *perfeito* se M cobre todos os vértices de G. Note que a condição do teorema 2 precisa ser válida para as duas partes de um grafo bipartido para que esse possua um emparelhamento perfeito.

Parte I

Algoritmo combinatório

Capítulo 2

Solução combinatória

A seguir descrevemos o Método Húngaro, proposto inicialmente por Kuhn[Kuh55] para resolver o chamado *Problema da Designação*, definido como: dadas pontuações para n pessoas na execução de n tarefas distintas, encontrar uma designação de cada pessoa para uma única tarefa que maximize a soma das pontuações. Como veremos na seção seguinte, esse problema é equivalente ao de encontrar um emparelhamento de peso máximo num grafo bipartido. Na seção 2.2, apresentamos um algoritmo devido a Edmonds[Edm65a] que resolve o problema do emparelhamento máximo para grafos arbitrários.

2.1 Caso bipartido - Método húngaro

Nesta seção apresentamos o método húngaro para obtenção de emparelhamentos de peso máximo em grafos bipartidos. O nome método húngaro foi cunhado por Kuhn [Kuh91] em homenagem aos húngaros Kőnig e Egerváry, autores de importantes resultados que levaram ao desenvolvimento do método.

O teorema de Kőnig diz que a cardinalidade de um emparelhamento de cardinalidade máxima em um grafo bipartido é igual à cardinalidade de uma cobertura mínima nesse grafo. Uma cobertura é um conjunto de vértices $C \subseteq V(G)$ tal que toda aresta de G incide em pelo menos um vértice de C. O teorema de Egerváry é uma generalização do resultado de Kőnig para o caso em que as arestas têm pesos. A prova do teorema 3 abaixo está fortemente ligada a esse resultado.

O método húngaro reduz o problema do emparelhamento de peso máximo em uma sucessão de problemas de emparelhamentos de cardinalidade máxima.

Sejam $G = (V, E)$ um grafo bipartido com bipartição $V = X \cup Y$ e $w : E \to \mathbb{R}^+$ uma função que atribui pesos às arestas de G. Na descrição que segue, suporemos que $|X| = |Y|$ e que G é bipartido completo. Caso alguma dessas suposições não seja verdadeira, adicionamos vértices à parte de V que tem menos vértices e tornamos o grafo bipartido completo adicionando arestas de peso zero. Se M' é um emparelhamento de peso máximo no grafo obtido, então o emparelhamento $M = \{e \in M' : w(e) > 0\}$ é um emparelhamento de peso máximo em G.

Dizemos que uma função $l : V \to \mathbb{R}^+$ é uma *rotulação viável* dos vértices de G se para toda aresta $uv \in E(G)$,

$$l(u) + l(v) \geq w(uv).$$

Para uma rotulação viável l, definimos o *grafo de equivalência* de G em relação a l como $G_l = (V, E_l)$

onde $E_l = \{xy : l(x) + l(y) = w(xy)\}$. Note que G_l é um subgrafo de G. Denotaremos por $N_l(u)$ o conjunto $\{v \in N(u) : uv \in E_l\}$ para algum $u \in V$ e por $N_l(S)$ o conjuto $\bigcup_{u \in S} N_l(u)$ para algum conjunto $S \subseteq V$.

Durante a execução do algoritmo, temos uma rotulação viável l, que definiremos a seguir, e um emparelhamento M, inicialmente vazio, em G_l. A cada iteração procura-se um caminho M-aumentante em G_l. Se um tal caminho existe expandimos M como mostrado na seção 1. Caso contrário l é alterada de forma que o grafo de equivalência inclua mais arestas. O processo termina quando M é um emparelhamento perfeito em G_l. Isso sempre é possível pois G é bipartido completo, $|X| = |Y|$ e a inclusão de arestas em G_l eventualmente torna-o igual a G.

O seguinte teorema garante que o emparelhamento obtido tem peso máximo em G.

Teorema 3 ([BM76])**.** *Se l é uma rotulação viável e M um emparelhamento perfeito em G_l então M é um emparelhamento de peso máximo em G.*

Demonstração. Note que qualquer emparelhamento perfeito M' em G satisfaz

$$\sum_{xy \in M'} w(xy) \leq \sum_{xy \in M'} \big(l(x) + l(y)\big) = \sum_{v \in V} l(v),$$

ou seja, $\sum_{v \in V} l(v)$ limita superiormente o peso de um emparelhamento perfeito. Por outro lado, como M é um emparelhamento perfeito em G_l, temos que

$$\sum_{xy \in M} w(xy) = \sum_{xy \in V} \big(l(x) + l(y)\big) = \sum_{v \in V} l(v).$$

portanto, M é um emparelhamento perfeito de peso máximo. Como não existem arestas de peso negativo e o grafo é bipartido completo, existe um emparelhamento perfeito cujo peso é máximo entre todos os emparelhamentos em G. Logo M tem peso máximo. □

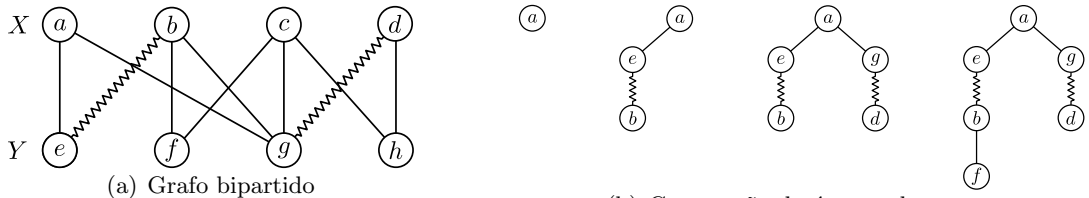
Uma rotulação viável inicial l pode ser obtida da seguinte forma:

$$\begin{aligned} l(x) &= \max_{y \in N(x)} \{w(xy)\} && \text{se } x \in X, \\ l(y) &= 0 && \text{se } y \in Y. \end{aligned}$$

A procura por um caminho aumentante consiste na construção de uma árvore alternante $H \subseteq G_l$ tendo como raiz um vértice livre $r \in X$. Sejam $S = V(H) \cap X$ e $T = V(H) \cap Y$. Inicialmente a árvore contém apenas a raiz e é expandida, mantendo a condição de que $|T| = |S| - 1$ até que um caminho aumentante seja encontrado ou $N_l(S) = T$. No segundo caso, sabemos pelo teorema 2 que G_l não admite um emparelhamento perfeito e precisaremos alterar l como veremos posteriormente.

A expansão da árvore é feita adicionando uma aresta de G_l entre um vértice $x \in S$ e um vértice $y \in N_l(S) - T$. Se y não é coberto por M, então existe um caminho aumentante entre r e y e a busca termina. Caso contrário, adicionamos a H a aresta $yz \in M$ que cobre y. Note que $z \in X$ e portanto a igualdade $|T| = |S| - 1$ é mantida.

Se G_l não admite um emparelhamento perfeito, l precisa ser alterado de forma a incluir mais



(a) Grafo bipartido

(b) Construção da árvore alternante

Figura 2.1: *Exemplo de construção de árvore alternante a partir de um grafo bipartido. Marcelo: "?" para a figura inteira.*

arestas de $E \setminus E_l$. Tome

$$\alpha_l = \min_{\substack{x \in S \\ y \in Y \setminus T}} \{l(x) + l(y) - w(xy)\}$$

e l' tal que

$$l'(v) = \begin{cases} l(v) - \alpha_l, & \text{se } v \in S, \\ l(v) + \alpha_l, & \text{se } v \in T, \\ l(v), & \text{caso contrário.} \end{cases}$$

Por construção, $G_{l'}$ contém pelo menos uma aresta a mais que G_l, uma vez que todas as arestas de G_l estão em $G_{l'}$. Então substituímos l por l' e continuamos a procura por caminhos aumentantes no novo grafo G_l.

Em cada iteração, ou encontramos um caminho aumentante ou adicionamos mais arestas a G_l. Portanto, com a construção de $O(|V|+|E|)$ árvores alternantes, o algoritmo termina. A construção das árvores alternantes é basicamente uma busca em profundidade e executa em tempo $O(|E|)$. Portanto, a complexidade do método húngaro é $O(|V||E| + |E|^2)$. Veja na figura 2.1 um exemplo de emparelhamento e respectiva contrução de árvore alternante.

Iremos mostrar a execução do algoritmo para o grafo da figura 2.2(a). A figura 2.2(b) mostra a rotulação inicial dos vértices. As arestas que não pertencem a E_l são representadas por linhas pontilhadas.

Começando a construção de uma árvore alternante a partir do vértice a, adicionamos a aresta af (figura 2.2(c)). Como f é livre, a aresta af passa a fazer parte de M (figura 2.2(d)) devido a aumentação através do caminho aumentante $\langle a, f\rangle$. Após a aumentação, construímos outra árvore alternante com raiz b. A aresta bd é adicionada (figura 2.2(e)) e o caminho aumentante $\langle b, d\rangle$ encontrado. A aresta bd passa a fazer parte de M (figura 2.2(f)). Construímos uma nova árvore tendo c como raiz.

A árvore resultante (figura 2.2(g)) não pode mais ser expandida. Por isso, calculamos α como o mínimo entre $l(b) + l(e) - w(be) = 4$, $l(b) + l(f) - w(ef) = 1$, $l(c) + l(e) - w(ce) = 4$ e $l(c) + l(f) - w(cf) = 4$, de onde segue que $\alpha = 1$. A figura 2.2(h) mostra o novo valor de l. Podemos então adicionar as arestas bf e fa à árvore com raiz c (figura 2.2(i)). Novamente a árvore não pode ser expandida resultando num ajuste dos rótulos.

Calculamos α como o mínimo entre $l(a) + l(e) - w(ae) = 3$, $l(b) + l(e) - w(be) = 3$ e $l(c) + l(e) - w(ce) = 4$. Ou seja, $\alpha = 3$. A figura 2.2(j) mostra o novo valor para l. Com os novos rótulos, a aresta ae pertence a E_l e pode ser adicionada a árvore com raiz c (figura 2.2(k)). Nesse ponto, o caminho aumentante $\langle c, d, b, f, a, e\rangle$ é encontrado. A figura 2.2(l) mostra o emparelhamento

obtido pela aumentação. Como ele é perfeito, o algoritmo termina.

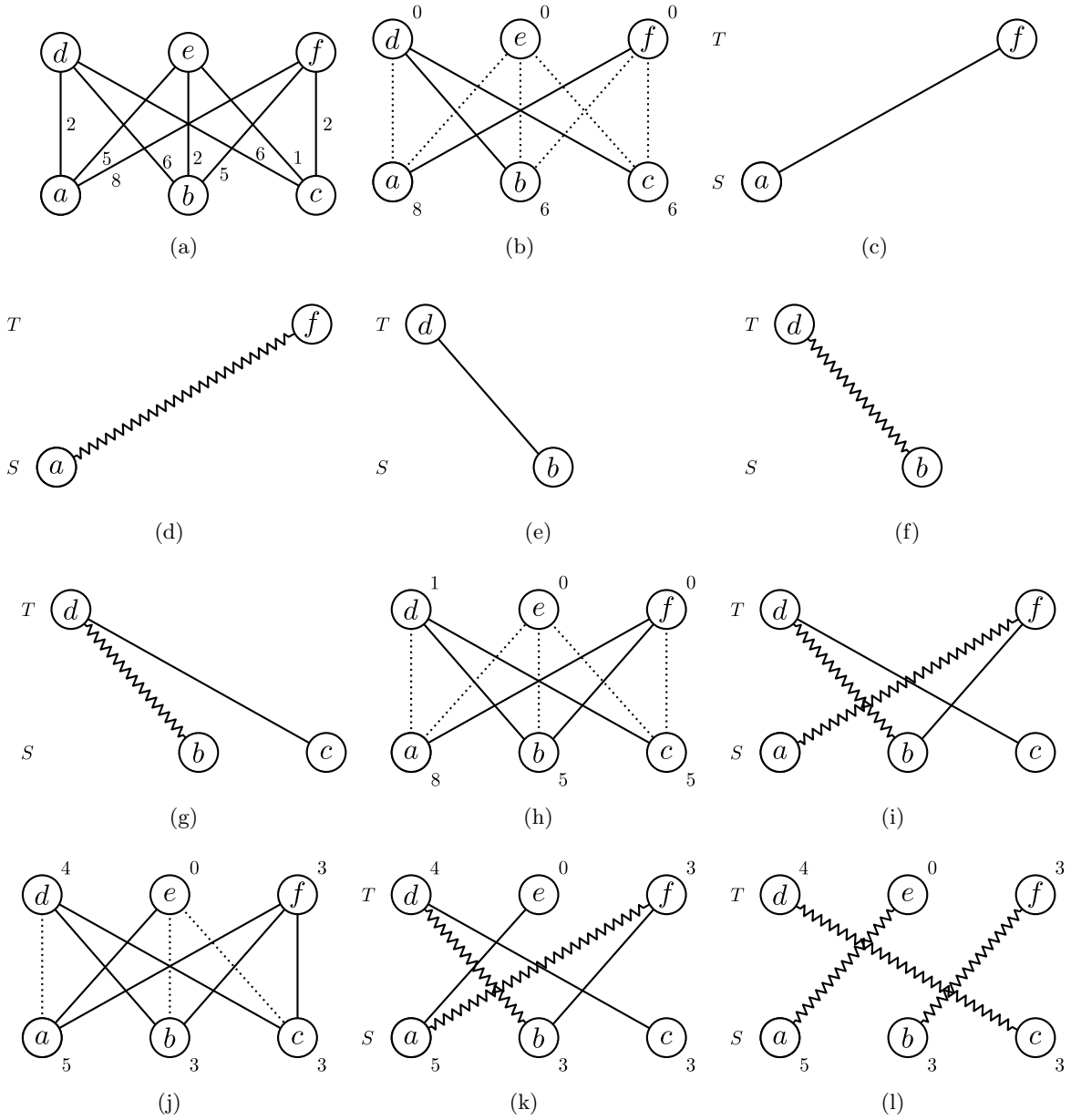


Figura 2.2: *Exemplo de execução do método húngaro.*

2.2 Caso geral - Algoritmo de Edmonds

O algoritmo apresentado na seção anterior é bastante simples, porém não funciona em grafos não-bipartidos. O problema está no algoritmo de busca por caminhos aumentantes, que pode falhar devido à existência de circuitos ímpares.

Considere a construção de uma árvore alternante a partir do grafo da figura 2.3 tendo o vértice a como raiz. Inicialmente adicionamos as arestas ab e bc. Se o primeiro vizinho visitado de c é o vértice d, adicionamos as arestas cd e de. Nesse ponto, $N(S) - T = \{c\}$. Se permitimos a adição de c em T, encontramos erroneamente o passeio a, b, c, d, e, c, b, a como um caminho aumentante. Por outro lado, se simplesmente ignoramos a aresta ce, a busca termina sem que o caminho aumentante

a, b, c, e, d, f seja encontrado.

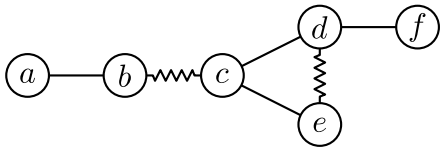


Figura 2.3: *Exemplo de grafo não bipartido onde a procura por caminhos aumentantes mostrada na seção 2.1 falha, apesar da existência do caminho aumentante a, b, c, e, d, f.*

Em 1965, Edmonds[Edm65b] propôs uma extensão para o algoritmo de busca por caminhos aumentantes que funciona para grafos arbitrários. A descrição que apresentaremos aqui é baseada no artigo original de Edmonds e também nos livros de Gibbons [Gib85] e Nemhauser e Wolsey [NW88].

A idéia básica do algoritmo é que todo emparelhamento máximo em um circuito ímpar deixa um vértice livre. Assim, os circuitos ímpares encontrados na busca por caminhos aumentantes são contraídos em um único vértice. Ao término da busca, obtemos um caminho aumentante num grafo que pode ter vários desses pseudo-vértices e, como veremos a seguir, é possível obter um caminho aumentante no grafo original a partir dele.

2.2.1 Blossoms

Formalizamos a operação de contração descrita acima como segue. Dado um conjunto $B = \{b_1, b_2, \ldots, b_r\} \subseteq V$, o grafo $G/B = (V_B, E_B)$ resultante da *contração* de B em G é dado por

$$V_B = (V \setminus B) \cup \{b\}$$
$$E_B = E(V \setminus B) \cup \{xb : xb_i \in E, 1 \leq i \leq r\}.$$

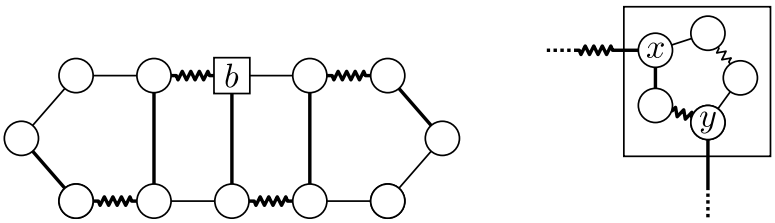
Dizemos que b é o *pseudo-vértice* associado a B. Note que se o grafo G foi obtido por contração a partir de algum outro grafo, o conjunto B pode conter pseudo-vértices.

Dado um emparelhamento M, dizemos que B é um *blossom* se os vértices de B formam um circuito ímpar e $|M \cap E(B)| = \frac{|B|-1}{2}$.

Lema 1. *Sejam B um blossom de um grafo G e T uma árvore alternante em G/B para um dado emparelhamento M. Se existe um caminho alternante $P = (u, \ldots, b, \ldots, v)$ em T então existe um caminho alternante $P' = (u, \ldots, v)$ em G que contém todas as arestas de P que não incidem em b, onde b é o pseudo-vértice associado ao blossom B em T.*

Demonstração. Seja $P = (u, \ldots, a, b, c, \ldots, v)$. Claramente, o caminho P_{ua} (resp. P_{cv}) de u para a (resp. de c para v) está em G. Suponha, s.p.g., que $ab \in M$. Sejam x e y vértices em B tais que $ax, cy \in E(G)$. O circuito ímpar B induz dois caminhos em G de x para y sendo um deles de comprimento par que chamaremos de Q. Seja M_B um emparelhamento máximo em B que deixa x livre (figura 2.4(b)). Claramente, Q é M_B-alternante. Seja $M' = (M \setminus \{ab, bc\}) \cup M_B \cup \{ax\}$. Então, o caminho $P' = P_{ua} \cdot Q \cdot P_{cv}$ é M'-alternante e $E(P) \cap E(P') = E(P) \setminus \{ab, bc\}$. □

Na próxima seção utilizaremos os resultados acima para descrever o algoritmo de procura por caminhos aumentantes proposto por Edmonds. Esse algoritmo é uma peça fundamental do algo-



(a) Grafo com conjunto contraído representado por quadrado.

(b) Emparelhamento que deixa x livre.

Figura 2.4: *Ilustração sobre a prova do lema 1. As arestas do caminho aumentante estão destacadas.*

ritmo que resolve o problema do emparelhamento máximo em grafos arbitrários, que descreveremos na seção 2.2.3.

2.2.2 Busca pelo caminho aumentante

A procura por caminhos aumentantes consiste na construção de uma floresta alternante. Cada vértice livre é a raiz de uma árvore e, inicialmente, cada árvore contém apenas a raiz. Diremos que um vértice v dessa floresta é *par* se o comprimento do caminho único de v até a raiz da árvore que o contém é par. Caso contrário, dizemos que v é *ímpar*. Iremos dizer ainda que um vértice que não pertence à floresta é um vértice *fora da floresta*.

A floresta é expandida iterativamente, visitando-se arestas entre um vértice par u e algum outro vértice v. O passo do algoritmo é descrito a seguir.

Se v é par, acontece um dos casos abaixo.

u **e** v **pertencem a árvores alternantes distintas.** Sejam r_u e r_v as raízes das árvores que contêm u e v, respectivamente. Seja P_u o caminho único de r_u a u e seja P_v o caminho único de v a r_v. Então, o caminho $P_u \cdot \langle u, v\rangle \cdot P_v$ é um caminho aumentante, pois r_u e r_v são livres.

u **e** v **pertencem à mesma árvore.** Então, existe um circuito B ímpar formado pela aresta uv e os caminhos únicos de u até w e v até w, onde w é o menor ancestral comum de u e v. Nesse caso, contraímos B. A procura continua no grafo G/B. Note que B é um blossom e que o pseudo-vértice associado a B é par.

É possível que um dos vértices no circuito ímpar seja na verdade um pseudo-vértice. Em alguns casos abusaremos da definição de blossom chamando de B o conjunto de vértices de G contidos direta ou indiretamente através de um pseudo-vértice em um blossom. Chamaremos de *blossom exterior* um blossom cujo pseudo-vértice não está contido em nenhum outro blossom contraído. Um blossom exterior será dito *par* (*ímpar*) se o seu pseudo-vértice em T for par (ímpar).

Se v é ímpar, a aresta uv é ignorada.

Se v não pertence à floresta, então v está emparelhado a um vértice $w \in V$, caso contrário, v seria a raiz de uma árvore alternante. Nesse caso, adiciona-se uv e vw à floresta. Note que v é ímpar e w é par.

Se todas as arestas que contêm um vértice par foram exploradas sem que um caminho aumentante tenha sido encontrado, então o emparelhamento M tem cardinalidade máxima, como demonstraremos a seguir.

Dizemos que uma coleção de vértices S e conjuntos de cardinalidade ímpar $O_1, \ldots, O_t$ é uma *cobertura por conjuntos ímpares* de $G = (V, E)$, se, para toda aresta e de G, ou e incide em algum vértice de S ou e está contida em $E(O_i)$ para algum i. A *capacidade* de uma cobertura por conjuntos ímpares C é definida como

$$cap(C) = |S| + \sum_{i=1}^{t} \frac{|O_i| - 1}{2}.$$

Lema 2. *Se M é um emparelhamento em G e C uma cobertura por conjuntos ímpares de G, então*

$$|M| \leq cap(C).$$

Demonstração. Sejam $M' \subseteq M$ o conjunto das arestas de M que incidem em vértices de S e M'' o conjunto das arestas de M que estão contidas em $E(O_i)$ para algum i. Claramente,

$$|M'| \leq |S|. \tag{2.1}$$

Cada conjunto O_i pode conter no máximo $\frac{1}{2}(|O_i| - 1)$ arestas de M, ou seja,

$$|M \cap O_i| \leq \frac{|O_i| - 1}{2} \quad \text{para todo } i. \tag{2.2}$$

De 2.1 e 2.2, segue que

$$|M| \leq |M' \cup M''| \leq |S| + \sum_{i=1}^{t} \frac{|O_i| - 1}{2} = cap(C).$$

□

Ao final do algoritmo descrito acima, temos um emparelhamento M, uma floresta alternante e um conjunto de vértices que não faz parte da floresta, que estão todos cobertos. Seja S um conjunto contendo todos os vértices ímpares da floresta alternante e mais exatamente um extremo de cada aresta entre vértices fora da floresta. Sejam $O_1, \ldots O_t$ o conjunto de vértices dos blossoms contraídos durante a execução do algoritmo. Vamos demonstrar que S e $O_1, \ldots, O_t$ formam uma cobertura por conjuntos ímpares de G que chamaremos de C.

