Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação

Uma abordagem baseada em emparelhamento em grafos para o problema da conversão de triangulações de superfícies em quadrilaterizações

Lucas Araújo de Lima

Natal-RN

Março de 2011

Lucas Araújo de Lima

Uma abordagem baseada em emparelhamento em grafos para o problema da conversão de triangulações de superfícies em quadrilaterizações

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador

Prof. Dr. Marcelo Ferreira Siqueira

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

Março de 2011

Monografia de Graduação sob o título *Uma abordagem baseada em emparelhamento em grafos para o problema da conversão de triangulações de superfícies em quadrilaterizações* apresentada por Lucas Araújo de Lima e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Prof. Dr. Marcelo Ferreira Siqueira
Orientador
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Profª. Dra. Elizabeth Ferreira Gouvêa Goldbarg
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Prof. Dr. Bruno Motta de Carvalho
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte

Natal-RN, 11 de março de 2011.

Agradecimentos

Meus agradecimentos especiais destinam-se: ao orientador Marcelo Ferreira Siqueira, que ofereceu suporte e dedicação excepcionais a este trabalho; ao colega de pesquisa Ícaro Lins Leitão da Cunha, que forneceu ideias e ajuda na implementação dos algoritmos; e aos meus pais, pelo apoio sempre constante.

Uma abordagem baseada em emparelhamento em grafos para o problema da conversão de triangulações de superfícies em quadrilaterizações

Autor: Lucas Araújo de Lima
Orientador: Prof. Dr. Marcelo Ferreira Siqueira

Resumo

Esta monografia apresenta um trabalho que consiste no estudo e implementação de um algoritmo para conversão de triangulações de superfícies em quadrilaterizações. O algoritmo proposto é uma modificação de um algoritmo guloso para o problema. A modificação proposta tem como motivação a eliminação de uma desvantagem prática do algoritmo guloso. Para isto, transformamos o problema da conversão em um problema de encontrar emparelhamentos perfeitos de custo máximo em grafos gerais. Com isso, podemos nos aproveitar de uma rica fundamentação teórica e de uma gama de algoritmos bem estabelecidos para encontrar emparelhamento em grafos, os quais resolvem o problema da conversão sem a desvantagem inerente ao algoritmo guloso.

Palavras-chave: triangulações, quadrilaterizações, grafos, grafo dual, emparelhamento.

Lista de figuras

Lista de tabelas

Lista de algoritmos

Sumário

1 Introdução

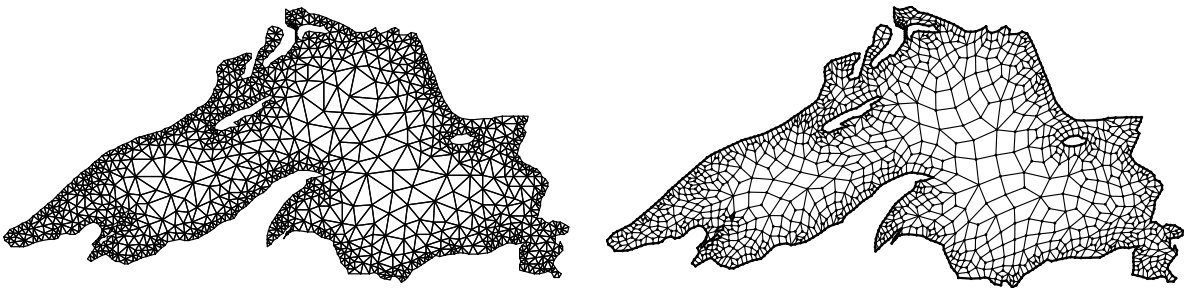
Este capítulo introduz o trabalho de conclusão de curso de graduação descrito nesta monografia. A Seção 1.1 contextualiza o trabalho. A Seção 1.2 apresenta os seus objetivos e contribuições, ressaltando a importância do trabalho desenvolvido para a formação intelectual do autor. Finalmente, a Seção 1.3 descreve como o restante desta monografia está organizada.

1.1 Contextualização

Triangulações e quadrilaterizações são casos particulares e importantes de subdivisão de um "objeto" geométrico em regiões mais simples, conexas, disjuntas e com forma triangular e quadrangular, respectivamente. A Figura 1.1 ilustra uma triangulação e uma quadrilaterização de uma mesma região planar. A Figura 1.2 ilustra uma triangulação e uma quadrilaterização de uma mesma superfície no espaço Euclidiano. Subdivisões de um objeto, tais como triangulações e quadrilaterizações, nos permitem obter, *mais facilmente*, informações locais sobre a topologia e geometria do objeto. De certa forma, a motivação para subdividir um objeto geométrico em partes mais simples é muito semelhante à do paradigma de divisão-e-conquista, que é bastante utilizado na solução de problemas computacionais.

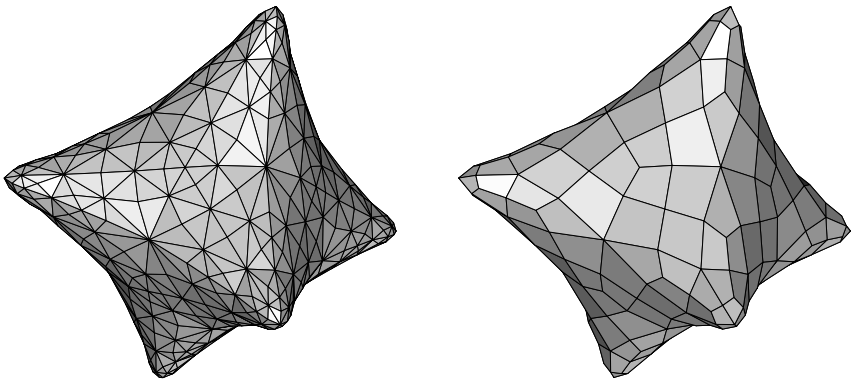
O uso de triangulações e quadrilaterizações em Matemática vai de topologia algébrica a análise numérica (CARVALHO; VELHO; GOMES, 1992). Em particular, um dos métodos numéricos mais populares para resolução de equações diferenciais parciais em Engenharia e Ciências Aplicadas, o Método dos Elementos Finitos (MEF), requer como entrada uma subdivisão do domínio do problema (JOHNSON, 2009), que em geral é uma triangulação ou quadrilaterização do domínio. Finalmente, triangulações e quadrilaterizações são utilizadas com muita frequência em computação gráfica (WATT, 1999), modelagem geométrica (SCHUMAKER, 1993), geometria computacional (GUIBAS; STOLFI, 1985) e outras áreas relacionadas, tais como processamento digital de imagens (GEVERS; SMEULDERS,

1997).



(a) (b)

Figura 1.1: (a) Uma triangulação e (b) uma quadrilaterização de região planar.



(a) (b)

Figura 1.2: (a) Uma triangulação e (b) uma quadrilaterização de uma superfície em $\mathbb{R}^3$.

Em geral, triangulações de objetos geométricos bidimensionais são mais facilmente geradas do que quadrilaterizações. Isto motivou o surgimento de inúmeros algoritmos para triangulações de regiões do plano euclidiano, tais como polígonos (CHAZELLE, 1991; BERN; EPPSTEIN, 1992; PAV; WALKINGTON, 2005) e superfícies no espaço euclidiano (VLASSOPOULOS, 1990; NING; BLOOMENTHAL, 1993; VELHO; FIGUEIREDO; GOMES, 1999; BOISSONNAT; OUDOT, 2005; CHENG; DEY; RAMOS; RAY, 2007; MEYER; KIRBY; WHITAKER, 2007). Muitos desses algoritmos oferecem garantias teóricas para alguns atributos da triangulação gerada por eles, tais como um limite inferior para o menor ângulo de um triângulo da triangulação. Em muitas aplicações, a garantia de limites "justos" para a faixa de valores dos atributos das triangulações é fundamental para a obtenção de soluções robustas através de métodos baseados em subdivisões do domínio do problema, tais como o MEF (BERN; PLASSMANN, 2000).

Apesar da existência de "bons" algoritmos para a geração de triangulações de objetos

geométricos bidimensionais, quadrilaterizações são mais apropriadas do que triangulações em algumas aplicações importantes. Este é o caso, por exemplo, de interpolação e certas simulações numéricas baseadas no MEF (ALLMAN, 1988; LAI, 1996; MALANTHARA; GERSTLE, 1997).

Embora existam vários algoritmos para calcular quadrilaterizações de regiões planares (BLACKER, 1991; JOE, 1995; BERN; EPPSTEIN, 1997; OWEN; STATEN; CANNAN; SAIGAL, 1999; VISWANATH; SHIMADA; ITOH, 2000; RAMASWAMI; SIQUEIRA; SUNDARAM; GALLIER; GEE, 2005; ATALAY; RAMASWAMI; XU, 2008), eles não oferecem as mesmas garantias teóricas oferecidas pelos algoritmos para triangulações. Isto se deve ao fato das propriedades matemáticas para se gerar quadrilaterizações, com garantias teóricas para certos atributos, ainda não serem conhecidas. As dificuldades são ainda maiores quando se trata da geração de quadrilaterizações de superfícies definidas implicitamente, isto é, para as quais não temos parametrizações explícitas (ZHANG; BAJAJ, 2006).

Mais recentemente, a comunidade de pesquisa em computação gráfica e modelagem geométrica tem voltado sua atenção para o desenvolvimento de algoritmos para gerar quadrilaterizações de superfícies a partir de triangulações das mesmas (RAMASWAMI; RAMOS; TOUSSAINT, 1998; VELHO, 2000; MARINOV; KOBBELT, 2004; DONG; KIRCHER; GARLAND, 2005; RAY; LI; LéVY; SHEFFER; ALLIEZ, 2006; TONG; ALLIEZ; COHEN-STEINER; DESBRUN, 2006; KÄLBERER; NIESER; POLTHIER, 2007; LAI; KOBBELT; HU, 2008; BOMMES; ZIMMER; KOBBELT, 2009; TARINI; PIETRONI; CIGNONI; PANOZZO; PUPPO, 2010). Isto é, ao invés de usarem a superfície a partir da qual a triangulação foi obtida (que, em geral, é desconhecida), esses algoritmos utilizam a própria triangulação. A motivação por trás desses algoritmos é que a existência de uma triangulação torna mais simples o processo de geração de quadrilaterizações de superfícies com "boas" propriedades. Além disso, a maioria dos algoritmos existentes na literatura se baseia em técnicas oriundas de geometria diferencial discreta, as quais lhes permitem executar operações discretas simples e eficientes (em contrapartida às correspondentes operações baseadas em matemática contínua).

No contexto das aplicações dos algoritmos de geração de quadrilaterizações mencionados acima, os atributos que mais importam são: *regularidade*, *forma*, *alinhamento* e *adaptatividade*. Uma quadrilaterização deveria ser tão *regular* quanto possível, isto é, todos ou quase todos os seus vértices deveriam possuir exatamente quatro arestas incidentes sobre eles. A *forma* dos quadriláteros também deveria ser tão "ortogonal" quanto possível. As arestas deveriam estar *alinhadas* em relação às direções das (duas) curvaturas principais

da superfície e com certas características "afiadas" (do inglês *sharp features*), tais como "quinas" e "dobras". Finalmente, o tamanho dos quadriláteros deveria se *adaptar* localmente à curvatura da superfície; isto é, deveria ser menor em regiões da superfície com maior curvatura e deveria ser maior em regiões de menor curvatura (LAI; KOBBELT; HU, 2008).

Nenhum dos algoritmos existentes oferece garantias teóricas (por exemplo, limites superiores e inferiores) para as métricas usadas para avaliar os atributos acima. No entanto, vários deles otimizam, através de heurísticas bem fundamentadas matematicamente, todos ou alguns desses atributos. Infelizmente, alguns atributos, tais como regularidade e adaptatividade, são quase antagônicos e nenhum dos algoritmos existentes possui um conjunto de heurísticas que seja capaz de otimizar todos os atributos ao mesmo tempo. Alguns dos algoritmos que otimizam vários atributos não são capazes sequer de gerar uma subdivisão contendo apenas quadriláteros; isto é, o resultado é um misto de quadriláteros e triângulos.

1.2 Objetivos e contribuições

Este trabalho teve como objetivos: (1) a utilização de uma abordagem baseada em grafos para converter triangulações de *superfícies* em quadrilaterizações e (2) a comparação experimental desta com uma abordagem gulosa proposta por Luiz Velho (VELHO, 2000). A escolha desta abordagem, para o propósito de comparação, deve-se ao fato dessa abordagem gulosa ter a mesma finalidade da abordagem descrita nesta monografia: gerar uma quadrilaterização que serve como "ponto de partida" para algoritmos de quadrilaterização mais sofisticados e mais complexos, que são capazes de gerar quadrilaterizações de boa qualidade a partir de quadrilaterizações de qualidade mais baixa (TARINI; PIETRONI; CIGNONI; PANOZZO; PUPPO, 2010).

A abordagem utilizada neste trabalho possui dois passos. Primeiro, triângulos que compartilham uma aresta são emparelhados de tal forma que cada triângulo faça parte de, no máximo, um par. Segundo, a aresta comum aos dois triângulos de um mesmo par é removida para gerar um quadrilátero. Já a abordagem proposta por Luiz Velho (VELHO, 2000) possui três passos. O segundo deles é idêntico ao da abordagem desenvolvida aqui, enquanto o primeiro possui o mesmo propósito: emparelhar triângulos. A diferença entre o primeiro passo das duas abordagens reside na forma de escolher os triângulos a serem emparelhados.

Em (VELHO, 2000), o emparelhamento de triângulos adjacentes por aresta é realizado através da *heurística da aresta mais longa*. De acordo com esta heurística, os pares de triângulos são escolhidos de acordo com o comprimento da aresta comum. Quanto mais longa a aresta comum, melhor. O algoritmo utilizado identifica a aresta mais longa, emparelha os dois triângulos que a compartilham e "marca" todas as arestas dos dois triângulos emparelhados. No próximo passo, a aresta mais longa entre as arestas que não estão marcadas é identificada e a tarefa de emparelhamento e marcação de arestas é repetida. O algoritmo termina quando não houver mais arestas não marcadas. A subdivisão resultante pode conter quadriláteros e triângulos "isolados", isto é, cercados por quadriláteros. Quando este é o caso, o algoritmo executa um passo extra de subdivisão dos quadriláteros e triângulos. A subdivisão resultante deste terceiro passo é uma quadrilaterização do objeto definido pela triangulação inicial. A Figura 1.3 ilustra o passo extra de subdivisão.

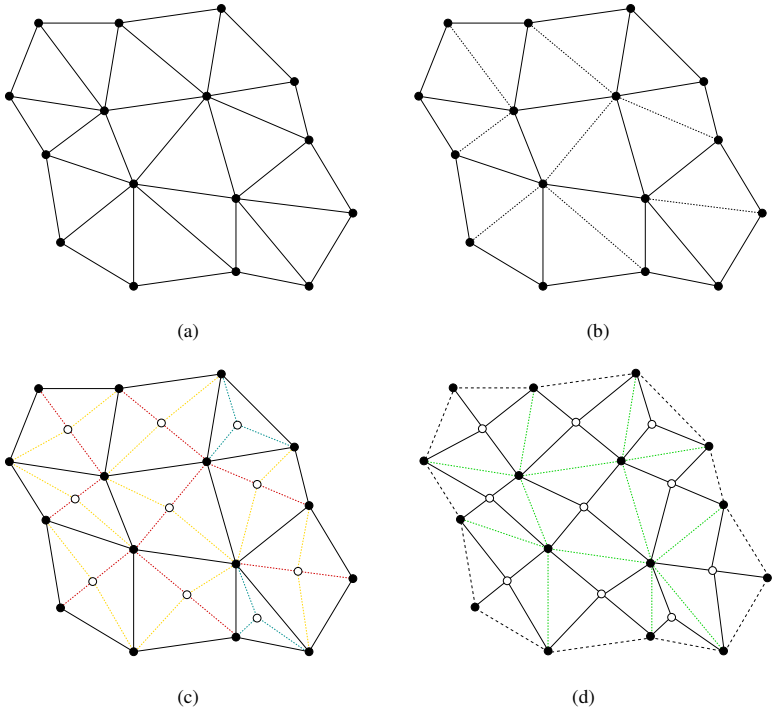


Figura 1.3: (a) Triangulação de entrada. (b) Arestas pontilhadas são removidas. (c) Se houver triângulos isolados, esses são subdivididos por subdivisão baricêntrica em três triângulos e os quadriláteros são subdivididos em quatro triângulos. (d) Em seguida, arestas "internas" e "externas" são invertidas e os quadriláteros são definidos pelas novas arestas externas.

A subdivisão executada pelo algoritmo em (VELHO, 2000) possui duas desvantagens. Primeiro, ela gera uma subdivisão com mais vértices, arestas e quadriláteros do que se o passo não fosse executado. Segundo, cada triângulo isolado gera um vértice com exata-

mente três arestas incidentes nele, contribuindo para tornar a quadrilaterização resultante mais irregular.

A abordagem utilizada neste trabalho não possui nenhuma das duas desvantagens do algoritmo em (VELHO, 2000). Para que isso fosse possível, três observações foram fundamentais. A primeira é que toda triangulação da *classe de superfícies consideradas neste trabalho* possui um número par de triângulos, o que, em princípio, é necessário para realizar um emparelhamento perfeito dos triângulos (isto é, sem que haja triângulos "isolados"). A segunda é que o *grafo dual* da triangulação de entrada é um grafo 3-regular (isto é, cúbico) e sem arestas de corte. A terceira é que este grafo admite um emparelhamento perfeito.

Com as três observações acima em mente, utilizamos um algoritmo para calcular um emparelhamento perfeito no grafo dual da triangulação de entrada. O emparelhamento resultante nos fornece um emparelhamento de triângulos adjacentes por aresta, pois cada vértice do grafo dual corresponde a um triângulo distinto da triangulação e vice-versa. Como o emparelhamento é perfeito, não há necessidade de um passo extra de subdivisão de triângulos e quadriláteros, como em (VELHO, 2000), para gerar uma quadrilaterização da superfície.

Embora o cálculo de um emparelhamento perfeito nos permita gerar quadrilaterizações "menores" e menos irregulares do que aquelas geradas por (VELHO, 2000), este tipo de emparelhamento não leva em conta a qualidade da forma dos quadriláteros gerados. Para incorporar este benefício, criamos duas variações da nossa abordagem. A primeira delas calcula um *emparelhamento perfeito de custo máximo* e a segunda, um *emparelhamento de custo máximo* (não necessariamente perfeito). Essas duas variações de emparelhamento nos permitiram realizar uma comparação experimental mais conclusiva entre as duas abordagens.

As comparações entre as quadrilaterizações geradas pela nossa abordagem e por aquela em (VELHO, 2000) se utilizam de uma métrica de qualidade proposta recentemente em (DANIELS; NONATO; SIQUEIRA; LIZIER; SILVA, 2011). Para enriquecer as comparações experimentais descritas nesta monografia, geramos quadrilaterizações a partir de emparelhamentos em grafos ponderados cujos pesos foram dados pela própria métrica usada para medir a qualidade das quadrilaterizações.

É importante ressaltar que tanto a abordagem descrita aqui quanto a abordagem descrita em (VELHO, 2000) são incapazes, em geral, de gerar quadrilaterizações de "boa" qualidade, tais como as geradas por algoritmos mais complexos e menos robustos (MARINOV;

KOBBELT, 2004; DONG; KIRCHER; GARLAND, 2005; RAY; LI; LéVY; SHEFFER; ALLIEZ, 2006; TONG; ALLIEZ; COHEN-STEINER; DESBRUN, 2006; KÄLBERER; NIESER; POLTHIER, 2007; LAI; KOBBELT; HU, 2008; BOMMES; ZIMMER; KOBBELT, 2009). No entanto, a importância de abordagens mais simples e robustas está no fato delas servirem para gerar quadrilaterizações "iniciais" para algoritmos de simplificação, tais como em (TARINI; PIETRONI; CIGNONI; PANOZZO; PUPPO, 2010), que geram quadrilaterizações de "boa" qualidade a partir de uma quadrilaterização inicial. Como a qualidade da quadrilaterização inicial influencia o desempenho do algoritmo de simplificação, uma quadrilaterização "menor" e menos irregular do que a gerada pela abordagem em (VELHO, 2000) é sempre mais desejável.

Entre as contribuições do trabalho ora descrito, ressaltamos:

- *Uma abordagem baseada em grafos para converter triangulações de superfícies em quadrilaterizações.* Até onde saibamos, tal abordagem havia sido utilizada apenas para converter triangulações de regiões poligonais em quadrilaterizações (RAMASWAMI; RAMOS; TOUSSAINT, 1998). Ainda assim, o emparelhamento utilizado não era perfeito e nem baseado nas mesmas técnicas usadas aqui. A abordagem utilizada aqui foi descrita no artigo (DANIELS; NONATO; SIQUEIRA; LIZIER; SILVA, 2011) de co-autoria do orientador desta monografia.

- *Uma comparação experimental entre uma abordagem baseada em grafos e uma abordagem gulosa para converter triangulações de superfícies em quadrilaterizações.* A abordagem baseada em grafos possui complexidade de tempo maior do que a da abordagem gulosa. Logo, a comparação experimental descrita nesta monografia pode servir como guia para aqueles interessados em saber se os resultados obtidos com a abordagem baseada em grafos justificam a sua utilização, quando comparados aos resultados obtidos por uma abordagem gulosa e mais "barata", como aquela em (VELHO, 2000).

- *Uma implementação de um algoritmo para calcular emparelhamentos perfeitos em grafos gerais.* O algoritmo utilizado para calcular emparelhamentos perfeitos na nossa abordagem foi implementado e o código será disponibilizado de forma aberta e gratuita.

O desenvolvimento do trabalho descrito contribuiu para a formação do autor da monografia de diversas formas. Em particular, o autor teve oportunidade de ler diversos artigos e capítulos de livro sobre emparelhamentos em grafos bipartidos e em grafos gerais, alguns

deles descrevendo o estado-da-arte em algoritmos para o problema do emparelhamento. O autor também teve uma breve exposição à programação linear e aprendeu a utilizar bibliotecas importantes com rotinas para cálculo de emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo (não necessariamente perfeitos) em grafos gerais.

1.3 Organização

Este texto está organizado em seis capítulos além deste. O Capítulo 2 apresenta algumas definições necessárias ao entendimento da descrição da nossa abordagem. O Capítulo 3 faz uma revisão da bibliografia relevante para este trabalho. O Capítulo 4 descreve o algoritmo usado por nossa abordagem para cálculo de emparelhamentos perfeitos na solução do problema-alvo. O Capítulo 5 descreve, brevemente, os fundamentos dos algoritmos para cálculo de emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo (não necessariamente perfeitos). O Capítulo 6 discute alguns aspectos da implementação dos algoritmos de emparelhamento e apresenta os resultados da comparação experimental da nossa abordagem com a abordagem em (VELHO, 2000). Finalmente, o Capítulo 7 conclui o presente texto com um resumo dos pontos mais importantes e contribuições do trabalho, relata as dificuldades encontradas pelo autor e sugere algumas extensões futuras para o trabalho.

2 Conceitos Básicos

Este capítulo contém algumas definições necessárias para o pleno entendimento do problema-alvo deste trabalho e, também, de sua solução. A Seção 2.1 introduz o conceito de superfície topológica. A Seção 2.2 discute subdivisões de superfícies conexas, compactas e orientáveis no espaço euclidiano tridimensional e estabelece uma relação entre subdivisões e imersões de grafos em superfícies. A Seção 2.3 revisa o conceito de emparelhamento em grafos e apresenta alguns resultados importantes relacionados a este conceito.

2.1 Superfícies

Esta monografia lida com subdivisões de certos subconjuntos de $\mathbb{E}^3$ denominados superfícies, onde $\mathbb{E}^3$ denota o espaço euclidiano afim dotado do produto escalar como produto interno. Informalmente, uma *superfície*, S, em $\mathbb{E}^3$ é um um subconjunto de $\mathbb{E}^3$ que, localmente, "assemelha-se" ao plano euclidiano. Para formalizar o que entendemos por uma superfície em $\mathbb{E}^3$, necessitamos dos conceitos de bola aberta e homeomorfismo. Esses conceitos e todos os demais conceitos apresentados nesta seção foram retirados de (BLOCH, 1997).

Definição 2.1.1. *A* bola aberta, $\mathbb{B}_r(p, \mathbb{E}^n)$, de centro p e raio r em $\mathbb{E}^n$, *onde* $p \in \mathbb{E}^n$ *e* $r \in \mathbb{R}$, *com* $r > 0$, *é o conjunto*

$$\mathbb{B}_r(p, \mathbb{E}^n) = \{q \in \mathbb{E}^n \mid \text{dist}(p, q) < r\},$$

onde $\text{dist}(p, q)$ *é a* distância euclidiana do ponto p ao ponto q; *isto é,* $\text{dist}(p, q)$ *é igual a* $\sqrt{\langle \mathbf{pq}, \mathbf{pq} \rangle}$, *onde* $\langle \cdot, \cdot \rangle$ *denota produto escalar e* $\mathbf{pq}$ *é o vetor (único) que satisfaz* $q = p + \mathbf{pq}$.