Seja e uma aresta de G. Se um dos extremos de e é um vértice ímpar ou fora da floresta, então e incide em algum vértice de S. Se e é uma aresta contida em um blossom então e está contida em $E(O_i)$ para algum i. Um dos dois casos acontece, pois uma aresta entre dois vértices pares estaria contraído em um blossom. De fato, C é uma cobertura por conjuntos ímpares de G.

Mostraremos agora que $|M| = cap(C)$. Note que cada aresta $e \in M$ é uma aresta entre vértices fora da floresta, uma aresta contida num blossom ou uma aresta entre um vértice ímpar e outro vértice par ou contido num blossom. Sejam $M' \subseteq M$ o conjunto das arestas de M que incidem em vértices de S e M'' o conjunto das arestas de M que estão contidas em $E(O_i)$ para algum i. Por contrução, $|M'| = |S|$. Como $O_1, \ldots, O_t$ são blossoms, temos que $|M \cap O_i| = \frac{1}{2}(|O_i| - 1)$ para todo

i. Portanto, temos que

$$|M''| = \sum_{i=1}^{t} \frac{|O_i| - 1}{2}.$$

Como M' e M'' são disjuntos, temos que

$$|M| = |M'| + |M''| = |S| + \sum_{i=1}^{t} \frac{|O_i| - 1}{2} = cap(C).$$

Do lema 2, segue que M tem cardinalidade máxima.

2.2.3 Encontrando emparelhamentos de peso máximo

Nesta seção apresentaremos um algoritmo combinatório devido a Edmonds [Edm65a] que resolve o problema do emparelhamento de peso máximo em grafos arbitrários. Esse algoritmo, assim como o método húngaro, procura emparelhamentos de cardinalidade máxima em subgrafos de G. Para tanto, utilizamos uma extensão da rotulação usada na seção 2.1. Seja

$$O = \{B \subseteq V : |B| \geq 3 \text{ e ímpar}\}.$$

Diremos que uma função $l : V \cup O \to \mathbb{R}^+$ é uma *rotulação viável* de G se

$$\pi_{uv} := l(u) + l(v) + \sum_{\substack{B \in O \\ uv \in E(B)}} l(B) \geq w(uv) \tag{2.3}$$

para toda aresta uv.

O algoritmo é iterativo, sendo que em cada passo temos um emparelhamento M e uma rotulação viável l. A partir dessa rotulação, definimos o grafo $G_l = (V, E_l)$, onde $E_l = \{uv \in E : \pi_{uv} = w(uv)\}$. Cada iteração consiste na busca por um caminho aumentante em G_l, utilizando o algoritmo descrito no final da seção anterior. Se a busca tem sucesso, aumentamos M da maneira mostrada na seção 1.1. Caso contrário, uma nova rotulação viável l' é obtida e a procura continua no subgrafo $G_{l'}$. Ao término do algoritmo, vale que

$$l(u) > 0 \Rightarrow u \text{ é coberto por } M \tag{2.4}$$

e

$$l(B) > 0 \Rightarrow |M \cap E(B)| = \frac{|B| - 1}{2}. \tag{2.5}$$

Essas condições são suficientes para garantir que M tem peso máximo, como mostra o teorema a seguir.

Teorema 4. *Se l é uma rotulação viável e M um emparelhamento em G_l tal que 2.4 e 2.5 são satisfeitas, então M é um emparelhamento de peso máximo em G.*

Demonstração. Note que o peso de um emparelhamento qualquer N é

$$\begin{aligned}\sum_{uv\in N} w(uv) &\leq \sum_{uv\in N}\Big(l(u)+l(v)+\sum_{\substack{B\in O\\ uv\in E(B)}} l(B)\Big)\\ &= \sum_{uv\in N}\Big(l(u)+l(v)\Big)+\sum_{uv\in N}\sum_{\substack{B\in O\\ uv\in E(B)}} l(B)\\ &\leq \sum_{v\in V} l(v)+\sum_{B\in O}\frac{|B|-1}{2}l(B).\end{aligned}$$

Por outro lado, toda aresta $uv \in M$ satisfaz $\pi_{uv} = w(uv)$. Portanto,

$$\begin{aligned}\sum_{uv\in M} w(uv) &= \sum_{uv\in M}\Big(l(u)+l(v)+\sum_{\substack{B\in O\\ uv\in E(B)}} l(B)\Big)\\ &= \sum_{uv\in M}\Big(l(u)+l(v)\Big)+\sum_{uv\in M}\sum_{\substack{B\in O\\ uv\in E(B)}} l(B)\\ &= \sum_{v \text{ coberto}} l(v)+\sum_{B:l(B)>0}|M\cap E(B)|\,l(B).\end{aligned}$$

Como l e M satisfazem 2.4 e 2.5, temos que

$$\sum_{uv\in M} w(uv) = \sum_{v\in V} l(v)+\sum_{B\in O}\frac{|B|-1}{2}l(B).$$

□

O algoritmo inicia com um emparelhamento vazio M e a rotulação viável l dada por

$$\begin{aligned}&l(v) = \frac{1}{2}\max_{e\in E}\{w(e)\} \quad \text{para todo } v\in V,\\ &l(B) = 0 \quad \text{para todo } B\in O.\end{aligned}$$

Note que esse par não satisfaz 2.4 mas satisfaz 2.5. Ao longo do algoritmo, alteramos l para que 2.4 seja satisfeita para mais vértices, sempre mantendo 2.5 satisfeita. Para tanto, o valor de $l(B)$ para $B \in O$ será diferente de zero apenas se B é um blossom contraído, pois assim podemos garantir que $|M \cap E(B)| = \frac{1}{2}(|B|-1)$.

Quando encontramos um emparelhamento de cardinalidade máxima em G_l, se 2.4 está satisfeita para todos os vértices, o algoritmo termina. Note que toda expansão de M faz com que 2.4 seja satisfeita para os extremos do caminho aumentante utilizado na expansão.

Se o emparelhamento M tem cardinalidade máxima em G_l mas ainda existem vértices para os quais 2.4 não seja satisfeita, calculamos uma nova rotulação viável l' em função de um $\delta > 0$ escolhido de forma apropriada.

Sejam $\mathcal{B}_P$ o conjunto dos blossoms exteriores pares e $\mathcal{B}_I$ o conjunto dos blossoms exteriores ímpares. Ainda, sejam $V_P, V_I \subseteq V$, tais que V_P (V_I) é o conjunto dos vértices pares (ímpares) ou contidos num blossom exterior par (ímpar). Denotaremos por $B(v)$ o blossom exterior que contém um vértice $v \in V$. Se v não pertence a nenhum blossom, então $B(v) = v$.

A nova rotulação l' é definida como segue:

$$\begin{aligned} l'(v) &= l(v) - \delta, && \text{se } v \in V_P, \\ l'(v) &= l(v) + \delta, && \text{se } v \in V_I, \\ l'(B) &= l(B) + 2\delta, && \text{se } B \in \mathcal{B}_P, \\ l'(B) &= l(B) - 2\delta, && \text{se } B \in \mathcal{B}_I. \end{aligned}$$

δ é escolhido de forma que a rotulação l' seja viável. Para tanto, tomamos δ como o mínimo entre δ_1, δ_2, δ_3 e δ_4 definidos a seguir:

$$\delta_1 = \frac{1}{2} \min \{l(B) : B \in \mathcal{B}_I\}; \tag{2.6}$$

$$\delta_2 = \min \{l(v) : v \in V_P\}; \tag{2.7}$$

$$\delta_3 = \min \{l(u) + l(v) - w_{uv} : v \in V_P \text{ e } u \notin V_P \cup V_I\} \tag{2.8}$$

$$\delta_4 = \frac{1}{2} \min \{l(u) + l(v) - w_{uv} : u, v \in V_P \text{ tais que } B(u) \neq B(v)\} \tag{2.9}$$

δ_1 e δ_2 garantem que l' é não negativa. δ_3 garante que 2.3 é satisfeita para arestas entre um vértice par ou contido num blossom exterior par e um vértice fora da floresta. δ_4 garante que 2.3 continua satisfeita para arestas entre vértices pares, entre vértices contidos em blossoms exteriores pares distintos ou entre um vértice par e um vértice contido num blossom exterior par.

Essa escolha de δ implica que para algum vértice v ou para algum blossom B, o valor de l' se torna zero ou então 2.3 passa a valer com igualdade, ou seja, $\pi_{uv} = w(uv)$ para pelo menos uma aresta adicional uv. Nesse último caso, $G_{l'}$ passa a conter tais arestas e podemos continuar a procura por caminhos aumentantes usando a floresta alternante atual como ponto de partida.

Se l' torna-se zero para algum vértice u, é possível alterar M de forma que 2.4 seja satisfeita por r. Se $u = r$, então não é necessário nenhuma alteração. Suponha que $u \neq r$. Seja P o caminho único entre u e r. Como u é par, P tem um número par de arestas. Fazendo $M \leftarrow M \Delta E(P)$, o vértice r passa a ser coberto e portanto 2.4 fica satisfeita para esse vértice. Como $l'(u) = 0$, 2.4 também é satisfeita para u.

Se l' torna-se zero para algum blossom B, não precisamos mais exigir que M cubra $|B| - 1$ vértices. Portanto, expandimos B. Como B é ímpar, exatamente duas arestas incidem no pseudo-vértice associado a B, sendo uma aresta de M e outra que não pertence a M. Sejam elas $e_1 \in M$ e $e_2 \in E \setminus M$. Sejam v_1 e v_2 vértices de B tais que e_1 incide em v_1 e e_2 incide em v_2. Se removermos o pseudo-vértice associado a B da floresta alternante e adicionarmos o caminho de comprimento par de v_1 a v_2, a procura por caminhos aumentantes pode continuar com a mesma floresta.

Isso termina a descrição do algoritmo de Edmonds para o problema do emparelhamento de peso máximo. No próximo capítulo discutiremos a estrutura de dados que utilizamos para implementá-lo fazendo uma análise de complexidade. Vejamos agora um exemplo de execução do algoritmo.

Considere o grafo da figura 2.5(a). Uma execução do algoritmo descrito acima usará 50 como rótulo inicial para todos os vértices. Inicialmente as arestas $3, 4$ e $5, 6$ estarão ativas (figura 2.5(b)). A princípio, todos os vértices são pares pois são raízes de árvores alternantes que contém apenas a raiz. A expansão da árvore com raiz 3 encontra o caminho aumentante $\langle 3, 4 \rangle$. Similarmente, o caminho aumentante $\langle 5, 6 \rangle$ é encontrado na expansão da árvore com raiz 5. A figura 2.6(a) mostra

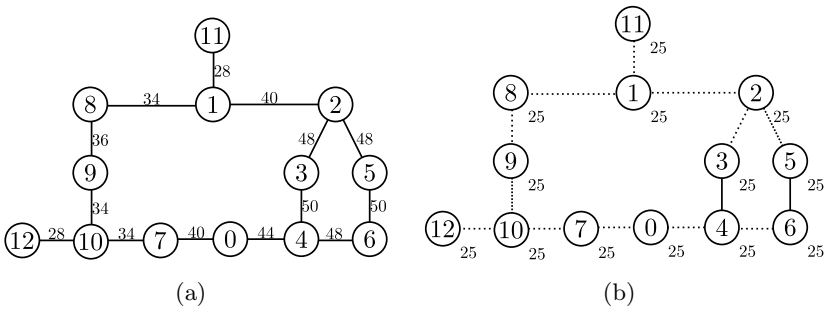


Figura 2.5: *Exemplo de execução do Algoritmo de Edmonds (parte 1).*

o emparelhamento obtido depois do uso dos dois caminhos.

Como não é possível expandir mais nenhuma das árvores alternantes, um ajuste dos rótulos se torna necessário. Das equações 2.6 a 2.9, obtemos $\delta_1 = \infty$, $\delta_2 = 25$, $\delta_3 = 2$, $\delta_4 = 5$ e $\delta = \delta_2$. A figura 2.6(b) mostra o grafo obtido após a alteração dos rótulos. Note que as arestas $2,3$ e $2,5$ se tornam ativas. Expandindo a árvore alternante com raiz no vértice 2, obtemos a árvore representada pela figura 2.6(c).

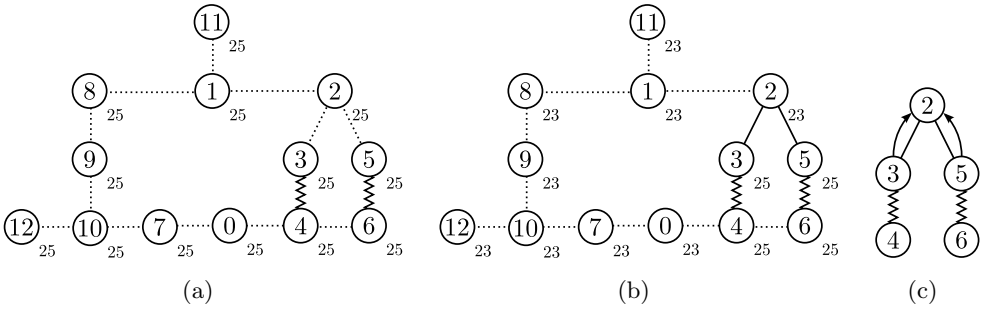


Figura 2.6: *Exemplo de execução do Algoritmo de Edmonds (parte 2).*

Nesse ponto, outro ajuste de rótulos é necessário. Com $\delta = \delta_4 = l(4)+l(6)-w(4,6) = 1$, a aresta $4,6$ se torna ativa. Continuando a expansão a partir da árvore da figura 2.6(c), essa aresta resulta na contração dos vértices 2, 3, 4, 5 e 6 no blossom B_1 (figura 2.7(a)). Depois da contração, outro ajuste dos rótulos se faz necessário. A alteração através de $\delta = \delta_3 = \frac{1}{2}(l(0) + l(4) - w(0,4)) = 1$ ativa a aresta $0,4$ (figura 2.7(b)).

O pseudo-vértice B_1 é par e, portanto, a expansão da árvore que contém 0 encontra o caminho aumentante $\langle 0, B_1 \rangle$. Substituindo B_1 pelo caminho alternante de comprimento par entre o vértice 4 e a base do blossom, 2, obtemos o caminho aumentante no grafo original $\langle 0,4,3,2 \rangle$. A figura 2.7(c) mostra o resultado da aumentação por esse caminho.

Novamente, se torna impossível expandir qualquer árvore alternante e por isso é feito um novo ajuste dos rótulos com $\delta = \delta_3 = 2$. Note que o vértice 0 e o pseudo-vértice B_1 não pertencem a nenhuma árvore alternante e por isso seus rótulos se mantêm inalterados. Depois do ajuste, as arestas 1,2 e 0,7 são ativadas (figura 2.8(a)).

Começando uma nova árvore alternante com raiz 1, a aresta $1,2$ faz com que o pseudo-vértice B_1 seja adicionado como um vértice ímpar e o vértice 0 seja adicionado como par, devido à aresta $0,4$. Continuando a expansão, a aresta 0,7 induz o caminho aumentante $\langle 7,0,B_1,1 \rangle$ que corresponde a $\langle 7,0,4,3,2,1 \rangle$ no grafo original. A figura 2.9(a) mostra o resultado da aumentação através desse

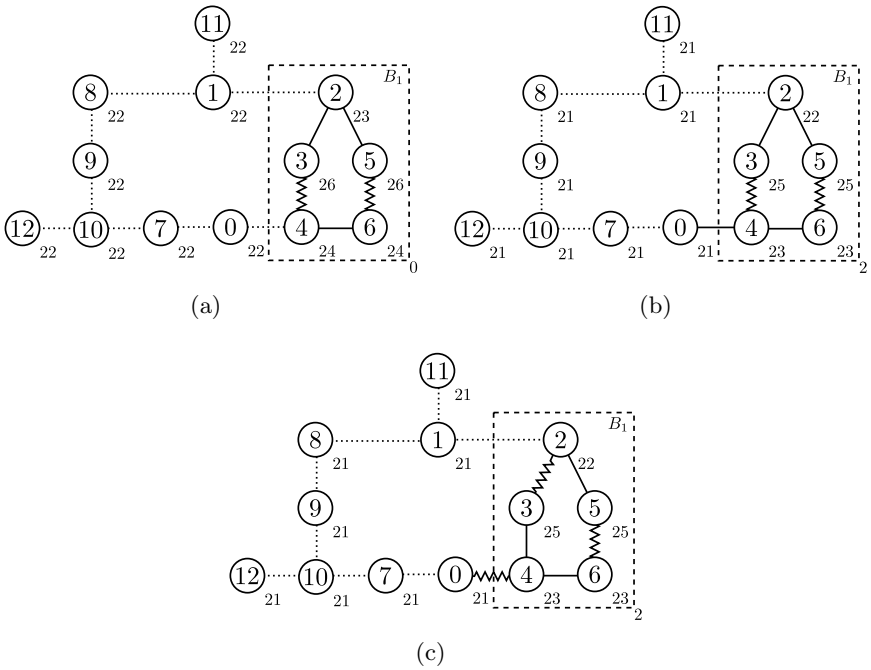


Figura 2.7: *Exemplo de execução do Algoritmo de Edmonds (parte 3).*

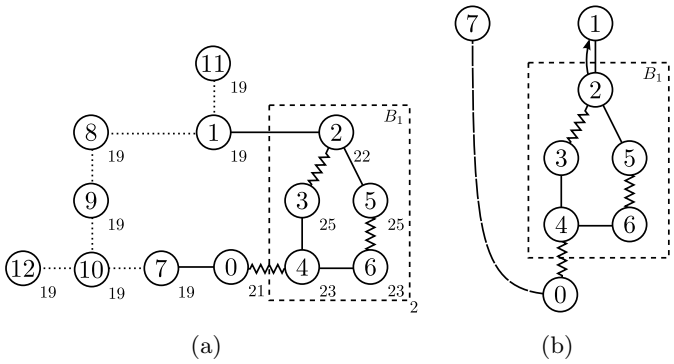


Figura 2.8: *Exemplo de execução do Algoritmo de Edmonds (parte 4).*

caminho aumentante. Em seguida, uma nova floresta alternante é construída. No entanto, não é possível expandi-la a partir de nenhum vértice (figura 2.9(b)) livre e um novo ajuste dos rótulos é feito.

Obtemos $\delta = \delta_4 = \frac{1}{2}(l(8) + l(9) - w(8,9))$. Após o ajuste a aresta 8,9 se torna ativa e o caminho aumentante 8,9 é encontrado. O resultado da aumentação é mostrado pela figura 2.9(c). Novamente, os únicos vértices pares são aqueles em que não incide nenhuma aresta ativa. Um novo ajuste, com $\delta = \delta_3 = l(9) + l(10) - w(9,10) = 2$, é feito, ativando a aresta $9,10$ (figura 2.9(d)).

A expansão da árvore a partir do vértice livre 10 resulta na árvore da figura 2.9(e). Novamente, um ajuste dos rótulos é necessário. Com $\delta = \delta_3 = l(10) + l(7) - w(10,7) = 1$, a aresta $7,10$ se torna ativa (figura 2.9(f)). Continuando a expansão da árvore de raiz 10, obtemos a árvore alternante ilustrada na figura 2.10(a). Essa árvore não pode ser mais expandida.

Um novo ajuste dos rótulos é feito, com $\delta = \delta_1 = l(B_1) = \delta_3 = l(10) + l(12) - w(10,12) = 1$. Expandimos B_1 pois seu rótulo se tornou zero. A aresta $10,12$ também é ativada por esse ajuste. O resultado é ilustrado pela figura 2.10(b).

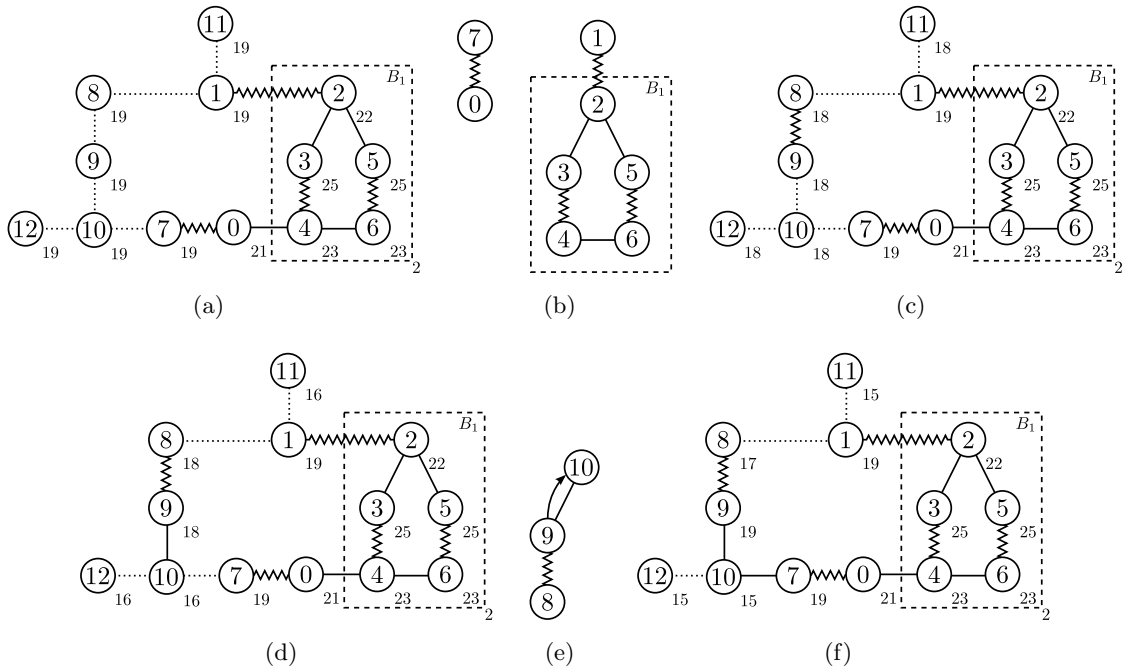


Figura 2.9: *Exemplo de execução do Algoritmo de Edmonds (parte 5).*

O caminho aumentante $\langle 10, 12\rangle$ é encontrado pela expansão da árvore de raiz 10, resultado mostrado pela figura 2.10(c). Um ajuste dos rótulos com $\delta = \delta_3 = l(1) + l(11) - w(1, 11) = 4$ torna a aresta $1, 11$ ativa (figura 2.10(d)) e o resultado da expansão da árvore com raiz 11 é ilustrado na figura 2.10(e). Um último ajuste dos rótulos é feito, com $\delta = \delta_2 = l(11) = 10$, que torna o rótulo da raiz 11 zero. Nesse ponto, as restrições 2.4 e 2.5 são satisfeitas e portanto o emparelhamento obtido é de peso máximo.