Note que uma bola aberta em $\mathbb{E}^2$ é igual ao interior de uma circunferência em $\mathbb{E}^2$, enquanto uma bola aberta em $\mathbb{E}^3$ é igual ao interior de uma esfera em $\mathbb{E}^3$. Quando $p = \mathbf{O} =$

$(0,0) \in \mathbb{E}^n$ e $r = 1$ na Definição 2.1.1, denotamos $\mathbb{B}_r(p, \mathbb{E}^n) = \mathbb{B}_1(\mathbf{O}, \mathbb{E}^n)$ simplesmente por $\mathbb{B}^n$. Em particular, quando $n = 2$, chamamos a bola aberta $\mathbb{B}^2$ de *disco unitário aberto* em $\mathbb{E}^2$.

Definição 2.1.2. *Sejam $X \subset \mathbb{E}^n$ e $Y \subset \mathbb{E}^m$ dois conjuntos quaisquer e seja $f : X \to Y$ uma função de X em Y. Então, a função f é um* homeomorfismo *se ela for bijetora e se f e sua inversa, $f^{-1} : Y \to X$, forem ambas funções contínuas. Se a função f é um homeomorfismo, então dizemos que X e Y são* homeomorfos *e denotamos este fato por $X \approx Y$.*

Por exemplo, se $X = (-1, 1) \subset \mathbb{E}$ e $Y = (-2, 2) \subset \mathbb{E}$, então a função $g : X \to Y$, dada por $g(x) = 2 \cdot x$, é bijetora e contínua. A função inversa, $g^{-1} : Y \to X$, de g dada por $g^{-1}(y) = (1/2) \cdot y$, também é contínua. Logo, pela Definição 2.1.2, a função g é um homeomorfismo. Intuitivamente, se os conjuntos X e Y são homeomorfos, então eles são equivalentes com respeito a uma certa "deformação" (isto é, o homeomorfismo). Isto significa que podemos esticar, encolher, amassar e dobrar (mas sem grudar partes) o conjunto X para que ele se torne o conjunto Y. Por exemplo, a função g do exemplo acima "estica" o intervalo $(-1, 1)$, transformando-o no intervalo $(-2, 2)$. Analogamente, a função inversa, g^{-1}, encolhe o intervalo aberto $(-2, 2)$, transformando-o no intervalo $(-1, 1)$.

Definição 2.1.3. *Um subconjunto $S \subset \mathbb{E}^n$ é dito ser uma* superfície topológica*, ou simplesmente* superfície*, se para cada ponto $p \in S$, existir $r \in \mathbb{R}$, com $r > 0$, tal que a interseção da bola aberta $\mathbb{B}_r(p, \mathbb{E}^n)$ com S é homeomorfa ao disco unitário aberto, $\mathbb{B}^2$, em $\mathbb{E}^2$.*

Informalmente, a Definição 2.1.3 nos diz que todo ponto de uma superfície possui uma pequena vizinhança homeomorfa a uma pequena porção do plano euclidiano na forma de disco. Este é o caso de qualquer plano definido em $\mathbb{E}^3$, assim como de qualquer esfera ou n-toro em $\mathbb{E}^3$. No entanto, um cone de duas folhas em $\mathbb{E}^3$, como ilustrado na Figura 2.1, não é uma superfície, pois o conjunto interseção do cone com *qualquer* bola aberta centrada no ponto comum às duas folhas não se assemelha ao disco $\mathbb{B}^2$. Uma reta em $\mathbb{E}^3$ também não é uma superfície, pois o conjunto interseção da reta com *qualquer* bola aberta centrada em um ponto da reta é um segmento de reta. Porém, um segmento de reta não é homeomorfo a $\mathbb{B}^2$.

Neste trabalho, restringiremos nossa atenção às esferas e n-toros em $\mathbb{E}^3$. Além disso, quando nos referimos a esferas (resp. n-toros), estamos nos referindo à classe de todas as

superfícies em $\mathbb{E}^3$ homeomorfas a uma esfera (resp. um n-toro). As esferas e n-toros são as únicas superfícies conexas, compactas e orientáveis em $\mathbb{E}^3$. Informalmente, uma superfície $S \subset \mathbb{E}^3$ é *conexa* se, e somente se, existir um caminho (isto é, uma curva) formado apenas por pontos de S conectando qualquer par de pontos de S. Por sua vez, uma superfície $S \subset \mathbb{E}^3$ é *compacta* se, e somente se, existir um cubo em $\mathbb{E}^3$ que contenha S. Essas duas propriedades nos permitem definir triangulações e quadrilaterizações, com representações finitas, nas superfícies. Finalmente, uma superfície conexa e compacta em $\mathbb{E}^3$ que também é *orientável* particiona $\mathbb{E}^3$ em três regiões disjuntas: a própria superfície e seu interior e exterior.

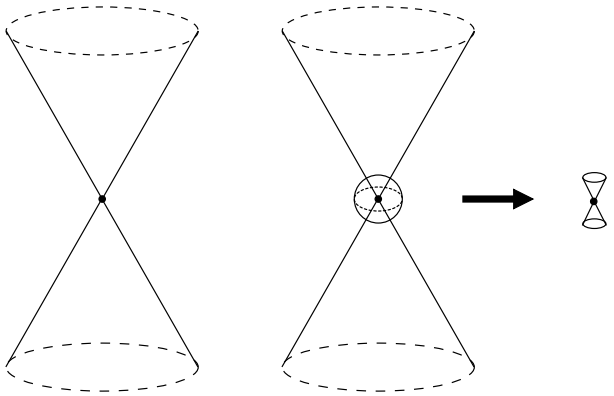


Figura 2.1: Um cone de duas folhas em $\mathbb{E}^3$ não é uma superfície.

2.2 Subdivisões, triangulações e quadrilaterizações

De agora em diante, o termo superfície será usado para denotar uma superfície homeomorfa a uma esfera ou a um n-toro. Pelo exposto na seção anterior, sabemos que tais superfícies são conexas e compactas. Nesta seção, introduzimos os conceitos de subdivisão, triangulação e quadrilaterização de superfícies. Em seguida, estabelecemos uma conexão entre o conceito de subdivisão de uma superfície e o de imersão de um grafo em uma superfície. A definição de subdivisão apresentada a seguir pode ser encontrada em (GUIBAS; STOLFI, 1985), enquanto as definições de triangulações, quadrilaterizações e outras relacionadas à teoria dos grafos topológica podem ser encontradas em (GROSS; TUCKER, 2001).

Definição 2.2.1. *Uma* subdivisão *de uma superfície $S \subset \mathbb{E}^n$ é uma partição $\mathcal{P}$ de S em três coleções finitas de partes disjuntas: os* vértices*, as* arestas *e as* faces*, que denotamos por $V_S(\mathcal{P})$, $E_S(\mathcal{P})$ e $F_S(\mathcal{P})$, respectivamente, e que satisfazem as condições enumeradas a seguir:*

S1. *Todo vértice é um ponto de S.*

S2. *Toda aresta é uma curva em S.*

S3. *Toda face é um subconjunto de S homeomorfo a $\mathbb{B}^2$.*

S4. *A fronteira de toda face é um caminho fechado em S formado por vértices e arestas.*

Os vértices, arestas e faces de uma subdivisão são chamados de elementos *da subdivisão.*

A Figura 2.2 contém alguns exemplos de subdivisões de superfícies.

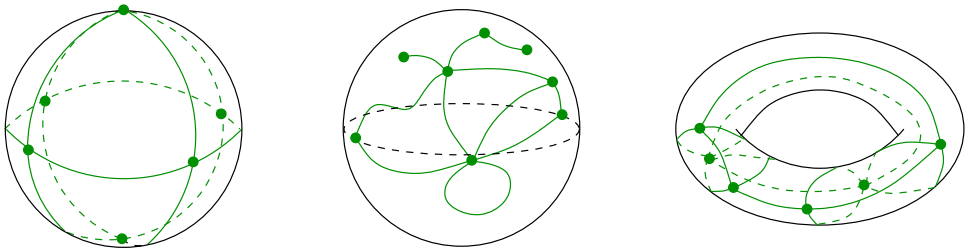


Figura 2.2: Alguns exemplos de subdivisões de superfícies em $\mathbb{E}^3$.

A condição **S4** requer uma explicação detalhada. Seja $\overline{\mathbb{B}}^2 = \{q \in \mathbb{E}^2 \mid \text{dist}(q, \mathbf{O}) \leq 1\}$ o *disco unitário fechado em* $\mathbb{E}^2$ e seja $\mathbb{S}^1 = \{q \in \mathbb{E}^2 \mid \text{dist}(q, \mathbf{O}) = 1\}$ a circunferência de raio unitário em $\mathbb{E}^2$ centrada em $\mathbf{O} = (0, 0) \in \mathbb{E}^2$. Note que $\overline{\mathbb{B}}^2 = \mathbb{B}^2 \cup \mathbb{S}^1$. Nós definimos um *caminho simples* em $\mathbb{S}^1$ como sendo uma partição de $\mathbb{S}^1$ em uma seqüência de pontos isolados e arcos abertos. Então, a condição **S4** tem o seguinte significado: para toda face τ de $F_S(\mathcal{P})$, existe um caminho simples, π, em $\mathbb{S}^1$ e uma função contínua, $g_\tau : \overline{\mathbb{B}}^2 \to \overline{\tau}$, de $\overline{\mathbb{B}}^2$ no fecho, $\overline{\tau}$, de τ tal que (1) a restrição de g_τ a $\mathbb{B}^2$ é um homeomorfismo de $\mathbb{B}^2$ em τ, (2) a restrição de g_τ a cada arco de π é um homeomorfismo deste arco em uma aresta de $\mathcal{P}$ e (3) a restrição de g_τ a cada ponto isolado de π é um homeomorfismo do ponto isolado em um vértice de $\mathcal{P}$. A Figura 2.3 ilustra a nossa formalização do significado da condição **S4**.

A condição **S4** possui uma série de implicações importantes. Por definição, os pontos isolados e arcos de π se alternam em um percurso ao redor de $\mathbb{S}^1$. Logo, se α é um arco entre dois pontos consecutivos, p e q, de π, então a imagem, $g_\tau(\alpha)$, de α é uma aresta de de $\mathcal{P}$ incidente nos vértices $g_\tau(p)$ e $g_\tau(q)$ de $\mathcal{P}$. Isto significa que as imagens dos elementos de π, tomadas na ordem em que percorremos $\mathbb{S}^1$, constituem um caminho, π_τ, conexo e fechado, de arestas e vértices de $\mathcal{P}$ cuja união é a fronteira de τ. É importante ressaltar que π_τ não é necessariamente simples, pois g_τ pode associar dois ou mais arcos ou pontos

isolados distintos de $\mathbb{S}^1$ a uma mesma aresta ou vértice de $\mathcal{P}$. Isto quer dizer que, embora toda face de $\mathcal{P}$ seja homeomorfa a $\mathbb{B}^2$, o fecho da face pode não ser homeomorfo a $\overline{\mathbb{B}}^2$ (veja a Figura 2.2).

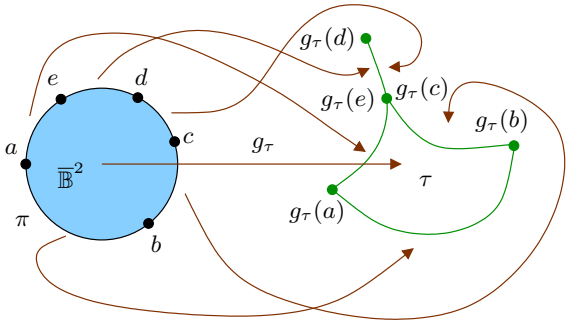


Figura 2.3: Ilustração da condição **S4** da Definição 2.2.1. Note que $g_\tau(c) = g_\tau(e)$.

Como é impossível cobrir, com um número finito de vértices e arestas, uma região qualquer de S homeomorfa a $\mathbb{B}^2$, toda aresta e todo vértice de $\mathcal{P}$ deve ser incidente em alguma face de $\mathcal{P}$. Usando a condição **S4**, podemos concluir que toda aresta de $\mathcal{P}$ está inteiramente contida na fronteira de alguma face de $\mathcal{P}$ e que ela é incidente em dois (não necessariamente distintos) vértices de $\mathcal{P}$, denominados *extremos* da aresta. Quando os dois extremos são o mesmo vértice, então a aresta é um *laço* e o fecho deste laço é homeomorfo a $\mathbb{S}^1$.

Definição 2.2.2. *Uma* triangulação *de uma superfície $S \subset \mathbb{E}^n$ é uma subdivisão de S na qual a fronteira de cada face consiste de exatamente três arestas e três vértices distintos. De forma análoga, uma* quadrilaterização *de uma superfície $S \subset \mathbb{E}^n$ é uma subdivisão de S na qual a fronteira de cada face consiste de exatamente quatro arestas e quatro vértices distintos.*

O seguinte fato será usado na Seção 3.1:

Lema 2.2.3. *Seja S uma superfície em $\mathbb{E}^n$ e seja $\mathcal{T}$ uma triangulação de S. Então, toda aresta de $\mathcal{T}$ é incidente em exatamente duas faces de $\mathcal{T}$.*

Demonstração. Uma aresta não pode ser incidente em mais de duas faces, pois isto implicaria no fato de nenhum ponto da aresta possuir uma vizinhança homeomorfa a $\mathbb{B}^2$, o que é impossível já que S é uma superfície. Então, há apenas duas possibilidades: a aresta é incidente em apenas uma face ou em duas faces, pois toda aresta deve ser incidente em alguma face. Pela Definição 2.2.2, os vértices de cada face de $\mathcal{T}$ são distintos. Pela Definição 2.2.1, condição **S3**, toda face é homeomorfa a $\mathbb{B}^2$. Esses dois fatos implicam que

as três arestas de toda face são distintas também. Como uma única face com três vértices distintos e três arestas distintas não pode cobrir uma esfera nem um n-toro por inteiro, o complemento do fecho da face com respeito a S deve conter pelo menos mais uma face de $\mathcal{T}$. Logo, concluímos que cada aresta da triangulação $\mathcal{T}$ é incidente em exatamente duas faces de $\mathcal{T}$. □

O seguinte teorema de Radó estabelece que toda superfície compacta admite uma triangulação:

Teorema 2.2.4. *Toda superfície compacta admite uma triangulação.*

As provas existentes para o teorema acima são extremamente técnicas (veja, por exemplo, (RADÓ, 1925; ALFORS; SARIO, 1960)). Do ponto de vista deste trabalho, o importante é que o teorema estabelece a existência de subdivisões (em particular, triangulações) para qualquer superfície compacta, o que inclui as superfícies conexas, compactas e orientáveis em $\mathbb{E}^3$.

Existem inúmeros algoritmos para construir triangulações de superfícies em $\mathbb{E}^3$ (VLASSOPOULOS, 1990; NING; BLOOMENTHAL, 1993; VELHO; FIGUEIREDO; GOMES, 1999; BOISSONNAT; OUDOT, 2005; CHENG; DEY; RAMOS; RAY, 2007; MEYER; KIRBY; WHITAKER, 2007). A grande maioria deles produz aproximações para triangulações da superfície. Em particular, as arestas das triangulações produzidas são segmentos de reta, o que implica que elas podem não estar inteiramente contidas na superfície. No entanto, alguns desses algoritmos oferecem garantias teóricas (topológicas e geométricas) para a triangulação aproximada, tais como a existência de um homeomorfismo da superfície na superfície resultante da união dos elementos da triangulação e um limite inferior para o ângulo mínimo dos triângulos (BOISSONNAT; OUDOT, 2005; CHENG; DEY; RAMOS; RAY, 2007). Os algoritmos que calculam triangulações e até subdivisões mais gerais de forma exata são bem mais raros e, em geral, restringem-se a certos tipos de superfície ou representação de superfície (SUGIHARA, 2002; BERBERICH; KERBER, 2008; BERBERICH; FOGEL; HALPERIN; MEHLHORN; WEIN, 2007; FOGEL; SETTER; HALPERIN, 2008).

Subdivisões podem ser definidas através de "imersões" de grafos em superfícies. Como lidaremos essencialmente com grafos na solução proposta aqui para o problema-alvo descrito na Seção 3.1, esta forma de definir subdivisões é mais natural. Em particular, o nosso interesse não é na imersão em si, mas no fato de poder manipular subdivisões através de grafos.

Informalmente, uma imersão de um grafo em uma superfície é um "desenho" do grafo

na superfície de tal forma que duas arestas quaisquer possam se intersectar apenas em seus vértices extremos. Isto é uma generalização da noção de planaridade. Formalmente, temos:

Definição 2.2.5. *Sejam G um grafo e S uma superfície (conexa e compacta) em $\mathbb{E}^n$, com $n \geq 3$. Então, uma* imersão de G em S *é uma função contínua e injetora, $i : G \to S$, que leva cada vértice de G em um ponto de S e cada aresta de G em uma curva simples em S tal que*

I1. *Os pontos extremos da curva associada com uma aresta de G são os pontos associados com os vértices da aresta.*

I2. *Nenhuma curva contém um ponto de S associado com outros vértices que não sejam os vértices da aresta associada com a curva.*

I3. *A interseção de quaisquer duas curvas não pode ser um ponto interior de nenhuma delas.*

Se a função $i : G \to S$ é uma imersão do grafo G na superfície S, então o complemento, $\overline{i(G)} = S - i(G)$, de $i(G)$ com respeito a S é uma coleção de *regiões*, denominadas de *faces*. Se cada face em $\overline{i(G)}$ é homeomorfa ao disco aberto unitário, então a imersão i é denominada uma *imersão 2-celular* ou um *mapa*. Porém, a tripla formada pelas coleções de pontos e curvas em $i(G)$ e das faces em $\overline{i(G)}$ nada mais é do que uma subdivisão de S. Mais especificamente, o fato de S ser compacta e as condições **I1**-**I3** da Definição 2.2.5 são equivalentes à condição **S4** da Definição 2.2.1 (GUIBAS; STOLFI, 1985; HENLE, 1994). Logo, dizemos que i *induz* uma subdivisão $\mathcal{P}$ em S e, muitas vezes, referimo-nos a $\mathcal{P}$ através de i.

2.3 Emparelhamento em grafos

Como veremos na Seção 3.1, o problema-alvo desta monografia foi formulado como um problema de emparelhamento em grafos arbitrários. Em particular, percebemos que os grafos, embora arbitrários, são 3-regulares e sem arestas de corte. Esse fato nos motivou a estudar o problema e tentar solucioná-lo com a utilização de algum algoritmo para emparelhamento nesse tipo de grafo. Por acaso, houve avanços recentes e importantes no que diz respeito a algoritmos eficientes para emparelhamento em grafos 3-regulares sem arestas de corte, como mencionado no Capítulo 3. Nesta seção, introduzimos o conceito

de emparelhamento em grafos, assim como outros conceitos relacionados e resultados relevantes para a monografia. Esses conceitos e resultados foram retirados do livro (BONDY; MURTY, 2007).

Definição 2.3.1. *Seja $G = (V, E)$ um grafo, onde V é o conjunto de vértices e E é o conjunto de arestas. Então, dizemos que um subconjunto de arestas, M, de E é um* emparelhamento no grafo G *se não existem duas arestas em M com um vértice em comum. Um vértice de G sobre o qual uma das arestas de M incide é dito* ocupado *com respeito a M. Contrariamente, um vértice que não incide em nenhuma das arestas de M é dito* livre *com respeito a M. De forma similar, uma aresta de G é dita* ocupada *com respeito a M se ela pertence a M, ou* livre *com respeito a M, caso contrário. Para cada aresta em M, dizemos que seus dois vértices extremos estão* emparelhados *com respeito a M.*

Há alguns emparelhamentos que são especialmente importantes neste trabalho:

Definição 2.3.2. *Seja $G = (V, E)$ um grafo e seja $\mathcal{C}$ a coleção de todos os emparelhamentos em G. Então, um emparelhamento $M \in \mathcal{C}$ é de* cardinalidade máxima*, ou simplesmente* máximo*, se, para todo $M' \in \mathcal{C}$, temos que $|M| \geq |M'|$, onde $|\cdot|$ denota cardinalidade.*

Definição 2.3.3. *Seja $G = (V, E)$ um grafo e seja M um emparelhamento em G. Então, o emparelhamento M é* perfeito *se todos os vértices de G estão ocupados com respeito a M.*

Dizemos que um grafo $G = (V, E)$ é *ponderado* se existe uma função, $c : E \to \mathbb{R}$, denominada *custo*, que associa a cada aresta de G um valor real. Seja $X \subseteq E$. Então, o total

$$c(X) = \sum_{e \in X} c(e)$$

é o *custo do conjunto* X. Em algumas aplicações, emparelhamentos que minimizam ou maximizam esse custo são objetos de interesse. Tais emparelhamentos são definidos como segue:

Definição 2.3.4. *Seja $G = (V, E)$ um grafo ponderado cuja função custo é c. Seja $\mathcal{C}$ a coleção de todos os emparelhamentos em G. Então, um emparelhamento $M \in \mathcal{C}$ é de* custo máximo *(resp.* mínimo*) se, para todo $M' \in \mathcal{C}$, temos $c(M) \geq c(M')$ (resp. $c(M) \leq c(M')$).*

Note que um emparelhamento de custo máximo não é necessariamente um emparelhamento máximo. Neste trabalho, estamos particularmente interessados em emparelhamentos perfeitos, emparelhamentos perfeitos de custo máximo e, também, emparelhamentos de custo máximo. Um emparelhamento perfeito é dito de *custo máximo* se seu custo é maior ou igual ao custo de qualquer outro emparelhamento perfeito no mesmo grafo.

Definição 2.3.5. *Um* caminho *em um grafo, $G = (V, E)$, é uma seqüência,*

$$[v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n],$$

em que, para cada $i = 1, \ldots, n$, e_i é uma aresta em E conectando o vértice v_{i-1} ao vértice v_i, com $v_{i-1}, v_i \in V$. Quando os vértices são distintos dois a dois, dizemos que o caminho é simples.

Como os grafos considerados nesta monografia são todos *simples* (isto é, sem arestas paralelas e laços), nós omitimos as arestas quando descrevemos um caminho. Por exemplo, o caminho

$$[v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n]$$

será escrito como

$$[v_0, v_1, v_2, \ldots, v_n].$$

Definição 2.3.6. *Um* ciclo *em um grafo, $G = (V, E)$, é um caminho,*

$$[v_0, v_1, v_2, \ldots, v_n],$$

em que os vértices $v_0, v_1, \ldots, v_{n-1}$ são distintos dois a dois e $v_0 = v_n$.

Definição 2.3.7. *Seja M um emparelhamento em um grafo G. Então, um* caminho alternante com respeito a M *é um caminho simples cujas arestas estão alternadamente nos conjuntos $E - M$ e M, isto é, um caminho simples que alterna arestas livres e ocupadas com respeito a M. Caso os vértices extremos desse caminho sejam livres com respeito a M, dizemos também que o caminho é* aumentante com respeito a M. *Assim, as arestas extremas de um caminho aumentante também são livres com respeito ao emparelhamento. Dizemos ainda que um emparelhamento M é aumentado ao longo de um caminho aumentante P quando todas as arestas comuns a M e P têm o "estado" alternado. Isto é, as que forem livres (resp. ocupadas) com respeito a M se tornam ocupadas (resp. livres) com respeito a M.*

O teorema a seguir estabelece uma relação entre emparelhamentos máximos e caminhos aumentantes:

Teorema 2.3.8 (Berge, 1957)**.** *Seja G um grafo simples e M um emparelhamento em G. Então, M é máximo se, e somente se, G não tiver nenhum caminho aumentante com respeito a M.*

Definição 2.3.9. *Um grafo $G = (V, E)$ é dito* k-regular *se todos os vértices em V tiverem grau k com respeito às arestas em E. Algumas vezes, dizemos simplesmente que o grafo é* regular*.*

Definição 2.3.10. *Um grafo $G' = (V', E')$ é dito* subgrafo *de $G = (V, E)$ se $V' \subseteq V$ e $E' \subseteq E$. Quando $V' = V$, dizemos também que G' é um* subgrafo gerador *de G. Se um grafo G' for subgrafo gerador de outro grafo G e, além disso, G' for k-regular, dizemos que G' é um* k-fator *de G.*

Pela Definição 2.3.10, podemos concluir que um subgrafo, $G' = (V', E')$, de um grafo $G = (V, E)$ é um 1-fator de G se, e somente se, E' define um emparelhamento perfeito em G.

Definição 2.3.11. *Uma aresta, e, de um grafo $G = (V, E)$ é denominada* aresta de corte*,* ponte *ou* ístimo *se não existir nenhum caminho conectando os dois vértices de e no grafo $G' = (V, E - \{e\})$.*

Finalmente, temos o seguinte teorema:

Teorema 2.3.12 (Petersen, 1891)**.** *Todo grafo 3-regular sem arestas de corte possui um 1-fator.*

3 Revisão Bibliográfica

Este capítulo descreve, formalmente, o problema-alvo desta monografia e apresenta as principais referências que utilizamos para desenvolver a nossa abordagem de solução. Mais especificamente, a Seção 3.1 descreve o problema-alvo, usando as definições vistas no Capítulo 2. A Seção 3.2 descreve duas variações do problema-alvo que são de interesse deste trabalho. A Seção 3.3 revisa os principais algoritmos e resultados disponíveis na literatura para cálculo de emparelhamentos perfeitos em grafos não necessariamente bipartidos[1] e, em particular, grafos 3-regulares sem aresta de corte. A Seção 3.4 revisa a literatura sobre emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo em grafos gerais. A Seção 3.5 fornece mais detalhes sobre o algoritmo guloso em (VELHO, 2000).

3.1 O problema-alvo

Em linhas gerais, o problema-alvo desta monografia é o de converter uma triangulação, $\mathcal{T}$, de uma superfície $S \subset \mathbb{E}^3$ em uma quadrilaterização de S. Por "converter", entendemos a obtenção de uma quadrilaterização cujo conjunto de vértices contém o conjunto de vértices da triangulação. Em outras palavras, o processo de conversão pode adicionar novos vértices ao conjunto de vértices da triangulação de entrada, mas não pode remover nenhum deles. Ao contrário dos vértices, arestas da triangulação podem ser removidas livremente.

Pelo Lema 2.2.3, cada aresta de uma triangulação da superfície é incidente em exatamente duas faces adjacentes da triangulação. Então, podemos formar uma face de "quatro lados" removendo a aresta comum a duas faces de "três lados" de $\mathcal{T}$. Logo, podemos tentar converter $\mathcal{T}$ em uma quadrilaterização através da remoção de arestas de $\mathcal{T}$. Para que este processo possa gerar uma quadrilaterização, o número de faces de $\mathcal{T}$ deve ser necessariamente par e apenas uma das três arestas incidentes em uma mesma face de $\mathcal{T}$ pode ser

[1] Denominados, aqui, de grafos gerais.

removida.

O que acabamos de descrever acima pode ser formalizado como um problema de emparelhamento em grafos. De fato, sejam $S \subseteq \mathbb{E}^3$ uma superfície e $i : G \to S$ uma imersão de um grafo $G = (V, E)$ em S que induz uma triangulação, $\mathcal{T}$, em S. Então, temos:

Definição 3.1.1. *O* grafo dual, $G' = (V', E')$, *de* G *é tal que (1)* $v_\tau \in V'$ *se, e somente se, existe uma face* τ *em* $i(G)$ *e (2)* $e' = (v_\sigma, v_\mu) \in E'$ *se, e somente se, existe uma aresta* e *em* G *tal que* $i(e)$ *é incidente nas faces* σ *e* τ *da triangulação* $\mathcal{T}$ *induzida por* $i(G)$. *O vértice* v_τ *é* dual *da face* τ *e vice-versa, assim como a aresta* e' *é* dual *da aresta* e *e vice-versa.*

A Figura 3.1 ilustra a Definição 3.1.1.

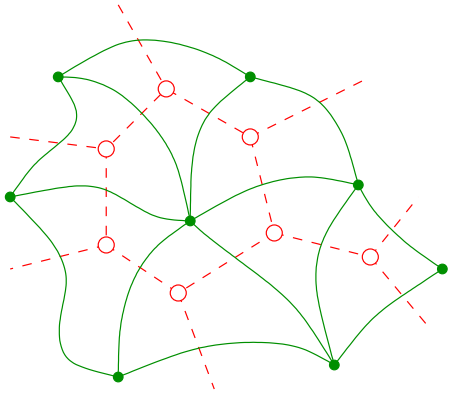


Figura 3.1: Representação gráfica de parte de um grafo dual (com arestas tracejadas).

Seja M um emparelhamento qualquer no grafo dual, G'. Então, como quaisquer duas arestas de M não podem compartilhar um mesmo vértice de G', podemos formar um conjunto de faces quadrangulares "independentes" removendo da triangulação $\mathcal{T}$ toda aresta $e \in G$ cuja aresta dual, e', pertence a M. Isto significa que o problema de converter $\mathcal{T}$ em uma quadrilaterização de S pode ser resolvido através de um emparelhamento perfeito em G', se um existir. Porém, como veremos mais adiante, tal emparelhamento sempre existe!

Algumas observações sobre o grafo dual, G', nos permitem estabelecer fatos importantes sobre a existência de emparelhamentos perfeitos em G'. Primeiro, observamos que o grafo G' é 3-regular, pois cada face τ de $\mathcal{T}$ possui três arestas de $i(G)$ incidentes nela, o que implica que há três arestas incidentes em v_τ em G': as arestas duais das arestas de $i(G)$ incidentes em τ. Segundo, temos que o grafo G' não possui nenhuma aresta de

corte. Esta segunda afirmação não é óbvia e, portanto, é enunciada abaixo como uma proposição:

Proposição 3.1.2. *O grafo dual da Definição 3.1.1 não possui arestas de corte.*

Demonstração. Seja e' qualquer aresta do grafo dual $G' = (V', E')$. Então, considere a aresta, e, dual de e' em G. Seja v qualquer um dos dois vértices de e (os quais são distintos, por definição de triangulação). Pelo Lema 2.2.3, a imagem, $i(e)$, da aresta e pela imersão i é incidente em duas faces triangulares, digamos τ e μ, de $i(G)$. Tanto τ quanto μ são incidentes em v. Como S é orientável, podemos enumerar todas as faces de $i(G)$ incidentes em $i(v)$ em um percurso anti-horário ao redor de $i(v)$. Considere a seqüência, $e_1, \ldots, e_k$, de arestas de $i(G)$ que são encontradas durante este percurso, como ilustrado pela Figura 3.2.

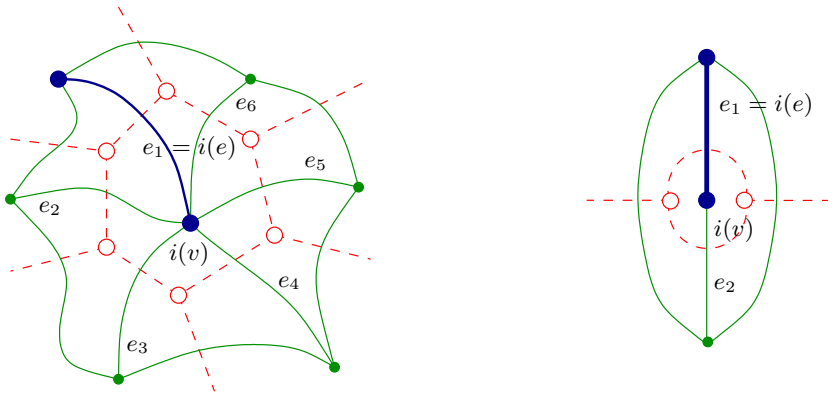


Figura 3.2: Ilustração da prova da Proposição 3.1.2.

Note que há pelo menos duas arestas distintas incidentes em $i(v)$, isto é, $k \geq 2$. De fato, há exatamente duas arestas distintas (a aresta $i(e)$ e uma outra) quando τ e μ compartilham exatamente duas arestas e, pelo menos, três arestas distintas quando τ e μ compartilham apenas a aresta $i(e)$. Logo, as arestas duais, $e'_1, \ldots, e'_k$, de $e_1, \ldots, e_k$ e seus vértices formam um ciclo com $k \geq 2$ arestas em G'. Ao retiramos e' de E', obtemos um grafo, $G'' = (V', E' - \{e'\})$, no qual os dois vértices extremos de e' estão conectados por um caminho formado pelas arestas em $\{e'_1, \ldots, e'_k\} - \{e'\}$. Logo, a aresta e' não é aresta de corte do grafo dual, G'. Como e' foi escolhida arbitrariamente, nossa afirmação é verdadeira. □

As duas observações acima e o Teorema 2.3.12 nos permitem concluir que G' sempre admite um emparelhamento perfeito. À primeira vista, este fato pode causar estranheza, pois ele implica que o número de faces de $i(G)$ tem de ser par. Porém, este é mesmo o

caso, como pode ser constatado através da *característica de Euler-Poincaré*. Mais especificamente, se a superfície S possui *genus* igual a n, então a seguinte fórmula deve ser satisfeita para qualquer triangulação $\mathcal{T}$ de S: $n_v - n_e + n_f = 2 \cdot (n-1)$, onde n_v, n_e e n_f são, respectivamente, os números de vértices, arestas e faces de $\mathcal{T}$ (TRUDEAU, 1993). Mas, como cada aresta é incidente em exatamente duas faces e cada face é limitada por exatamente três arestas, temos que $3 \cdot n_f = 2 \cdot n_e$. Multiplicando ambos os lados da fórmula por 2 e substituindo $2 \cdot n_e$ por $3 \cdot n_f$, temos

$$2 \cdot n_v - 3 \cdot n_f + 2 \cdot n_f = 4 \cdot (n-1) \Rightarrow n_f = 2 \cdot (n_v - 2 \cdot (n-1)) .$$

Logo, o número de faces, n_f, de $\mathcal{T}$ é mesmo um número par. Então, o nosso problema-alvo pode ser formulado como aquele de encontrar um emparelhamento perfeito no grafo dual, G'.

3.2 Duas variações do problema-alvo

Como mencionado no Capítulo 1, a solução do problema-alvo via emparelhamento perfeito no grafo dual da triangulação de entrada gera uma quadrilaterização menor (isto é, com menos quadrados, vértices e arestas) e menos irregular (isto é, com menos vértices de grau diferente de 4) do que aquelas geradas pelo algoritmo guloso em (VELHO, 2000). Mas, ao contrário das quadrilaterizações geradas por este algoritmo guloso, as quadrilaterizações geradas por emparelhamento perfeito não levam em conta a qualidade de forma dos quadriláteros.

Felizmente, uma pequena variação da nossa abordagem baseada em grafos nos permite incorporar o atributo qualidade de forma na solução. A ideia básica é representar o grafo dual, G', como um grafo ponderado no qual o custo associado a cada uma de suas arestas possui uma relação direta com a qualidade do emparelhamento dos dois triângulos duais dos vértices da aresta. Por exemplo, podemos associar a cada aresta e' de G' o comprimento de $i(e)$, onde $i(e)$ é a imersão de e em S e e é a aresta dual de e'. No entanto, é importante frisar que qualquer outra medida de custo pode ser utilizada. A partir deste ponto, o problema-alvo passa a ser aquele de encontrar um *emparelhamento perfeito de custo máximo* no grafo dual, G'.

Uma outra possibilidade é encontrar um *emparelhamento de custo máximo* no grafo dual, G'. Neste caso, o emparelhamento resultante pode não ser perfeito e o mesmo passo de subdivisão utilizado em (VELHO, 2000) pode ser usado aqui para gerar a quadrilateri-

zação final. Obviamente, esta solução possui as mesmas duas desvantagens da abordagem em (VELHO, 2000), mas ela serve para comparar a abordagem gulosa com a abordagem baseada em grafos de uma forma mais ampla. Por exemplo, além de podermos comparar a qualidade de forma dos quadriláteros (como nos outros dois tipos de emparelhamento), podemos verificar também qual das duas abordagens de conversão gera mais triângulos "isolados".

3.3 Emparelhamentos perfeitos

Seja $G = (V, E)$ um grafo e seja M um emparelhamento máximo em G. Note que M é perfeito se um emparelhamento perfeito em G existir. Sejam $|V|$ e $|E|$ o número de vértices e arestas de G, respectivamente. Um algoritmo em tempo polinomial em $|V|$ (e/ou $|E|$) para encontrar M não era conhecido até o trabalho pioneiro de Jack Edmonds (EDMONDS, 1965b). O algoritmo desenvolvido por Edmonds foi denominado *blossom*[2] e possui complexidade de tempo $\mathcal{O}(|V|^4)$. Este algoritmo explora a importante relação entre emparelhamentos máximos e caminhos aumentantes estabelecida por Claude Berge (Teorema 2.3.8).

O trabalho de Edmonds inspirou uma série de novos algoritmos e implementações computacionais robustas. A grande maioria dos algoritmos que se seguiram ao algoritmo blossom se trata, na verdade, de modificações do algoritmo com o intuito de reduzir sua ordem de complexidade de pior caso. Na década de 1970, Gabow e Lawler mostraram, independentemente, implementações mais cuidadosas do algoritmo blossom que possuem complexidade de tempo $\mathcal{O}(|V|^3)$ (GABOW, 1976; LAWLER, 1976). A partir daí, muitas outras modificações que simplificaram e reduziram a complexidade de tempo do algoritmo foram propostas.

Até recentemente, os algoritmos propostos por Micali e Vazirani (MICALI; VAZIRANI, 1980), Blum (BLUM, 1990) e Gabow e Tarjan (GABOW; TARJAN, 1991), todos com complexidade de tempo $\mathcal{O}(|E| \cdot \sqrt{|V|})$, ainda eram o que havia de melhor em termos de algoritmos determinísticos para o problema de emparelhamento máximo em grafos gerais. Vale salientar que o algoritmo de Blum utiliza uma estratégia distinta daquela usada pelo algoritmo blossom. Em 2003, Fremuth-Paeger e Jungnickel (FREMUTH-PAEGER; JUNGNICKEL, 2003) apresentaram um algoritmo determinístico para o problema de emparelhamento máximo em grafos gerais com complexidade de tempo $\mathcal{O}(|E| \cdot \sqrt{|V|} \log(|V|^2/|E|)/\log|V|)$.

[2]Em português, o algoritmo é denominado de "contração de botão" do inglês *blossom-shrinking*.

Em 2004, um outro algoritmo determinístico com a mesma complexidade foi proposto por Goldberg e Karzanov (GOLDBERG; KARZANOV, 2004). Note que o ganho computacional desses dois algoritmos só é verificado quando o grafo é "denso", ou seja, quando $|E|$ é $\Theta(|V|^2)$.

Também em 2004, Mucha e Sankowski apresentaram um algoritmo aleatorizado do tipo Las Vegas para encontrar M, ou seja, um algoritmo cujo tempo de execução *esperado* é limitado por uma função polinomial no tamanho da entrada (MUCHA; SANKOWSKI, 2004). Mais especificamente, a complexidade de tempo (esperado) do algoritmo é $\mathcal{O}(|V|^{\omega})$, onde ω é o expoente que define a complexidade de tempo do melhor algoritmo para calcular a multiplicação de duas matrizes de ordem $|V|$. Como $\omega < 2.38$ (COPPERSMITH; WINOGRAD, 1987), quando o grafo é "denso", o algoritmo aleatorizado de Mucha e Sankowski se torna uma alternativa bastante atraente, na prática, aos algoritmos de Micali e Vazirani (MICALI; VAZIRANI, 1980), Blum (BLUM, 1990) e Gabow e Tarjan (GABOW; TARJAN, 1991).

Como o grafo dual das triangulações consideradas neste trabalho é 3-regular, temos a relação $2 \cdot |E| = 3 \cdot |V|$, pois $2 \cdot |E| = \sum_{v \in V} \deg(v)$, onde $\deg(v)$ é o grau do vértice v. Então, se usarmos qualquer um dos melhores algoritmos determinísticos para obter um emparelhamento perfeito no grafo dual, resolvemos o problema em tempo $\mathcal{O}(|V|^{1.5})$, pois $|E|$ é $\Theta(|V|)$.

Felizmente, o fato do nosso grafo dual ser 3-regular e sem arestas de corte nos permite utilizar algoritmos determinísticos ainda melhores, que são próprios para este tipo de grafo. Em particular, Biedl, Bose, Demaine e Lubiw desenvolveram um algoritmo com complexidade de tempo $\mathcal{O}(|V| \cdot \log^4 |V|)$ para encontrar emparelhamentos perfeitos em grafos 3-regulares e sem arestas de corte (BIEDL; BOSE; DEMAINE; LUBIW, 2001). No início de 2010, Diks e Stanczyk apresentaram uma simplificação do algoritmo de Biedl e colaboradores que originou um algoritmo com complexidade de tempo $\mathcal{O}(|V| \cdot \log^2 |V|)$ (DIKS; STANCZYK, 2010).

O aspecto mais intrigante do trabalho de Bield e colaboradores é que a estratégia usada para obter o emparelhamento não é a mesma do algoritmo blossom de Edmonds nem de todos os demais algoritmos baseados nele. Em particular, a estratégia adotada foi inspirada em provas antigas para o teorema de Petersen (veja o Teorema 2.3.12). Além disso, no mesmo artigo, Bield, Bose, Demaine e Lubiw também apresentam um algoritmo com complexidade de tempo linear em $|V|$ para encontrar emparelhamentos perfeitos em grafos 3-regulares, sem arestas de corte e *planares*, ou seja, que podem ser "desenhados"

em $\mathbb{E}^2$.

3.4 Emparelhamentos (perfeitos) de custo máximo

Seja $G = (V, E)$ um grafo e seja $c : E \to \mathbb{R}$ uma função custo associada com as arestas do grafo. Então, lembre-se de que o problema do emparelhamento máximo de custo máximo é aquele de encontrar um emparelhamento máximo, M, em G que maximiza a soma $\sum_{e \in M} c(e)$. Note que se o grafo G admitir um emparelhamento perfeito, então M é perfeito.

O trabalho pioneiro de Edmonds também contempla emparelhamentos perfeitos de custo máximo em grafos gerais (EDMONDS, 1965a). Uma implementação simples e direta do algoritmo dado por Edmonds possui complexidade de tempo $\mathcal{O}(|V|^4)$ (PAPADIMITRIOU; STEIGLITZ, 1998). Galil, Micali e Gabow desenvolveram um algoritmo para encontrar o emparelhamento perfeito de custo máximo, M, em tempo $\mathcal{O}(|E| \cdot |V| \cdot \log |V|)$ (GALIL; MICALI; GABOW, 1986). Esta complexidade foi, em seguida, reduzida para $\mathcal{O}(|V| \cdot (|E| \cdot \log \log \log_{\max\{|E|/|V|,2\}} |V| + |V| \cdot \log |V|))$ em um algoritmo criado por Gabow, Galil e Spencer (GABOW; GALIL; SPENCER, 1989). Finalmente, Gabow desenvolveu um outro algoritmo de complexidade de tempo $\mathcal{O}(|V| \cdot (|E| + |V| \cdot \log |V|))$, que é, atualmente, o melhor limite superior definido em função de $|V|$ e $|E|$ de uma solução para o problema (GABOW, 1990).

Paralelamente ao desenvolvimento dos algoritmos acima, muitos se dedicaram a implementações robustas e otimizadas dos algoritmos para emparelhamentos perfeitos de custo máximo, inclusive o próprio Edmonds (EDMONDS; JOHNSON; LOCKHART, 1969). As implementações dos algoritmos baseados no algoritmo blossom de Edmonds foram denominadas `BLOSSOM I`, `BLOSSOM II` e assim por diante. As implementações mais antigas se baseiam em estruturas de dados simples. Até 2002, a implementação tida como mais rápida e extremamente eficiente na prática, denominada `BLOSSOM IV`, foi aquela desenvolvida por Cook e Rohe (COOK; ROHE, 1999). A complexidade de tempo da `BLOSSOM IV` é $\mathcal{O}(|V|^2 \cdot |E|)$.

Em 2002, Mehlhorn e Schäfer usaram estruturas de dados mais sofisticadas do que as utilizadas por implementações anteriores e implementaram um algoritmo que possui melhor desempenho do que o de Cook e Rohe em várias instâncias práticas do problema (MEHLHORN; SCHÄFER, 2002). Finalmente, em 2009, Vladimir Kolmogorov imple-

mentou uma nova versão[3] do algoritmo blossom, denominada `BLOSSOM V`, que combina as melhorias de código usadas por Cook e Rohe e também por Mehlhorn e Schäfer (KOLMOGOROV, 2009).

O algoritmo blossom, de Edmonds, para encontrar um emparelhamento perfeito de custo máximo num dado grafo é baseado em programação linear. Edmonds formulou um simples programa linear (EDMONDS, 1965a), detalhado no Capítulo 5, que serviu de pontapé inicial para o desenvolvimento de algoritmos polinomiais para resolver o problema. A implementação do algoritmo blossom dada por Mehlhorn e Schäfer se baseia nesta formulação e, ao contrário de implementações anteriores e do `BLOSSOM V`, ela pode ser usada tanto para calcular um emparelhamento perfeito de custo máximo quanto um emparelhamento de custo máximo (MEHLHORN; SCHÄFER, 2002).

Até onde saibamos, não existe nenhum algoritmo para calcular emparelhamentos perfeitos de custo máximo em grafos 3-regulares e sem arestas de corte que não possa ser usado para grafos mais gerais. Como a implementação dada por Mehlhorn e Schäfer (MEHLHORN; SCHÄFER, 2002) e a do `BLOSSOM V` (KOLMOGOROV, 2009) possuem complexidade de tempo $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$, o melhor que temos ao nosso dispor, para calcular emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo nos grafos duais das triangulações de entrada do problema-alvo, são algoritmos de complexidade $\mathcal{O}(|V|^2 \cdot \log |V|)$, já que no nosso caso $|E|$ é $\Theta(|V|)$.

3.5 O algoritmo `TriQuad()`

Como mencionado no Capítulo 1, Luiz Velho desenvolveu um algoritmo que converte uma triangulação $\mathcal{T}$ de uma superfície em $\mathbb{E}^3$ numa quadrilaterização, $\mathcal{Q}$, da mesma superfície (VELHO, 2000). Para tanto, o algoritmo utiliza uma estratégia gulosa que emparelha pares de triângulos que compartilham uma aresta para formar quadriláteros. Os pares de triângulos são escolhidos de acordo com a heurística da aresta mais longa, isto é, os triângulos que compartilham a aresta mais longa "disponível" são emparelhados para formar um quadrilátero de $\mathcal{Q}$. Uma vez que dois triângulos adjacentes à aresta e de $\mathcal{T}$ sejam emparelhados para formar um quadrilátero com diagonal e, a aresta e é removida da subdivisão atual da superfície e as demais arestas dos dois triângulos se tornam indisponíveis.

Para efetuar o emparelhamento descrito acima, Velho utilizou uma fila de prioridades que armazena as arestas de $\mathcal{T}$ usando o comprimento delas como prioridade. Com o uso

[3] Veja `http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html`

desta estrutura de dados, uma versão iterativa do algoritmo se torna bastante intuitiva e simples. Em cada iteração, removemos uma aresta da fila de prioridades. Se a aresta estiver disponível, emparelhamos os dois triângulos adjacentes a ela e marcamos as demais arestas dos dois triângulos (ou seja, do quadrilátero obtido) como indisponíveis. O pseudocódigo deste procedimento de "conversão" de subdivisões, denominado `TriQuad()`, é dado no Algoritmo 3.1.

A complexidade de tempo do algoritmo `TriQuad`() é $\mathcal{O}(|E| \cdot \log |E|)$, pois H pode ser construída em tempo linear em $|E|$, o laço é repetido $|E|$ vezes e cada repetição consome $\mathcal{O}(\log |E|)$ unidades de tempo. Usando a característica de Euler-Poincaré, concluímos que $|E|$ é $\Theta(|V|)$ para qualquer triangulação de uma superfície. Então, podemos expressar a complexidade do algoritmo `TriQuad`(), em termos do número de vértices, como $\mathcal{O}(|V| \cdot \log |V|)$. É importante lembrar que a subdivisão, $\mathcal{Q}$, produzida pelo algoritmo `TriQuad`() pode conter triângulos também. Quando este é o caso, a subdivisão $\mathcal{Q}$ não é, por definição, uma quadrilaterização e o algoritmo em (VELHO, 2000) subdivide todos os triângulos e quadriláteros pertencentes a $\mathcal{Q}$ para obter uma quadrilaterização, como ilustrado pela Figura 1.3.

Algoritmo 3.1 `TriQuad`($\mathcal{T}$)

Entrada: uma triangulação $\mathcal{T}$ de uma superfície S em $\mathbb{E}^3$
Saída: um conjunto, $\mathcal{Q}$, de triângulos e quadriláteros

```
1: Q ← T
2: seja E o conjunto de arestas de Q
3: construa a fila de prioridades, H, com as arestas de E ordenadas por comprimento
4: enquanto H ≠ ∅ faça
5:    remova a aresta mais longa, e, de H
6:    se e está disponível então
7:       remova e de Q
8:       marque as arestas do quadrilátero obtido como indisponíveis
9:    fim se
10: fim enquanto
11: devolva Q
```

4 Emparelhamentos Perfeitos

Este capítulo faz uma descrição detalhada do algoritmo que utilizamos para calcular emparelhamentos perfeitos nos grafos duais das triangulações de superfícies convertidas em quadrilaterizações. A Seção 4.1 justifica a escolha do algoritmo que implementamos e descrevemos neste capítulo. A Seção 4.2 apresenta um algoritmo para o problema do emparelhamento máximo em grafos bipartidos, que serve como motivação para o algoritmo para grafos mais gerais. A Seção 4.3 introduz as modificações necessárias no algoritmo do caso bipartido, para adaptá-lo a grafos gerais. A Seção 4.4 descreve o algoritmo de Gabow para emparelhamento máximo em grafos gerais, implementado neste trabalho, que consiste numa versão de menor complexidade de tempo do algoritmo apresentado na Seção 4.3.