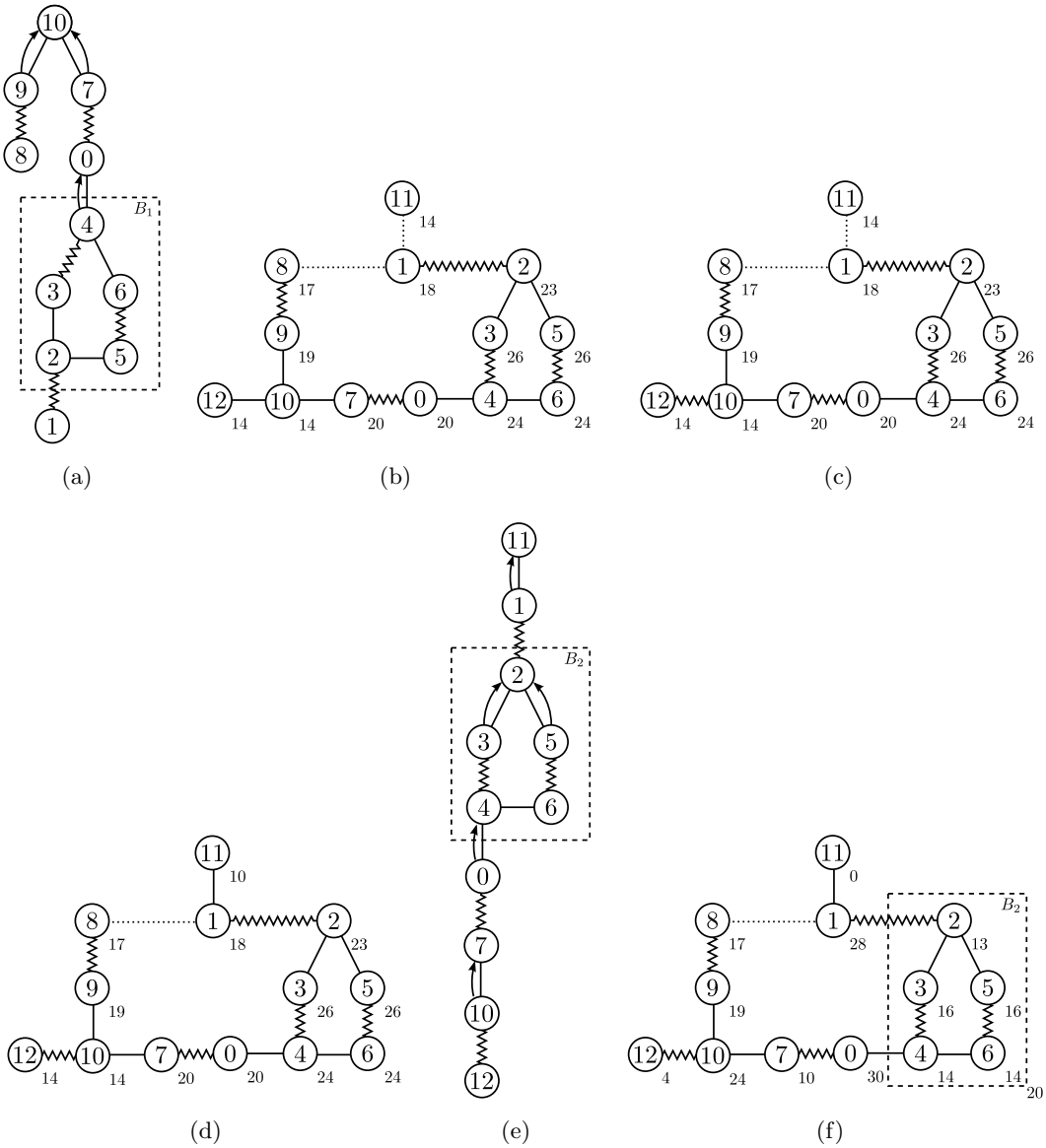


Figura 2.10: *Exemplo de execução do Algoritmo de Edmonds (parte 6).*

Capítulo 3

Implementação do Algoritmo de Edmonds

Descreveremos neste capítulo nossa implementação do Algoritmo de Edmonds. Numa tentativa de reduzir a complexidade do algoritmo, procuramos evitar a operação de contração nos baseando em idéias de Lovász e Plummer [LP86] para o problema de emparelhamento de cardinalidade máxima. Junto com a apresentação dessas idéias, faremos uma análise de complexidade.

A operação de contração foi substituída por uma estrutura de dados que permite tratar os vértices que deveriam estar contraídos como um único vértice. Dessa maneira, os emparelhamentos e caminhos aumentantes utilizados ao longo do algoritmo estão sempre associados ao grafo original.

Informações sobre os pseudo-vértices serão armazenadas em uma lista, sem alterar o grafo original e sem a criação de um novo grafo. Os elementos dessa lista conterão informações sobre os pseudo-vértices como o valor de l e outras que descreveremos mais tarde. O uso de uma lista é justificado pela necessidade de se adicionar e remover pseudo-vértices ao longo de todo o algoritmo. Essa escolha permite a adição e remoção em tempo constante.

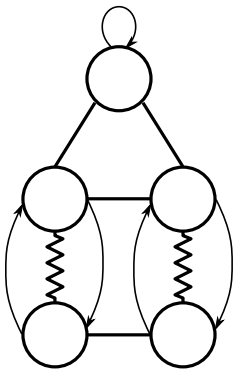
A cada passo da busca por caminhos aumentantes, uma aresta do grafo é analisada e uma ação é tomada de acordo com o tipo de um vértice (par, ímpar ou fora da floresta). Para que esse procedimento funcione corretamente sem a contração dos blossoms, a estrutura de dados precisa permitir que, dado um vértice v, seja possível identificar qual é o blossom exterior que contém v ou então determinar que não existe tal blossom. Para tanto, utilizaremos o mapeamento $ex : V \to O$, sendo $ex(v)$ o blossom exterior que contém v. Se nenhum blossom contém v, então $ex(v) = v$.

Em vários momentos será necessário explorar a estrutura interna de um pseudo-vértice, tratando-o como se esse não estivesse contraído. Para tanto, é necessário conhecer todos os blossoms que contêm um dado vértice. Por isso, cada vértice possui uma pilha dos blossoms que o contém. Para um vértice v, $ex(v)$ está sempre no topo dessa pilha. Denotaremos por $prev(v, B)$ o blossom que contém v e antecede B na pilha de blossoms de v. Se esse blossom não existe, $prev(v, B) = v$.

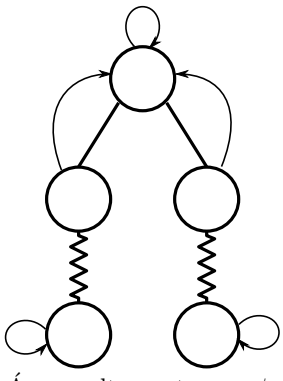
O emparelhamento M utilizado ao longo do algoritmo será representado por um mapeamento $\mu : V \to V$. Se $uv \in M$, então $\mu(u) = v$ e $\mu(v) = u$. Para um vértice livre w, $\mu(w) = w$.

A árvore alternante será definida por μ e um outro mapeamento ϕ que detalharemos a seguir. Os dois mapeamentos juntos devem indicar como obter um caminho alternante partindo de um vértice da árvore até sua raiz. Começando com um vértice inicial e um caminho vazio, iterativamente adicionamos arestas ao caminho usando alternadamente μ e ϕ. ϕ estará definido tanto para

vértices de G quanto para pseudo-vértices. Se o vértice atual está contido num pseudo-vértice B, acrescentamos o caminho que cruza o blossom exterior B antes de usar $\phi(B)$. Descreveremos esse algoritmo em detalhe no final da seção 3.1.



(a) Grafo com μ representado por setas.



(b) Árvore alternante com ϕ representada por setas.

Figura 3.1: *Exemplos mostrando o comportamento dos mapeamentos μ e ϕ.*

Inicialmente, $\phi(v) = v$ para todo $v \in V$. O valor de ϕ para um pseudo-vértice recém criado é NIL. ϕ é alterado à medida que a floresta alternante é construída. Quando uma aresta entre um vértice par u e um vértice v que não pertence a T é analisada, se concluímos que tal aresta deve ser incluída em T, então fazemos $\phi(ex(v)) \leftarrow u$. Quando uma aresta entre dois vértices pares u e v é analisada, um de dois casos acontece: uv forma um circuito ímpar que deveria ser contraído ou então uv pertence a um caminho aumentante. A operação de contração será detalhada na seção 3.1.

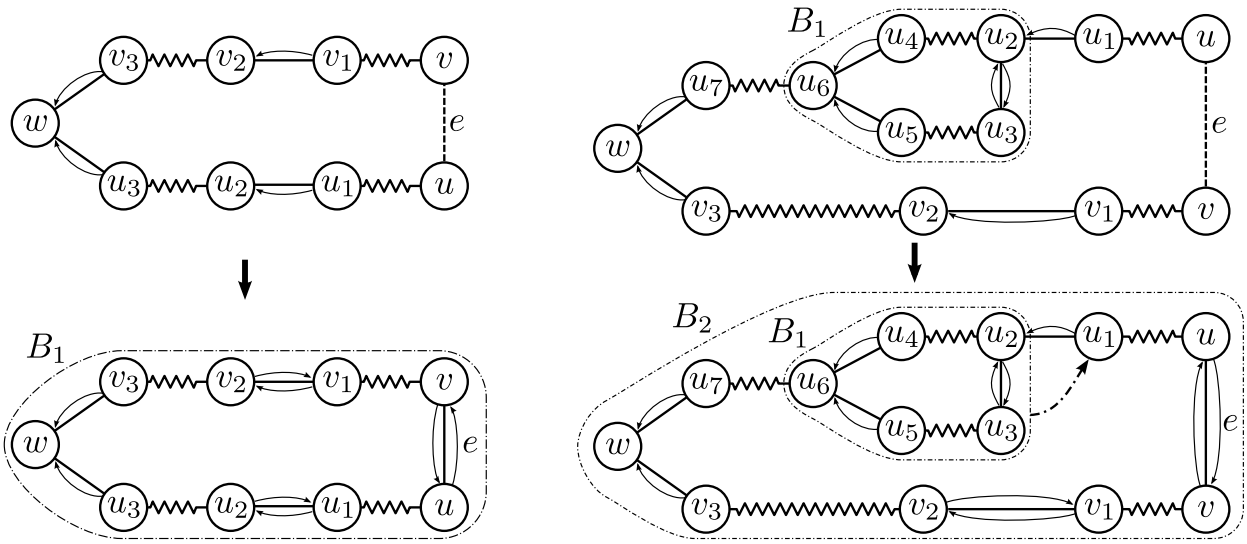
No caso de uv ser parte de um caminho aumentante, alteramos μ de forma a representar o emparelhamento correspondente a inverter o papel das arestas ao longo desse caminho. A alteração pode alterar bases de blossoms e por isso é necessário alterar ϕ também. Essas alterações são discutidas na seção 3.2. Depois de feitas as alterações de μ e ϕ, o algoritmo continuará com uma nova floresta alternante. Por isso, para todo vértice v tal que $ex(v) = v$, fazemos $\phi(v) \leftarrow v$ e para todo blossom B, $\phi(B) \leftarrow NIL$ e $exit(B) \leftarrow NIL$.

Se todas as arestas foram analisadas sem que um caminho aumentante tenha sido encontrado, então faremos uma alteração dos rótulos, que mostraremos na seção 3.3.

3.1 Contração

Seja uv uma aresta de G tal que $ex(u)$ e $ex(v)$ são pares. Seja w o mínimo ancestral comum entre u e v na árvore T e sejam P_u e P_v, respectivamente, os caminhos de u até w e de v até w. A contração do blossom formado por uv, P_u e P_v é feita, sem a alteração de G, da seguinte forma:

1. Suponha que
$$P_v = \langle v_o = v, v_1, \ldots, v_k = w \rangle .$$
Sejam
$$P'_v = \left\langle v'_0 = ex(v_0), v'_1 = ex(v_1), \ldots, v'_k = ex(v_k) \right\rangle$$



(a) Exemplo simples de contraçao, com $P_v = P'_v = P''_v = \langle v, v_1, v_2, v_3, w\rangle$ e $P_u = P'_u = P''_u = \langle u, u_1, u_2, u_3, w\rangle$.

(b) Exemplo de contração mais complexo. Aqui, $P_v = P'_v = P''_v = \langle v, v_1, v_2, v_3, w\rangle$, mas $P_u = \langle u, u_1, u_2, u_4, u_6, u_7, w\rangle$, $P'_u = \langle u, u_1, B_1, B_1, B_1, u_7, w\rangle$ e $P''_u = \langle u, u_1, B_1, u_7, w\rangle$.

Figura 3.2: *Exemplos de contração. As setas representam o valor de ϕ. Omitimos as setas para os vértices x com $\phi(x) = x$.*

.

e P''_v a sequência obtida a partir de P'_v eliminando-se repetições. Note que $P''_v = \langle v''_0, \ldots, v''_t\rangle$ é um caminho no grafo que seria obtido pela contração dos blossoms ao longo do algoritmo.

Para $0 < i < t$, se i é par, faça $\phi(v''_i) \leftarrow v''_{i-1}$. Caso contrário, faça $\phi(v''_i) \leftarrow v''_{i+1}$.

Defina P_u, P'_u e P''_u analogamente e repita o procedimento para P''_u.

2. Faça $\phi(v''_0) = u_0$ e $\phi(u''_0) = v_0$.

3. Faça $\phi(v''_t) = v''_t$.

4. Adicione um novo elemento B na lista de blossoms. Para todo vértice x em P_u e P_v, empilhe B a sua lista de blossoms e faça $ex(x) \leftarrow B$.

Descreveremos esse algoritmo em pseudo-código no final desta seção.

Como um blossom é um circuito ímpar, a configuração de ϕ construída pelo algoritmo acima nos permite encontrar um caminho alternante que começa em qualquer vértice e termina na raiz do blossom, com a primeira aresta pertencendo ao emparelhamento. Isso acontece porque para um vértice v contido num blossom B, $\phi(v)$ indica qual a aresta do blossom B que não pertence a M incide em v. Como a base b é um vértice livre, o caminho alternante par em B entre um vértice qualquer e b começará com uma aresta de M ou então esse caminho terá comprimento zero. O seguinte algoritmo é uma solução recursiva para encontrar esse caminho.

CAMINHO-BLOSSOM(v, B)

1 $v_0 \leftarrow v$
2 $i \leftarrow 0$
3 **while** $v_i \neq b(B)$
4 **do if** o primeiro blossom que contém v_i é B

```
5          then if  i é par
6                  then v_{i+1} ← μ(v_i)
7                  else  v_{i+1} ← φ(v_i)
8          else   seja B' o blossom anterior a B na pilha de blossoms de v_i
9                 P ← Caminho-Blossom(v, B')
10                concatene P ao caminho atual
11                i ← i + |P|
12                if v_i = b(B)
13                  then  pare
14                if  i é par
15                  then v_{i+1} ← μ(v_i)
16                  else  v_{i+1} ← φ(ex(v_i))
17        i ← i + 1
18
19 return  ⟨v_0, v_1, ..., v_i − 1⟩
```

O algoritmo acima supõem que pseudo-vértices são sempre pares. Porém, isso não é sempre verdade, como mostra a figura 3.3. Quando a árvore é expandida, é possível que um blossom contraído numa iteração anterior se torne um blossom ímpar. Nesse caso, o primeiro vértice do blossom a ser visitado será sua base. No entanto, o último vértice desse caminho deve ser o vértice que forma a aresta com o vértice indicado pelo valor de ϕ para esse blossom. Se soubermos qual é esse vértice, podemos encontrar o caminho desse vértice até a base e invertê-lo. Para isso, usamos um mapeamento $exit : O \to V$. Toda vez que o valor de ϕ é alterado para um blossom, alteramos $exit$ de forma que $\{\phi(B), exit(B)\}$ seja a aresta do grafo pela qual o blossom B está "pendurado" na árvore.

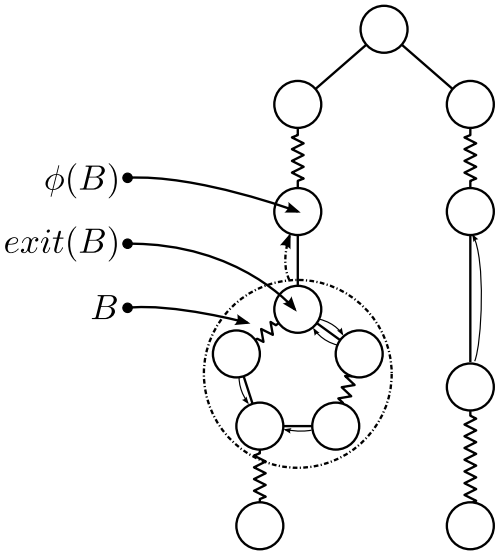


Figura 3.3: *Exemplo de árvore alternante com um pseudo-vértice ímpar.*

Abaixo, mostramos a versão do algoritmo acima que funciona para blossoms ímpares. Note que essa versão tem um parâmetro a mais: *emp*. Se *emp* é verdadeiro, então a primeira aresta do caminho será uma aresta do emparelhamento. Caso contrário, então a primeira aresta será livre.

```
CAMINHO-BLOSSOM-2(v, B, emp)
if emp e v = b(B)
   then v ← exit(B)
        emp ← emp̄
v_0 ← v
i ← 0
while v_i ≠ b(B)
    do if  o primeiro blossom que contem v_i é B
          then if emp
                  then v_{i+1} ← μ(v_i) ELSE v_{i+1} ← φ(v_i)
          else   seja B' o blossom anterior a B na pilha de blossoms de v_i
                 P ← CAMINHO-BLOSSOM-2(v, B', emp)
                 concatene P ao caminho atual
                 i ← i + |P|
                 if v_i = b(B)
                    then  pare
                 if emp
                    then v_{i+1} ← μ(v_i)
                    else  v_{i+1} ← φ(ex(v_i))
       emp ← emp̄
       i ← i + 1

return  ⟨v_0, v_1, ..., v_i − 1⟩
```

Usando esse algoritmo, fica fácil determinar o caminho alternante entre qualquer vértice e a raiz da árvore que o contém. Começamos com um vértice v e, enquanto uma raiz não é atingida, substituímos o vértice atual por $\mu(v)$ ou $\phi(v)$, dependendo se o passo atual é par ou ímpar. Se atingirmos um blossom, concatenamos o resultado do algoritmo CAMINHO-BLOSSOM-2 e continuamos. Segue a descrição em pseudo-código desse algoritmo.

```
CAMINHO-ATE-RAIZ(v)
v_0 ← v
i ← 0
repeat
        if  nenhum blossom contém v_i
           then if  i é par
                   then v_{i+1} ← μ(v_i)
                   else  v_{i+1} ← φ(v_i)

           else  B ← ex(v_i)
                 P ⟨p_0, p_1, ..., p_k⟩ ← CAMINHO-BLOSSOM-2(v_i, ex(v_i),  i é par )
                 for 0 < j ≤ k
                     do v_{i+j} ← p_j
                 i ← i + k
```

```
                    if  i é par
                       then v_{i+1} ← μ(v_i)
                       else  if φ(B) ≠ NIL
                                then v_{i+1} ← φ(v_i)
                                else  v_{i+1} ← v_i


          i ← i + 1

    until v_i ≠ v_{i-1}
  return ⟨v_0, v_1, ..., v_{i-1}⟩
```

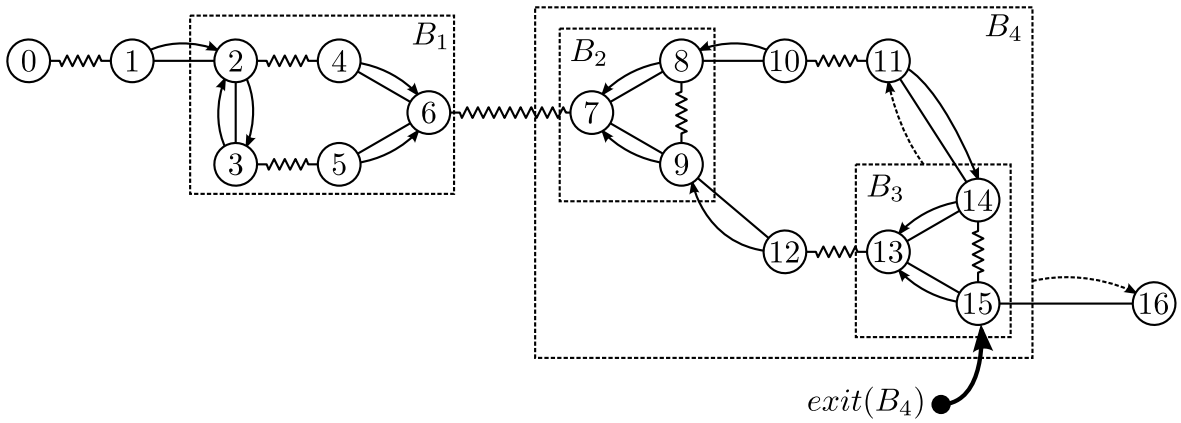


Figura 3.4: *Árvore alternante com blossoms pares e ímpares.*

Considere a árvore alternante da figura 3.4, que tem o vértice 16 como raiz. Se executarmos o algoritmo Caminho-Ate-Raiz iniciando com o vértice 0, teremos como resposta o caminho $\langle 0, 1, 2, 4, 6, 7, 8, 9, 12, 13, 14, 15, 16\rangle$. Vamos mostrar essa execução em detalhes.

A execução inicia com o vértice 0 e o primeiro passo será par. Depois do primeiro passo, o caminho parcial é $\langle 0, \mu(0) = 1\rangle$. O passo seguinte é ímpar e então obtemos $\langle 0, 1, \phi(1) = 2\rangle$. Como B_1 contém o vértice 2, a subrotina Caminho-Blossom-2(B_1, 2, True) será invocada.

Caminho-Blossom-2 irá alternar o uso de μ e ϕ, iniciando com μ, devolvendo o caminho $\langle 2, \mu(2) = 4, \phi(4) = 6\rangle$. Esse caminho é concatenado ao anterior, resultando em $\langle 0, 1, 2, 4, 6\rangle$. O passo que levou até o blossom era ímpar, então o passo seguinte será par. $\mu(6)$ leva ao vértice 7, contido nos blossoms B_2 e B_4, provocando a invocação de Caminho-Blossom-2(B_4, 7, False).

Essa chamada a Caminho-Blossom-2 inicia com um passo ímpar. No entanto, $\phi(7) = 7$ e $\phi(B_2) = NIL$, pois eles são a base do blossom B_2 e B_4, respectivamente. Em outras palavras, B_4 é um blossom ímpar e o algoritmo terá que encontrar o caminho começando pelo fim, usando o vértice $exit(B_4) = 15$. Devido à inversão, o passo inicial será par. Mas como 15 pertence a B_3, Caminho-Blossom-2 é chamada recursivamente para obter o caminho que cruza B_3, $\langle 15, 14, 13\rangle$. Tendo cruzado B_3, o algoritmo dará um passo par e outro ímpar, obtendo o caminho $\langle 15, 14, 13, 12, 9\rangle$ terminando no vértice 9, contido pelo blossom B_2. Com mais uma chamada a Caminho-Blossom-2, o caminho $\langle 15, 14, 13, 12, 9, 8, 7\rangle$, que termina no vértice 7, base do blossom B_4, é encontrado. Como a procura por esse caminho foi feita ao contrário, o caminho $\langle 7, 8, 9, 12, 13, 14, 15\rangle$ é devolvido.

O passo final que levou a B_4 foi par, portanto o passo seguinte é ímpar. $\phi(B_4) = 16$ nos leva a raiz da árvore, completando o caminho $\langle 0, 1, 2, 4, 6, 7, 8, 9, 12, 13, 14, 15, 16\rangle$.

Voltemos agora ao algoritmo de contração de blossoms. O pseudo-código abaixo implementa o procedimento descrito no início desta seção. No entanto, ele não cria os caminho P' e P''. Em vez disso, os caminhos P_u e P_v são percorridos até que o ponto comum entre os dois (w) seja atingido. Durante o percurso, os valores de ϕ são alterados apropriadamente. Quando encontramos um blossom, no entanto, alguns cuidados adicionais são necessários.

Além do valor de ϕ, os blossoms devem ter um valor apropriado para *exit*. Também é preciso cuidado com as listas de blossoms. O empilhamento do novo blossom B não se resume aos vértices ao longo do caminho. É necessário empilhar B na lista de blossoms de todos os vértices contidos em blossoms ao longo do caminho, mesmo que esses não façam parte do caminho que induziu a contração. Segue o pseudo-código do algoritmo.

```
CONTRAI-BLOSSOM(P_u, P_v, w)
 1  Crie o novo blossom B.
 2  b(B) ← w
 3  for P = ⟨x_0, x_1, ..., x_k⟩ ∈ {P_u, P_v}
 4      do i ← 0
 5          while x_i ≠ w
 6              do if ex(x_i) ≠ x_i
 7                     then if i > 0  e par
 8                              then φ(ex(x_i)) ← x_{i-1}
 9                                   exit(ex(x_i)) ← x_i
10                          if i  é impar
11                              then while x_i ≠ exit(ex(x_i))
12                                         do i ← i + 1
13                              else  while x_i ≠ b(ex(x_i))
14                                         do i ← i + 1
15                          if i > 0  e ímpar
16                              then φ(ex(x_i)) ← x_{i+1}
17                                   exit(ex(x_i)) ← x_i
18                          for y ∈ ex(x_i)
19                              do  Empilhe B na lista de blossoms de y.
20                     else  if i > 0
21                              then if  i é par
22                                       then φ(x_i) ← x_{i-1}
23                                       else  φ(x_i) ← x_{i+1}
24                          Empilhe B na lista de blossoms de x_i.
25                 i ← i + 1
26  φ(ex(v_0)) ← u_0
27  if ex(v_0) ≠ v_0
28     then exit(ex(v_0)) = v_0
29  φ(ex(u_0)) ← v_0
30  if ex(u_0) ≠ u_0
```

```
    then exit(ex(u_0)) = u_0
if ex(w) ≠ w
    then φ(ex(w)) = NIL
         exit(ex(w)) = NIL
         for x ∈ ex(w)
             do Empilhe B a lista de blossoms de x.
    else φ(w) ← w
         Empilhe B a lista de blossoms de w.
```

Note que esse algoritmo executa em tempo $O(|V|)$, uma vez que o número de vértices nos caminhos P_u e P_v e também o número de vértices contidos em blossoms é limitado pelo número de vértices do grafo.

3.2 Usando um caminho aumentante

Um outro resultado possível da análise de uma aresta entre vértices pares é a identificação de um caminho aumentante. Sejam u e v vértices pares tais que $uv \in E_l$, $P_v = \langle v_0, v_1, \ldots, v_t \rangle$ o caminho alternante de v até a raiz da árvore que contém v e $P_u = \langle u_0, u_1, \ldots, u_s \rangle$ o caminho alternante de u até a árvore que contém u. Como as raízes são livres, sabemos que $u_0u_1, v_0v_1 \in M$ e portanto, $P = \langle v_t, \ldots, v_1, v_0, u_0, u_1, \ldots, u_s \rangle$ é um caminho aumentante. Alterar μ de forma a representar o emparelhamento $M \Delta E(P)$ é simples.

1. Para $0 < i \leq t$, faça $\mu(v_i) \leftarrow v_{i+1}$ se i é ímpar ou $\mu(v_i) = v_{i-1}$, caso contrário.
2. Para $0 < i \leq s$, faça $\mu(u_i) \leftarrow u_{i+1}$ se i é ímpar ou $\mu(u_i) = u_{i-1}$, caso contrário.
3. Faça $\mu(v_0) \leftarrow u_0$ e $\mu(u_0) \leftarrow v_0$.

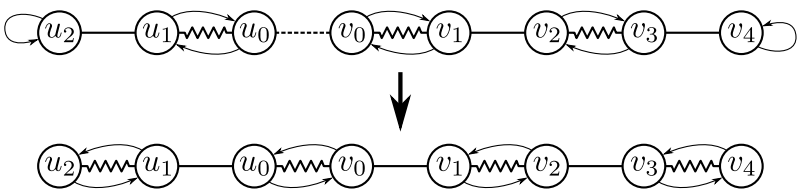


Figura 3.5: *Exemplo de alteração de papel de arestas usando um caminho aumentante. As setas representam o valor de μ.*

Depois de uma aumentação, uma nova floresta alternante será construída. No entanto, blossoms continuam contraídos e como P pode passar por um pseudo-vértice, precisamos alterar ϕ de forma que continue sendo possível encontar caminhos alternantes que cruzam blossoms. Como a implementação das alterações de ϕ que descreveremos abaixo necessitam do valor de μ antes das alterações, essa alteração será feita por último.

Seja $P^B = \langle v_0^B, \ldots, v_k^B \rangle$ a intersecção entre P e um blossom exterior B. Sabemos que P^B tem comprimento par. Suponha, s.p.g., que $v_0^B, v_1^B \notin M$. Então, $prev(v_0^B, B)$ é base de B. Após a aumentação, a aresta $v_0^B v_1^B$ passa a fazer parte de M e a aresta $v_{k-1}^B v_k^B$ é removida de M e portanto $prev(v_0^B, B)$ deixa de ser a base de B, que passa a ser $prev(v_k^B, B)$.

Para a nova base $prev(v_k^B, B)$, fazemos $\phi(prev(v_k^B, B)) \leftarrow prev(v_k^B, B)$. Para os vértices intermediários desse caminho, a alteração é bastante similar a de μ. Se P^B não cruza nenhum blosom, a alteração de ϕ é

$$\begin{aligned} \phi(v_i^B) &\leftarrow v_{i-1}^B \quad \text{se } i \text{ é ímpar,} \\ \phi(v_i^B) &\leftarrow v_{i+1}^B \quad \text{se } i \text{ é par,} \end{aligned} \tag{3.1}$$

que pode ser interpretada como fazer ϕ apontar para o lado contrário de μ. Essa caso é ilustrado pela figura 3.6. No entanto, a existência de blossoms em P^B complica bastante o processo. Além de modificar ϕ para os vértices ao longo do caminho, teremos que modificar ϕ para os blossoms contidos em B.

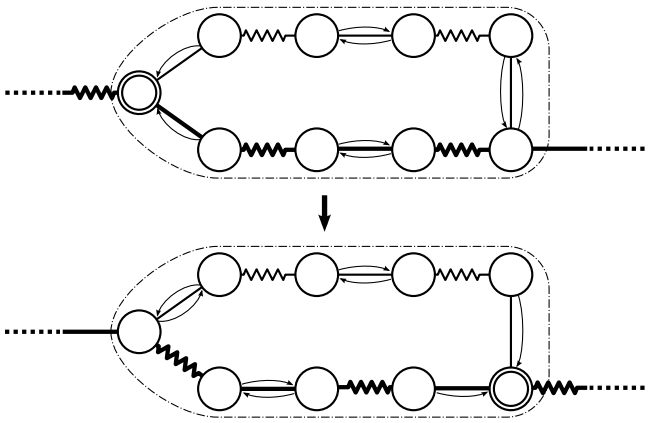


Figura 3.6: *Exemplo de alteração de ϕ utilizando um caminho aumentante que não passa por blossoms.*

O algoritmo que faz a alteração de ϕ é recursivo. Percorremos o caminho aumentante fazendo as alterações descritas em 3.1 quando um vértice é visitado. Quando visitamos um blossom B', a alteração de ϕ é feita recursivamente para B' e alteramos $\phi(B')$ e $exit(B')$ como segue.

Seja i tal que v_i^B é o primeiro vértice de P_B contido em B'. Se a aresta $\{v_{i-1}^B, v_i^B\}$ pertence ao emparelhamento, então

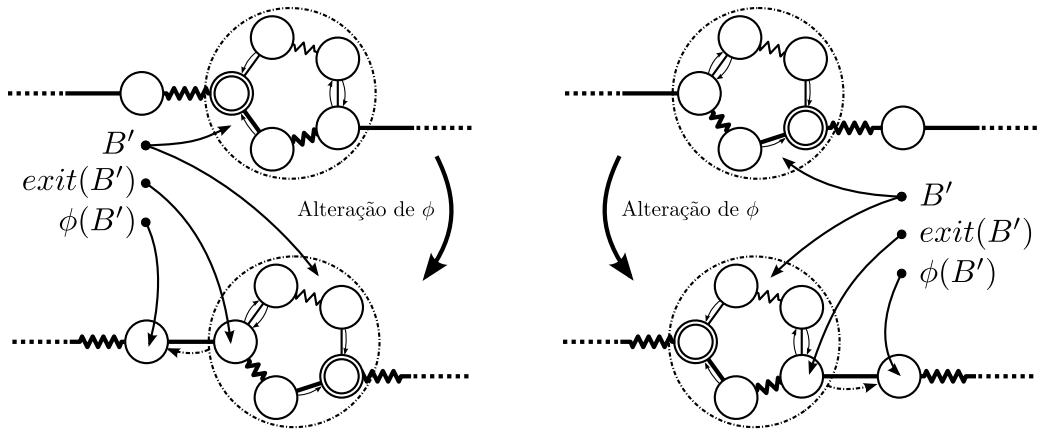
$$\begin{aligned} \phi(B') &\leftarrow v_{i-1}^B \quad \text{e} \\ exit(B') &\leftarrow v_i^B. \end{aligned}$$

Caso contrário, seja j tal que v_j^B seja o último vértice de P^B contido em B'. Então, fazemos

$$\begin{aligned} \phi(B') &\leftarrow v_{j+1}^B \quad \text{e} \\ exit(B') &\leftarrow v_j^B. \end{aligned}$$

Essa alteração é ilustrada pela figura 3.7.

Como $prev(v_0^B, B)$ era a base de B, $\phi(prev(v_0^B, B)) = prev(v_0^B, B)$. Com a aumentação, os caminhos alternantes irão terminar em v_k^B ao invés de v_0^B e portanto precisamos alterar ϕ de forma que seja possível dar um passo "ímpar" partindo de $prev(v_0^B, B)$. Como o caminho até a base deve ter comprimento par, alteraremos $\phi(prev(v_0^B, B))$ para que prossiga até a base pelo caminho de comprimento ímpar até a base no circuito que forma B. Esse caminho é disjunto de P^B e portanto as alterações de ϕ para $v_1^B, \ldots, v_{k-1}^B$ não o altera. Para fazer esse alteração, precisamos



(a) $\{v_{i-1}^B, v_i^B\} \in M$ (b) $\{v_{i-1}^B, v_i^B\} \notin M$

Figura 3.7: *Exemplo de alteração de ϕ quando o caminho aumentante cruza um blossom.*

identificar qual é o caminho ímpar até a base antiga antes de fazer qualquer alteração a μ e ϕ. Esse caminho pode ser calculado por Caminho-Blossom-2$(\phi(prev(v_k^B, B)), B, \text{True})$. O novo valor de $\phi(prev(v_0^B, B))$ deve ser o último vértice v desse caminho tal que $prev(v, B) \neq prev(v_0^B, B)$. Se $prev(v_k^B, B) = prev(v_0^B, B)$, o blossom $prev(v_k^B, B)$ deve ser alterado recursivamente e B fica inalterado.

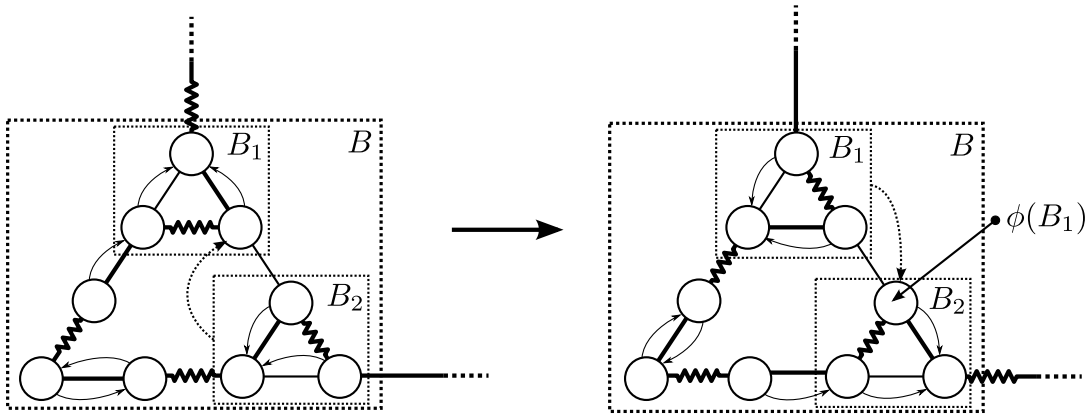


Figura 3.8: *Exemplo mostrando o valor de ϕ para a antiga base de um blossom. Note que a base do blossom B é o blossom B_1. Após a alteração de ϕ usando-se o caminho alternante destacado, $\phi(B_1)$ passa apontar para o primeiro vértice de B_2.*

Segue a descrição do algoritmo que faz as alterações de ϕ. B é o blossom que deve ser alterado, v é o primeiro vértice do caminho aumentante que pertence a B e *emp* uma variável booleana que indica se a próxima aresta do caminho aumentante pertence ou não ao emparelhamento. A determinação de qual deve ser o valor de ϕ e *exit* para a antiga base do blossom foi separada no procedimento Novo-Phi. O procedimento Atualiza-Blossom devolve o último vértice do caminho aumentante que pertence ao blossom B.

Novo-Phi(B, v, emp)

1 $\phi' \leftarrow$ NIL

2 $exit' \leftarrow$ NIL

3

```
if prev(v, B) = prev(b(B), B)
   then return (NIL, NIL)

if φ(prev(v, B)) = b(B)
   then φ' ← b(B)
        exit' ← exit(prev(v, B))
   else P ⟨v_0, ..., v_k⟩ ← CAMINHO-BLOSSOM-2(B, φ(prev(v, B)), emp)
        if prev(b(B), B) = b(B)
           then φ' ← v_{k-1}
           else i ← k
                repeat
                       i ← i − 1
                  until prev(v_i, B) = prev(b(B), b)
                φ' ← v_i
                exit' ← v_{i+1}

return (φ', exit')
```

ATUALIZA-BLOSSOM(B, v, emp)

```
inverter ← FALSE
if emp = FALSE e v = b(B)
   then v ← exit(B)
        emp ← TRUE
        inverter ← TRUE
(φ', exit') ← NOVO-PHI(B, v, tmp)
P ⟨v_0, v_1, ..., v_k⟩ ← CAMINHO-BLOSSOM-2(v, B, emp)
i ← 0
while v_i ≠ b(B)
    do if prev(v_i, B) = v_i
          then if i = 0
                  then φ(v_i ← v_i)
                  else if emp = TRUE
                          then φ(v_i) ← v_{i+1}
                          else φ(v_i) ← v_{i-1}
          else B' ← prev(v_i, B)
               v ← ATUALIZA-BLOSSOM(B', v_i, emp)
               if emp = FALSE
                  then φ(B') ← v_{i-1}
                       exit(B') ← v_i
               while v_i ≠ v
                   do v ← v + 1
               if v = b(B)
                  then break
```

```
25                  if emp = TRUE
26                     then φ(B') ← v_{i+1}
27                          exit(B') ← v_i
28          i ← i + 1
29          emp ← ¬emp
30
31  if φ' ≠ NIL
32     then φ(prev(b(B), B) ← φ'
33          if prev(b(B), B) é um blossom
34             then exit(prev(b(B), B) ← exit'
35
36  if prev(v, B) é um blossom
37     then φ(prev(v, B)) ← NIL
38          exit(prev(v, B)) ← NIL
39     else φ(prev(v, B)) = prev(v, B)
40
41  b(B) ← v
42
43  if inverter = TRUE
44     then return v
45     else return v_k
```

Determinar *prev* para um dado vértice e um blossom tem complexidade de tempo $O(|V|)$, uma vez que essa operação envolve percorrer a pilha de blossoms do vértice e o número de blossoms é limitado pelo número de vértices de um grafo. Devido à necessidade de percorrer o caminho de comprimento ímpar do blossom sendo atualizado comparando-se o valor de valores na pilha de blossoms, a complexidade de NOVO-PHI é $O(|B| \cdot |V|)$.

Vamos mostrar por indução no tamanho dos blossoms, que o tempo de execução $t(i)$ do laço da linha 9 de ATUALIZA-BLOSSOM para um blossom B tal que $|B| = i$ é $O(|V|^2)$. Note que se $|B| = 3$, então B não contém nenhum pseudo-vértice. Então, a comparação da linha 10 é sempre verdadeira. O laço da linha 9 será executado $O(n)$ vezes e a determinação de *prev* custa tempo $O(n)$ fazendo com que $t(i) = O(n)$, pois $|B|$ é constante.

Suponha que $t(i) = O(in)$, para $i = 1, \ldots, k-1$ e que $|B| = k$. Sejam $B_1, \ldots, B_t$ os blossoms diretamente inferiores a B cruzados pelo caminho alternante percorrido pelo algoritmo. Note que

$$\sum_{i=1}^{t} |B_i| < |B|. \tag{3.2}$$

Seja $f = |B| - \sum_{i=1}^{t} |B_i|$. Note que $f \geq 2$. Temos que

$$t(k+1) = tO(n) + \sum_{i=1}^{t} t(|B_i|) + fO(n),$$

pois *prev* será determinado uma vez para cada vértice ou blossom visitado. De onde segue que

$$t(k+1) = tO(n) + \sum_{i=1}^{t} O(|B_i| \cdot |V|) + fO(n). \tag{3.3}$$

Note que de 3.2, temos que

$$\sum_{i=1}^{t} O(|B_i|) = O(|B|). \tag{3.4}$$

Aplicando 3.4 em 3.3, temos que

$$t(k+1) = tO(|V|) + O(|B| \cdot |V|) + fO(|V|).$$

Como $t + f = O(|B|)$, segue que

$$t(k+1) = O(|B| \cdot |V|).$$

O resultado segue do fato que $|B| = O(|V|)$.

Como já vimos a execução de Novo-Phi tem tempo de execução $O(|B| \cdot |V|)$ e as atribuições feitas depois do laço da linha 9 dependem apenas de *prev*, tendo tempo de execução $O(|V|)$. Portanto, a complexidade de Atualiza-Blossom é $O(|B| \cdot |V|)$.

O algoritmo abaixo percorre um caminho alternante, supondo que a primeira aresta pertence ao emparelhamento e chama Atualiza-Blossom para os blossoms no meio no caminho.

Atualiza-Blossoms-No-Caminho($P = \langle v_0, \ldots, v_k \rangle$)

1 $i \leftarrow 0$
2 **while** $i \neq k$
3 **do if** $ex(v_i) \neq v_i$
4 **then if** i é par
5 **then** $x \leftarrow$ Atualiza-Blossom($ex(v_i), v_i,$ True)
6 **else** $x \leftarrow$ Atualiza-Blossom($ex(v_i), v_i,$ False)
7 **while** $v_i \neq x$
8 **do** $i \leftarrow i + 1$
9 $i \leftarrow i + 1$

Usando um argumento similar ao usado anteriormente para Atualiza-Blossom, temos que a complexidade desse algoritmo é $O(|V|^2)$. Com isso podemos descrever o algoritmo que faz uma aumentação. Dados dois caminhos P_u e P_v tais que $P_u^{-1} \cdot P_v$ é um caminho aumentate, o algoritmo abaixo inverte quais arestas estão e quais arestas não estão no emparelhamento e altera ϕ de forma que seja possível encontrar caminhos alternantes que cruzam blossoms.