4.1 Considerações iniciais

Como visto no Capítulo 3, o grafo dual de qualquer triangulação das superfícies consideradas neste trabalho é 3-regular e sem aresta de corte. Isto nos possibilita calcular emparelhamentos perfeitos neste tipo de grafo usando o algoritmo proposto recentemente por Diks e Stanczyk (DIKS; STANCZYK, 2010), que tem menor complexidade de tempo do que os algoritmos existentes para grafos mais gerais. Embora este algoritmo seja a melhor escolha de que dispomos, o algoritmo implementado em nossa solução foi o algoritmo de Gabow (GABOW, 1976), que tem complexidade bem mais alta do que o de Diks e Stanczyk.

A escolha de um algoritmo de complexidade mais alta pode causar estranheza, mas esta escolha se deveu a um erro de planejamento no cronograma do trabalho. O plano inicial era estudar e implementar pelo menos três algoritmos para calcular emparelhamentos perfeitos em grafos gerais, incluindo o algoritmo de Diks e Stanczyk (DIKS; STANCZYK, 2010), além de estudar e implementar algoritmos para o cálculo de emparelhamentos (perfeitos) de custo máximo. Quando percebemos que não cumpriríamos o cronograma,

já havíamos estudado o algoritmo de Gabow, entre outros, mas não havíamos estudado o algoritmo de Diks e Stanczyk e nem havíamos iniciado nosso estudo de algoritmos para cálculo de emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo.

É importante ressaltar que a escolha de um algoritmo de menor ou maior complexidade é pouco relevante para os objetivos deste trabalho, pois o que desejamos é comparar, à luz de uma métrica de qualidade, duas abordagens para converter triangulações de superfícies em quadrilaterizações. Logo, o que mais importa aqui é a qualidade da quadrilaterização produzida pelo tipo de emparelhamento usado (independentemente do algoritmo específico usado para obtê-lo). Além disso, o algoritmo de Gabow é elegante e simples e nos oportunizou a enriquecedora experiência de estudar a teoria de contração de botões (EDMONDS, 1965b).

4.2 Grafos bipartidos

Como forma de motivar e justificar a teoria de contração de botões, apresentamos um algoritmo para cálculo de emparelhamentos de cardinalidade máxima em grafos bipartidos. Mais adiante, mostraremos que a tentativa de estender este algoritmo para grafos mais gerais falha. Edmonds percebeu a causa e desenvolveu a teoria usada em muitos algoritmos para cálculo de emparelhamentos de cardinalidade máxima em grafos gerais (EDMONDS, 1965b).

Definição 4.2.1. *Seja $G = (V, E)$ um grafo simples, onde V é o conjunto de vértices e E é o conjunto de arestas de G. Dizemos que G é* bipartido *se, e somente se, V puder ser particionado em dois conjuntos, U e W, tal que toda aresta em E seja incidente a um vértice em U e a outro em W; ou seja, se, e somente se, todo ciclo em G tem comprimento par.*

A operação básica de inúmeros algoritmos de emparelhamentos em grafos (bipartidos ou não) é a *busca por caminhos aumentantes* (veja a Definição 2.3.7). Esta operação é motivada pelo Teorema 2.3.8. Este Teorema sugere que podemos começar a busca por um emparelhamento máximo a partir de um emparelhamento vazio (ou um emparelhamento qualquer), M. Em seguida, procuraremos por caminhos aumentantes com respeito a M. A cada caminho aumentante, P, encontrado, o emparelhamento atual M é *aumentado* ao longo de P. Este emparelhamento resultante do *aumento* possui exatamente uma aresta a mais do que o emparelhamento anterior. Finalmente, a busca por caminhos aumentantes

é reiniciada, mas, desta vez, com respeito ao novo emparelhamento obtido. Este processo é repetido até que não seja mais possível encontrar nenhum caminho aumentante com respeito ao emparelhamento atual. Neste momento, o Teorema 2.3.8 nos permite afirmar que o emparelhamento atual é um emparelhamento máximo no grafo em questão (veja Algoritmo 4.1).

Algoritmo 4.1 `EmparCardMax`(G)

Entrada: um grafo $G = (V, E)$ simples (bipartido ou não)
Saída: um emparelhamento M em G

```
1: M ← ∅
2: achou ← falso
3: repetir
4:    se houver um caminho aumentante P, com respeito a M então
5:       aumente M ao longo de P
6:    senão
7:       achou ← verdadeiro
8:    fim se
9: até que achou
10: devolva M
```

Em um grafo bipartido, a busca por um caminho aumentante pode ser realizada por uma simples modificação de uma busca em largura. De fato, seja $G = (V, E)$ um grafo bipartido e seja $\{U, W\}$ uma partição de V. Para simplificar nossa exposição, assuma que os vértices de U representam "meninos" e os vértices de W representam "meninas". Denote por M o emparelhamento atual. Note que M pode ser o conjunto vazio. A busca por um caminho aumentante em G com respeito a M pode ser iniciada em um menino ou menina livre. Assuma que a busca seja iniciada em um menino livre. Como os vértices extremos de um caminho aumentante são vértices livres, a busca deve terminar em uma menina livre.

A busca por um caminho aumentante em G com respeito a M é uma busca em largura que gera um caminho simples (isto é, o caminho não pode ser um ciclo). Denote por V_i os vértices encontrados no i-ésimo passo da busca. Como G é bipartido, temos que V_i é um conjunto apenas de meninos ou apenas de meninas. A busca se inicia pelo conjunto V_0, que consiste de todos os meninos livres com respeito a M. O primeiro passo da busca cria o conjunto V_1, que consiste de todas as meninas adjacentes a algum menino de V_0. O segundo passo da busca cria o conjunto V_2, que consiste de todos os meninos emparelhados, com respeito a M, a alguma menina de V_1. De forma geral, temos a seguinte regra: se V_i é um conjunto de meninos, então V_{i+1} é o conjunto de meninas adjacentes a algum menino em V_i e que ainda não foram visitadas pela busca; caso contrário, V_i é um conjunto de

meninas e V_{i+1} é o conjunto de meninos emparelhados a alguma menina em V_i. A busca se encerra tão logo alguma menina livre seja encontrada ou quando o conjunto de meninas, V_{i+1}, for o conjunto vazio. No primeiro caso, um caminho aumentante em G com respeito a M é encontrado. No segundo caso, não há nenhum caminho aumentante em G com respeito a M.

Para ilustrar o algoritmo, considere o grafo $G = (V, E)$ bipartido da Figura 4.1. O conjunto $V = U \cup W$ é tal que $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ e $W = \{w_1, w_2, w_3, w_4, w_5\}$. Suponha que $M = \{u_3w_4, u_4w_5, u_6w_3\}$ seja o emparelhamento atual. Então, temos que o conjunto, V_0, de meninos livres de G com respeito a M é igual a $\{u_1, u_2, u_5\}$. Se executarmos a busca descrita no parágrafo anterior, obteremos os conjuntos $V_1 = \{w_3, w_4\}$, $V_2 = \{u_3, u_6\}$ e $V_3 = \{w_1, w_2, w_5\}$. Como V_3 contém uma menina livre, a busca se encerra no passo $i = 2$. Todos os caminhos (aumentantes ou não) encontrados pela busca em largura modificada são ilustrados na Figura 4.2. Note que há seis caminhos aumentantes (que são não-disjuntos dois a dois). Nós podemos escolher qualquer um deles para aumentar M.

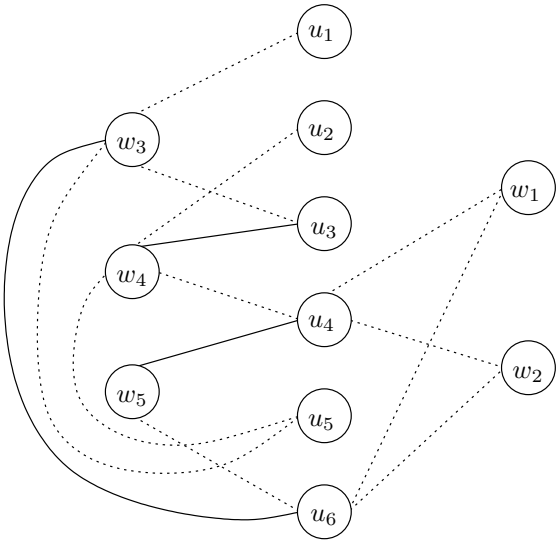


Figura 4.1: Um grafo bipartido G e um emparelhamento M em G (arestas sólidas).

Suponha que o algoritmo retorne o caminho aumentante $P = [u_1, w_3, u_6, w_1]$ com respeito a M. Se aumentarmos M ao longo de P, obtemos um novo emparelhamento, $M = \{u_1w_3, u_3w_4, u_4w_5, u_6w_1\}$ (veja a Figura 4.3). Se iniciarmos uma nova busca por caminhos aumentantes, o conjunto de meninos livres é agora igual a $V_0 = \{u_2, u_5\}$. A partir deste conjunto, obtemos $V_1 = \{w_3, w_4\}$, $V_2 = \{u_1, u_3\}$ e $V_3 = \emptyset$. O fato do conjunto, V_3, de meninas ser vazio faz com que a busca seja finalizada e que nenhum caminho aumentante seja retornado. Quando a busca não encontra nenhum caminho aumentante

em G com respeito a M, podemos mostrar que tais caminhos não existem (BONDY; MURTY, 2007).

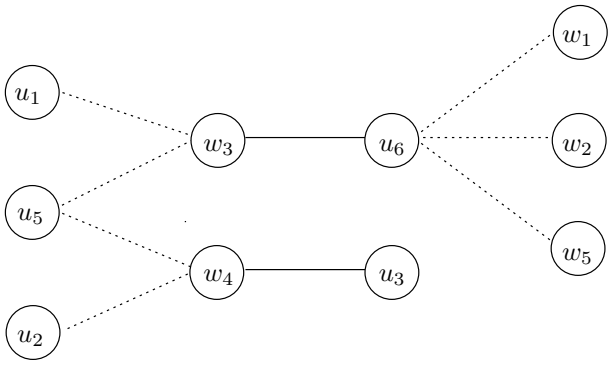


Figura 4.2: Caminhos aumentantes no grafo da Figura 4.1 com respeito a M.

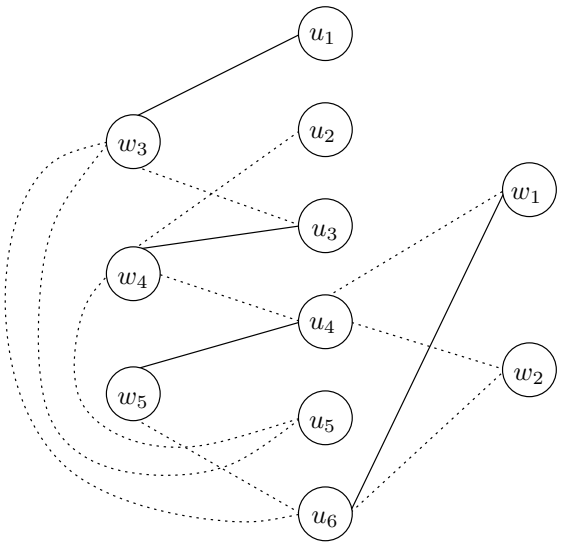


Figura 4.3: Um emparelhamento de cardinalidade máxima no grafo da Figura 4.1.

4.3 Grafos gerais

O Teorema 2.3.8 vale para grafos gerais, o que nos permite adotar a mesma estratégia de cálculo de emparelhamento em grafos bipartidos para o caso geral: procurar sucessivamente por caminhos aumentantes com respeito ao emparelhamento atual, até que não seja mais possível aumentar a cardinalidade do emparelhamento. Esta é, de fato, a operação básica utilizada pelos algoritmos mais conhecidos de emparelhamento de cardinalidade máxima.

Infelizmente, a busca em largura modificada que usamos para encontrar caminhos aumentantes em grafos bipartidos não funciona para qualquer grafo geral. Em particular, a busca só funciona para grafos bipartidos porque esses grafos não possuem ciclos de comprimento ímpar. Esta propriedade nos permite ignorar qualquer ciclo encontrado durante a busca sem excluir nnehum caminho aumentante. Em grafos gerais, nós não podemos ignorar ciclos no grafo, como demonstrado pelos dois exemplos discutidos a seguir[1].

Seja $G = (V, E)$ o grafo (não-bipartido) da Figura 4.4. Considere o emparelhamento $M = \{u_2u_3, u_4u_5\}$ em G cujas arestas estão desenhadas como linhas sólidas na Figura 4.4. Se tentarmos aplicar a estratégia de busca por caminhos aumentantes para grafos bipartidos, obteremos $V_0 = \{u_1\}$, $V_1 = \{u_2\}$, $V_2 = \{u_3\}$, $V_3 = \{u_4, u_5\}$ e $V_4 = \{u_5, u_4\}$. Neste momento, encontramos uma anomalia: os vértices u_4 e u_5 já estão em V_3, ou seja, eles representavam "meninas" e, agora, representam "meninos". Se não permitirmos que um mesmo vértice represente "menino" e "menina" (em passos pares e ímpares da busca), teremos $V_4 = \emptyset$ e, logo, não encontraremos o único caminho aumentante em G com respeito a M:

$$[u_1, u_2, u_3, u_5, u_4, u_6] .$$

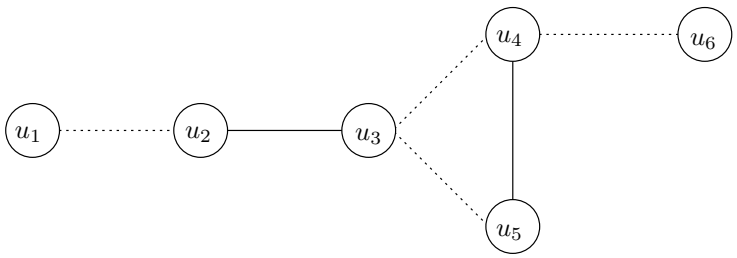


Figura 4.4: Um grafo não-bipartido no qual um caminho aumentante não foi encontrado.

O exemplo anterior mostra que devemos permitir que um mesmo vértice represente "menino" e "menina", em passos distintos da busca, a fim de garantir que um caminho aumentante sempre seja encontrado. Infelizmente, esta "permissão" causa um outro problema. Seja $G = (V, E)$ o grafo (não-bipartido) da Figura 4.5. Considere o emparelhamento $M = \{u_2u_3, u_4u_5\}$ em G cujas arestas estão desenhadas como linhas sólidas na Figura 4.5. Se aplicarmos a estratégia de busca por caminhos aumentantes para grafos bipartidos e permitirmos que um mesmo vértice represente "menino" e "menina", em passos distintos da busca, obteremos $V_0 = \{u_1, u_6\}$, $V_1 = \{u_2\}$, $V_2 = \{u_3\}$, $V_3 = \{u_4, u_5\}$, $V_4 = \{u_5, u_4\}$, $V_5 = \{u_3\}$ e $V_6 = \{u_1, u_2\}$. No passo $i = 5$, os vértices u_1 e u_6 representam "meninas" e

[1]Retirados de notas de aula da disciplina CS294-5, 2006, da U. C. Berkeley, ministrado por Richard Karp.

ambos estão livres. Logo, o algoritmo deveria retornar um dos "caminhos" aumentantes, digamos

$$[u_1, u_2, u_3, u_5, u_4, u_3, u_2, u_6]\,.$$

O problema é que a seqüência acima não define um caminho *simples* de u_1 a u_6. De fato, o grafo G da Figura 4.5 não possui nenhum caminho aumentante com respeito a M.

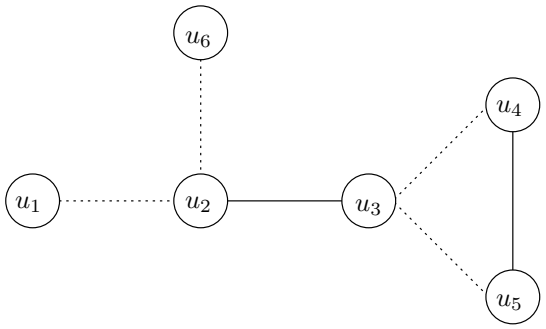


Figura 4.5: Um grafo não-bipartido sem caminhos aumentantes.

Como os exemplos acima mostram, a busca em largura modificada que usamos para encontrar caminhos aumentantes em grafos bipartidos não pode ser utilizada para o mesmo fim em grafos gerais. Uma busca diferente deve ser utilizada, como veremos a seguir:

4.3.1 O algoritmo de Edmonds

Jack Edmonds percebeu que os problemas ilustrados pelos dois exemplos vistos anteriormente decorrem da possibilidade de ocorrência de ciclos com um número ímpar de nós em grafos gerais, os quais não estão presentes em grafos bipartidos (EDMONDS, 1965b). Mais especificamente, Edmonds notou que o problema se deve à ocorrência de *botões*.

Definição 4.3.1. *Um ciclo, B, com um número ímpar de nós, em um grafo G é um* botão*, com respeito a um emparelhamento M em G, se M é um emparelhamento máximo em B; isto é, se todos os vértices em B são ocupados com relação a M, exceto um deles, denominado* base *de B.*

A Figura 4.6 mostra um botão com relação ao emparelhamento definido pelas arestas sólidas do grafo: o ciclo formado pelos vértices u_3, u_4, u_5, u_6 e u_7. O vértice u_3 é a base do botão. Uma propriedade importante é que, em um botão, existe um caminho alternante da sua base para cada um de seus outros vértices, o qual termina com uma aresta do emparelhamento.

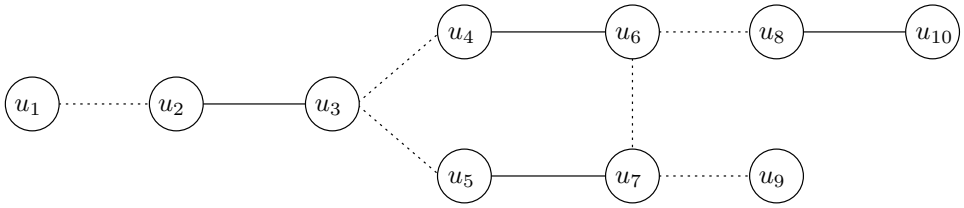


Figura 4.6: Um botão em um grafo (arestas sólidas pertencem ao emparelhamento).

A ideia central do algoritmo de Edmonds para encontrar caminhos aumentantes em grafos gerais é a de se livrar dos botões, através de uma operação denominada *contração de botão*.

Definição 4.3.2. *Seja B um botão em um grafo $G = (V, E)$. O grafo resultante da* contração *de B em G é*

$$G/B = (V/B, E/B)\,,$$

onde V/B consiste de V com todos os vértices de B omitidos e um novo vértice, v_B, que substitui B, e E/B consiste de E com todas as arestas com ambas as extremidades em B omitidas e todas as arestas (v, u), tais que u está em B e v não está em B, substituídas por (v, v_B).

A Figura 4.7 ilustra o grafo resultante da contração do botão B do grafo da Figura 4.6.

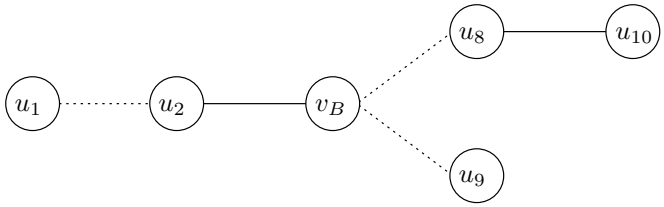


Figura 4.7: O grafo resultante da contração do botão B do grafo da Figura 4.6.

Seja $G = (V, E)$ um grafo (bipartido ou não) e seja M um emparelhamento em G. O algoritmo de Edmonds procura por um caminho aumentante em G com respeito a M a partir de qualquer vértice livre de G, digamos u, de maneira similar à do caso bipartido. Em particular, o algoritmo utiliza uma busca em largura para construir uma árvore, chamada de *árvore alternante*, enraizada em u. A árvore construída tem a propriedade de que todos os caminhos que conectam o vértice u a qualquer outro vértice da árvore são *caminhos alternantes*. O vértice raiz, u, e todos os vértices a uma distância par de u são denominados *externos*, enquanto vértices a uma distância ímpar de u são denominados *internos*.

No início da busca em largura, a árvore alternante consiste apenas de seu vértice raiz, u. Com o decorrer da busca, a árvore alternante é acrescida de novas arestas e vértices. A busca considera um vértice externo da árvore por vez, começando com a raiz, u. Se v é o vértice externo atualmente considerado pela busca, a operação de crescimento da árvore consiste em considerar cada vértice w de V tal que vw é uma aresta de G. Há apenas duas possibilidades para w: (1) ele não pertence à árvore ou (2) ele pertence à árvore. Se (1) ocorrer, então o algoritmo verifica se w é livre ou ocupado com respeito a M. Se w é livre, a busca acaba de encontrar um caminho *aumentante* de u para w. Caso contrário, w foi emparelhado com um vértice x de V e a árvore é acrescida das arestas vw e wx. O vértice w é designado interno, enquanto o vértice x, externo. Se (2) ocorrer, o algoritmo verifica se w é externo ou interno. Se w for externo, a adição da aresta vw à árvore define um botão. Caso contrário, o vértice w é ignorado e o próximo vértice adjacente a v é considerado.

Quando um botão, B, é encontrado (isto é, quando w pertence à árvore e é um vértice externo), o algoritmo o contrai. Para isto, o algoritmo deve determinar todos os vértices de B. Isto é feito através de um algoritmo que, primeiro, encontra o ancestral comum mais próximo dos vértices v e w. Este ancestral é a base de B. Em seguida, todos os vértices do caminho único que conecta v (resp. w) à base de B são encontrados. Esses vértices, juntamente com v, w e a base de B, formam o conjunto de vértices de B. A partir daí, todos os vértices de B e todas as arestas que conectam qualquer vértice de G a um vértice de B são removidas. Em seguida, o novo vértice v_B é adicionado ao grafo e conectado a todos os vértices restantes que se conectavam a algum vértice de B. Finalmente, o vértice v_B é designado como externo. Após a operação de contração de B, a busca por um caminho aumentante segue com um outro vértice exterior da árvore que ainda não tenha sido considerado.

Considere o grafo da Figura 4.6 e o emparelhamento definido pelas arestas sólidas da figura. Suponha que iniciemos a busca pelo vértice u_1. De acordo com o algoritmo, a árvore é iniciada apenas com o vértice u_1. Usando a busca em largura, encontramos o vértice u_2. Como ele não pertence à árvore e está ocupado, as arestas u_1u_2 e u_2u_3 são adicionadas à árvore. O vértice u_2 passa a ser interno e, u_3, externo. A busca segue a partir do próximo vértice externo, u_3, pois u_1 não possui nenhum outro vértice adjacente a ele. Os vértices adjacentes a u_3 são u_4 e u_5. Ambos estão ocupados e não pertencem à árvore. Logo, as arestas u_3u_4, u_4u_6, u_3u_5 e u_5u_7 são adicionadas à árvore, os vértices u_4 e u_5 são designados como internos e os vértices u_6 e u_7 são designados como externos. A Figura 4.8(a) ilustra a árvore construída até então. O próximo vértice considerado pela

busca em largura é u_6 ou u_7, pois u_3 não possui nenhum outro vértice adjacente a ele além de u_2, u_4 e u_5.

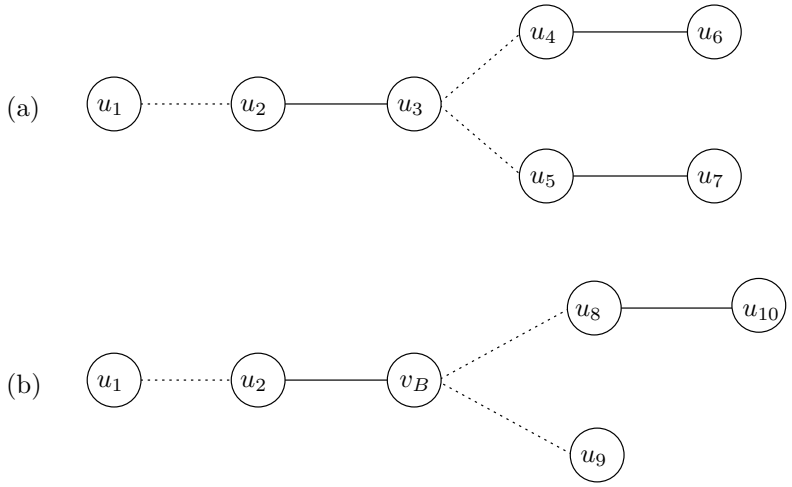


Figura 4.8: Árvores construídas pelo algoritmo de Edmonds no grafo da Figura 4.6.