Aumenta-Emparelhamento($P_u = \langle u_0, \ldots, u_s \rangle, P_v = \langle v_0, \ldots, v_t \rangle$)

Atualiza-Blossoms-No-Caminho(P_u)
Atualiza-Blossoms-No-Caminho(P_v)
for $P = \langle x_0, \ldots, x_k \rangle \in \{P_u, P_v\}$
do $i \leftarrow 1$
while $i \neq k$
do if i é par
then $\mu(x_i) \leftarrow x_{i-1}$

```
            else  μ(x_i) ← x_{i+1}
        i ← i + 1
μ(v_0) = u_0
μ(u_0) = v_0
```

Claramente, o algoritmo acima tem complexidade $O(|V|^2)$.

3.3 Ajuste dos rótulos

O ajuste dos rótulos é uma operação bastante simples. Consiste em determinar δ, fazer as alterações de l e, de acordo com os rótulos que passem a ter valor zero, fazer pequenas alterações na floresta alternante.

Para determinar δ, é necessário percorrer todos os vértices, arestas e blossoms do grafo para determinar os mínimos descritos pelas equações 2.6, 2.7, 2.8 e 2.9. Como o número de blossoms é limitado pelo número de vértices e o número de arestas é $O(|V|^2)$, a determinação de δ tem complexidade $O(|V|^2)$.

É necessário percorrer todos os vértices e blossoms novamente para alterar o valor de l. Portanto, essa operação leva tempo $O(|V|)$.

Depois de feito o ajuste dos rótulos, é necessário verificar se o rótulo de algum vértice ou blossom se tornou zero. No primeiro caso, o papel das arestas no caminho entre o vértice cujo rótulo se tornou zero e a raiz é invertido. Esse procedimento envolve percorrer o caminho até o topo e alterar μ e ϕ através do algoritmo Atualiza-Blossoms-No-Caminho. O número de vértices com rótulo zero é $O(|V|)$ e portanto essa operação tem complexidade $O(|V|^3)$. No entanto, todos os vértices livres são pares e portanto eles são decrementados em todo ajuste dos rótulos. Como a condição de parada do algoritmo é exatamente que todo vértice livre tenha rótulo zero, então esse caso acontece no máximo uma vez durante a execução do algoritmo.

No segundo caso, os blossoms que tiverem seus rótulos alterados para zero serão expandidos. Expandir um blossom envolve remover um elemento da pilha de cada vértice contido nele e removê-lo da lista de blossoms. Esse procedimento pode ser executado em tempo $O(|V|)$.

Se desconsiderarmos o caso em que o rótulo de algum vértice se torna zero, o ajuste dos rótulos pode ser feito em tempo $O(|V|^2)$. No caso geral, a sua complexidade é $O(|V|^3)$.

Devido a simplicidade do algoritmo de ajuste de rótulos, omitiremos a sua descrição em pseudo-código.

3.4 Camada mais externa do algortimo

Apresentamos até agora as partes intrincadas de nossa implementação. Descrevemos agora a parte mais externa do algoritmo que junta todas as rotinas descritas acima.

A rotina Expande-Floresta recebe como parâmetro um vértice par x tal que x ainda não foi visitado e adiciona à floresta alternante todas as arestas possíveis incidentes em x.

```
Expande-Floresta(x)
aumentou ← False
for e = xy ∈ δ(v)
```

```
    do if e ∉ E_l
          then continue
       if  y é par e ex(x) ≠ ex(y)
          then S_x ← Caminho-Ate-Raiz(x)
               S_y ← Caminho-Ate-Raiz(y)
               Seja w o mínimo ancestral comum de x e y ou NIL se ele não existe.
               if w = NIL
                  then  Aumenta-Emparelhamento(S_x, S_y)
                        aumentou ← True
                        break
                  else  Contrai-Blossom(S_x, S_y, w)
       if  y está fora da floresta
          then φ(ex(y)) ← x
               if ex(y) ≠ y
                  then exit(ex(y)) ← y
visitado(x) ← True
return aumentou
```

Note que o algoritmo acima tem complexidade $O(|\delta(v)|)$. Considerando que a construção da floresta alternante termina quando todos os vértices foram explorados, a complexidade da construção é $O(|V|^2)$ devido ao número de arestas visitadas.

Finalmente, descrevemos a rotina mais externa do algoritmo.

```
Emparelhamento-Peso-Maximo()
 Rotulo-Inicial()
for v ∈ V
    do visitado(x) ← False
       μ(x) ← x
       φ(x) ← x
       ex(x) ← x
while  existe vértice livre v com l(v) > 0
    do while  existe um vértice x ∈ V tal que x é par e visitado(x) = False .
            do if  Expande-Floresta(x) = True
                  then  Descarta a floresta alternante atual.
                        Marca todos os vértices como não visitados.
               if  existe vértice livre v com l(v) > 0
                  then break
         Ajuste-dos-Rotulos()
```

Como discutimos acima, a complexidade de construir uma floresta alternante é $O(|V|^2)$. A complexidade do algoritmo será determinada pelo número de ajustes de rótulos e aumentações executados. Como o emparelhamento nunca diminui, o número de caminhos aumentantes utilizados é $O(|V|)$. Entre uma aumentação e outra, são realizados um número $O(|V|)$ de ajustes dos rótulos, como mostraremos a seguir.

Quando um ajuste é realizado, o valor de δ é igual a um dos valores $\delta_1, \delta_2, \delta_3$ ou δ_4. Considerando-se que a floresta alternante é mantida entre alterações de l, a seguinte análise é possível.

Se $\delta = \delta_1$, então o rótulo de algum blossom ímpar se torna zero e ele é expandido. Como o número de blossoms é limitado pelo número de vértices e como qualquer blossom criado em seguida será par, esse caso pode acontecer apenas $O(|V|)$ vezes.

Se $\delta = \delta_2$, então o rótulo de algum vértice se torna zero. Mas como discutimos na seção anterior, nesse caso a condição de parada do algoritmo é atingida.

Se $\delta = \delta_3$, então uma aresta entre um vértice par e um vértice fora da floresta entra em E_l. Nesse caso, adicionamos dois vértices a floresta alternante unidos por uma aresta do emparelhamento. O número de vezes que esse caso pode acontecer é claramente $O(|V|)$.

Se $\delta = \delta_4$, então uma aresta entre dois vértices pares entra em E_l. Nesse caso, ou ocorre uma aumentação ou então uma contração. Como os blossoms quando criados são sempre pares e continuam sendo pares até que uma nova floresta alternante seja construída, o número de vezes que esse caso pode acontecer é $O(|V|)$.

Portanto, o número de ajustes de rótulos executados entre aumentações é $O(|V|)$. Dado que o número de aumentações é $O(|V|)$, temos que o número total de ajustes é $O(|V|^2)$. A construção da floresta alternante entre dois ajustes consome tempo $O(|V|^2)$ resultando numa complexidade total do algoritmo de $O(|V|^4)$.

Parte II

Métodos de planos-de-corte

Capítulo 4

Fluxos multi-terminais

Considere uma rede de computadores conectados por *links* de diferentes larguras de banda em que os computadores funcionam também como roteadores. Se queremos transmitir dados entre dois computadores específicos nessa rede, a seguinte pergunta parece natural: qual é a quantidade máxima de banda disponível entre esses dois computadores? Essa é uma possível interpretação informal de um problema de fluxo. Formalizaremos esse conceito na seção 4.1. O problema de fluxo multi-terminal é o de responder a mesma pergunta quando ao invés de dois computadores fixos, desejamos saber a quantidade máxima de banda disponível entre quaisquer dois computadores pertencentes a um conjunto de terminais. Discutiremos esse problema na seção 4.2 e apresentaremos um algoritmo que o resolve na seção 4.3.

À primeira vista, fluxos multi-terminais podem parecer não ter relação com emparelhamentos. No entanto, como veremos no capítulo 5, o algoritmo apresentado aqui será de grande valia na construção de um método de planos-de-corte para o problema de emparelhamento máximo.

4.1 Fluxo

Antes de definir o problema de fluxo, precisamos definir uma rede. Uma rede é uma tupla $(G = (V, E), c, v_s, v_t)$, onde G é um grafo conexo que pode ser dirigido ou não, $c : E \to \mathbb{R}^+$ é uma função que atribui uma *capacidade* a cada aresta de G e v_s e v_t são dois vértices especiais, chamados de *origem* e *destino*. Trataremos grafos não dirigidos como grafos dirigidos em que entre dois vértices v_i e v_j existem os arcos a_{ij} e a_{ji} com igual capacidade. Se uma aresta é formada pelos vértices v_i e v_j, denotaremos por c_{ij} a sua capacidade.

Um *fluxo* numa rede é uma função $x : E \to \mathbb{R}^+$ que satisfaz as seguintes condições:

$$\sum_{ij \in E} x_{ij} - \sum_{jk \in E} x_{jk} = \begin{cases} -f_x, & \text{se } j = s, \\ 0, & \text{se } j \neq s, t, \\ f_x, & \text{se } j = t, \end{cases} \qquad \text{para todo } j \in V, \tag{4.1}$$

$$0 \leq x_{ij} \leq c_{ij} \quad \text{para todo } ij \in E, \tag{4.2}$$

onde f_x é um número não negativo que chamaremos de *valor do fluxo* x. A equação 4.1 representa o fato de que o fluxo é conservado em todos os vértices diferentes da origem e do destino. A restrição 4.2 garante que o fluxo em cada arco está dentro de sua capacidade. Definimos então o *problema do fluxo máximo* como, dada uma rede, determinar um fluxo de valor máximo.

Claramente, a capacidade dos arcos afeta diretamente o valor de um fluxo máximo em uma rede. Por exemplo, numa rede formada por um caminho entre a origem e o destino, o valor de um fluxo máximo será a menor capacidade nessa rede. Esse arco é um gargalo nessa rede. Iremos generalizar o conceito de gargalo através da definição de cortes.

Definimos um *corte* $(X, \overline{X})$ numa rede, onde $X \subset V$ e $\overline{X} = V - X$ como o conjunto de todos os arcos de de G com um extremo em X e outro em $\overline{X}$. Se $v_s \in X$ e $v_t \in \overline{X}$ dizemos que o corte $(X, \overline{X})$ separa v_s e v_t. A *capacidade* de um corte é definida como

$$c(X, \overline{X}) = \sum_{\substack{ij \in E \\ v_i \in X \text{ e } v_j \in \overline{X}}} c_{ij}.$$

Chamaremos de *corte mínimo* um corte que tem capacidade mínima entre aqueles que separam v_s e v_t.

Vamos mostrar que o valor de um fluxo máximo x é menor ou igual à capacidade de um corte $(X, \overline{X})$ que separa v_s e v_t. Note que

$$f_x = \sum_{j \in \overline{X}} \left(\sum_{ij \in E} x_{ij} - \sum_{jk \in E} x_{jk} \right),$$

pois a soma $\sum_i x_{ij} - \sum_k x_{jk}$ é igual a f_x para $j = t$ e igual 0 para qualquer outro vértice v_j de $\overline{X}$. Como para todo arco a_{ij} tal que $v_i, v_j \in \overline{X}$, x_{ij} aparece duas vezes na soma acima, uma vez com sinal positivo e outra com sinal negativo, então

$$f_x = \sum_{\substack{ij \in E \\ v_i \in X, v_j \in \overline{X}}} x_{ij} - \sum_{\substack{jk \in E \\ v_j \in \overline{X}, v_k \in X}} x_{jk}. \tag{4.3}$$

Como $\displaystyle\sum_{\substack{jk \in E \\ v_j \in \overline{X}, v_k \in X}} x_{jk} \geq 0$, temos que

$$f_x \leq \sum_{\substack{ij \in E \\ v_i \in X, v_j \in \overline{X}}} x_{ij} = c(X, \overline{X}). \tag{4.4}$$

Portanto, o valor de um fluxo máximo é menor ou igual à capacidade de um corte que separa v_s e v_t e, em especial, à capacidade de um corte mínimo. A relação entre cortes mínimos e fluxos máximos é ainda mais forte como mostra o teorema a seguir.

Teorema 5 ([FF62])**.** *Para qualquer rede, o valor de um fluxo máximo é igual à capacidade de um corte mínimo.*

Demonstração. Seja x um fluxo máximo que separa v_s e v_t. Já vimos que, para qualquer corte $(X, \overline{X})$ tal que $v_s \in X$ e $v_t \in \overline{X}$, $f_x \leq c(X, \overline{X})$. Portanto, resta apenas mostrar que $f_x \geq c(Y, \overline{Y})$, para algum corte mínimo $(Y, \overline{Y})$.

Considere o conjunto S como o conjunto dos vértices atingíveis a partir de v_s por caminhos da forma $P = \langle v_s = v_0, v_1, \ldots, v_k \rangle$ tais que $x_{i,i+1} < c_{i,i+1}$ ou $x_{i+1,i} > 0$ e tome $\overline{S} = V \setminus S$.

Vamos mostrar que $v_t \in \overline{S}$. Suponha que $v_t \in S$. Seja P um caminho de v_s e v_t como descrito

acima e sejam $\epsilon_1 = \min_{a_{ij} \in E(P)} \{c_{ij} - x_{ij}\}$ e $\epsilon_2 = \min_{a_{ji} \in E(P)} \{x_{ij}\}$. Tome $\epsilon = \min\{\epsilon_1, \epsilon_2\}$. Então, podemos somar ϵ nos arcos x_{ij} e subtrair ϵ dos arcos x_{ji} e obter um fluxo com valor maior que x, o que é uma contradição, pois x é máximo. Logo, $v_t \in \overline{S}$.

Pela construção de S, para todo arco a_{ij} tal que $v_i \in S$ e $v_j \in \overline{S}$, temos que $x_{ij} = c_{ij}$ e para todo arco a_{jk} tal que $v_j \in \overline{S}$ e $v_k \in S$, $x_{jk} = 0$. Logo, pela equação 4.3, temos que o valor de x é

$$f_x = \sum_{\substack{ij \in E \\ v_i \in S, v_j \in \overline{S}}} x_{ij} - \sum_{\substack{ij \in E \\ v_j \in \overline{S}, v_k \in S}} x_{jk} = \sum_{\substack{ij \in E \\ v_i \in S, v_j \in \overline{S}}} x_{ij} = \sum_{\substack{ij \in E \\ v_i \in S, v_j \in \overline{S}}} c_{ij} = c(S, \overline{S}).$$

Portanto, $(S, \overline{S})$ é um corte mínimo pois, caso contrário, a inequação 4.4 seria violada. □

O teorema a seguir será usado na seção 4.3.

Teorema 6. *Sejam $(X, \overline{X})$ e $(Y, \overline{Y})$ cortes mínimos em uma rede qualquer. Então, $(X \cup Y, \overline{X \cup Y})$ e $(X \cap Y, \overline{X \cap Y})$ também são cortes mínimos nessa rede.* □

Ao leitor interessado na prova do teorema acima ou mais informações sobre fluxos, recomendamos a leitura de [Hu69].

4.2 Fluxos multi-terminais

Formalizamos o problema de fluxo multi-terminal como, dados uma rede $(G = (V, E), c, v_s, v_t)$ e um conjunto de vértices terminais $Q \subseteq V$, determinar

$$\max_{(v_i, v_j) \in Q \times Q} \{f(G, c, v_i, v_j)\},$$

onde $f(G, c, v_i, v_j)$ é o valor de um fluxo máximo entre os vértices v_i e v_j, ou seja, o valor de um fluxo máximo na rede (G, c, v_i, v_j). Os resultados apresentados a seguir são válidos apenas para redes em que G é não dirigido. Portanto assumiremos que para quaisquer dois vétices v_i e v_j, o valor do fluxo máximo entre v_i e v_j é igual ao valor do fluxo máximo entre v_j e v_i.

Um algoritmo trivial para resolver esse problema seria simplesmente calcular o fluxo máximo entre todos os pares possíveis de vértices terminais. Se tivermos p terminais, esse algoritmo faria $p(p-1)/2$ cálculos de fluxo máximo. O algoritmo de Gomory e Hu [GH61], no entanto, resolve o mesmo problema com apenas $p-1$ execuções de um algoritmo de fluxo máximo. Note que se todos os vértices são terminais, seriam executados $n-1$ cálculos. Isso é possível porque entre todos os $n(n-1)/2$ pares de vértices numa rede, existem apenas $n-1$ valores distintos de fluxos máximos, como mostraremos a seguir.

Denotaremos por f_{ij} o valor do fluxo máximo entre os vértices v_i e v_j. Vamos mostrar que

$$f_{ij} \geq \min(f_{ik}, f_{kj}), \quad \text{para qualquer } k.$$

Seja $(X, \overline{X})$ um corte mínimo que separa v_i e v_j. Então, $c(X, \overline{X}) = f_{ij}$. Suponha que $k \in X$. Nesse caso, $(X, \overline{X})$ separa v_k e v_j, logo $f_{kj} \leq c(X, \overline{X}) = f_{ij}$. Caso contrário, $k \in \overline{X}$, e portanto $(X, \overline{X})$ separa v_i e v_k. Logo $f_{ik} \leq c(X, \overline{X}) = f_{ij}$. Como pelo menos um dos casos acontece, $f_{ij} \geq \min(f_{ik}, f_{kj})$.

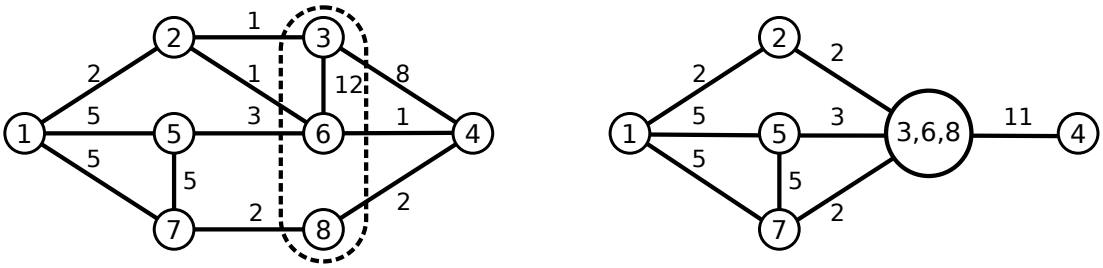
Por indução, temos que $f_{1n} \geq \min(f_{1,2}, f_{2,3}, \ldots, f_{n-1,n})$, para qualquer sequência de vértices $v_1, v_2, \ldots, v_n$.

Usando o resultado acima podemos mostrar que o número de valores de fluxos máximos entre todos os vértices é limitado a $n-1$. Considere um grafo completo em que o peso de cada uma de suas arestas correponde ao valor do fluxo máximo entre seus extremos em uma rede. Seja T uma árvore geradora de peso máximo desse grafo. Suponha que o valor do fluxo f_{ij} entre os vértices v_i e v_j é diferente de todos os pesos das arestas de T. Seja $P = \langle v_1 = v_i, v_2, \ldots, v_k = v_j \rangle$ o caminho entre v_i e v_j em T. Então, $f_{ij} \geq \min(f_{1,2}, f_{2,3}, \ldots, f_{k-1,k})$. Porém, se f_{ik} é maior do que o mínimo, podemos obter uma nova árvore geradora com peso maior do que o de T, o que é uma contradição. Portanto, temos que o valor do fluxo máximo entre quaisquer dois vértices da rede tem um dos $n-1$ valores na árvore geradora T.

O algoritmo de Gomory e Hu irá construir uma árvore similar à descrita acima. Essa árvore terá p vértices e cada aresta será determinada por um corte mínimo obtido através da execução de um algoritmo de fluxo máximo. No início, teremos apenas um vértice e nenhuma aresta. A cada iteração, o fluxo máximo entre dois vértices será calculado e uma aresta da árvore determinada. O algoritmo de fluxo máximo não utilizará a rede original, mas sim uma rede mais condensada obtida a partir dela.

Na figura abaixo, o corte mínimo que separa os vértices 2 e 5 é formado pelas arestas a_{21}, a_{23} e a_{26}. A alteração de qualquer outra aresta da rede não influencia no valor do fluxo máximo entre esses dois vértices, que permanece sendo 4. Usando essa idéia, podemos contrair alguns conjuntos de vértices da rede tornando-a bastante condensada.

A rede resultante da contração de um conjunto de vértices $S \subseteq V$ possui conjunto de vértices $V' = V \setminus S + \{v' \notin V\}$. As arestas entre vértices de $V \setminus S$ estão presentes na rede contraída com suas capacidades originais. As arestas entre um vértice $v_i \in S$ e outro vértice $v_j \notin S$ são substituídas por uma única aresta ligando v_j ao vértice v' e sua capacidade é a soma da capacidade de todas as arestas entre v_j e algum vértice de S. Veja o exemplo da figura 4.1.



(a) Rede antes da contração. (b) Rede após a contração.

Figura 4.1: *Exemplo contração em uma rede. Retirado de [Hu69]*

Para formalizar a idéia de que podemos contrair alguns conjuntos de vértices sem alterar o valor de um fluxo máximo entre certos vértices iremos utilizar a definição de cortes que (não) se cruzam. Dizemos que dois cortes $(X, \overline{X})$ e $(Y, \overline{Y})$ *se cruzam* se e somente se as intersecções $X \cap Y, X \cap \overline{Y}, \overline{X} \cap Y$ e $\overline{X} \cap \overline{Y}$ não são vazias.

Dados dois cortes que não se cruzam $(X, \overline{X})$ e $(Y, \overline{Y})$, um dos seguintes casos acontece:

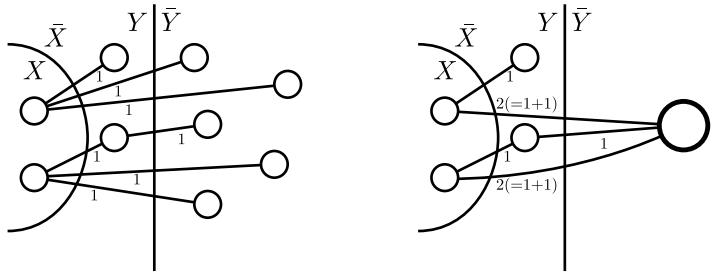
(i) $X \cap Y = \emptyset \Rightarrow X \subseteq \overline{Y} \Rightarrow Y \subseteq \overline{X}$;

(ii) $X \cap \overline{Y} = \emptyset \Rightarrow X \subseteq Y \Rightarrow \overline{Y} \subseteq \overline{X}$;

(iii) $\overline{X} \cap Y = \emptyset \Rightarrow \overline{X} \subseteq \overline{Y} \Rightarrow Y \subseteq X$;

(iv) $\overline{X} \cap \overline{Y} = \emptyset \Rightarrow \overline{X} \subseteq Y \Rightarrow \overline{Y} \subseteq X$.

Ou seja, sempre um dos lados de $(X, \overline{X})$ está contido em um dos lados de $(Y, \overline{Y})$ enquanto que o outro lado de $(X, \overline{X})$ contém o outro lado de $(Y, \overline{Y})$. Se a parte de $(Y, \overline{Y})$ que contém a parte de $(X, \overline{X})$ for contraída como da maneira descrita acima, a capacidade do corte $(X, \overline{X})$ não é alterada porque as arestas entre um vértice de X e vértices de $\overline{Y}$ serão substituídas por uma única arestas cuja capacidade é a soma da capacidade das arestas removidas (veja figura 4.2).



(a) Cortes que não se cruzam $(X, \overline{X})$ e $(Y, \overline{Y})$.

(b) Corte $(X, \overline{X})$ tem a mesma capacidade se contraímos $\overline{Y}$.

Figura 4.2: *Exemplo de cortes que não se cruzam.*

Vamos agora provar alguns resultados sobre a existência de cortes que não se cruzam. Esses resultados serão importantes na prova de corretude do algoritmo de Gomory e Hu.

Lema 3. *Seja $(X, \overline{X})$ um corte mínimo que separa $v_i \in X$ de outro vérice e sejam v_e e v_k dois vértices de $\overline{X}$. Então, existe um corte mínimo $(Z, \overline{Z})$ que separa v_e e v_j tal que $(X, \overline{X})$ e $(Z, \overline{Z})$ não se cruzam.*

Demonstração. Suponha que existe um corte mínimo $(Y, \overline{Y})$ que separa v_e e v_k que cruza $(X, \overline{X})$. Sejam

$$X \cap Y = Q, \quad X \cap \overline{Y} = S,$$
$$\overline{X} \cap Y = P, \quad \overline{X} \cap \overline{Y} = R.$$

Caso 1. Suponha que $v_i \in Q$, $v_e \in P$ e $v_k \in R$ (figura 4.3(a)). Como $(X, \overline{X})$ é um corte mínimo, temos que

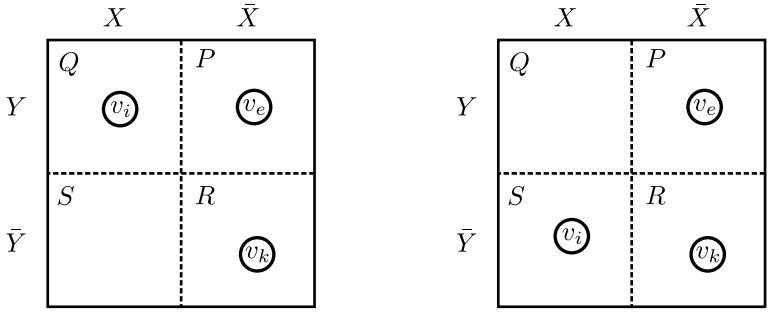
$$c(Q, P) + c(Q, R) + c(S, P) + c(S, R) \leq c(Q, P) + c(Q, R) + c(Q, S),$$

onde $c(S_1, S_2) = \sum_{e \in E(S_1, S_2)} c_e$. Como $c(S, P) \geq 0$, então

$$c(S, R) \leq c(Q, S).$$

Por outro lado, temos que

$$0 \leq c(P, S).$$



(a) Caso 1. (b) Caso 2.

Figura 4.3: *Representação gráfica dos casos da prova do lema 3.*

Somando as duas desigualdades e somando $c(P,R)+c(Q,R)$ em ambos os lados, obtemos

$$\begin{aligned} c(P,R)+c(Q,R)+c(S,R) &\leq c(P,R)+c(Q,R)+c(Q,S)+c(P,S) \\ &= c(Y,\overline{Y}). \end{aligned}$$

Então, $(Z,\overline{Z}) = (P \cup Q \cup S, R)$ é um corte mínimo que separa v_e e v_k e não cruza $(X,\overline{X})$.

Caso 2. Suponha que $v_i \in S$ (figura 4.3(b)). Analogamente, podemos mostrar que $(Z,\overline{Z}) = (P, Q \cup S \cup R)$ é um corte que separa v_e e v_k e cuja capacidade não é maior do que a de $(Y,\overline{Y})$.

□

O resultado do lema 3 implica que podemos contrair X em um único vértices quando formos calcular o fluxo máximo entre vértices de $\overline{X}$.

Lema 4. *Seja $(X,\overline{X})$ um corte mínimo separando v_i e outro vértice e seja $v_e \in \overline{X}$ um vértice qualquer. Então existe um corte mínimo $(Z,\overline{Z})$ que separa v_i e v_e tal que $(X,\overline{X})$ e $(Z,\overline{Z})$ não se cruzam.*

Demonstração. Suponha que existe um corte mínimo $(Y,\overline{Y})$ que separa v_i e v_e tal que $(X,\overline{X})$ e $(Y,\overline{Y})$ se cruzam. Sejam

$$\begin{aligned} X \cap Y = Q, \quad X \cap \overline{Y} = S, \\ \overline{X} \cap Y = P, \quad \overline{X} \cap \overline{Y} = R. \end{aligned}$$

Note que $v_e \in R$ e $v_i \in Q$. Usando o mesmo argumento do caso 1 do lema 3, temos que

$$\begin{aligned} c(P,R)+c(Q,R)+c(S,R) &\leq c(P,R)+c(Q,R)+c(Q,S)+c(P,S) \\ &= c(Y,\overline{Y}). \end{aligned}$$

Então, $(Z,\overline{Z}) = (P \cup Q \cup S, R)$ é um corte mínimo que separa v_i e v_e e que não cruza $(X,\overline{X})$. □

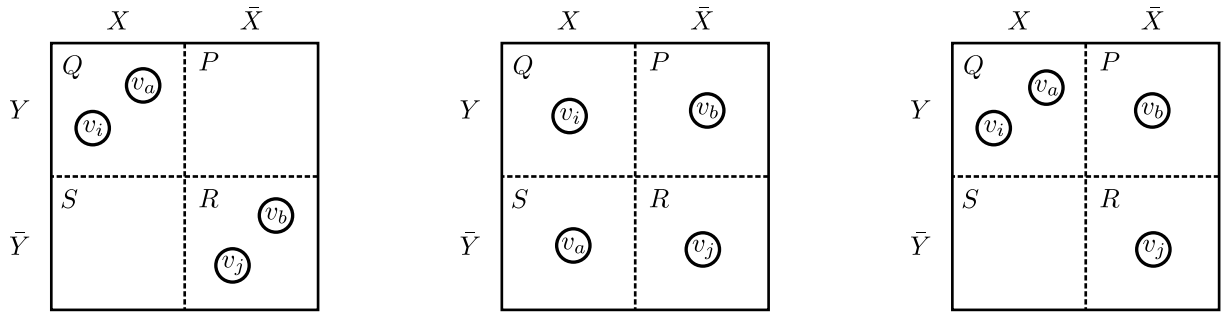
O resultado do lema 4 nos diz que podemos contrair X se estivermos procurando o fluxo máximo f_{ie}.

Lema 5. *Seja $f_{ab} = c(X, \overline{X})$ o valor de um fluxo máximo entre dois vértice v_a e v_b e sejam $v_i \in X$ e $v_j \in \overline{X}$ dois vértices quaisquer. Então, existe um corte mínimo $(Z, \overline{Z})$ que não cruza $(X, \overline{X})$ tal que $c(Z, \overline{Z}) = f_{ij}$.*

Demonstração. Suponha que existe um corte mínimo $(Y, \overline{Y})$ que separa v_i e v_j tal que $(X, \overline{X})$ e $(Y, \overline{Y})$ se cruzam. Sejam

$$X \cap Y = Q, \quad X \cap \overline{Y} = S,$$
$$\overline{X} \cap Y = P, \quad \overline{X} \cap \overline{Y} = R.$$

Note que $v_i \in Q$ e $v_j \in R$.



(a) Caso 1. (b) Caso 2. (c) Caso 3.

Figura 4.4: *Representação gráfica dos casos da prova do lema 5.*

Caso 1. $v_a \in Q$ e $v_b \in R$ (figura 4.4(a)). Podemos usar o mesmo argumento dos lemas 3 e 4 e mostrar que $(Z, \overline{Z}) = (P \cup Q \cup S, R)$ é um corte mínimo que separa v_i e v_j e não cruza $(X, \overline{X})$.

Caso 2. $v_a \in S$ e $v_b \in P$ (figura 4.4(b)). Então, $(S, P \cup Q \cup R)$ e $(P, Q \cup R \cup S)$ são cortes que separam v_a e v_b. Como $(X, \overline{X})$ é um corte mínimo, temos

$$c(Q, P) + c(Q, R) + c(S, P) + c(S, R) \leq c(S, P) + c(S, Q) + c(S, R) \tag{4.5}$$

e

$$c(Q, P) + c(Q, R) + c(S, P) + c(S, R) \leq c(Q, P) + c(R, P) + c(S, P). \tag{4.6}$$

Considerando que $c(Q, R) \geq 0$, de 4.5, temos

$$c(Q, P) \leq c(S, Q) = c(Q, S). \tag{4.7}$$

Como $c(Q, R) \geq 0$, de 4.6, temos que

$$c(S, R) \leq c(R, P) = c(P, R). \tag{4.8}$$

Ainda, temos que

$$c(P, R) + 2c(Q, R) + c(Q, S) \leq c(P, R) + 2c(Q, R) + c(Q, S) + 2c(P, S). \tag{4.9}$$

Somando as inquações 4.7, 4.8 e 4.9, temos

$$[c(Q,P)+c(Q,R)+c(Q,S)]+[c(P,R)+c(Q,R)+c(S,R)] \leq 2[c(P,R]+c(P,S)+c(Q,R)+c(Q,S)],$$

ou seja,

$$c(Q, P \cup R \cup S) + c(P \cup Q, \cup S, R) \leq 2c(Y, \overline{Y}). \tag{4.10}$$

Por outro lado, como $(Q, P \cup R \cup S)$ e $(P \cup Q, \cup S, R)$ separam v_i e v_j, temos que $c(Q, P \cup R \cup S) \geq c(Y, \overline{Y})$ e $c(P \cup Q, \cup S, R) \geq c(Y, \overline{Y})$. Portanto, temos que

$$c(Q, P \cup R \cup S) = c(P \cup Q, \cup S, R) = c(Y, \overline{Y}).$$

Como nenhum dos cortes cruza $(X, \overline{X})$, podemos tomar $(Z, \overline{Z}) = (Q, P \cup R \cup S)$ ou $(Z, \overline{Z}) = c(P \cup Q, \cup S, R)$.

Caso 3. $v_a \in Q$ e $v_b \in P$ (figura 4.4(c)). Então, $(Q, P \cup R \cup S)$ é um corte que separa v_a e v_b. Logo,

$$c(X, \overline{X}) \leq c(Q, P \cup R \cup S)$$

ou

$$c(Q,P) + c(S,P) + c(Q,R) + c(S,R) \leq c(Q,P) + c(Q,R) + c(Q,S).$$

Como $c(S,P) \geq 0$, temos que

$$c(S,R) \leq c(Q,S)$$

e

$$c(P,R) + c(Q,R) \leq c(P,R) + c(Q,R) + c(P,S).$$

Somando as duas desigualdades acima, obtemos

$$c(P,R) + c(Q,R) + c(S,R) \leq c(P,R) + c(Q,R) + c(P,S) + c(Q,S),$$

ou seja,

$$c(P \cup Q \cup S, R) \leq c(Y, \overline{Y}).$$

Portanto, $(Z, \overline{Z}) = (P \cup Q \cup S, R)$ é um corte mínimo que separa v_i e $_j$ e que não cruza $(X, \overline{X})$.

□

Os lemas acima serão importantes para provar a corretude do algoritmo de Gomory e Hu, que descrevemos a seguir.

4.3 Algoritmo de Gomory e Hu

O algoritmo de Gomory e HU [GH61] resolve o problema de fluxo multi-terminal com apenas $|Q| - 1$ execuções de um algoritmo de fluxo máximo. O resultado do algoritmo é uma árvore cujos vértices são conjuntos de vértices da rede original e as arestas correspondem a cortes mínimos nessa

rede. Chamaremos os vértices da árvore de *círculos*. Ao término do algoritmo, cada círculo contém um vértice terminal e zero ou mais vértices não terminais.

Inicialmente, a árvore é composta de um único círculo contendo todos os vértices da rede. A cada iteração, um círculo contendo pelo menos dois vértices terminais é dividido em outros dois círculos, de forma que cada terminal fica em um dos círculos. Quando não existem mais círculos nessa condição, o algoritmo termina.

A divisão de um círculo C é feita através do cálculo de um corte mínimo $(X, \overline{X})$ obtido como resultado da execução de um algortimo de fluxo máximo. Dois vértices terminais arbitrários contidos no círculo, digamos v_a e v_b, são usados como origem e destino. O algoritmo, no entanto, é executado sob uma rede simplificada através da contração de alguns conjuntos de vértices. O conjunto de vértices de cada componente conexa do grafo obtido pela remoção de C da árvore é contraído em um único vértice.

O círculo C é substituído por dois outros círculos, C^X e $C^{\overline{X}}$, conectados por uma aresta de peso f_{ab}. Os vértices originais da rede que pertencem a X fazem parte de C^X. Analogamente, os vértices originais da rede que pertencem a $\overline{X}$ fazem parte de $C^{\overline{X}}$. As arestas da árvore entre C a um outro círculo C' são substituídas da seguinte forma: se o vértice associado a C' pertence a X, então a aresta $\{C, C'\}$ é substituída por $\{C^X, C'\}$, caso contrário, ela é substituída por $\{C^{\overline{X}}, C'\}$.

A figura 4.5 mostra a execução do algoritmo para a rede representada em 4.5(a) considerando os vértice v_1, v_3, v_4 e v_5 como terminais. Inicialmente, o fluxo máximo entre v_1 e v_3 é calculado e o corte $(\{v_1, v_2, v_6\}, \{v_3, v_4, v_5\})$ é obtido. A figura 4.5(b) ilustra a árvore depois da divisão do círculo inicial.

Em seguida, o fluxo máximo entre v_3 e v_4 é calculado na rede da figura 4.5(c) obtida através da contração de v_1, v_2 e v_6. O corte mínimo obtido é $(\{v_1, v_2, v_3, v_5, v_6\}, \{v_4\})$ e árvore resultante é exibida na figura 4.5(d). Nesse ponto, o único círculo contendo vértices terminais é $\{v_3, v_5\}$ e o cálculo do fluxo máximo é executada na mesma rede da figura 4.5(c). O corte mínimo entre esses dois vétices é $(\{v_1, v_2, v_4, v_5, v_6\}, \{v_3\})$. A figura 4.5(e) mostra a árvore obtida no fim da execução.

Teorema 7. *O valor de um fluxo máximo entre dois vértice terminais quaisquer v_i e v_j é igual a*

$$\min(w_{i,v_1}, w_{v_1,v_2}, \ldots, w_{v_k,v_j}),$$

onde $w_{i,v_1}, w_{v_1,v_2}, \ldots, w_{v_k,v_j}$ são os pesos das arestas que formam o caminho único entre v_i e v_j na árvore gerada pelo algoritmo de Gomory-Hu.

Demonstração. Primeiro, vamos mostrar que se v_a e v_b estão contidos em círculos adjacentes, então a hipótese vale. Considere as duas subárvores obtidas pela remoção da aresta e entre os círculos que contém v_a e v_b. Seja X o conjunto de vértices da subárvore que contém v_a e $\overline{X}$ o conjunto de vértices da subárvore que contém v_b. Então, $(X, \overline{X})$ é um corte que separa v_a e v_b, e portanto, $c(X, \overline{X}) \geq f_{ab}$.

O corte $(X, \overline{X})$ é obtido pelo algoritmo através de um cálculo de fluxo máximo. Logo, ele é um corte mínimo que separa dois vértices, digamos v_i e v_j. Suponha que $v_i \in X$ e $v_j \in \overline{X}$. Pelo lema 5, sabemos que existe um corte mínimo, digamos $(Z, \overline{Z})$, que separa v_a e v_b que não cruza $(X, \overline{X})$. Suponha que $Z \subset X$ e $\overline{X} \subset \overline{Z}$. Os casos em que $Z \subset \overline{X}$, $\overline{Z} \subset X$ e $\overline{Z} \subset \overline{X}$ são análogos, bastando inverter v_i e v_j e/ou v_a e v_b.

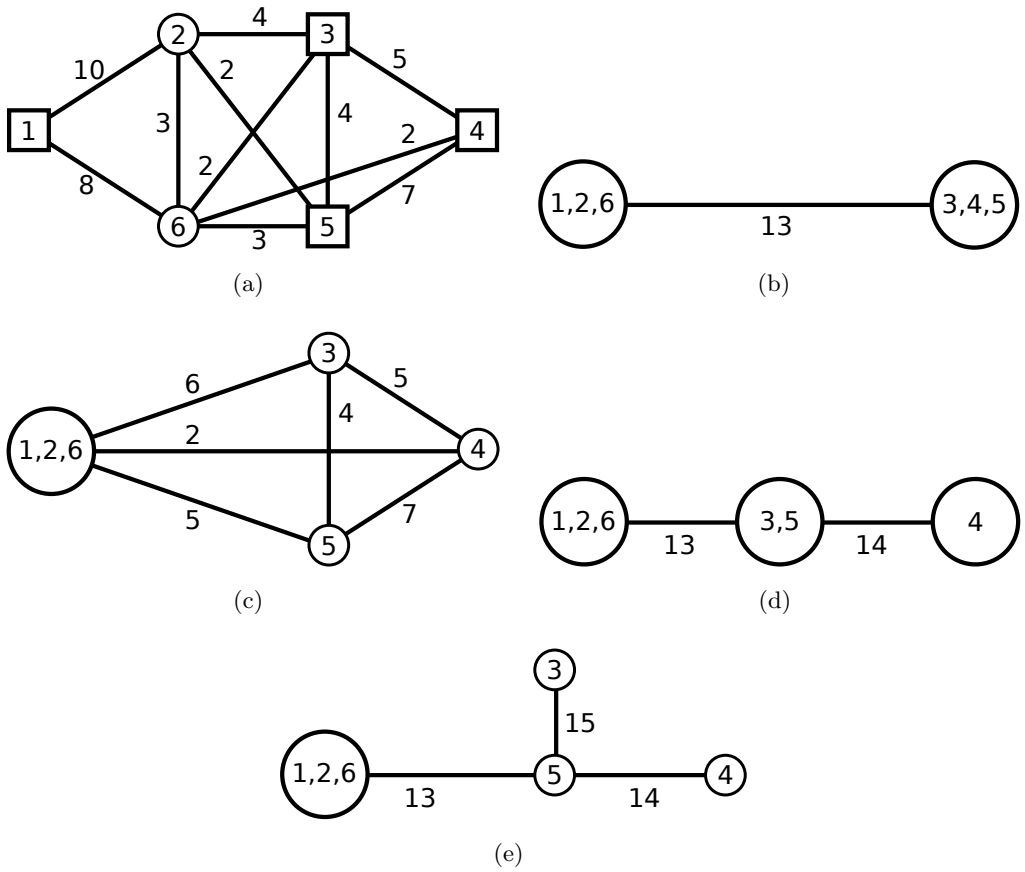


Figura 4.5: *Exemplo de execução do algoritmo de Gomory-Hu retirado de [Hu69]. Os vértices 1, 3, 4 e 5 são terminais.*

Caso 1. Suponha que $v_i \in Z$ (figura 4.6(a)). Note que $v_j \in \overline{X} \subset \overline{Z}$. Logo, $(Z, \overline{Z})$ separa v_i e v_j. Como $(X, \overline{X})$ é um corte mínimo que separa v_i e v_j, temos que

$$c(Z, \overline{Z}) \geq c(X, \overline{X}) = f_{ij}.$$

Caso 2. Suponha que $v_i \in \overline{Z} \cap X$ (figura 4.6(b)). Então, seja $(Y_1, \overline{Y_1})$ um corte mínimo que separa v_i e v_a e que não cruza $(X, \overline{X})$ (o lema 3 garante a existência de um tal corte). Seja $(Y_2, \overline{Y_2})$ um corte mínimo que separa v_i e v_a e que não cruza $(Z, \overline{Z})$. Então, pelo teorema 6, temos que $(Y, \overline{Y}) = (Y_1 \cap Y_2, \overline{Y_1 \cap Y_2})$ é um corte mínimo que separa v_i e v_a. Note que $(Y, \overline{Y})$ não cruza $(X, \overline{X})$ nem $(Z, \overline{Z})$. Logo, temos que v_a, v_b e v_j pertencem a $\overline{Y}$, pois $(Y, \overline{Y})$ separa v_i e v_a por construção e v_b e v_j estão em $\overline{X}$. Portanto, $(Y, \overline{Y})$ separa v_i e v_j. Como $(X, \overline{X})$ é um corte mínimo que separa v_i e v_j, temos que

$$c(Y, \overline{Y}) \geq c(X, \overline{X}) = f_{ij}$$

Mas como $(Z, \overline{Z})$ também separa v_i e v_a, temos que

$$c(Z, \overline{Z}) \geq f_{ia} = c(Y, \overline{Y}) \geq c(X, \overline{X}).$$

Note que nos dois casos, temos que $c(Z, \overline{Z}) \geq c(X, \overline{X})$. Por outro lado, $(X, \overline{X})$ é um corte que

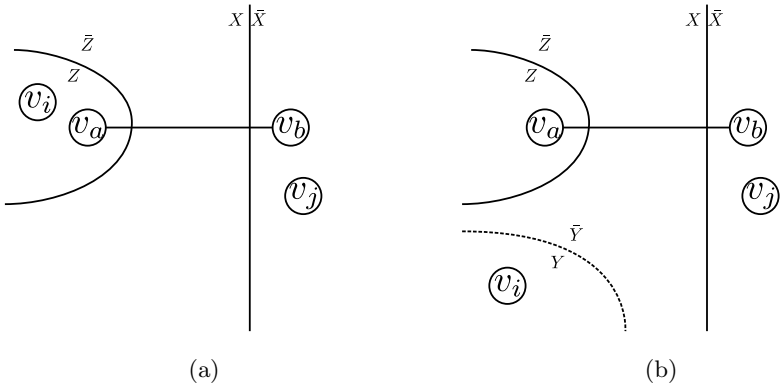


Figura 4.6: *Representação gráfica dos casos da prova do teorema 7.*

separa v_a e v_b e portanto, $c(X, \overline{X}) \geq c(Z, \overline{Z}) = f_{ab}$. Logo, $c(Z, \overline{Z}) = c(X, \overline{X}) = f_{ab}$. Isso prova que o valor do fluxo máximo entre dois vértices em círculos vizinhos é igual ao peso da aresta que os conecta.