Suponha que u_6 seja o próximo vértice externo considerado pela busca em largura. Quando o vértice u_7 é visitado, o algoritmo verifica que ele pertence à árvore e é um vértice externo, o que implica que um botão foi encontrado e deve ser contraído. O grafo resultante da contração deste botão está na Figura 4.7. A árvore resultante possui apenas os vértices u_1 e u_2 e o novo vértice v_B. A busca continua a partir do próximo vértice externo da árvore, que só pode ser v_B, pois ele é o único vértice externo que ainda não foi considerado. A partir daí, a árvore é acrescida das arestas v_Bu_8 e u_8u_{10} (veja a Figura 4.8(b)), pois u_8 não pertence à árvore e é um vértice ocupado. Em seguida, a árvore é acrescida da aresta v_Bu_9. Como u_9 é livre, o algoritmo acaba de encontrar um caminho aumentante no grafo da Figura 4.7:

$$[u_1, u_2, v_B, u_9]\,.$$

Quando um caminho aumentante é encontrado, a busca em largura é encerrrada. No entanto, note que o caminho aumentante foi encontrado no grafo "reduzido" e não no grafo original. No entanto, o seguinte teorema permite que este caminho aumentante seja transformado em um caminho aumentante no grafo original mediante uma *expansão* do botão contraído:

Teorema 4.3.3. *Suponha que, ao buscarmos um caminho aumentante em um grafo G, com respeito a um emparelhamento M, a partir de um vértice u, descobrimos um botão, B. Então, existe um caminho aumentante em G, com respeito a M, partindo de u se, e somente se, existe um caminho aumentante em G/B, com respeito a M/B, partindo de u (ou de v_B se u for a base de B).*

Demonstração. Veja (EDMONDS, 1965b) para uma demonstração detalhada. □

Como garantido pelo Teorema 4.3.3, ao expandirmos o botão que contraimos anteriormente, verificamos que também existe um caminho aumentante no grafo da Figura 4.6, com respeito ao emparelhamento descrito pelas arestas sólidas da figura e partindo do vértice u_1:

$$[u_1, u_2, u_3, u_5, u_7, u_9] .$$

Para determinar o caminho aumentante acima, o algoritmo expande o botão e encontra o caminho único na árvore, de comprimento par, que conecta a base do botão (no exemplo, o vértice u_3) ao vértice de "saída" do botão (no exemplo, o vértice u_7) que está conectado ao vértice que segue v_B no caminho aumentante no grafo "reduzido" (no exemplo, o vértice u_9).

Depois que um caminho aumentante é encontrado no grafo original, o algoritmo de Edmonds aumenta o emparelhamento ao longo do caminho e escolhe um outro vértice livre para construir uma árvore alternante a partir dele. A construção de uma árvore termina quando nenhum botão for mais encontrado e a árvore não puder mais ser expandida. Neste caso, todos os vértices da árvore podem ser descartados, pois eles jamais participarão de qualquer caminho aumentante com respeito ao emparelhamento atual. O algoritmo termina quando todos os vértices livres, com respeito ao emparelhamento atual, tiverem sido descartados (isto é, já tiverem sido usados como raiz para construir árvores alternantes).

Uma implementação ingênua do algoritmo de Edmonds resolve o problema do emparelhamento máximo em tempo $\mathcal{O}(|V|^4)$. No entanto, várias outras modificações do algoritmo foram propostas com o intuito de reduzir sua complexidade, como a apresentada na seção a seguir. É importante salientar que o algoritmo de Edmonds foi o primeiro algoritmo, com complexidade de tempo polinomial, para o problema do emparelhamento de cardinalidade máxima.

4.4 O algoritmo de Gabow

O algoritmo utilizado neste trabalho para encontrar um emparelhamento perfeito no grafo dual de uma triangulação de uma superfície em $\mathbb{R}^3$ é o proposto por Harold Gabow em (GABOW, 1976). O algoritmo de Gabow encontra emparelhamentos de cardinalidade máxima em grafos gerais. O algoritmo decorre de uma implementação cuidadosa das ideias de Edmonds. A abordagem proposta Gabow resolve o problema em $\mathcal{O}(|V|^3)$ unidades de tempo.

Alguns trabalhos anteriores já haviam reduzido a complexidade do algoritmo de Edmonds para $\mathcal{O}(|V|^3)$ (veja (WITZGALL; ZAHN, 1965; BALINSKI, 1967)), mas não traziam a generalidade inerente ao algoritmo de Edmonds, tornando-os de difícil adaptação para a resolução de outros problemas relacionados. O algoritmo de Gabow pode ser generalizado para resolver o problema do emparelhamento de custo máximo, além de proporcionar uma implementação relativamente simples e elegante para o emparelhamento de cardinalidade máxima.

O ganho em eficiência obtido pelo algoritmo de Gabow se deve à utilização de um sistema de ponteiros para codificar a estrutura dos caminhos aumentantes. Esta estratégia permite eliminar os processos de contração e expansão de botões *de forma explícita*, presentes no algoritmo de Edmonds. Para a descrição do algoritmo são necessárias algumas notações:

Definição 4.4.1. *Sejam G um grafo simples e $P = [v_1, v_2, ..., v_n]$ um caminho em G. Então, o* reverso *de P, denotado por P_r, é o caminho $[v_n, ..., v_2, v_1]$ em G, denotado também por*

$$[v_1, v_2, ..., v_n]_r \, .$$

Definição 4.4.2. *Seja $G = (V, E)$ um grafo simples e sejam $P = [v_1, v_2, ..., v_n]$ e $Q = [w_1, w_2, ..., w_m]$ caminhos em G tais que $v_n w_1 \in E$. Então, a* concatenação *de P e Q, denotada por*

$$P * Q \, ,$$

é o caminho

$$[v_1, v_2, ..., v_n, w_1, w_2, ..., w_m]$$

em G.

Como de praxe, este algoritmo baseia-se no Teorema 2.3.8, mantendo a mesma ideia básica do de Edmonds. Similarmente, o algoritmo inicia com um emparelhamento vazio

(ou qualquer emparelhamento) e procura, de forma iterativa, por caminhos aumentantes com respeito ao emparelhamento atual, até que não seja mais possível aumentar a cardinalidade deste emparelhamento. Quando isto ocorre, o algoritmo devolve o emparelhamento atual, que é máximo.

Assim como no algoritmo de Edmonds, a busca por caminhos aumentantes é, portanto, a operação crucial do algoritmo de Gabow. Em linhas gerais, a busca começa por um vértice livre, u, e consiste em processar arestas do grafo com o objetivo de encontrar caminhos alternantes de outros vértices até u. Quando um caminho aumentante é encontrado, o emparelhamento atual é aumentado ao longo deste caminho, como no algoritmo de Edmonds.

Um vértice v é dito *rotulado* quando a busca encontra um caminho *alternante* que vai de v até u. Note que este caminho tem de começar com uma aresta do emparelhamento atual (incidente em v), pois o vértice u é livre. Denotamos tal caminho por $P(v, u)$. Se, durante a busca, um vértice livre, w, com $vw \in E$ e $w \neq u$, for encontrado, um caminho aumentante é descoberto. Este caminho é $w * P(v, u)$. Por outro lado, caso tal vértice w não exista e nenhum outro vértice possa se tornar rotulado, podemos concluir que não há caminhos aumentantes de outros vértices para o vértice u. Neste momento, a busca a partir do vértice u é encerrada e uma nova busca é iniciada a partir de um outro vértice livre, que ainda não tenha sido usado para iniciar uma busca. Se tal vértice não existir, o algoritmo termina e devolve o emparelhamento atual, que é garantido ter cardinalidade máxima.

Para manter as informações produzidas pela busca, o algoritmo de Gabow utiliza duas tabelas e numera os vértices e arestas do grafo, $G = (V, E)$, dado como entrada para o algoritmo, com o intuito de indexar as entradas dessas tabelas. Os vértices em V são numerados, obrigatoriamente, de 1 a $|V|$, mas o algoritmo faz uso do número 0 para o índice de um vértice fictício[2]. As arestas em $|E|$ são numeradas de $|V| + 1$ a $|V| + |E|$. Uma das principais finalidades das tabelas usadas pelo algoritmo é a de permitir que ele calcule todos os vértices de um caminho alternante, $P(v, u)$, de v a u encontrado pela busca.

A primeira tabela é denotada por L. Esta tabela possui $|V|$ entradas. Denotamos por $L(v)$ a entrada de L correspondente ao vértice de índice v, ou simplesmente vértice v, de G. O algoritmo insere informações na entrada $L(v)$ se, e somente se, o vértice v é visitado durante a busca por caminhos alternantes conectando vértices do grafo ao vértice livre u.

[2]Uma espécie de sentinela.

Para cada $v \in \{1, \ldots, |V|\}$, a entrada $L(v)$ de L é um número de 1 a $\max\{|V|, |E|\}$, acompanhado de dois outros números que designam (1) o vértice emparelhado a v, se algum, e (2) o tipo da entrada. A Figura 4.9 ilustra um grafo no qual a busca do algoritmo de Gabow é realizada. Os vértices rotulados são desenhados como círculos sólidos, enquanto os não-rotulados, como círculos vazados. A Tabela 4.1 mostra os valores armazenados em L durante a busca por um caminho aumentante conectando os vértices do grafo ao vértice livre, 13.

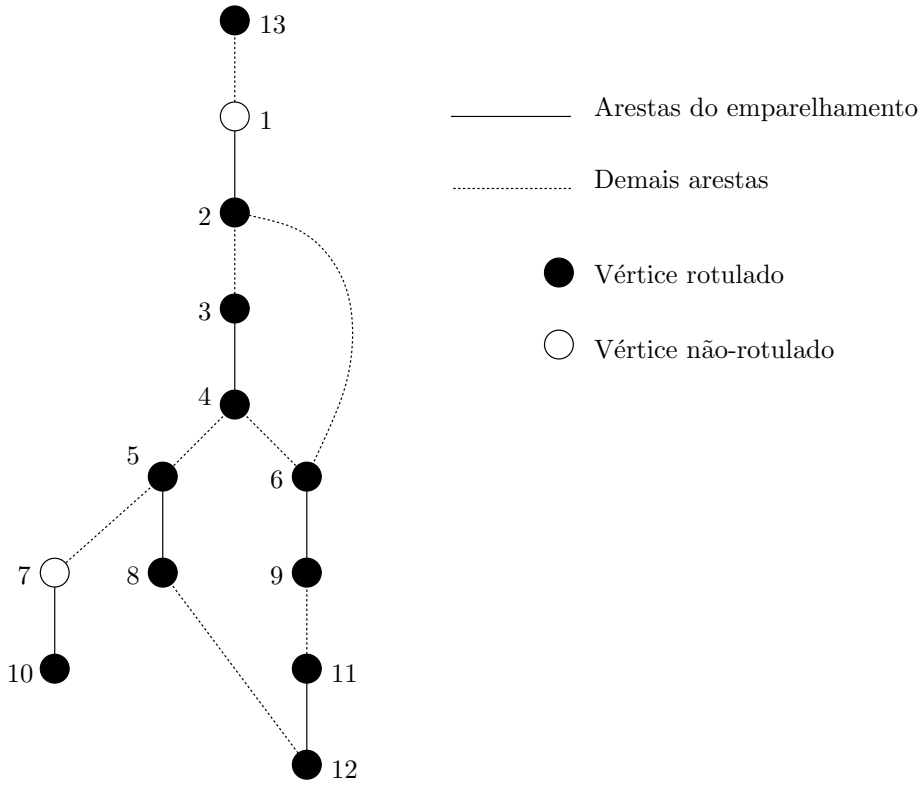


Figura 4.9: Um grafo no qual a busca do algoritmo de Gabow é executada.

Cada vértice do grafo G é classificado, durante a busca, como rotulado ou não. Um vértice, v, é classificado como rotulado se ele é o vértice inicial da busca, isto é, se $v = u$, se a busca encontrou um caminho alternante do vértice, v, com $v \neq u$, para o vértice u que começa com uma aresta do emparelhamento incidente em v. Logo, no início da busca, apenas u é rotulado e todos os demais vértices do grafo são não-rotulados. Quando um vértice v se torna rotulado durante a busca, o algoritmo associa um valor à entrada de $L(v)$.

O valor associado à entrada $L(v)$ depende do tipo do vértice rotulado, que pode ser degenerado, ponteiro ou par. Um vértice rotulado é do tipo *degenerado* se ele é o vértice livre, u, a partir do qual a busca foi iniciada. Logo, toda busca mantém um, e apenas

um, vértice degenerado. A entrada em L para o vértice degenerado u (o valor $L(u)$) é inicialmente "nulo" (e ignorado).

Tabela 4.1: Conteúdo da tabela L durante uma busca no grafo da Figura 4.9.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	não-rotulado	–
2	1	ponteiro	13
3	4	par	2
4	3	ponteiro	2
5	8	par	1
6	9	par	1
7	10	não-rotulado	–
8	5	ponteiro	4
9	6	ponteiro	4
10	7	ponteiro	5
11	12	par	1
12	11	ponteiro	9
13	–	degenerado	–

Se um vértice rotulado, v, é do tipo *ponteiro*, então $L(v)$ é o índice do próximo vértice rotulado no caminho que vai de v a u. Como veremos, o caminho alternante $P(v, u)$ pode ser escrito como $[v, \text{PAR}(v)] * P(L(v), u)$, onde $\text{PAR}(v)$ é o vértice que está emparelhado a v. Usando esta definição e a Tabela 4.1, podemos escrever o caminho $P(4, 13)$ como

$$P(4,13) = [4, \text{PAR}(4)] * P(L(4), 13) = [4, 3] * P(2, 13) = [4, 3] * [2, 1, 13] = [4, 3, 2, 1, 13]$$

pois

$$P(2,13) = [2, \text{PAR}(2)] * P(L(2), 13) = [2, 1] * P(13, 13) = [2, 1, 13]\,.$$

Por fim, se um vértice rotulado, v, é do tipo *par* então $\text{PAR}(v)$ deve ser do tipo ponteiro. Neste caso, $L(v)$ consiste num índice para a segunda tabela, denominada *BASE*, contendo os campos BASE_1 e BASE_2, tal que $\text{BASE}_1(L(v))$ e $\text{BASE}_2(L(v))$ são índices usados para obter $P(v, u)$. Como exemplo, considere o vértice 3, do tipo par, do grafo da Figura 4.9 e a Tabela 4.2, com os dados da tabela BASE para a busca atual no grafo. Os valores $\text{BASE}_1(L(3))$ e $\text{BASE}_2(L(3))$ são os vértices 2 e 6, respectivamente. O caminho $P(3, 13)$ pode ser definido em termos dos caminhos $P(2, 13)$ e $P(6, 13)$, um dos quais deve, obrigatoriamente, conter o vértice 3. No exemplo, o caminho $P(6, 13)$. A saber, $P(3, 13)$ é dado

por

$$\begin{aligned} P(3,13) &= P(6,3)_r * P(2,13) \\ &= [6,9,11,12,8,5,4,3]_r * P(2,13) \\ &= [3,4,5,8,12,11,9,6] * [2,\mathrm{PAR}(2)] * P(13,13) \quad = [3,4,5,8,12,11,9,6,2,1,13]\,, \end{aligned}$$

onde $P(6,3)$ denota a porção do caminho $P(6,13)$ que vai do vértice 6 ao vértice 3. É importante notar que a definição acima só é consistente porque existe uma aresta entre os vértices 2 e 6.

Tabela 4.2: Conteúdo da tabela BASE durante uma busca no grafo da Figura 4.9.

Entrada do tipo PAR	**BASE$_1$**	**BASE$_2$**	**TOPO**
1	8	12	1
2	2	6	1

A tabela BASE contém um outro campo, *TOPO*. Se v é um vértice cuja entrada em L é do tipo par, TOPO($L(v)$) informa o índice do primeiro vértice não-rotulado do caminho $P(v,u)$. Se o caminho $P(v,u)$ não contiver nenhum vértice não-rotulado, TOPO($L(v)$) recebe o valor especial 0. A utilização do campo TOPO é crucial para obter complexidade de tempo $\mathcal{O}(|V|^3)$. Em particular, TOPO possibilita a obtenção do primeiro vértice não-rotulado em $P(v,u)$ em tempo constante. Sem TOPO, esta consulta requereria $\mathcal{O}(|V|)$ passos.

Agora, podemos detalhar a busca caminhos aumentantes realizada pelo algoritmo de Gabow. Uma busca começa por um vértice livre, u, que ainda não tenha sido tomado como vértice inicial de uma busca anterior. O tipo deste nó, com respeito à busca atual, é degenerado (e, portanto, rotulado). Os demais vértices do grafo são classificados como não-rotulados. O campo PAR da tabela L contém os dados do emparelhamento atual, mas as entradas $L(v)$, para todo vértice $v \in V$, são "nulas". Na implementação do algoritmo, usamos um valor inválido; por exemplo, -1. A busca é iniciada a partir de um vértice, v, rotulado que ainda não tenha sido considerado por ela. Inicialmente, apenas u é rotulado. A partir daí, os vértices adjacentes a v são visitados, um após o outro, como numa busca em largura. Seja w o vértice adjacente a v atualmente visitado pela busca. Então, temos que $vw \in E$ e o algoritmo considera uma dentre as seguintes quatro possibilidades mutuamente exclusivas:

(1) Se w for um vértice livre, com $w \neq u$, um caminho aumentante foi encontrado:

$[w] * P(v, u)$ (veja a Figura 4.10(a)). Logo, o emparelhamento é aumentado ao longo de $P(w, u)$. A computação de $P(w, u)$ é feita através da tabela L, como descrito anteriormente.

(2) Se w estiver emparelhado a um vértice x e ambos forem não-rotulados, x se torna rotulado e recebe uma entrada em L do tipo ponteiro: $L(x) = w$, para indicar que w é o próximo vértice rotulado no caminho, $P(x, u)$, que vai de x a u (veja a Figura 4.10(b)).

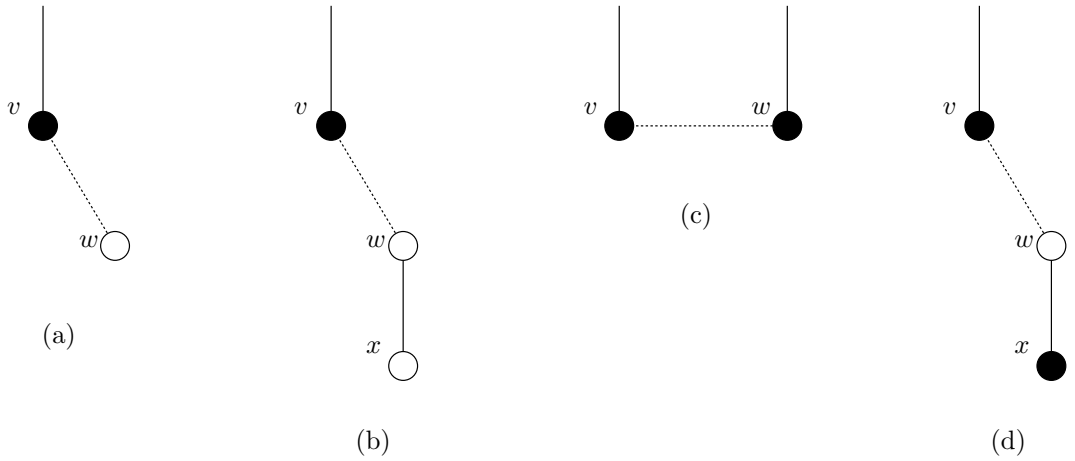


Figura 4.10: Os quatro casos da busca realizada pelo algoritmo de Gabow.

(3) Se w for um vértice rotulado, uma entrada do tipo par é atribuída a alguns vértices não-rotulados (veja a Figura 4.10(c)). Primeiro, um vértice t deve ser obtido. Este vértice é o primeiro vértice não-rotulado que é comum aos caminhos $P(v, u)$ e $P(w, u)$.

Para obter t, o campo TOPO é usado. Em seguida, todos os vértices não-rotulados que precedem t em $P(v, u)$ ou em $P(w, u)$ recebem uma entrada do tipo par. Se y é tal vértice, uma nova entrada em L e BASE é associada a y: $\text{BASE}_1(L(y)) = v$, $\text{BASE}_2(L(y)) = w$ e $\text{TOPO}(L(y)) = t$. Daqui em diante, diremos que $L(y) = (v, w)$ e $t = \text{TOPO}(v, w)$.

(4) Se nenhum dos casos anteriores ocorrer, temos que w não é um vértice rotulado. Além disso, temos que $x = \text{PAR}(w)$ é um vértice rotulado; caso contrário, teríamos o caso 2 (veja a Figura 4.10(d)). Isto significa que a aresta (v, w) é "inútil", pois

ela não faz com que o caminho alternante atual cresça e, portanto, nenhuma ação é tomada.

Enquanto o caso 1 faz com que uma nova busca seja iniciada, os casos 2, 3 e 4 podem ser considerados como um laço do procedimento de busca. Após cada um deles, a busca atual continua a partir de outro vértice, w, adjacente a v, que ainda não tenha sido considerado por ela. Se tal vértice não existir, um outro vértice, v, rotulado, que ainda não tenha sido considerado pela busca atual, é escolhido. Finalmente, se tal vértice não existir, a busca atual é encerrada e outra busca é iniciada a partir de outro vértice livre u que ainda não tenha sido usado para iniciar uma busca, como feito após o término do caso 1.

A busca realizada pelo algoritmo de Gabow cria um "grafo" a partir do vértice u. Este grafo cumpre o mesmo papel da árvore alternante do algoritmo de Edmonds. Os casos 1-4 acima podem encerrar a busca, adicionar arestas ao grafo ou mudar os atributos dos vértices do grafo. O caso 1 é o mais simples e corresponde à obtenção de um caminho *aumentante*. O caso 2 faz, simplesmente, com que um caminho *alternante* de um dado vértice ao vértice u seja acrescido em duas arestas, aumentando o grafo. O caso 3 corresponde à detecção de um botão. O vértice t, comum aos caminhos $P(v, u)$ e $P(w, u)$, está emparelhado à base do botão (veja a Figura 4.11). Este é o caso mais interessante, pois é justamente nele que reside a contribuição de Gabow. Em particular, ao invés do algoritmo explicitamente contrair ou expandir um botão, ele realiza uma "rotulação" especial nos vértices, criando entradas do tipo par, e seguindo adiante com a busca. Finalmente, o caso 4 corresponde à detecção de um ciclo de comprimento par, que em nada contribui para a busca por um caminho aumentante até o vértice u. O algoritmo de Gabow é detalhado na próxima seção.

4.4.1 Pseudocódigo

No algoritmo de Gabow, os vértices do grafo de entrada são numerados de 1 a $|V|$, com um vértice de índice 0 sendo usado apenas para condições especiais, enquanto as arestas são rotuladas de $|V| + 1$ a $|V| + |E|$. O grafo é armazenado como uma coleção de listas de adjacência, uma para cada vértice. A lista de adjacência correspondente a um dado vértice contém os índices de todas as arestas incidentes a ele. A saída do algoritmo é um vetor correspondente à coluna PAR da tabela L. Este vetor descreve o emparelhamento obtido. Sejam v e u dois vértices adjacentes no grafo sobre o qual o algoritmo é executado. Após a execução do algoritmo, u é livre se, e somente se, $PAR(u) = 0$; a aresta (u, v) faz

parte do emparelhamento se, e somente se, $PAR(u) = v$ (e $PAR(v) = u$). O mesmo vale para os emparelhamentos intermediários que vão sendo obtidos no decorrer da execução do algoritmo.