Na seção 4.2, mostramos que para dois vértices v_i e v_j, o fluxo máximo f_{ij} é tal que

$$f_{ij} \geq \min(f_{i,v_1}, f_{v_1,v_2}, \ldots, f_{v_k,v_j}) = \min(w_{i,v_1}, w_{v_1,v_2}, \ldots, w_{v_k,v_j}).$$

Por outro lado, como todos os valores no lado direito da equação acima representam cortes separando v_i e v_j, temos que

$$f_{ij} \leq \min(f_{i,v_1}, f_{v_1,v_2}, \ldots, f_{v_k,v_j}) = \min(w_{i,v_1}, w_{v_1,v_2}, \ldots, w_{v_k,v_j}).$$

Logo,

$$f_{ij} = \min(w_{i,v_1}, w_{v_1,v_2}, \ldots, w_{v_k,v_j}).$$

□

Segue uma descrição em pseudo-código do algoritmo.

GOMORY-HU$(G, w : E(G) \to \mathbb{R}^+)$

$C_0 \leftarrow V(G)$
$T \leftarrow (\{C_0\}, \emptyset)$
while Existe um círculo C_i tal que C_i contém dois vértices terminais
  do Sejam $t_1, t_2 \in C_i$ dois vértices terminais.
    $V_i \leftarrow \{v \in V(G) : v \in C_i\} \cup \{c_j \notin V(G) : C_j \neq C_i\}$
    $E_i = \{uv \in E(G) : u, v \in C_i\} \cup \{vc_j : v \in C_i, uv \in E(G) \text{ e } u \in C_j\}$
    $G_i \leftarrow (V_i, E_i)$
    for $uv \in E_i$
      do if $u \in C_i$ **e** $v = c_j$
        then $w_i(uv) \leftarrow \sum_{v \in C_j} w(uv)$
        else $w_i(uv) \leftarrow w(uv)$
    Seja $(X, \overline{X})$ um corte mínimo que separa t_1 e $t2$ na rede $(G_i, w_i, t1, t2)$.
    $C_i \leftarrow C_i \setminus \{v \in \overline{X} | v \in V(G)\}$
    $C_j \leftarrow \{v \in \overline{X} | v \in V(G)\}$

```
15        for  C_iC_k ∈ E(T)
16            do if  c_k ∈ X̄
17                   then E(T) ← E(T) \ {C_iC_k}
18                        E(T) ← E(T) ∪ {C_jC_k}
19        E(T) ← E(T) ∪ {C_iC_j}
20        w(C_iC_j) ← c(X, X̄)
21  return T
```

Note que a cada iteração, um vértice terminal é removido de C_i e acionado a um novo círculo C_j. Portanto, o número de iterações do laço da linha 3 é $|Q| - 1$. As linhas 5 a 11 constroem um grafo contraído. Note que esse processo pode ser executado em tempo $O(|E|)$. A linha 12 faz uso de um algoritmo de fluxo máximo para determinar um corte mínimo no grafo contraído. Em nossa implementação, usamos o algoritmo de Edmonds-Karp, que tem complexidade de tempo $O(|V| \cdot |E|^2)$. As linhas 13 a 18 adicionam um novo círculo a árvore T e altera suas arestas se necessário, tendo complexidade $O(|V|^2)$. A linhas 19 e 20 executam em tempo constante.

Como o número de iterações é $|Q| - 1 = O(|V|)$ e em cada iteração um algoritmo de complexidade $O(|V| \cdot |E|^2)$ é utilizado, a complexidade total de nossa implementação do algoritmo de Gomory e Hu é $O(|V|^2 \cdot |E|^2)$. Existem algoritmos mais eficientes para o problema de fluxo máximo, como o algoritmo Push-Relabel de Goldberg e Tarjan [GT86], que tem complexidade $O(|V| \cdot |E| \cdot \log(|V|^2/|E|))$. Portanto, implementações mais eficientes do algoritmo de Gomory e Hu são possíveis.

No entanto, a implementação utilizando o algoritmo de Edmonds-Karp demonstrou desempenho satisfatório para o método de planos-de-corte que descrevemos no próximo capítulo. Nos testes que apresentamos no capítulo 6, o algoritmo combinatório demonstrou desempenho bastante pior que o método poliédrico e, por esse motivo, não houve necessidade de melhorar o desempenho deste último.

Capítulo 5

Método de planos-de-corte

Neste capítulo, descreveremos como podemos usar o método de planos-de-corte seguindo a implementação feita por Grötschel e Holland [GH85] para a resolução do problema do emparelhamento de peso máximo. A próxima seção descreve métodos de planos-de-corte em geral e definimos o problema da separação. Na seção 5.2, mostraremos o algoritmo de Padberg e Rao [PR82] utilizado para a resolução do problema da separação para o caso específico do problema do emparelhamento máximo e finalmente, na seção 5.3, descreveremos o método de planos-de-corte devido a Grötschel e Holland.

5.1 Métodos de planos-de-corte

O objetivo de um método de planos-de-corte é resolver um problema de otimização combinatória através de programação linear. A partir de um programa linear inicial, outros programas são obtidos através de planos-de-corte, que definiremos mais tarde, até que seja encontrada uma solução ótima inteira para o programa. A partir dessa solução, encontra-se uma solução para o problema original. Expandiremos essa idéia ao longo desta seção, que é baseada em [FW96].

Antes de entrarmos em detalhes sobre esses métodos, teremos que introduzir alguns conceitos de teoria dos poliedros.

Um vetor $x \in \mathbb{R}^n$ é dito uma *combinação convexa* de $x_1, \ldots, x_k$ se existem escalares $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$ tais que

$$0 \leq \alpha_i \leq 1, \quad \text{para } i = 1, \ldots, k,$$

$$\alpha_1 + \alpha_2 + \cdots + \alpha_k = 1$$

e

$$x = \sum_{i=1}^{k} \alpha_i x_i.$$

Denotaremos por $\operatorname{conv}\{x_1, \ldots, x_k\}$ o *fecho convexo* dos vetores $x_1, \ldots, x_k$, definido como

$$\operatorname{conv}\{x_1, \ldots, x_k\} = \{x \in \mathbb{R}^n : x \text{ é combinação convexa de } x_1, \ldots, x_k\}.$$

Dizemos que $P \subseteq \mathbb{R}^n$ é um *poliedro* se

$$P = \{x \in \mathbb{R}^n : Ax \leq b\},$$

para uma matriz $A \in \mathbb{R}^{m \times n}$ e um vetor $b \in \mathbb{R}^m$. Definimos o *fecho inteiro* de um poliedro P como

$$P_I = \operatorname{conv}\{x \in P : x \text{ é inteiro}\}.$$

Se $P = P_I$ então dizemos que P é um *poliedro inteiro*.

Sejam $a \in \mathbb{R}^n$ e $\alpha \in \mathbb{R}$ tais que para todo $x \in P$, $a^T x \leq \alpha$. Então dizemos que a inequação $a^T x \leq \alpha$ é *válida* para P. Um conjunto $F \subseteq P$ é uma *face* de P se existe uma inequação $a^T x \leq \alpha$ válida para P tal que

$$F = P \cap \left\{x : a^T x = \alpha\right\}.$$

Dizemos que F é a face *induzida* por $a^T x \leq \alpha$. Se $F \neq P$, então F é uma *face própria* de P. Se F não está contida em nenhuma outra face própria de P, então dizemos que F é uma *faceta*.

Um *Problema de Otimização Combinatória Linear* é uma tripla $(E, c, \mathcal{S})$, onde E é um conjunto finito, $c : E \to \mathbb{R}$ é uma função que atribui pesos aos elementos de E e $\mathcal{S}$ é um conjunto de subconjuntos de E. Dizemos que um elemento $S \in \mathcal{S}$ é uma *solução* do problema. Dada uma tripla, deseja-se encontrar $S \in \mathcal{S}$ tal que S maximiza (ou minimiza) $\sum_{e \in S} c(e)$.

Dada uma solução $S \in \mathcal{S}$, definimos o *vetor característico* de S como $\chi^S \in \mathbb{R}^E$ tal que

$$\chi_e^S = \begin{cases} 1, & \text{se } e \in S, \\ 0, & \text{caso contrário.} \end{cases}$$

Para um problema de otimização combinatória linear $\mathbf{POC} = (E, c, \mathcal{S})$ podemos definir o poliedro $P_{\mathbf{POC}}$ como

$$P_{\mathbf{POC}} = \operatorname{conv}\left\{\chi^S \in \mathbb{R}^E : S \in \mathcal{S}\right\}.$$

Note que esse é um poliedro inteiro em $\mathbb{R}^E$.

Como $P_{\mathbf{POC}}$ é um poliedro, sabemos que existem $A \in \mathbb{R}^{n \times m}$ e $b \in \mathbb{R}^m$, onde $m = |E|$, tais que

$$P_{\mathbf{POC}} = \left\{x \in \mathbb{R}^E : Ax \leq b \text{ e } x \text{ é inteiro}\right\}.$$

Portanto, o problema **POC** é equivalente ao programa linear

$$\max \sum_{e \in E} c(e) x_e, \tag{5.1}$$

sujeito a

$$Ax \leq b, \tag{5.2}$$

$$x \text{ inteiro}. \tag{5.3}$$

Removendo a restrição de integralidade, obtemos um programa linear que pode ser resolvido pelo método simplex ou outro algoritmo para resolução de programas lineares. Chamaremos esse problema de *relaxação linear* de **POC**. Seja P_0 o poliedro $\left\{x \in \mathbb{R}^E : Ax \leq b\right\}$. Note que $P_0 \supseteq P_{\mathbf{POC}}$, logo,

$$\max_{x \in P_{\mathbf{POC}}} \sum_{e \in E} c(e) x_e \leq \max_{x \in P_0} \sum_{e \in E} c(e) x_e.$$

Se o argumento que maximiza $\sum_{e \in E} c(e) x_e$ em P_0 é inteiro, então ele está contido em $P_{\mathbf{POC}}$.

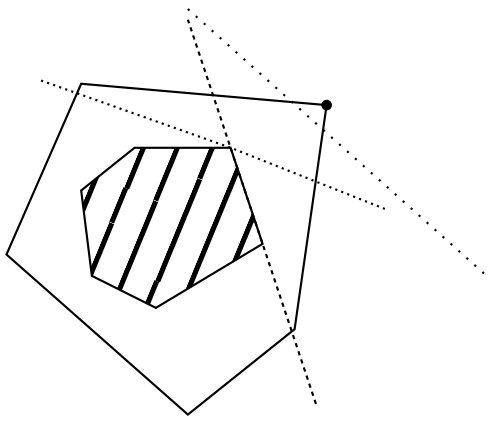


Figura 5.1: *Exemplo de cortes em um poliedro. A figura hachurada representa um poliedro inteiro contido em uma relaxação linear. Os linhas pontilhadas representam possíveis cortes para a solução do problema linear relaxado. A linha tracejada demonstra um corte que induz uma faceta.*

Portanto, se resolvermos a relaxação linear de **POC** e obtivermos uma solução inteira, sabemos que essa solução resolve o problema de otimização original. No entanto isso nem sempre acontece.

Se a solução x^* é fracionária, então ela pertence a P_0 mas não pertence $P_{\mathbf{POC}}$. Se pudermos encontrar uma inequação válida para $P_{\mathbf{POC}}$ mas que não é satisfeita por x^* então podemos obter um novo poliedro $P_1 = \left\{x \in \mathbb{R}^E : Ax \leq b, ax \leq \alpha\right\}$ tal que $P_{\mathbf{POC}} \subseteq P_1 \subset P_0$ e $x^* \notin P_1$. Dizemos que uma inequação válida que não é satisfeita por x^* é um *plano-de-corte*. O problema de, dado um vetor $x^* \in P_i$, encontrar uma inequação válida para $P_{\mathbf{POC}}$ tal que x^* não satisfaz essa inequação é chamado de *Problema da Separação*.

Se resolvermos o programa linear $\max_{x \in P_1} \sum_{e \in E} c(e)x_e$, obteremos uma nova solução x^* diferente da anterior e que pode ou não ser inteira. Se x^* é inteira, então x^* pertence a $P_{\mathbf{POC}}$ e portanto é uma solução do problema original. Caso contrário, se podemos encontrar um plano-de-corte, podemos obter um novo poliedro $P_2 \subset P_1$ e resolver um novo programa linear. Contanto que saibamos resolver o problema da separação, podemos repetir esse processo até que encontremos uma solução para o problema original. Gomory [Gom63] demonstrou que para uma sucessão de planos-de-corte gerados de forma sistemática, existe um número finito t tal que o poliedro P_t é inteiro (veja também [Chv73] e [Sch80]).

Diferentes planos-de-corte resultam em poliedros diferentes. Por isso, temos um particular interesse em cortes que induzem faces de P_I, chamados de *cortes faciais*. Nesse caso, sabemos que esse corte "encosta" em P_I, como mostra a figura 5.1. Dentre todos os cortes faciais, aqueles que induzem facetas são de particular interesse, já que esses tem uma intersecção maximal com P_I.

Podemos então resumir um método de planos-de-corte como segue.

1. Seja PL a relaxação linear de **POC**.

2. Resolva PL obtendo uma solução x^*.

3. Se x^* é inteira, pare. Caso contrário, encontre um plano-de-corte e o adicione a PL. Volte para o passo 2.

5.1.1 O poliedro dos emparelhamentos

Vamos agora restringir os conceitos apresentados acima para o problema do emparelhamento de peso máximo num grafo $G = (V, E)$ com uma função peso $w : E \to \mathbb{R}^+$. Note que esse é um problema de otimização combinatória linear $(E, w, \mathcal{M})$, onde

$$\mathcal{M} = \{M \subseteq E : M \text{ é um emparelhamento em } G\}.$$

Definimos então o *poliedro dos emparelhamentos* do grafo G como

$$P_{\text{Emp}}(G) := \text{conv}\left\{\chi^M \in \mathbb{R}^E : M \text{ é um emparelhamento em } G\right\}.$$

Considere as seguintes inequações:

$$x_e \geq 0 \quad \text{para toda aresta } e \in E. \tag{5.4}$$

$$\sum_{e \in \delta(v)} x_e \leq 1 \quad \text{para todo } v \in V \tag{5.5}$$

É fácil ver que o vetor característico de qualquer emparelhamento satisfaz essas restrições e que, por outro lado, qualquer vetor $x \in \mathbb{Z}^E$ que satisfaz 5.4 e 5.5 é o vetor característico de um emparelhamento. Portanto,

$$P_{\text{Emp}}(G) = \text{conv}\left\{x \in \mathbb{Z}^E : x \text{ satisfaz 5.4 e 5.5}\right\}.$$

Seja $P'(G)$ o poliedro $\left\{x \in \mathbb{R}^E : x \text{ satisfaz 5.4 e 5.5}\right\}$. Note que o fecho inteiro de $P'(G)$ é igual a $P_{\text{Emp}}(G)$. No entanto, $P'(G)$ não é um poliedro inteiro para todo grafo G. Tome $G = K_3$ e $\hat{x} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. Note que $\hat{x} \in P'(G)$ mas $\hat{x} \notin P_{\text{Emp}}(G)$.

A solução fracionária $\hat{x}$ viola a restrição de que para qualquer conjunto de vértices B com tamanho ímpar, o número máximo de arestas entre vértices de B que um emparelhamento pode ter é $\frac{|B|-1}{2}$. Podemos reescrever essa restrição como

$$\sum_{e \in E(B)} x_e \leq \frac{|B| - 1}{2} \quad \text{para todo } S \in O, \tag{5.6}$$

onde $O = \{B \subseteq V : |B| \geq 3 \text{ e ímpar}\}$. Logo, o poliedro $P(G) = \left\{x \in \mathbb{R}^E : x \text{ satisfaz 5.4, 5.5 e 5.6}\right\}$ não contém a solução fracionária $\hat{x}$. O poliedro $P(G)$ é igual $P_{\text{Emp}}(G)$, como demonstrou Edmonds [Edm65a]. Uma prova alternativa, baseada em Lovász [Lov79] pode ser encontrada em [FW96].

O problema de separação para o poliedro $P_{\text{Emp}}(G)$ é: dado um vetor $\hat{x} \in \mathbb{R}^E$ determine que $\hat{x} \in P_{\text{Emp}}(G)$ ou então encontre uma inequação válida para $P_{\text{Emp}}(G)$ que não é satisfeira por $\hat{x}$. Note que, no segundo caso, pelo menos uma das restrições 5.4, 5.5 e 5.6 será violada por $\hat{x}$.

Podemos testar por substituição se alguma restrição dos tipos 5.4 ou 5.5 é violada. Por outro lado, existe um número exponencial de restrições do tipo 5.6. Apesar disso, Padberg e Rao desenvolveram um algoritmo polinomial capaz de encontrar uma restrição do tipo 5.6 violada por um vetor $\hat{x} \notin P_{\text{Emp}}(G)$, como veremos na próxima seção.

5.2 Resolvendo o problema da separação para o poliedro dos emparelhamentos

Mostraremos agora um algoritmo que resolve o seguinte problema de separação: dado um grafo G e um vetor $\hat{x} \in \mathbb{R}^M$, deteminar se $\hat{x} \in P_{\text{Emp}}(G)$ ou então achar uma inequação de P_{Emp} violada por $\hat{x}$. Podemos verificar em tempo polinomial se alguma restrição do tipo 5.4 ou 5.5 é violada por $\hat{x}$. Devido ao número exponencial de restrições do tipo 5.6, testar todas as restrições não é uma opção se queremos um algoritmo de tempo polinomial.

Padberg e Rao [PR82] demonstraram que é possível resolver esse problema em tempo polinomial. O algoritmo que propuseram baseia-se na equivalência entre esse problema e o problema do corte ímpar mínimo que definiremos a seguir.

5.2.1 Poliedro dos emparelhamentos e cortes ímpares mínimos

Em um grafo $G = (V, E)$ cujos vértices são rotulados como terminais ou não-terminais, dizemos que um *corte ímpar* é uma partição dos vértices de G em dois conjuntos V_1 e V_2 tais que V_1 e V_2 possuam um número ímpar de vértices terminais. Dados pesos não negativos c_e nas arestas de G, o *Problema do Corte Ímpar Mínimo* é encontrar um corte ímpar (V_1, V_2) que minimiza a soma

$$c(V_1, V_2) := \sum_{e \in \delta(V_1)} c_e.$$

Para reduzir o problema de separação a um problema de corte ímpar mínimo, precisamos notar a equivalência entre 5.6 e a inequação 5.8 abaixo. Seja s_v a variável de folga da restrição 5.5 para o vértice v. Se somarmos essas restrições para algum conjunto $B \in O$, temos

$$\sum_{v \in B} \sum_{e \in \delta(v)} x_e + \sum_{v \in B} s_v = |B|$$

e portanto

$$2 \sum_{e \in E(B)} x_e + \sum_{e \in \delta(B)} x_e + \sum_{v \in B} s_v = |B|. \tag{5.7}$$

Substituindo 5.7 em 5.6, obtemos

$$\sum_{e \in E(B)} x_e \leq \frac{1}{2}\big(2 \sum_{e \in E(B)} x_e + \sum_{e \in \delta(B)} x_e + \sum_{v \in B} s_v - 1\big)$$

e portanto, para algum $B \in O$, 5.6 vale se e somente se

$$\sum_{e \in \delta(B)} x_e + \sum_{v \in B} s_v \geq 1. \tag{5.8}$$

Dado $\hat{x} \in \mathbb{R}^E$ definimos o grafo $G_{\hat{x}} = (V_{\hat{x}}, E_{\hat{x}})$ que contém todos os vértices de G e um vértice adicional S que representa as variáveis de folga, ou seja, $V_{\hat{x}} = V \cup \{S \notin V\}$. As arestas de $G_{\hat{x}}$ são definidas como segue

$$E_{\hat{x}} = \{e \in E, \hat{x}_e > 0\} \cup \{vS : v \in V, s_v > 0\}.$$

Rotulamos os vértices $v \in V$ como terminais. Se $|V|$ é ímpar, então S também é rotulado terminal.

Caso contrário, o rótulo de S é não-terminal.

Seja (V_1, V_2) um corte ímpar mínimo em $G_{\hat{x}}$. Suponha, s.p.g., que $S \in V_2$. Se o custo desse corte é menor que 1, então V_1 viola 5.6 pois

$$c(V_1, V_2) = \sum_{e \in \delta_{G_{\hat{x}}}(V_1)} x_e = \sum_{e \in \delta_G(V_1)} x_e + \sum_{v \in V_1} s_i < 1.$$

Vejamos agora como encontrar um corte ímpar mínimo num grafo.

5.2.2 Resolvendo o problema do corte ímpar mínimo

Nesta subseção, iremos provar o seguinte teorema.

Teorema 8 ([PR82])**.** *Seja G um grafo cujos vértices foram rotulados terminais e não-terminais de tal forma que o número total de vértices terminais é par. Seja G_T a árvore devolvida pela execução do algoritmo de Gomory e Hu para G. Sejam $e_1, \ldots, e_k \in E(G_T)$ arestas cuja remoção divide G_T em duas subárvores com número ímpar de vértices e sejam $f_1, \ldots, f_k$ os pesos dessas arestas em G_T. Então $f_* = \min\{f_1, \ldots, f_k\}$ define um corte ímpar mínimo em G.*

O teorema acima nos diz que podemos resolver o problema do corte ímpar mínimo de um grafo através do algoritmo de Gomory e Hu, descrito no capítulo 4. Depois de construída a árvore que nos diz o valor de um fluxo entre quaisquer dois vértices terminais, encontrar um corte ímpar envolve apenas percorrer a árvore a procura de arestas que a dividem em duas subárvores mínimas. Uma tal aresta de peso mínimo define um corte ímpar mínimo.

Devido ao cálculo de fluxos em redes contraídas, teremos de fazer uma prova muito similar à dos lemas do capítulo anterior. Utilizando esse resultado, provaremos o teorema. O lema 6 abaixo, nos diz que podemos contrair o conjunto de vértices de um dos lados de um corte mínimo $(X, \overline{X})$ de forma que se existe um corte ímpar mínimo que separa um certo conjuntos de vértices terminais no grafo original, então existe um outro corte ímpar mínimo no grafo contraído que separa o mesmo conjunto de vértices terminais e tem a mesma capacidade.

Lema 6. *Dado um grafo $G = (V, E)$ com pesos c_e em suas arestas tal que seu conjunto de vértices é uma partição $V = V_0 \cup V_1$, onde os vértices de V_0 são não terminais e os vértices de V_1 são terminais. Seja $(X, \overline{X})$ um corte mínimo em relação a todos os pares de vértices de V_1. Então existe um corte ímpar mínimo $(Z, \overline{Z})$ tal que $(X, \overline{X})$ e $(Z, \overline{Z})$ não se cruzam.*

Demonstração. Suponha que $|X \cap V_1|$ é par, pois, caso contrário, não há nada a provar. Seja $(Y, \overline{Y})$ um corte ímpar mínimo. Suponha que $(Y, \overline{Y})$ cruza com $(X, \overline{X})$. Sejam,

$$X \cap Y = Q, \quad X \cap \overline{Y} = S,$$
$$\overline{X} \cap Y = P, \quad \overline{X} \cap \overline{Y} = R.$$

Como X tem um número par e Y um número ímpar de vértices terminais, extamente um dos conjuntos Q e R tem um número ímpar de vértices terminais. Suponha, s.p.g., que esse conjunto é Q. Como $|Q \cap V_1| + |S \cap V_1| = |X \cap V_1|$, $|Q \cap V_1|$ é ímpar e $|X \cap V_1|$ é par, então $|S \cap V_1|$ é ímpar. Como $\overline{X}$ tem pelo menos um vértice terminal, então pelo menos um dentre os conjuntos P e R tem um vértice terminal.

Caso 1. R tem um vértice terminal. Então, $(R, P \cup Q \cup S)$ é um corte ímpar. Como $(X, \overline{X})$ é mínimo, temos que

$$c(Q,P) + c(Q,R) + c(S,P) + c(S,R) \leq c(P,R) + c(Q,R) + c(S,R).$$

Como $c(S,P) = c(P,S) \geq 0$, temos

$$c(Q,P) \leq c(P,R) + c(P,S).$$

Somando $c(Q,S) + c(Q,R)$ aos dois lados da inequação acima, obtemos

$$c(Q,P) + c(Q,S) + c(Q,R) \leq c(P,R) + c(P,S) + c(Q,S) + c(Q,R),$$

ou seja,

$$c(Q, P \cup S \cup R) \leq c(Y, \overline{Y}).$$

Como Q tem um número ímpar de vértices terminais e $(Y, \overline{Y})$ é um corte ímpar mínimo, temos que o corte $(Z, \overline{Z}) = (Q, P \cup S \cup R)$ é um corte ímpar mínimo tal que $(Z, \overline{Z})$ não cruza com $(X, \overline{X})$.

Caso 2. P tem um vértice terminal. Trocando Q e S no caso acima, podemos mostrar que $(S, P \cup Q \cup R)$ é um corte ímpar mínimo que não cruza $(X, \overline{X})$. □

O lema acima garante que o uso de contração feito pelo algoritmo de Gomory e Hu também é válido quando estamos interessado em cortes ímpares mínimos. Se o corte mínimo $(X, \overline{X})$ entre todos os pares de terminais de G é tal que X tem um número ímpar de terminais, então $(X, \overline{X})$ é um corte ímpar mínimo. Se esse não é o caso, podemos dividir G em dois novos grafos G^1 e G^2 de forma que G^1 é o grafo obtido pela contração de $\overline{X}$ e G^2 o grafo obtido pela contração de X. Como o número de terminais em X e $\overline{X}$ é par, os pseudo-vértices são rotulados como não terminais, de forma que o número total de terminais permanece par.

Como o lema 6 se aplica a G^1 e G^2, podemos executar o algorimo de Gomory e Hu para esses grafos e encontrar o corte mínimo que separa quaisquer dois de seus vértices terminais. Se algum desses cortes é ímpar, então o de menor valor é um corte ímpar mínimo. Se ambos são cortes pares, então podemos dividir G^1 e G^2 como fizemos para G e continuar o processo até que um corte ímpar mínimo seja encontrado.

Pelos resultados do capítulo 4, sabemos que o valor do fluxo máximo entre quaisquer dois vértices de G^i é igual ao valor do fluxo máximo entre os vértices correspondentes de G, para $i = 1, 2$. Portanto, as árvores resultantes da execução do algoritmo de Gomory e Hu para G são precisamente as subárvores que obteríamos se removêssemos a aresta referente ao corte $(X, \overline{X})$ da árvore resultado da execução para G.

Portanto, podemos encontrar um corte mínimo entre quaisquer dois vértices de G^1 através da aresta de menor peso da subárvore de G_T correspondente a G^1 e similarmente para G^2. Então, podemos encontrar um corte ímpar mínimo dividindo a árvore G_T sucessivamente até que encontremos uma aresta que divide G_T em duas subárvores de cardinalidade ímpar. Cada aresta desses representa um corte mínimo num dos grafos obtidos pela divisão através de um corte mínimo. Portanto, entre as arestas que dividem G_T em duas subárvores de cardinalidade ímpar, aquelas de peso mínimo descrevem um corte ímpar mínimo.

Com isso provamos o teorema 8 acima. Iremos agora descrever um método de planos-de-corte que utiliza esse algoritmo de separação para resolver o problema do emparelhamento de peso máximo.

5.3 Método de Grötschel e Holland

O método de planos-de-cortes proposto por Grötschel e Holland resolve o problema do emparelhamento perfeito de peso mínimo, que é equivalente ao problema do emparelhamento de peso máximo. A equivalência fica clara se, dado um grafo $G = (V, E)$ com pesos $w(e)$ em suas arestas, construírmos um grafo $G' = (V', E')$ como segue: se $|V|$ é ímpar, $V' = V \cup \{a\}$, caso contrário $V' = V$. G' é um grafo completo, ou seja, $E' = \{uv : u, v \in V'\}$. Definimos o peso $w'(e)$ das arestas de G' como

$$w'(e) = \begin{cases} -w(e), & \text{se } e \in E, \\ 0, & \text{caso contrário.} \end{cases}$$

Claramente, se M' é um emparelhamento perfeito de peso mínimo em G', $M = \{e \in M' : w'(e) < 0\}$ é um emparelhamento de peso máximo em G. Por outro lado, se M é um emparelhamento de peso máximo em G podemos obter um emparelhamento perfeito de peso mínimo em G' completando M com arestas de peso zero em G'.

O poliedro considerado no método que descreveremos contém apenas emparelhamentos perfeitos. Sendo assim, substituímos a restrição 5.5 por

$$\sum_{e \in \delta(v)} x_e = 1 \quad \text{para todo } v \in V. \tag{5.9}$$

Note que essa restrição também pode ser separada por substituição.

O algoritmo proposto por Grötschel e Holland, consiste na resolução de vários programas lineares em subgrafos de G, sendo o primeiro deles o seguinte:

$$z := \min \sum_{e \in E'} w(e) x_e$$

sujeito às restrições 5.4 e 5.9, onde $E' \subseteq E$. A escolha de E' influencia fortemente o desempenho do algoritmo. Seja E_v^k o conjunto das k arestas de menor peso que incidem em v. Então, uma boa esolha de E' é

$$E' = \bigcup_{v \in V} E_v^k,$$

para $5 \leq k \leq 10$[GH85].

Planos-de-cortes e variáveis para arestas de $E \setminus E'$ são adicionadas ao programa linear anterior que é então reotimizado até que se encontre um emparelhamento perfeito de peso mínimo. Essas adições são descritas a seguir. Seja $x^\star$ a solução obtida.

Se $x^\star$ é inteira, ela representa um emparelhamento perfeito de peso mínimo no grafo $G' = (V, E')$. Seja E^+ o conjunto das arestas de $E \setminus E'$ que potencialmente podem reduzir o valor da função objetivo (as arestas cujas variáveis têm custo reduzido negativo, veja [BT97, pp. 84]).

Se $E^+ \neq \emptyset$, adicionamos ao programa linear as variáveis x_e para $e \in E^+$.

Se $E^+ = \emptyset$, então a solução obtida é um emparelhamento perfeito de peso mínimo em G e o algoritmo termina.

Se $x^\star$ é fracionária, então ela não representa um emparelhamento em G. Nesse caso, encontra-se uma restrição violada do tipo 5.6 que é adicionada ao programa.

Como o procedimento de Padberg e Rao tem complexidade de tempo $O(n^4)$, o que o torna muito lento para instâncias grandes, Grötschel e Holland [GH85] propuseram duas heurísticas que são muito eficientes na prática para encontrar restrições do tipo 5.6 violadas. São elas:

Heurística 1: Seja $G_{x^\star} = (V, E_{x^\star})$ onde $E_{x^\star} = \{e \in E' : x_e^\star > 0\}$. Usando busca em profundidade, verifica-se se $G_{x^\star}$ possui componentes ímpares. Para cada componente (V_i, E_i) encontrada, adicionamos ao programa linear a restrição

$$\sum_{e \in E(V_i)} x_e \leq \frac{1}{2}\left(|V_i| - 1\right).$$

A procura por planos-de-corte continua apenas se nenhuma restrição foi adicionada.

Heuristíca 2: Seja $G^2_{x^\star}$ um subgrafo de $G_{x^\star}$ com conjunto de arestas $E^2_{x^\star} = \{e \in E_{x^\star} : x_e^\star \geq 0.3\}$. Procuramos por componentes ímpares usando busca em profundidade e para cada componente (V_i, E_i) encontrada cuja restrição correspondente

$$\sum_{e \in E(V_i)} x_e \leq \frac{1}{2}\left(|V_i| - 1\right)$$

é violada por $x^\star$ adicionamos o plano de corte correspondente ao programa linear.

Apenas se as duas heurísticas falham o procedimento de Padberg e Rao é utilizado. Os planos-de-corte encontrados são adicionados ao programa linear.

Os passos descritos acima são repetidos até que um emparelhamento perfeito de peso mínimo seja encontrado.

Parte III

Testes computacionais

Capítulo 6

Resultados computacionais

Testamos nossa implementação dos dois algoritmos com grafos aleatórios e alguns grafos geométricos disponibilizados pelo DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) [DIM].

A tabela abaixo mostra o tempo de execução dos algoritmos para os problemas geométricos do DIMACS. Esses testes foram realizados em uma máquina com processador Intel Core 2 Quad Q6600 de 2.4GHz e com 4GB de memória RAM. O problema com 20726 vértices foi abortado por falta de memória.

Instância	Algoritmo de Edmonds Tempo(s)	Método de planos-de-cortes Tempo(s)
r422.geom	55.591	6.271
r1002.geom	2270.029	94.683
r2392.geom	90309.627	1521.680
r20726.geom	Abortado	Abortado

Tabela 6.1: *Resultados computacionais para os grafos geométricos do DIMACS.*

Os testes com grafos aleatórios foram a parte principal dos testes computacionais, dada a facilidade de geração de instâncias. Utilizamos grafos com uma variedade de número de vértices e arestas. Devido ao interesse em verificar a eficiência da técnica de subgrafos esparsos utilizada pelo método de planos-de-cortes, fizemos testes com grafos aleatórios com densidades diferentes. Foram gerados grafos aleatórios com 50, 100, 200, 400, 800, 1600 e 3200 vértices, usando probabilidades de aresta de 0,1, 0,3, 0,8 e 1,0. Para cada configuração de número de vértices e probabilidade de aresta foram geradas 10 instâncias.

A tabela 6.2 resume o desempenho dos algoritmos para essas instâncias. Cada entrada nessa tabela corresponde a

$$\frac{1}{10}\sum_{i=1}^{10}\frac{t_i^{\text{Edmonds}}}{t_i^{\text{Planos-de-corte}}},$$

onde t_i^{Edmonds} é o tempo gasto pelo algoritmo de Edmonds para resolver a instância i e $t_i^{\text{Planos-de-corte}}$ é o tempo gasto pelo algoritmo de planos-de-corte gasto para resolver a instância i. Das instâncias com mais de 100 vértices, apenas em um caso a razão $\frac{t_i^{\text{Edmonds}}}{t_i^{\text{Planos-de-corte}}}$ foi menor ou igual 1. Ou seja, em todos menos um caso, o método de planos-de-corte foi mais rápido que nossa implementação do algoritmo de Edmonds. No entanto, a tabela 6.2 demonstra uma tendência de diminuição da razão entre o tempo de execução dos dois algoritmos à medida que o número de vértices aumenta.

Número de Vértices	Densidade			
	0,1	0,3	0,8	1,0
100	0,92	1,28	1,78	2,15
200	1,34	1,64	4,10	5,07
400	2,58	3,72	8,39	10,24
800	4,46	4,47	4,93	6,00
1600	3,54	3,24	3,96	4,59
3200	2,25	2,41	3,30	4,07

Tabela 6.2: *Média das razões entre o tempo de execução do algoritmo de Edmonds e do método de planos-de-corte para grafos aleatórios de diferentes tamanhos e densidades.*

Esses testes foram realizados numa máquina equipada com processadores Intel Xeon E5440 de 2.83GHz e 32GB de memória RAM. A tabela 6.3 mostra os tempos médio e máximo de execução dos algoritmos para as várias configurações de número de vértices e probabilidade de arestas.

Testamos também os algoritmos com grafos aleatórios em que todas as arestas têm peso 1, ou seja, instâncias do problema do emparelhamento de cardinalidade máxima. Nesse caso, o desempenho do algoritmo de Edmonds foi muito superior àquele do método de planos-de-corte. Para várias instâncias uma quantidade grande de cortes foi necessária tornando o tempo gasto para resolver os programas lineares muito grande. Em testes preliminares algumas instâncias não tinham terminado de executar após vários dias.

Para a execução sistemática dos testes, limitamos o tempo de execução do método de planos-de-corte em 60 vezes o tempo de execução do algoritmo de Edmonds. A tabela 6.4 mostra a quantidade de instâncias cuja execução do métodos de planos-de-corte foi interrompida devido ao limite de tempo imposto.

A tabela 6.5 mostra os tempos de execução médio e máximo para as instâncias do tipo cardinalidade.

Vértices	Densidade	Alg. de Edmonds		Alg. de planos-de-corte	
		Médio	Máximo	Médio	Máximo
50	0,1	0,005	0,010	0,004	0,010
	0,3	0,004	0,010	0,004	0,010
	0,8	0,011	0,020	0,010	0,020
	1,0	0,012	0,020	0,005	0,010
100	0,1	0,022	0,030	0,026	0,040
	0,3	0,042	0,060	0,034	0,040
	0,8	0,084	0,110	0,048	0,060
	1,0	0,112	0,140	0,053	0,070
200	0,1	0,172	0,190	0,130	0,160
	0,3	0,384	0,470	0,243	0,300
	0,8	1,258	1,640	0,309	0,380
	1,0	1,724	2,250	0,342	0,450
400	0,1	3,621	4,390	1,533	2,680
	0,3	8,115	14,020	2,262	3,860
	0,8	33,300	47,830	3,857	5,150
	1,0	42,500	66,090	4,263	5,760
800	0,1	26,724	31,560	6,005	6,760
	0,3	76,119	86,100	18,265	34,820
	0,8	162,787	197,740	33,334	38,600
	1,0	217,815	260,250	37,058	53,500
1600	0,1	221,769	263,860	70,783	167,150
	0,3	569,636	617,740	193,901	338,770
	0,8	1299,516	1374,530	330,992	381,260
	1,0	1703,396	2020,650	373,170	421,290
3200	0,1	1436,767	1624,780	860,716	1697,720
	0,3	4135,154	5121,940	2017,112	3331,580
	0,8	8490,872	10078,400	2571,752	2956,480
	1,0	9911,047	14929,190	2488,556	4094,050

Tabela 6.3: *Tempo de execução médio e máximo para os grafos aleatórios com pesos aleatórios.*

Número de Vértices	Densidade			
	0,1	0,3	0,8	1,0
50	0	0	0	0
100	0	0	0	0
200	1	3	0	0
400	5	6	0	0
800	7	8	2	0
1600	10	10	3	0
3200	10	10	9	0

Tabela 6.4: *Número de instâncias to tipo cardinalidade que tiveram a execução do método de planos-de-corte interrompida pois seu tempo de execução foi superior a 60 vezes o tempo de execução do algoritmo de Edmonds.*

Vértices	Densidade	Alg. de Edmonds		Alg. de planos-de-corte	
		Médio	Máximo	Médio	Máximo
50	0,1	0,000	0,000	0,006	0,010
	0,3	0,001	0,010	0,005	0,010
	0,8	0,001	0,010	0,007	0,020
	1,0	0,002	0,010	0,005	0,010
100	0,1	0,001	0,010	0,030	0,050
	0,3	0,001	0,010	0,048	0,130
	0,8	0,003	0,020	0,037	0,050
	1,0	0,009	0,020	0,007	0,020
200	0,1	0,006	0,020	0,268	0,570
	0,3	0,010	0,020	0,589	1,040
	0,8	0,029	0,040	0,305	0,770
	1,0	0,035	0,040	0,059	0,090
400	0,1	0,020	0,030	1,712	3,640
	0,3	0,050	0,070	1,583	2,960
	0,8	0,140	0,150	2,628	7,200
	1,0	0,164	0,180	0,279	0,300
800	0,1	0,103	0,130	6,463	8,560
	0,3	0,238	0,280	9,035	10,830
	0,8	0,623	0,750	13,121	28,410
	1,0	0,754	0,830	1,208	1,340
1600	0,1	0,423	0,460	-	-
	0,3	0,694	0,790	-	-
	0,8	2,970	3,690	118,153	176,570
	1,0	3,941	4,340	4,964	5,400
3200	0,1	1,390	1,490	-	-
	0,3	4,201	4,990	-	-
	0,8	16,318	21,880	530,360	530,360
	1,0	21,592	24,650	18,080	20,380

Tabela 6.5: *Tempos médio e máximo de execução dos algoritmos para instâncias do tipo cardinalidade. Para as médias de tempo do método de planos-de-corte foram considerados apenas os tempos de execução das instâncias cuja execução não foi interrompida.*

Capítulo 7

Conclusão

Como vimos no capítulo 5, métodos de planos-de-corte são uma ferramenta para resolver problemas de otimização combinatória linear, aplicável para um conjunto bastante amplo de problemas. O algoritmo de Edmonds, no entanto, é uma solução combinatória específica para o problema do emparelhamento de peso máximo. Por isso, esperávamos que esse algoritmo fosse mais eficiente do que o método de Grötschel e Holland.

O que vimos no entanto, foi um desempenho significativamente melhor do método de planos-de-corte. Isso confirma o resultado de Grötschel e Holland [GH85], mostrando que seu método de planos-de-corte para esse problema é eficiente na prática.

O algoritmo de Edmonds se mostrou eficiente apenas para instâncias do problema do emparelhamento de cardinalidade máxima. Caso em que o desempenho foi muito superior ao método de planos-de-corte. No entanto, uma grande simplificação desse algoritmo pode ser usada nesse caso. Na realidade, Edmonds publicou inicialmente um algoritmo que tratava apenas do caso específico do problema de cardinalidade máxima. Posteriormente, ele publicou a generalização desse algoritmo que apresentamos nesse trabalho para o problema de peso máximo. É provável que, apesar da preocupação em resolver tal problema, Edmonds não tenha se preocupado muito com o desempenho do algoritmo.

Os resultados que apresentamos indicam, no entanto, que para instâncias maiores do problema o algoritmo combinatório teria um melhor desempenho do que o método poliédrico. Não pudemos verificar esse resultado devido ao elevado tempo de execução de ambos os algoritmos para tais instâncias. Ainda assim, nossa conclusão é que o uso de métodos de planos-de-corte podem ser uma alternativa viável para a resolução de instâncias grandes do problema do emparelhamento de peso máximo, uma vez que o algoritmo de Edmonds é de difícil implementação e seu desempenho, em muitos casos, não justifica o trabalho adicional.

Referências Bibliográficas

[BM76] J. A. Bondy e U. S. R. Murty. *Graph Theory with Applications.* Macmillan, London, 1976. 2, 3, 8

[BT97] D. Bertsimas e J. Tsitsiklis. *Introduction to linear optimization.* Athena Scientific, Belmont, Massachusetts, 1997. 58

[Chv73] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973. 53

[DIM] DIMACS. ftp://dimacs.rutgers.edu/pub/netflow/instances/matching/. Verificado em 17/11/2010. 63

[Edm65a] J. Edmonds. Maximum matching and a polyhedron with $0, 1$ vertices. *J. of Res. the Nat. Bureau of Standards*, 69 B:125–130, 1965. iii, v, 7, 14, 54

[Edm65b] J. Edmonds. Paths, trees and flowers. *Can. J. Math.*, 17:449–467, 1965. 11

[FF62] L.R Ford, Jr. e D.R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962. 40

[FW96] C. E. Ferreira e Y. Wakabayashi. *Combinatória Poliédrica e Planos-de-Corte Faciais.* X Escola de Computação, 1996. 51, 54

[GH61] R. E. Gomory e T. C. Hu. Multi-terminal network flows. *J. SIAM*, 9(4):551–570, 1961. 41, 46

[GH85] M. Grötschel e O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33-3:243–259, 1985. iii, v, 1, 51, 58, 59, 67

[Gib85] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, Cambridge, 1985. 11

[Gom63] R. E. Gomory. An algorithm for integer solutions to linear programs. Em R. Graves andP. Wolfe, editor, *Recent Advances in mathematical programming.* McGraw-Hill, 1963. 53

[GT86] A V Goldberg e R E Tarjan. A new approach to the maximum flow problem. Em *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, páginas 136–146, New York, NY, USA, 1986. ACM. 50

[Hu69] T. C. Hu. *Integer Programming and Network Flows.* Addinson-Wesley Publishing Company, Reading, Massachussets, 1969. ix, x, 41, 42, 48

[Kuh55] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. 7

[Kuh91] H. W. Kuhn. On the origin of the Hungarian Method. *History of Mathematical Programming – A Collection of Personal Reminescences (J.K. Lenstra, A.H.G. Rinnooy Kan, A. Schrijver, eds) CWI Amsterdam and North-Holland*, páginas 77–81, 1991. 7

[Lov79] L. Lovász. Graph theory and integer programming. *Annals of Discrete Mathematics*, 4:141–158, 1979. 54

[LP86] L. Lovász e M. D. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics.* North-Holland, Amsterdam, 1986. 1, 21

[NW88] G. L. Nemhauser e L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, 1988. 11

[PR82] M. Padberg e M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7-1:67–80, 1982. iii, v, 51, 55, 56

[Sch80] A. Schrijver. On cutting planes. *Annals of Discrete Mathematics*, 9:291–296, 1980. 53