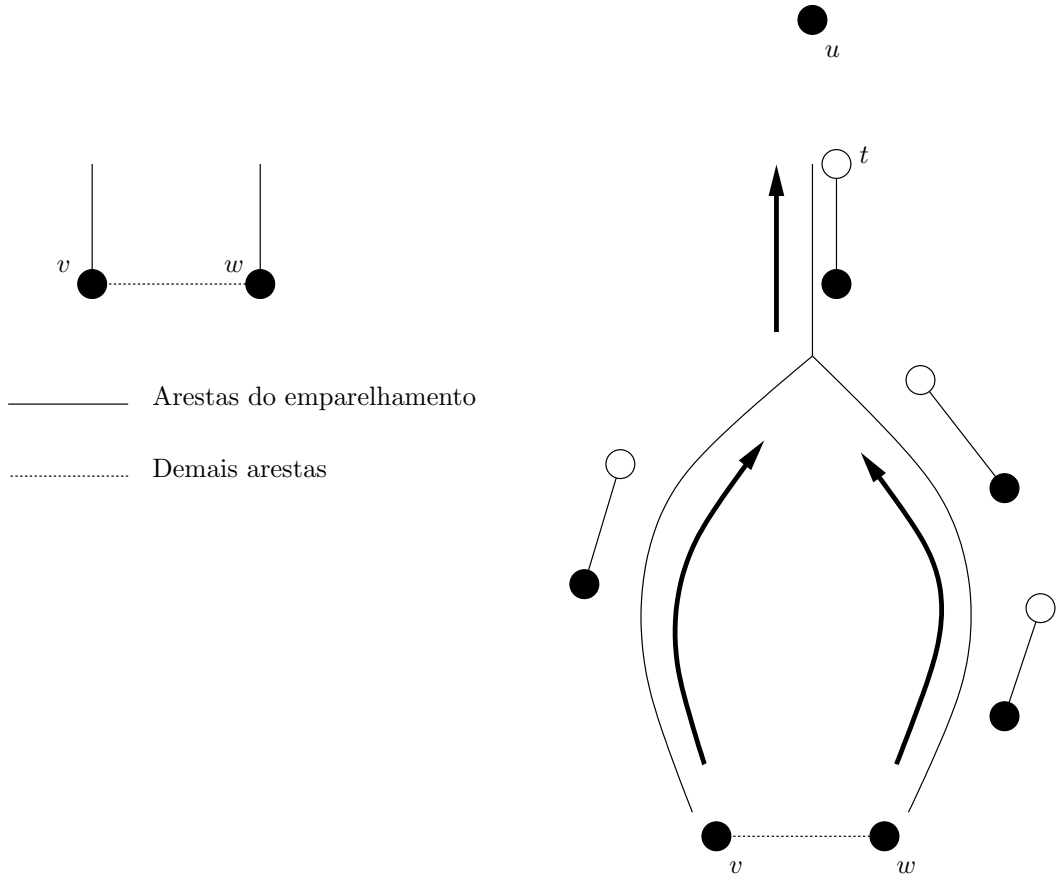


Figura 4.11: Ilustração do caso 3 do algoritmo de Gabow.

A tabela L tem uma entrada para cada vértice do grafo. Se um vértice v está rotulado, $P(v, u)$ é definido por $L(v)$. Pode ser utilizado um vetor auxiliar para, dado um vértice v, informar se o mesmo está rotulado e, em caso afirmativo, qual o tipo de sua entrada em L. A tabela BASE tem uma entrada para cada par (b_1, b_2) inserido em L como uma entrada do tipo par na busca atual. Gabow provou que o tamanho de BASE nunca ultrapassa $\lfloor(|V| - 1)/2\rfloor$. Outra propriedade a se observar é que, se um vértice v tem uma entrada, (b_1, b_2), em L do tipo par, então TOPO(b_1, b_2) é necessariamente o primeiro vértice não-rotulado em todos nos seguintes caminhos: $P(b_1, u)$, $P(b_2, u)$, $P(v, u)$ e $P(\text{PAR}(v), u)$. Entretanto, se TOPO$(b_1, b_2) = 0$, então não há vertices não-rotulados em nenhum desses caminhos.

A seguir, o algoritmo será descrito em pseudocódigo. Ele é composto de quatro rotinas: `EmparCardMaxGabow()`, a rotina principal; `EntradaPar()`, que adiciona entradas do tipo

par; `ObterTopo()`, uma rotina auxiliar para `EntradaPar()`; e `Aumentar()`, que modifica o emparelhamento ao longo de um caminho aumentante. A descrição do algoritmo assume que suas estruturas de dados (o grafo e as tabelas) podem ser acessadas por todas as rotinas. Assim, fazemos uso de parâmetros de entrada nas rotinas apenas para passagem de vértices.

Algoritmo 4.2 `EmparCardMaxGabow()`

Entrada: um grafo arbitrário, $G = (V, E)$
Saída: um emparelhamento de cardinalidade máxima em G

```
    % inicialização
 1: numere os vértices de 1 a |V| e as arestas de |V| + 1 a |V| + |E|
 2: crie um vértice especial de índice 0
 3: para i = 0 to |V| faça
 4:    PAR(i) ← 0 % o emparelhamento inicial também poderia ser não-vazio
 5: fim para
    % busca por caminhos aumentantes
 6: para u = 1 to |V| faça
 7:    se PAR(u) = 0 então % se u for livre, uma nova busca é iniciada a partir de u
 8:       L(u) ← 0 % u recebe uma entrada degenerada em L
 9:       enquanto existir uma aresta (v, w) não analisada nesta busca, sendo v um vértice
          rotulado E nenhum caminho aumentante tiver sido encontrado nesta busca faça
10:          se PAR(w) = 0 E w ≠ u então % caso (1): um caminho aumentante é encon-
             trado
11:             PAR(w) ← v
12:             Aumentar(w, v)
13:          senão se w está rotulado então % caso (3): criação de entradas do tipo par
14:             EntradaPar(w, v)
15:          senão
16:             x ← PAR(w)
17:             se x não está rotulado então % caso (2): x recebe uma entrada do tipo
                ponteiro
18:                L(x) ← v
19:             fim se
20:          fim se
21:       fim enquanto
22:    fim se
23: fim para
24: devolva PAR
```

A rotina `EmparCardMaxGabow()` é responsável por todo o processo de construção do emparelhamento para o dado grafo. Com o auxílio das demais rotinas, busca sucessivamente por caminhos aumentantes com respeito ao emparelhamento corrente, aumentando-o sempre que possível. A rotina recursiva `Aumentar()` re-emparelha as arestas que ocorrem ao longo de um caminho aumentante, incrementando o tamanho do emparelhamento corrente.

Algoritmo 4.3 `Aumentar`()

Entrada: um vértice, x, livre e um vértice, w, rotulado que será emparelhado a x.
Saída: nenhuma

```
1: y ← PAR(w)
2: PAR(w) ← x
3: se PAR(y) = w então
4:    se w tem uma entrada do tipo ponteiro então
5:       PAR(y) ← L(w)
6:       Aumentar(y, L(w))
7:    senão % neste ponto, w tem uma entrada do tipo par
8:       (b1, b2) ← L(w)
9:       Aumentar(b1, b2)
10:      Aumentar(b2, b1)
11:   fim se
12: senão
13:   retorne % o caminho aumentante chegou ao seu fim
14: fim se
```

A rotina `ObterTopo()` é chamada apenas por `EntradaPar()`. Dado um vértice rotulado, v, `ObterTopo()` encontra o primeiro vértice não-rotulado no caminho $P(v, u)$, que vai de v a u. No caso de não existir nenhum vértice não-rotulado em $P(v, u)$, o que ocorre, por exemplo, quando o próprio u é a base de um botão, `ObterTopo()` devolve o vértice fictício, 0.

Algoritmo 4.4 `ObterTopo`()

Entrada: um vértice, v, rotulado
Saída: o primeiro vértice não-rotulado em $P(v, u)$.

```
1: se PAR(v) não está rotulado então
2:    devolva PAR(v)
3: fim se
4: se v tiver uma entrada do tipo par então
5:    (b1, b2) ← L(v)
6: senão % neste ponto, PAR(v) tem uma entrada do tipo par
7:    (b1, b2) ← L(PAR(v))
8: fim se
9: devolva TOPO(b1, b2)
```

Finalmente, a rotina `EntradaPar()` tem como entrada dois vértices rotulados e adjacentes entre si, b_1 e b_2. Primeiramente, `EntradaPar()` encontra o primeiro vértice não-rotulado, t, que é comum a $P(b_1, u)$ e $P(b_2, u)$. Para a obtenção de t, os vértices não-rotulados em $P(b_1, u)$ e $P(b_2, u)$ são marcados, de forma alternada, até que um vértice não-rotulado, e comum aos dois caminhos, seja encontrado (ou seja, t). Em seguida, `EntradaPar()` associa uma entrada do tipo par em L, (b_1, b_2), a cada vértice não-rotulado

que precede t em $P(b_1, u)$ ou em $P(b_2, u)$. O vetor TOPO também é devidamente atualizado.

Algoritmo 4.5 `EntradaPar`()

Entrada: dois vértices, b_1 e b_2, rotulados e adjacentes entre si
Saída: nenhuma

```
    % inicialização
 1: v1 ← ObterTopo(b1)
 2: v2 ← ObterTopo(b2)
 3: se v1 = v2 então
 4:    retorne % nenhum vértice pode ser rotulado
 5: fim se
 6: marque v1 e v2 % vértices não-rotulados de P(b1, u) e P(b2, u) vão sendo marcados
    % computação do vértice t
 7: t ← −1 % o valor −1 assinala que t ainda não foi determinado
 8: enquanto t = −1 faça
 9:    se v2 ≠ 0 então
10:       troque v1 e v2 % v1 percorre os vértices não-rotulados em P(b1, u) e P(b2, u)
11:    fim se
12:    v1 ← ObterTopo(L(PAR(v1))) % próximo vértice não-rotulado em P(v1, u)
13:    se v1 não está marcado então
14:       marque v1
15:    senão
16:       t ← v1
17:    fim se
18: fim enquanto
    % atribuição de entradas do tipo par
19: para i = 1 to 2 faça
20:    w ← ObterTopo(bi)
21:    enquanto w ≠ t faça % w percorre os vértices não-rotulados em P(bi, u) de bi a t
22:       L(w) ← (b1, b2)
23:       desmarque w
24:       w ← ObterTopo(L(PAR(w)))
25:    fim enquanto
26: fim para
    % atualização de TOPO
27: TOPO(b1, b2) ← t
28: para toda entrada do tipo par, (x, y), inserida na busca atual partindo de u faça
29:    se TOPO(x, y) está rotulado então
30:       TOPO(x, y) ← t
31:    fim se
32: fim para
    % remoção das marcações dos vértices que ocorrem após t em P(b1, u) e P(b2, u)
33: enquanto t estiver marcado faça
34:    desmarque t
35:    t ← ObterTopo(L(PAR(t)))
36: fim enquanto
```

4.4.2 Exemplo de execução

O algoritmo de Gabow apresentado na seção anterior será executado detalhadamente, a seguir, para o grafo $G = (V, E)$, da Figura 4.12. Nesta execução, o algoritmo é iniciado com um emparelhamento vazio. O vértice tomado como inicial pela primeira busca é o vértice 1. Assim, tão logo a primeira aresta, $(1, 2)$, seja processada, um caminho aumentante é encontrado, uma vez que ambos os extremos da aresta são vértices expostos. Logo, `EmparCardMaxGabow()` efetua o aumento do emparelhamento com respeito ao caminho $[1, 2]$. A saber, faz as atribuições PAR(1) $\leftarrow 2$ e PAR(2) $\leftarrow 1$. Em seguida, as arestas $[3, 4]$ e $[5, 6]$ também são emparelhadas da mesma forma, em buscas iniciadas pelos vértices 3 e 5, respectivamente. O estado do emparelhamento neste momento é ilustrado na Figura 4.13.

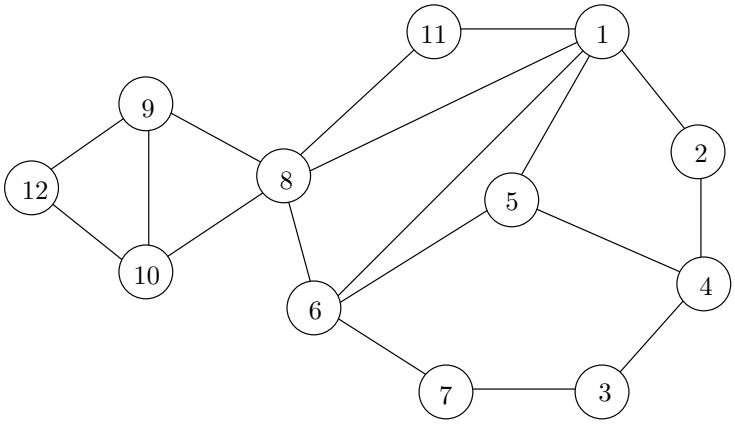


Figura 4.12: Grafo utilizado no exemplo de execução do algoritmo de Gabow.

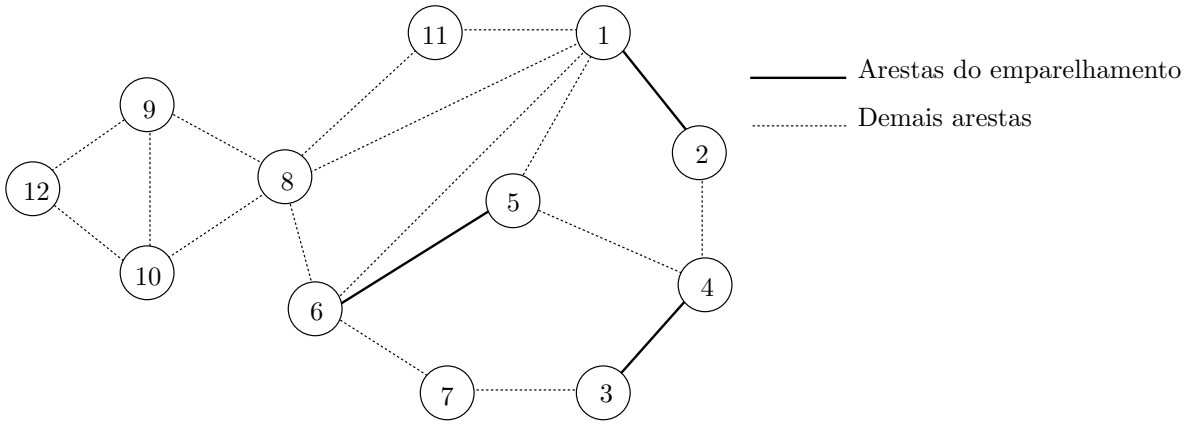


Figura 4.13: Emparelhamento obtido no grafo da Figura 4.12 após 3 buscas.

Note que apenas o caso 1 do algoritmo de Gabow ocorreu até agora.

A próxima busca é iniciada pelo vértice livre 7, que se torna o único vértice rotulado do grafo. `EmparCardMaxGabow()` processa a aresta $(7, 3)$ e adiciona, na tabela L, uma entrada do tipo ponteiro para o vértice 4. Isto porque os vértices 3 e 4 não são livres nem rotulados (caso 2). O vértice 4 se torna rotulado. Depois, `EmparCardMaxGabow()` escolhe, de maneira arbitrária e ainda na mesma busca iniciada no vértice 7, processar a aresta $(4, 5)$. De forma análoga, o vértice 6 se torna rotulado e uma entrada do tipo ponteiro, para o vértice 6, é adicionada à tabela L (caso 2). Em seguida, `EmparCardMaxGabow()` processa, de maneira arbitrária, a aresta $(6, 8)$. Como o vértice 8 está livre, o caminho aumentante,

$$[8] * P(6, 7) = [8, 6, 5, 4, 3, 7]\,,$$

é encontrado e o emparelhamento atual é aumentado. A Figura 4.14 ilustra o emparelhamento resultante. A Tabela 4.3 exibe as entradas adicionadas à tabela L ao longo da busca.

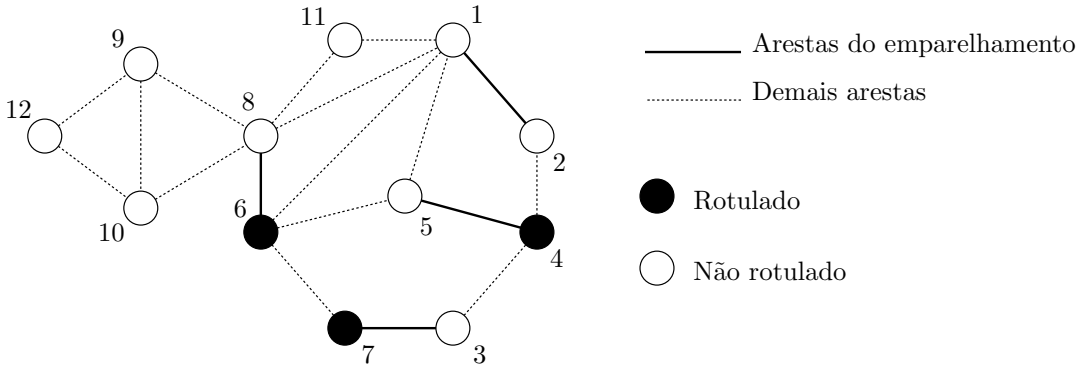


Figura 4.14: Emparelhamento obtido com aumento ao longo do caminho $P(8, 7)$.

`EmparCardMaxGabow()` prossegue com uma nova busca a partir do próximo vértice livre, o vértice 9. Novamente, na primeira aresta processada, $(9, 10)$, um caminho aumentante é encontrado (caso 1 do algoritmo), resultando no emparelhamento ilustrado na Figura 4.15.

A última busca realizada pelo algoritmo, antes de um emparelhamento de cardinalidade máxima ser encontrado, é iniciada pelo vértice 11. Para melhor acompanhar os passos desta busca, exibiremos várias figuras com alguns estados intermediários da busca. Cada estado é ilustrado por um desenho do grafo de busca e por algumas tabelas. O grafo de busca mostra as arestas de G que já foram processadas na busca atual. Adicionalmente, para cada nó v que se torna rotulado, o caminho $P(v, 11)$ é ilustrado sobre o grafo de busca.

Tabela 4.3: Tabela L logo após a busca que encontrou o caminho aumentante $P(8, 7)$.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	não-rotulado	–
2	1	não-rotulado	–
3	7	não-rotulado	–
4	5	ponteiro	7
5	4	não-rotulado	–
6	8	ponteiro	4
7	3	degenerado	–
8	6	não-rotulado	–
9	–	não-rotulado	–
⋮	⋮	⋮	⋮

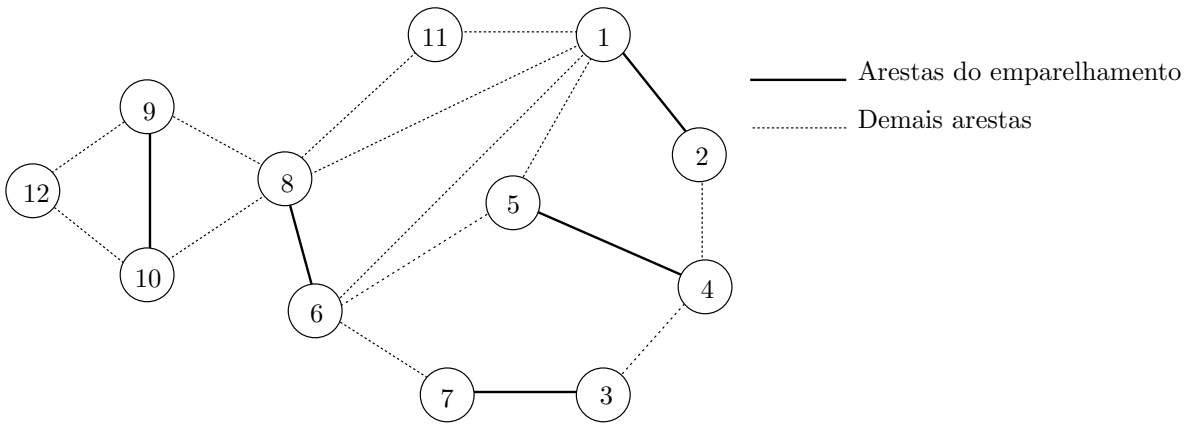


Figura 4.15: Emparelhamento obtido com aumento ao longo do caminho $P(10, 9)$.

A Figura 4.16(a) mostra o grafo de busca após as arestas $(11, 8)$, $(11, 1)$, $(6, 7)$ e $(6, 5)$ terem sido exploradas. Para cada uma delas, o caso 2 do algoritmo de Gabow ocorreu, o que resultou na rotulação dos vértices 6, 2, 3 e 4. Além disso, a tabela L foi acrescida de quatro entradas do tipo ponteiro para cada um desses quatro vértices. O conteúdo de L está na Tabela 4.4.

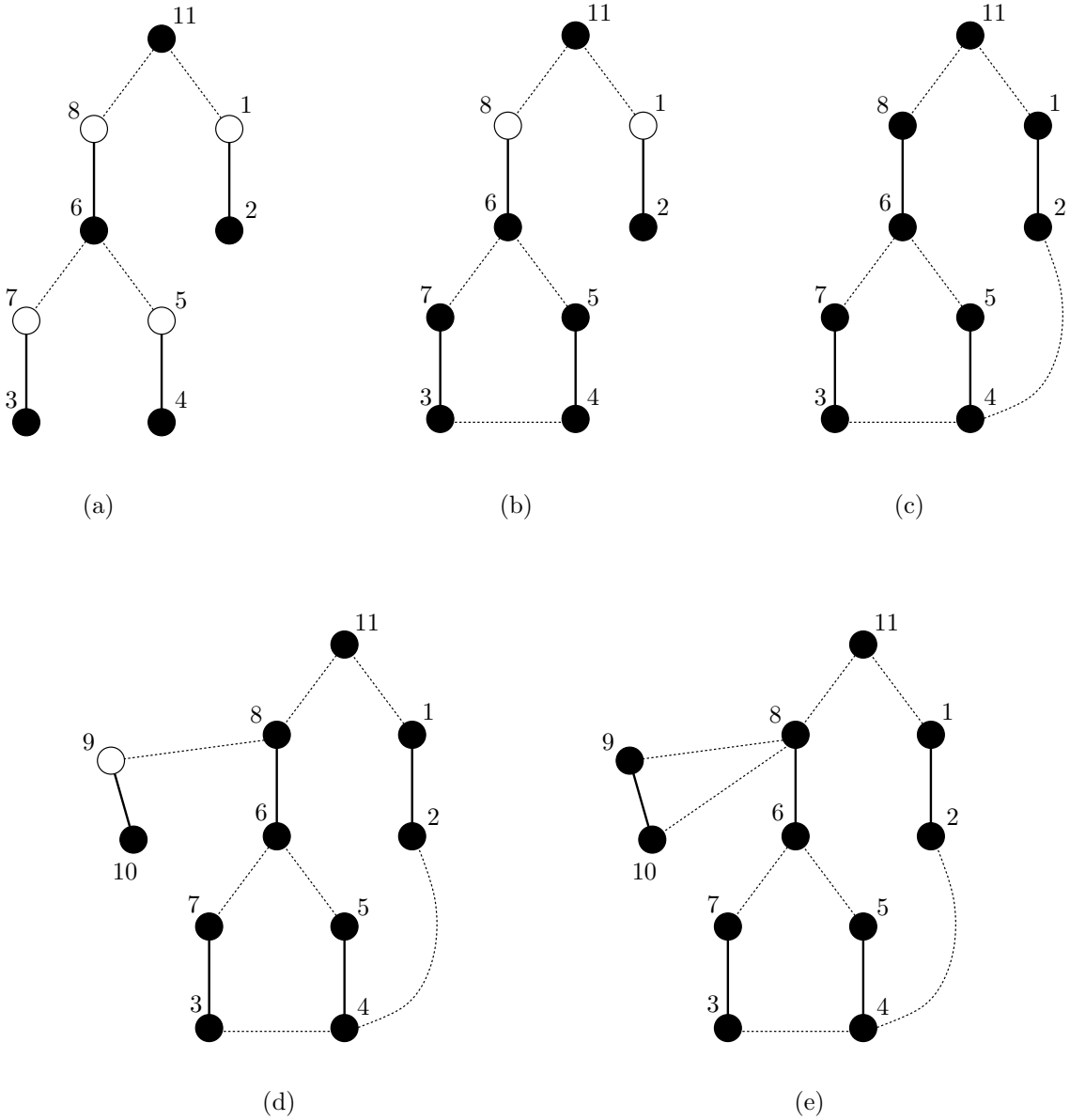


Figura 4.16: Evolução do grafo de busca construído a partir do vértice 11.

A partir deste ponto, a próxima aresta processada por `EmparCardMaxGabow()` é $(3, 4)$. Como ambos os extremos da aresta são vértices rotulados, a rotina `EntradaPar()` é chamada. Este é o caso 3 do algoritmo de Gabow, que corresponde à detecção de um botão. `EntradaPar()` encontra $\text{TOPO}(3, 4) = 8$, o primeiro vértice não-rotulado comum

a

$$P(3,11) = [3,7,6,8,11] \qquad \text{e} \qquad P(4,11) = [4,5,6,8,11]\,.$$

Note que o vértice 8 está emparelhado ao vértice 6, que é a base do botão encontrado. Em seguida, `EntradaPar()` associa entradas do tipo par aos vértices 5 e 7, aqueles vértices não-rotulados que precedem 8 em $P(3,11)$ ou em $P(4,11)$. O resultado é mostrado na Figura 4.16(b).

Tabela 4.4: Tabela L depois das arestas $(11,8)$, $(11,1)$, $(6,7)$ e $(6,5)$ serem exploradas.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	não-rotulado	–
2	1	ponteiro	11
3	7	ponteiro	6
4	5	ponteiro	6
5	4	não-rotulado	–
6	8	ponteiro	11
7	3	não-rotulado	–
8	6	não-rotulado	–
9	10	não-rotulado	–
10	9	não-rotulado	–
11	–	degenerado	–
12	–	não-rotulado	–

Os conteúdos de L e BASE estão na Tabela 4.5.

A seguir, a aresta $(2,4)$ é processada. Novamente, ambos os vértices envolvidos são vértices rotulados, o que faz com que o algoritmo execute `EntradaPar`$(4,2)$. Note que o botão que o algoritmo acabou de encontrar pode ser visto como um botão que contém o anterior. Para melhor entender como o algoritmo de Gabow lida com este "aninhamento" de botões, exibiremos os passos do algoritmo em maiores detalhes. A seguir, descrevemos como o primeiro vértice, t, não-rotulado e comum $P(4,11)$ e $P(2,11)$ é obtido por `EntradaPar()` (veja o Algoritmo 4.5):

- As linhas 1 a 6 encontram, isoladamente, os primeiros vértices, $u_1 = 8$ e $u_2 = 1$, não-rotulados em $P(4,11)$ e em $P(2,11)$, respectivamente. Os vértices 1 e 8 são, portanto, marcados.

- As linhas 7 a 18 encontram t. Ao entrar no laço da linha 8, os vértices u_1 e u_2 são trocados ($u_1 \leftarrow 1$ e $u2 \leftarrow 8$). Depois, u_1 passa a ser o próximo vértice não-rotulado

em $P(1, 11)$. Porém, como tal vértice não existe, u_1 recebe o vértice especial, 0. Logo, 0 é marcado.

- Os comandos internos ao laço da linha 8 são executados novamente, com $u_1 = 0$ e $u_2 = 8$. Primeiro, u_1 e u_2 são trocados ($u_1 \leftarrow 8$ e $u_2 \leftarrow 0$). Novamente, não há vértice não-rotulado em $P(8, 11)$ e, consequentemente, u_1 recebe o vértice 0. Mas desta vez, como 0 já está marcado, t recebe 0 e o laço é abortado. O fato de termos o valor $t = 0$ é consistente com o fato do vértice 11 ser a base do botão encontrado.

Tabela 4.5: Tabelas L e BASE depois da aresta $(3, 4)$ ser explorada.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	não-rotulado	–
2	1	ponteiro	11
3	7	ponteiro	6
4	5	ponteiro	6
5	4	par	$(4, 3)$
6	8	ponteiro	11
7	3	par	$(4, 3)$
8	6	não-rotulado	–
9	10	não-rotulado	–
10	9	não-rotulado	–
11	–	degenerado	–
12	–	não-rotulado	–

Entrada (PAR)	**BASE**	**TOPO**
1	$(4, 3)$	8

Nesse procedimento realizado para a obtenção de t, vale notar que o vértice u_1 percorre, alternadamente, os vértices não-rotulados dos caminhos $P(4, 11)$ e $P(2, 11)$. Cada um destes vértices é marcado quando "visitado" por u_1. Assim, logo que u_1 chega a um vértice já marcado, pode-se afirmar que ele é o primeiro vértice não-rotulado comum aos dois caminhos.

O próximo passo a ser realizado por `EntradaPar()` é associar, a cada vértice não-rotulado que precede t (0) em $P(4, 11)$ ou em $P(2, 11)$, a entrada do tipo par $(4, 2)$. Neste caso, já que t é o vértice especial 0, `EntradaPar()` associará a referida entrada do tipo par a todos os vértices não-rotulados que estão em $P(4, 11)$ ou em $P(2, 11)$ (os vértices 1 e 8). Para tal fim, `EntradaPar()` percorre novamente os devidos vértices não-rotulados, nas linhas de 19 a 26. As marcações dos vértices, não mais necessárias, são removidas ao longo do processo.

Nas linhas de 27 a 32, `EntradaPar()` atualiza o campo TOPO da tabela BASE para os

vértices v que receberam entradas do tipo par e tais que TOPO(v) passou a ser um vértice rotulado. Nas linhas de 33 a 36, `EntradaPar()` remove as entradas do tipo par excedentes, que ocorrem a partir de t em ambos os caminhos $P(4, 11)$ e $P(2, 11)$. Neste caso, como $t = 0$, apenas o vértice 0 é desmarcado. O resultado desta execução de `EntradaPar()` é mostrado na Figura 4.16(c). Os conteúdos das tabelas L e BASE são mostrados pela Tabela 4.6.

Tabela 4.6: Tabelas L e BASE depois da aresta $(4, 2)$ ser explorada.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	par	$(4, 2)$
2	1	ponteiro	11
3	7	ponteiro	6
4	5	ponteiro	6
5	4	par	$(4, 3)$
6	8	ponteiro	11
7	3	par	$(4, 3)$
8	6	par	$(4, 2)$
9	10	não-rotulado	–
10	9	não-rotulado	–
11	–	degenerado	–
12	–	não-rotulado	–

Entrada (PAR)	**BASE**	**TOPO**
1	$(4, 3)$	0
2	$(4, 2)$	0

Uma vez que o caso 3 do algoritmo tenha sido encerrado, `EmparCardMaxGabow()` prossegue com a busca, processando novas arestas. Primeiramente, a aresta $(8, 9)$ é escolhida, o que faz com que o vértice 10 seja rotulado (através do caso 2). A Figura 4.16(d) ilustra o grafo de busca após a rotulação do vértice 10. A próxima aresta escolhida é $(10, 8)$, o que faz com mais um botão seja detectado e o vértice 9 rotulado. A Figura 4.16(e) ilustra o grafo de busca após a rotulação do vértice 9. A Tabela 4.7 exibe os conteúdos de L e BASE.

Finalmente, a aresta $(9, 12)$ é escolhida para processamento e o caso 1 do algoritmo ocorre, pois o vértice 12 é livre. A Figura 4.17 ilustra o caminho aumentante, $P(12, 11)$, encontrado:

$$[12, 9, 10, 8, 6, 5, 4, 2, 1, 11]\,.$$

O aumento do emparelhamento, através do caminho aumentante $P(12, 11)$, é feito pela rotina `Aumentar()`. A seguir, detalharemos a execução desta rotina usando figuras que ilustram alguns estados intemediários do processo de aumento através de desenhos do

estado do grafo de busca e a pilha atual de chamadas recursivas feitas por `Aumentar()`. Nas ilustrações dos grafos, algumas arestas são desenhadas por segmentos de reta (ou curva) com duas partes: uma vermelha sólida e uma preta pontilhada. Se u e v são dois vértices rotulados por uma aresta desenhada desta forma e tal que a parte vermelha sólida é incidente em u, então o emparelhamento é interpretado como segue: $\text{PAR}(u) = v$, mas $\text{PAR}(v) \neq u$.

Tabela 4.7: Tabelas L e BASE depois da aresta $(10, 8)$ ser explorada.

Vértice	**PAR**	**Tipo**	**Entrada**
1	2	par	$(4, 2)$
2	1	ponteiro	11
3	7	ponteiro	6
4	5	ponteiro	6
5	4	par	$(4, 3)$
6	8	ponteiro	11
7	3	par	$(4, 3)$
8	6	par	$(4, 2)$
9	10	par	$(10, 8)$
10	9	ponteiro	8
11	–	degenerado	–
12	–	não-rotulado	–

Entrada (PAR)	**BASE**	**TOPO**
1	$(4, 3)$	0
2	$(4, 2)$	0
3	$(10, 8)$	0

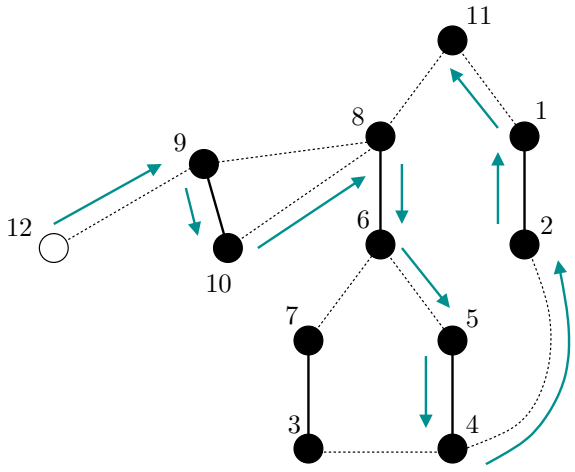


Figura 4.17: Ilustração do caminho aumentante $P(12, 11)$.

Quando a rotina `Aumentar()` é chamada, o estado do emparelhamento é o exibido pela Figura 4.18(a). Note que $\text{PAR}(12) = 9$. De fato, o vértice 9 é atribuído a PAR(12) por `EmparCardMaxGabow()` após o vértice exposto 12 ser encontrado (veja a linha 11 do

Algoritmo 4.2) e, portanto, antes da chamada a `Aumentar`$(12, 9)$ ser feita. O caminho $P(9, 11)$ é ressaltado na Figura 4.18(a). É ao longo dele que o emparelhamento atual será aumentado.

A Figura 4.18(b) mostra o resultado da execução de `Aumentar`$(12, 9)$. O vértice 9 já aparece completamente emparelhado a 12, e duas chamadas recursivas são adicionadas à pilha. Isto ocorre porque o vértice 9 possui uma entrada do tipo par na tabela L. Logo, o caminho $P(9, 11)$ é definido, através do vetor L, como a concatenação de dois caminhos, a saber:

$$P(9, 11) = P(10, 9)_{\mathrm{r}} * P(8, 11)\,.$$

Logo, ambos os caminhos devem ser processados (veja as linhas 8-10 do Algoritmo 4.3). É importante ressaltar que, por definição, um dos caminhos, $P(10, 11)$ ou $P(8, 11)$, contém o vértice 9. No exemplo, o caminho $P(10, 11)$. Entretanto, o algoritmo não calcula qual dos dois caminhos contém o vértice 9. As duas chamadas a `Aumentar()`, com os mesmos parâmetros passados em ordem inversa, realizam o aumento do emparelhamento, ao longo do caminho $P(9, 11)$, em partes disjuntas do caminho e seguindo direções inversas de percurso.

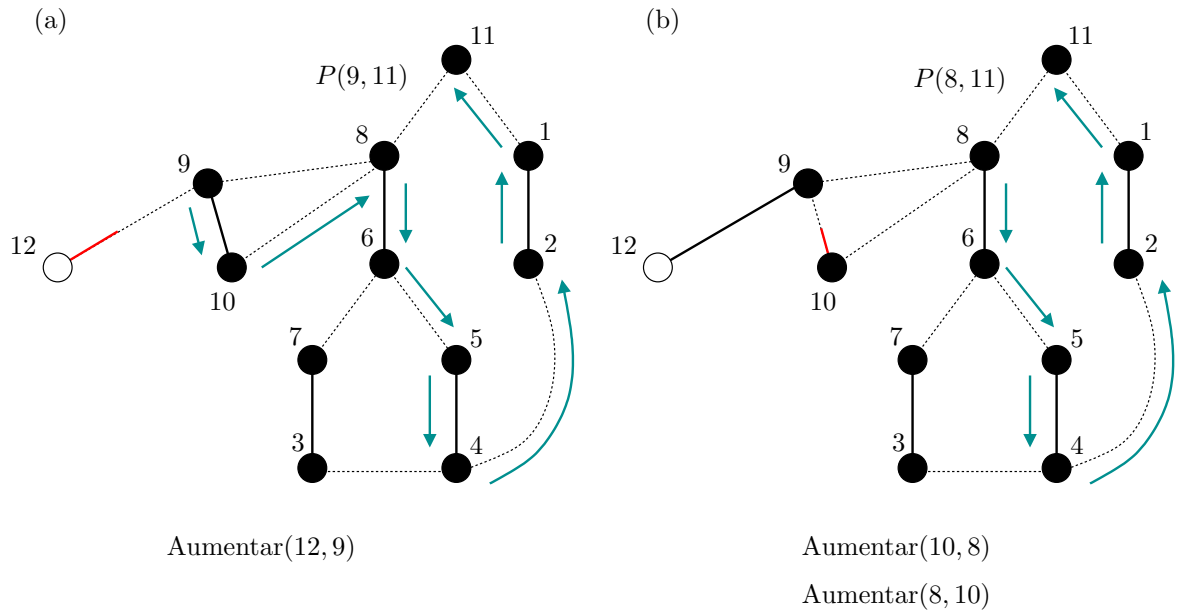


Figura 4.18: Estado do emparelhamento antes e depois de `Aumentar(12,9)`.

A chamada a `Aumentar`$(10, 8)$ processa $P(8, 11)$ de forma similar, visto que o vértice 8 também possui uma entrada do tipo par em L. A Figura 4.19(c) mostra o resultado desta execução.

O processo continua com a execução de `Aumentar`$(4, 2)$, que está agora no topo da pilha. O resultado desta nova execução é mostrado na Figura 4.19(d). Veja que PAR(1) e PAR(2) são modificados neste momento. Além disso, a chamada recursiva `Aumentar`$(1, 11)$ é adicionada à pilha, observando-se a entrada do tipo ponteiro do vértice 2. A chamada `Aumentar`$(1, 11)$, por sua vez, retorna sem fazer chamadas recursivas, porque 11 é o vértice degenerado da busca atual. Assim, o caminho $P(11, 11)$ é definido simplesmente como o próprio [11].

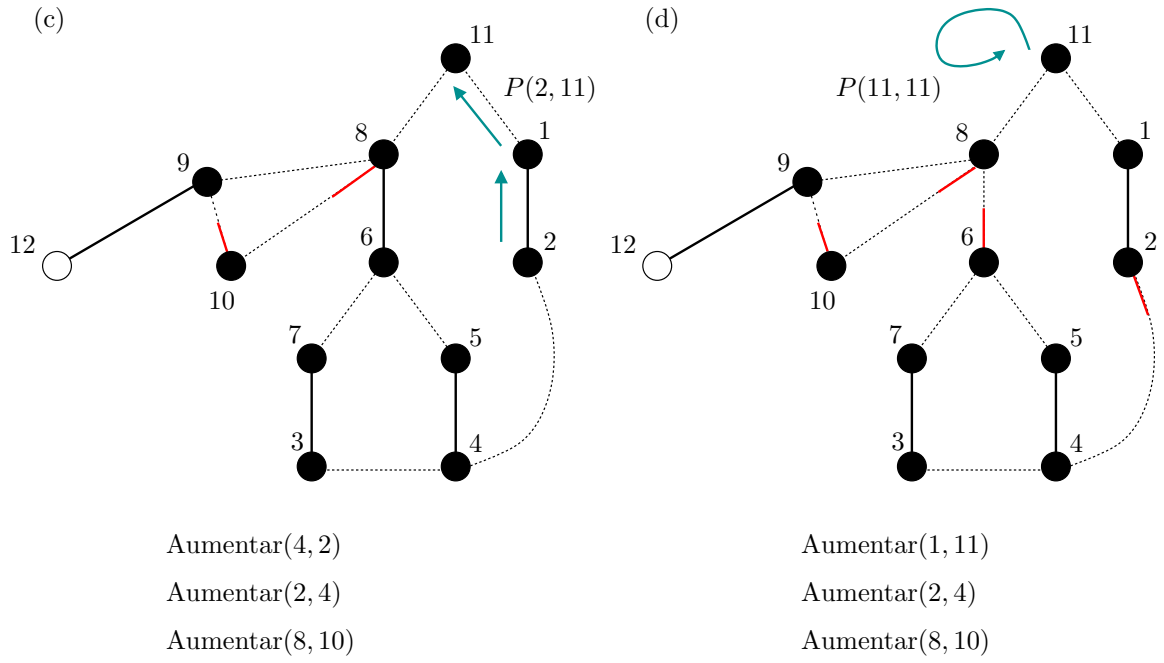


Figura 4.19: Estado do emparelhamento antes e depois de `Aumentar(10,8)`.

O restante do processamento do caminho aumentante é detalhado nas Figuras 4.20(e), 4.20(f) e 4.21(g). Já a Figura 4.21(h) mostra o novo emparelhamento obtido ao fim do processo. Neste ponto, não existem mais vértices livres e, portanto, o emparelhamento já é de cardinalidade máxima. Logo, a rotina `EmparCardMaxGabow()` continuará executando o laço da linha 6, que busca por vértices livres (neste caso, sem sucesso), até que u ultrapasse $|V|$.

Para finalizar esta seção, será feita agora uma breve demonstração de como o algoritmo de Edmonds, descrito na Seção 4.3, encontra o emparelhamento máximo para o mesmo grafo utilizado acima, G. Assim, será possível comparar o modo de operação de ambos os algoritmos.

Até chegar ao emparelhamento da Figura 4.15, ambos os algoritmos procedem da mesma forma. Portanto, o algoritmo de Edmonds será considerado a partir deste empar-

elhamento no grafo G. Recordemos que, neste estado, a próxima busca por um caminho aumentante parte do vértice 11. A Figura 4.22 mostra a evolução da árvore alternante, com raiz no vértice 11, construída pelo algoritmo de Edmonds. Esta árvore é equivalente aos grafos de busca da Figura 4.16, construídos pelo algoritmo de Gabow a partir do mesmo vértice.

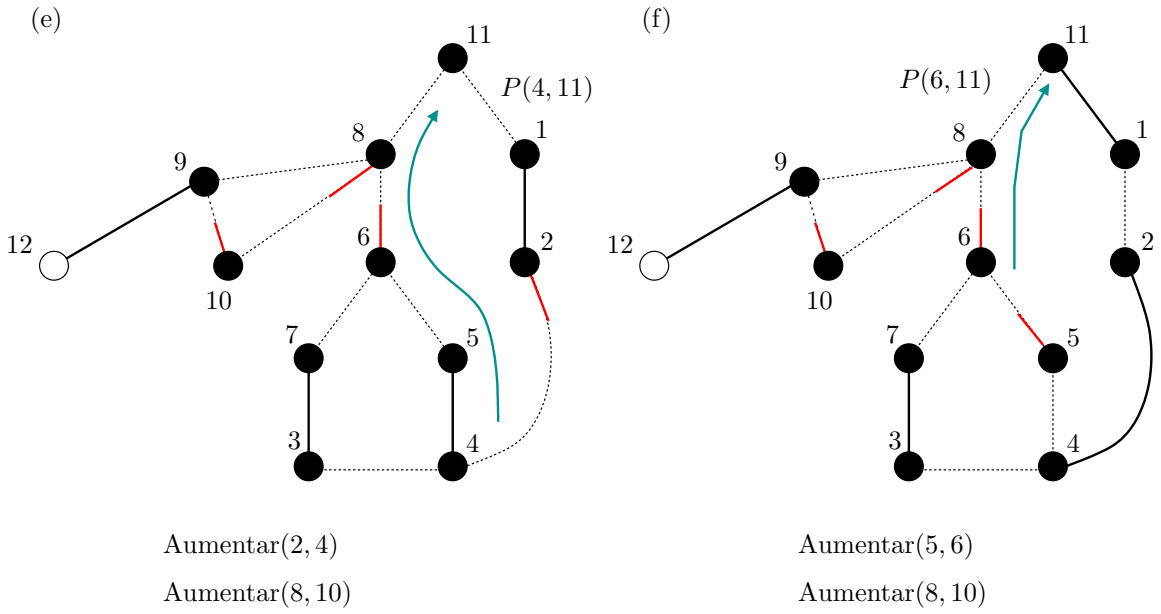
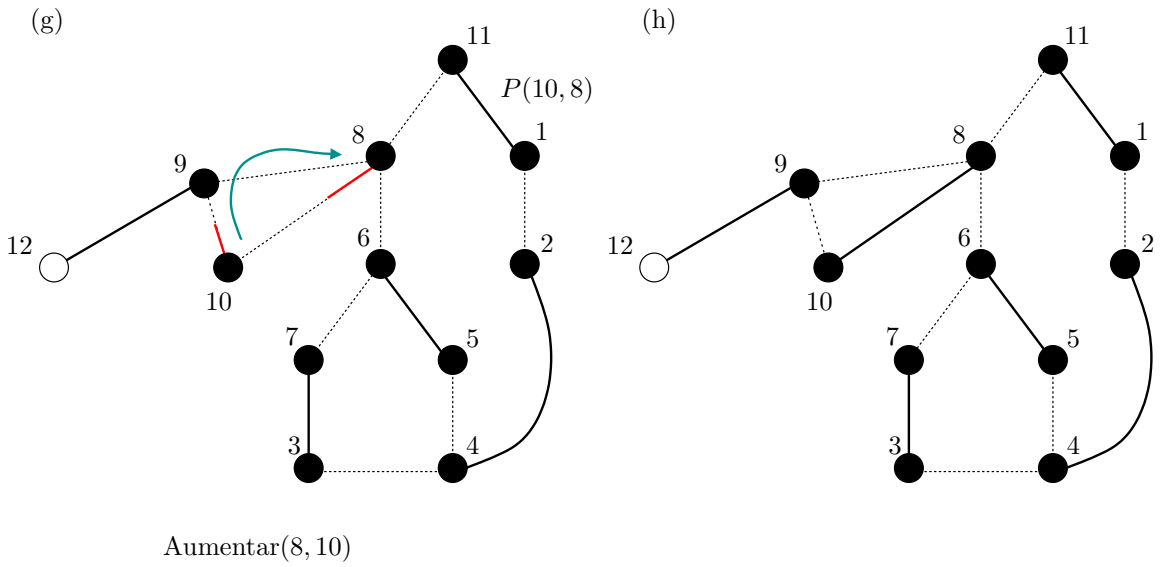


Figura 4.20: Estado do emparelhamento antes e depois de `Aumentar(2,4)`.

Inicialmente, a árvore é expandida até chegar ao estado mostrado na Figura 4.22(a). É importante perceber, comparando as Figuras 4.16(a) e 4.22(a), que a estrutura da árvore alternante é equivalente à das entradas do tipo ponteiro. Quando a próxima aresta, $(4, 3)$, é processada, um botão (veja a Definição 4.3.1) é formado: $[6, 7, 3, 4, 5, 6]$. Logo em seguida, o botão é contraído, transformando-se num único vértice a de acordo com as regras descritas na Definição 4.3.2,. A Figura 4.22 (b) mostra a árvore alternante resultante da contração do botão.

A busca prossegue, a partir desta nova árvore alternante, até que um caminho aumentante seja encontrado ou que a árvore não possa mais ser expandida. A aresta $(4, 2)$ é a próxima a ser processada. Mas, neste caso, o vértice 4 pertence ao botão contraído no passo anterior. Assim sendo, esta aresta é considerada como $(a, 2)$. Mais uma vez, a introdução desta aresta provoca a formação de um botão, que é contraído para a criação do vértice b, como exibido na Figura 4.22(c). Os demais passos realizados pela busca são semelhantes, até que um caminho aumentante é encontrado, $[c, 12]$, como mostra a Figura 4.22(f).



Aumentar$(8,10)$

Figura 4.21: Estado do emparelhamento antes e depois de `Aumentar(8,10)`.

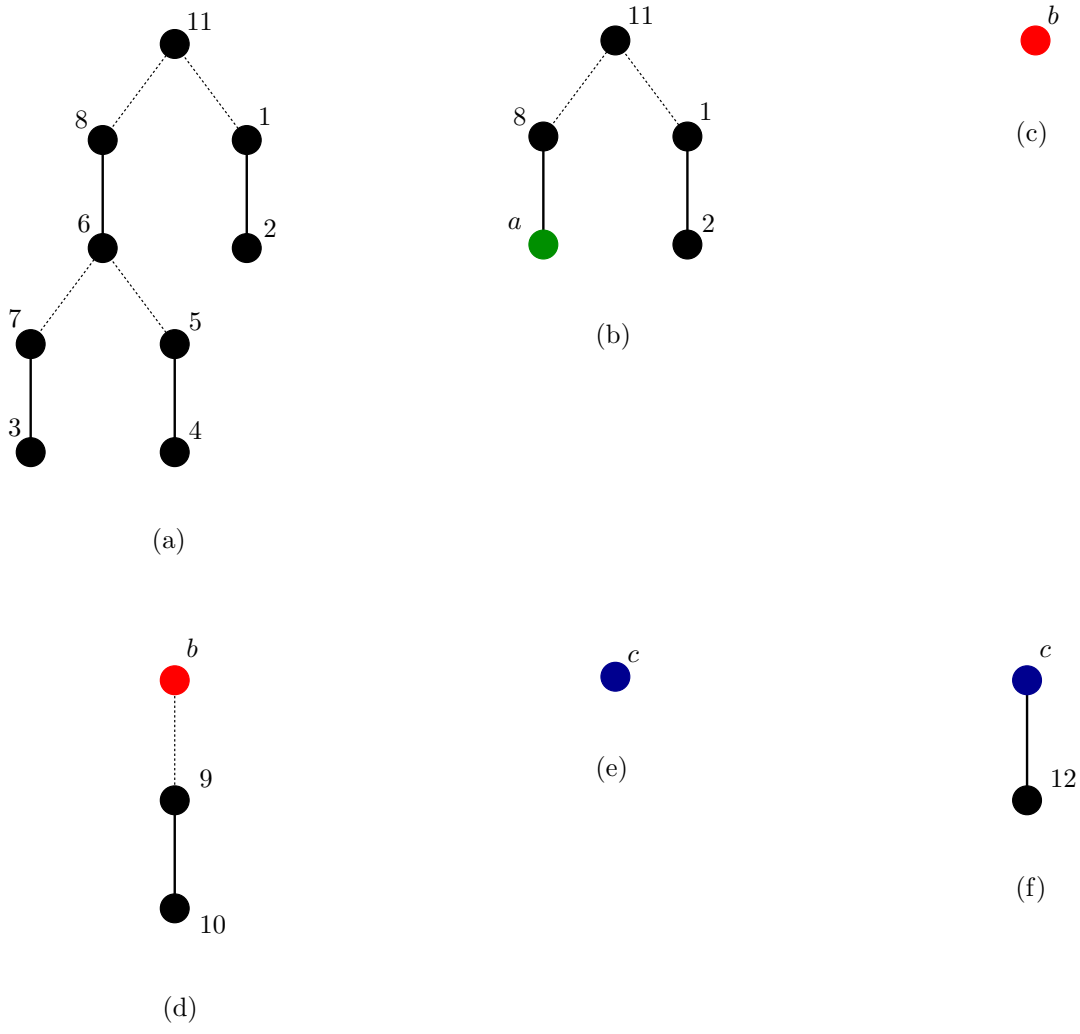


Figura 4.22: Evolução da árvore alternante do algoritmo de Edmonds.

Para efetuar o aumento do emparelhamento ao longo do caminho $[c, 12]$, os botões previamente contraídos precisam ser expandidos. A Figura 4.23 mostra os estados intermediários do processo de aumento do emparelhamento, o qual também realiza, gradativamente, a expansão dos devidos botões. Primeiramente, o vértice 12 é emparelhado ao vértice c, como mostra a Figura 4.23(a). A Figura 4.23(b) mostra o resultado da expansão do vértice c, recuperando o botão $[b, 9, 10, b]$. Além disso, note que as arestas do botão expandido são re-emparelhadas, de forma a manter o emparelhamento válido. O procedimento segue até que o novo emparelhamento seja obtido no grafo original, exibido na Figura 4.23(d).

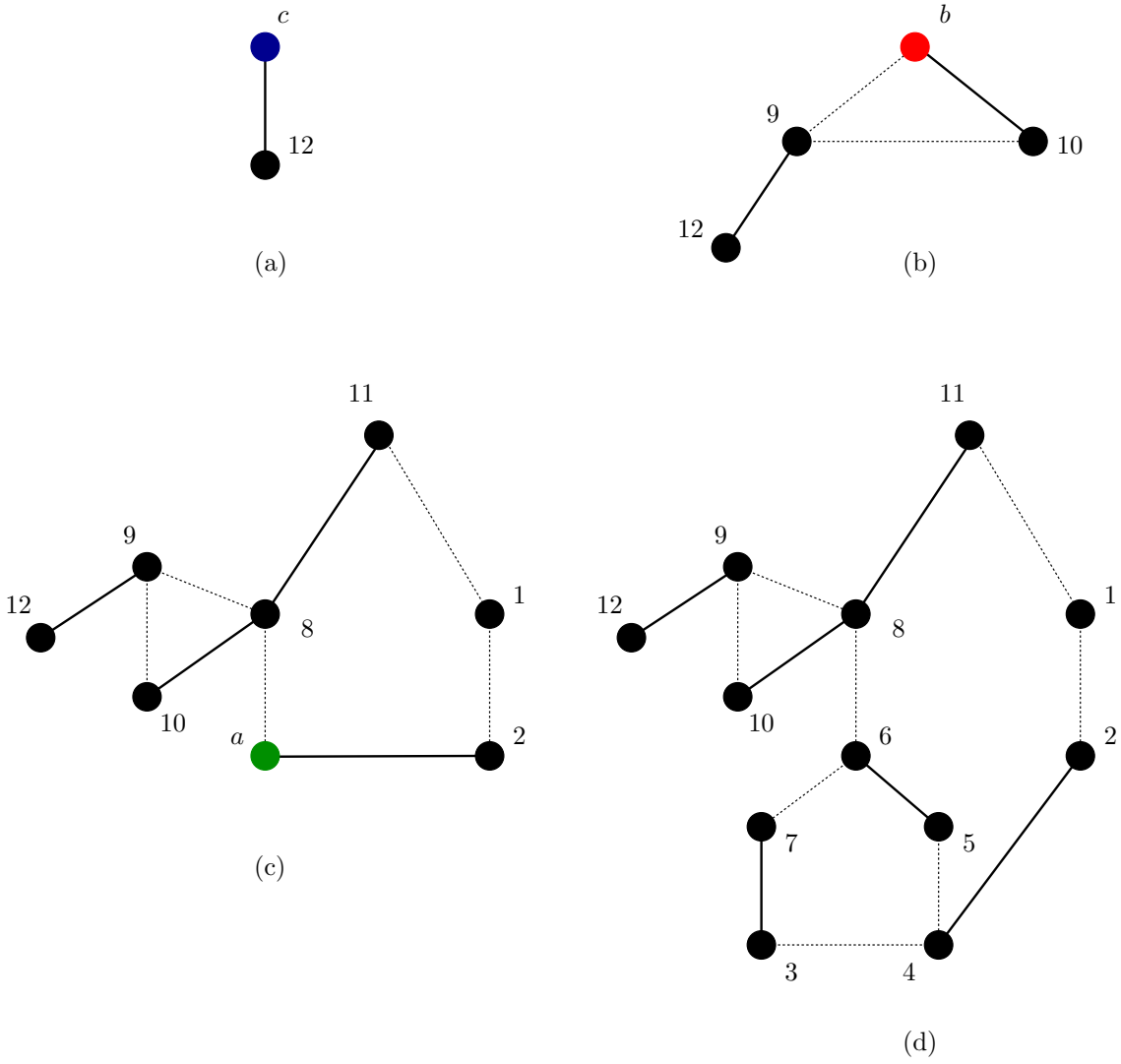


Figura 4.23: Expansão de botões e cálculo de caminho aumentante.

O exemplo acima ilustra a diferença entre os algoritmos de Gabow e Edmonds. O algoritmo de Gabow não efetua contrações e expansões de botões. Ao invés de contrair um botão, o algoritmo de Gabow realiza uma rotulação especial dos vértices não-rotulados

do botão (que passam a ser do tipo par) através da rotina `EntradaPar()`. Esta rotulação, representada pelas tabelas L e BASE, permite que a rotina `Aumentar()` penetre dentro de um botão (ou de um aninhamento de botões) quando um caminho aumentante está sendo calculado.

É especialmente interessante perceber como o trecho das linhas 27 a 32, da rotina `EntradaPar()`, atualiza os valores do campo TOPO de forma que os caminhos alternantes continuem bem definidos mesmo com a ocorrência de botões aninhados. Em particular, suponhamos que a execução da rotina chegue à linha 30, para uma dada entrada do tipo par (x, y) e um dado vértice t. Isto significa que a aresta (x, y) já pertence a um botão, a, encontrado na busca atual. Além disto, o botão, b, processado na chamada corrente a `EntradaPar()` (e cuja base é t) é "externo" em relação a a, de forma que ambos estão aninhados.

Dada esta situação, `EntradaPar()` efetua a atribuição $\text{TOPO}(x, y) \leftarrow t$, que tem o efeito de representar que os vértices do botão a, que possuem a entrada par (x, y), são englobados agora por um botão mais externo b, cuja base é t. Assim, o algoritmo vai definindo, implicitamente, uma relação de ordem parcial entre os botões descobertos numa dada busca. Esta ordem segue o mesmo princípio do algoritmo de Edmonds, no qual os botões precisam ser contraídos ou expandidos numa sequência apropriada, e é quem propicia posteriormente a correta recuperação dos caminhos aumentantes pela rotina `Aumentar()`.

Gabow mostrou que o seu esquema de rotulação reduz a complexidade do algoritmo de Edmonds de $\mathcal{O}(|V|^4)$ para $\mathcal{O}(|V|^3)$ (GABOW, 1976). De fato, uma implementação direta do algoritmo de Edmonds realiza uma única operação de contração (ou de expansão) em tempo $\mathcal{O}(|V|^2)$, enquanto as rotinas `EntradaPar()` e `Aumentar()` são lineares em $|V|$. Gabow também forneceu uma prova detalhada da corretude de seu algoritmo em (GABOW, 1972).

5 Emparelhamentos com Custo

Este capítulo apresenta duas variações do problema-alvo deste trabalho: o emparelhamento de custo máximo e o emparelhamento perfeito de custo máximo. A Seção 5.1 apresenta os conceitos básicos de Programação Linear, na qual os algoritmos de emparelhamento que usamos se baseiam. A Seção 5.2 descreve o problema do emparelhamento de custo máximo em grafos bipartidos e define o programa linear correspondente. A Seção 5.3 introduz as modificações necessárias no programa linear da seção anterior para grafos gerais.

5.1 Programação linear

Esta seção define o problema geral de Programação Linear e discute, muito superficialmente, alguns resultados e fatos relevantes para o entendimento do problema do emparelhamento em grafos ponderados. O conteúdo desta seção foi retirado do livro (GASS, 1964).

Sejam m e n inteiros positivos, $\boldsymbol{b} \in \mathbb{R}^m$ e $\boldsymbol{c} \in \mathbb{R}^n$ vetores colunas e $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ uma matriz $m \times n$. Então, o problema geral de Programação Linear (PL), enunciado na *forma padrão*, é o de encontrar um vetor coluna $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ tal que $\boldsymbol{Ax} = \boldsymbol{b}$ e $x_i \geq 0$, para todo $i \in \{1, \ldots, n\}$, e que a forma linear, $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$, denominada *função objetivo*, atinja seu valor mínimo. Um *programa linear* é uma instância do problema geral da programação linear.

Comumente, um programa linear é descrito na forma:

minimizar

$$\sum_{i=1}^{n} c_i \cdot x_i \tag{5.1}$$

sujeito a

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad \text{para todo } i = 1, \ldots, m \tag{5.2}$$

e

$$x_i \geq 0 \quad \text{para todo } i = 1, \ldots, n\,, \tag{5.3}$$

onde $(c_1, \ldots, c_n)$ e $(b_1, \ldots, b_m)$ são as coordenadas dos vetores $\boldsymbol{c}$ e $\boldsymbol{b}$, respectivamente, e a_{ij} é o elemento da i-ésima linha e j-ésima coluna da matriz $\boldsymbol{A}$, com $i = 1, \ldots, m$ e $j = 1, \ldots, n$.

Praticantes da PL têm por costume denominar qualquer vetor $\boldsymbol{x}$ de *solução*, mesmo que as restrições (5.2) e (5.3) do problema não sejam satisfeitas. Quando $\boldsymbol{x}$ satisfaz as restrições (5.2) e (5.3), dizemos que $\boldsymbol{x}$ é uma *solução viável*. Uma solução viável não necessariamente minimiza a função objetivo (5.1). Quando isto acontece, dizemos que a solução viável é *ótima*. Portanto, do ponto de vista matemático, o que os praticantes de PL tradicionalmente denominam solução viável ótima é o que, normalmente, entendemos por "solução".

Alguns problemas simplesmente não possuem soluções viáveis. Se este for o caso, o próprio problema é dito *inviável*. Um problema inviável se caracteriza pelo fato do conjunto

$$F = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \sum_{j=1}^{n} a_{ij}x_j = b_i \text{ e } x_j \geq 0, \text{ com } i = 1, \ldots, m,\ j = 1, \ldots, n \right\}$$

ser vazio. Um problema que não é inviável é dito *viável* ou *solúvel*. Em outras palavras, um problema é dito viável se F não é o conjunto vazio ou, de forma equivalente, se ele possui uma solução viável. Como já vimos, nem toda solução viável é ótima. Em particular, um problema viável pode não apresentar nenhuma solução ótima. Isto ocorre quando todos os vetores $\boldsymbol{x}$ em F estão associados a custos arbitrariamente negativos. Formalmente, para todo número ζ, existe $\boldsymbol{x}$ em F tal que $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} < \zeta$. Neste caso, dizemos que o problema é *ilimitado*.

A forma como definimos um programa linear nos diz que ele é um problema de otimização *contínua*, com uma quantidade não-enumerável de soluções viáveis em F. No entanto, alguns fatos importantes fazem com que um programa linear tenha natureza combinatória. O primeiro deles é que F é um subconjunto convexo do $\mathbb{R}^n$. Isto significa que F é igual ao fecho convexo de seus pontos extremos. O segundo fato é que se F não for ilimitado nem vazio, então a função objetivo (5.1) assume seu valor mínimo em um ponto extremo de F.

O terceiro fato é que há apenas um número finito de pontos extremos em todo conjunto convexo (ilimitado ou não), o que faz com que o problema tenha natureza combinatória.

As observações acima indicam que uma solução ótima pode ser encontrada se formos capazes de enumerar todos os pontos extremos de F (quando este conjunto não for vazio nem ilimitado), que é um número finito. No entanto, qualquer algoritmo que resolva qualquer instância do problema geral de PL deve ser capaz de *decidir* se F é vazio ou ilimitado. Pois, nesses dois casos, não há solução ótima. O revolucionário algoritmo *Simplex*, desenvolvido por George Dantzig (DANTZIG, 1963), é capaz de resolver esse problema de decisão e, se F não for vazio nem ilimitado, ele encontra uma solução ótima, reduzindo o programa linear original a um problema equivalente que pode ser resolvido por mera inspeção.

Mais especificamente, o Simplex devolve uma solução viável *básica*, que é ótima, quando o programa linear for solúvel e limitado. Uma solução viável, $\boldsymbol{x}$, é dita ser *básica* quando houver não mais do que m coordenadas positivas em $\boldsymbol{x}$ (isto é, quando pelo menos $n - m$ coordenadas de $\boldsymbol{x}$ forem zero). O Teorema Fundamental da Programação Linear garante que se uma solução ótima existir, então uma solução básica ótima existe. Por outro lado, quando uma solução ótima não existir, o programa linear é inviável ou ilimitado.

5.1.1 Forma geral

Na prática, muitos programas lineares são declarados na *forma geral*:

minimizar

$$\sum_{i=1}^{n} c_i \cdot x_i$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad \text{para } i \in M_1\,,$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \text{para } i \in M_2\,,$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{para } i \in M_3$$

e

$$x_i \geq 0 \quad \text{para } i \in N$$

onde

$$M_1 \cap M_2 = M_1 \cap M_3 = M_2 \cap M_3 = \emptyset,\ M_1 \cup M_2 \cup M_3 = \{1, \ldots, m\} \text{ e } N \subseteq \{1, \ldots, n\}\,.$$

Isto é, as restrições envolvendo $\boldsymbol{A}$, $\boldsymbol{x}$ e $\boldsymbol{b}$ podem ser igualdades ou desigualdades e não há restrição sobre um subconjunto (possivelmente vazio) de coordenadas do vetor $\boldsymbol{x}$. Felizmente, a forma mais geral de um programa linear pode ser convertida na forma padrão.

A ideia é nos livrarmos das desigualdades definindo variáveis de *folga*. Isto é, para cada

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i\,,$$

com $i \in M_2$, definimos a variável w_i tal que

$$\sum_{j=1}^{n} a_{ij}x_j - w_i = b_i\,.$$

Analogamente, para cada

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i\,,$$

com $i \in M_3$, definimos a variável w_i tal que

$$\sum_{j=1}^{n} a_{ij}x_j + w_i = b_i\,.$$

Em seguida, acrescentamos as restrições $w_i \geq 0$, para todo $i \in M_2 \cup M_3$, ao problema. Finalmente, para cada coordenada, x_j, com $j = \{1, \ldots, n\} - N$, nós definimos duas variáveis, x_j^+ e x_j^-, e substituimos toda ocorrência de x_j pelo lado direito da igualdade $x_j = x_j^+ - x_j^-$. Em seguida, adicionamos as restrições de sinal, $x_j^+ \geq 0$ e $x_j^- \geq 0$, ao problema.

Na prática, alguns problemas requerem a maximização da função objetivo. Do ponto de vista matemático, a maximização de uma função ζ é o mesmo que a minimização da função $-\zeta$. Então, a discussão que apresentamos até o presente momento serve para os dois casos.

5.1.2 Dualidade

Todo programa linear está associado a um outro, denominado seu *dual*. O dual do dual do programa linear original é o próprio programa linear original, que, por causa disso, é

denominado *programa linear primal*. Mais especificamente, dado um problema de PL na forma padrão,

minimizar

$$\sum_{i=1}^{n} c_i \cdot x_i$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad \text{para todo } i = 1, \ldots, m$$

e

$$x_i \geq 0 \quad \text{para todo } i = 1, \ldots, n\,,$$

o *programa linear dual* associado é dado por

maximizar

$$\sum_{i=1}^{m} b_i \cdot y_i \tag{5.4}$$

sujeito a

$$\sum_{i=1}^{m} y_i a_{ij} \geq c_j \quad \text{para todo } j = 1, \ldots, n \tag{5.5}$$

e

$$y_i \geq 0 \quad \text{para todo } i = 1, \ldots, m\,. \tag{5.6}$$

Um teorema clássico de Programação Linear, denominado Teorema da Dualidade (versão forte), estabelece que se o problema primal possui uma solução ótima, $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$, então o problema dual associado possui uma solução ótima, $\boldsymbol{y}^* = (y_1^*, \ldots, y_m^*)$, tal que

$$\sum_{i=1}^{n} c_i x_i^* = \sum_{j=1}^{m} b_j y_j^*\,.$$

Além disso, se um dos problemas não possui soluções viáveis, o outro é ilimitado e vice-versa.

A utilização prática das duas formas de um programa linear é motivada por vários fatores. Em geral, uma das formas revela informações importantes sobre a solução ótima da outra. Além disso, alguns algoritmos, tais como aqueles para calcular emparelhamentos de custo máximo, que veremos em breve, fazem uso das formas primal e dual simultane-

amente.

5.2 Grafos bipartidos

Como forma de motivar a nossa discussão sobre emparelhamentos em grafos ponderados arbitrários, introduzimos o problema de calcular emparelhamentos de custo máximo em grafos bipartidos ponderados. Dado um grafo bipartido $G = (V, E)$ (veja a Definição 4.2.1) e uma função de custo associada às arestas, $c : E \to \mathbb{R}$, na qual $c(e) \geq 0$, para toda aresta $e \in E$, o nosso objetivo é obter um emparelhamento de custo máximo, em G, com respeito a c. Como vimos no Capítulo 2, um emparelhamento, M, em G é de custo máximo com respeito c quando a soma $\sum_{e \in M} c(e)$ é máxima entre todos os emparelhamentos (máximos ou não) em G.

Para resolver o problema, nós modificamos (se necessário) o grafo original, G, e calculamos um emparelhamento *perfeito* de custo máximo no grafo modificado. Em seguida, convertemos este emparelhamento em um emparelhamento de custo máximo em G. Para modificar G, consideramos uma partição $\{U, W\}$ do conjunto, V, de vértices de G tal que toda aresta de G seja incidente a um vértice em U e outro em W. Em seguida, se $|U| \neq |W|$, então inserimos vértices extras no conjunto de menor cardinalidade de tal forma que as cardinalidades resultantes sejam as mesmas. Finalmente, acrescentamos arestas de custo zero conectando vértices dos dois conjuntos, de tal forma que todo vértice de um conjunto esteja conectado a um vértice do outro conjunto por uma aresta. Note que o número de vértices do grafo modificado é par e que ele sempre admite um emparelhamento perfeito.

O nosso objetivo é obter um emparelhamento *perfeito* de custo máximo no grafo *modificado*. Para simplificar, vamos denotar, também, por U e W os elementos da partição do conjunto de vértices do grafo modificado. Além disso, vamos supor que a cardinalidade desses elementos seja n. Então, o nosso problema pode ser formulado como um problema de programação linear. Para tanto, os conjuntos de vértices, U e W, do grafo modificado são enumerados de forma que $U = u_1, u_2, ..., u_n$ e $W = w_1, w_2, ..., w_n$ e as arestas são enumeradas de forma que a aresta $e_k = u_i w_j = u_j w_i$ recebe o índice $k = i + (j - 1) \cdot n$. Com isso, um emparelhamento no grafo modificado pode ser representado por um vetor, $\boldsymbol{x}$ tal que

$$\boldsymbol{x} = (x_1, \ldots, \ldots, x_{n^2})$$

e $x_k = 1$ se a aresta e_k está no emparelhamento e $x_k = 0$, caso contrário, com $k = 1, \ldots, n^2$.

Logo,

$$\sum_{k=1}^{n^2} c_k \cdot x_k \,,$$

onde c_k é o valor da função c na aresta e_k, é o custo do emparelhamento. Como o vetor $\boldsymbol{x}$ deve representar um emparelhamento perfeito no grafo modificado, temos a restrição

$$\sum_{k=1}^{n^2} a_{ik}x_k = 1, \text{ para todo } i = 1, \ldots, n \,,$$

onde $\boldsymbol{A} = (a_{ik})$ é a *matriz de incidência* do grafo modificado, isto é, uma matriz $n \times n^2$ tal que

$$a_{ik} = \begin{cases} 1 & \text{se o vértice } u_i \text{ é incidente na aresta } e_k \\ 0 & \text{caso contrário} \,. \end{cases}$$

Usando a notação acima, podemos definimos um problema de PL, na forma geral, equivalente ao problema de emparelhamento de custo máximo ou de emparelhamento perfeito de custo máximo. O único "porém" é a restrição sobre as coordenadas do vetor $\boldsymbol{x}$, que devem pertencer ao conjunto $\{0, 1\}$. Infelizmente, não podemos representar esta restrição usando igualdades ou desigualdades. O melhor que podemos fazer é definir as desigualdades $0 \leq x_k \leq 1$, para todo $k = 1, \ldots, n^2$. Na verdade, precisamos apenas de $0 \leq x_k$, pois a desigualdade $x_k \leq 1$ decorre de $0 \leq x_k$ e da restrição $\sum_{k=1}^{n^2} a_{ik}x_k = 1$, para todo $i = 1, \ldots, n$.

Pelo exposto acima, tudo que podemos fazer é definir um programa linear na forma geral

minimizar

$$\sum_{k=1}^{n^2} c'_k \cdot x_k \tag{5.7}$$

sujeito a

$$\sum_{k=1}^{n^2} a_{ik}x_k = 1, \text{ para todo } i = 1, \ldots, n \,, \tag{5.8}$$

e

$$x_i \geq 0 \quad \text{para todo } i = 1, \ldots, n \,, \tag{5.9}$$

onde

$$c'_k = P - c_k \quad \text{e} \quad P = \max\{c_k \mid k = 1, \ldots, n^2\} \,,$$

pois desejamos obter o emparelhamento perfeito de custo máximo no grafo modificado.

O programa linear que acabamos de definir parece permitir a obtenção de uma solução ótima "fracionária", ou seja, um vetor $\boldsymbol{x}$ com elementos maiores que 0 e menores que 1. Isto seria completamente indesejável, pois tal solução não poderia ser interpretada como um emparelhamento. Por exemplo, o vetor $\boldsymbol{x}$ de coordenadas $x_1 = x_2 = x_2 = x_4 = 0.5$ é uma solução viável ótima para o programa linear associado ao grafo ponderado da Figura 5.1.

Obviamente, uma solução "fracionária" é extremamente indesejável, pois ela não descreve um emparelhamento. Felizmente, podemos provar que todos os pontos extremos do conjunto, F, consistindo dos vetores $\boldsymbol{x}$ que satisfazem as restrições (5.8) e (5.9) do problema, são pontos em $\{0,1\}^{n^2}$. Logo, há pelo menos uma solução ótima viável que consiste apenas de coordenadas 0 e 1. Mais ainda, há uma solução viável básica que consiste apenas de coordenadas 0 e 1. Para chegarmos a esta conclusão, usamos a propriedade de que o determinante de toda submatriz quadrada de $\boldsymbol{A}$ é igual a 0, 1 ou -1 (LOVÁSZ; PLUMMER, 1986).

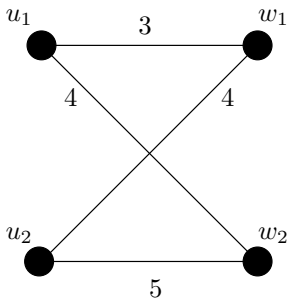


Figura 5.1: Um grafo bipartido ponderado cujo PL associado admite solução fracionária.

Uma vez que um emparelhamento perfeito de custo máximo, M, seja obtido no grafo modificado, nós retiramos os vértices e arestas adicionais deste grafo para obter o grafo, G, original. Em seguida, definimos um emparelhamento, M', no grafo original que consiste das arestas de M com os dois vértices em G. Por definição, as arestas que não constam em M estão associadas a custo zero, o que faz com que o custo total de M' seja igual ao de M.

É fácil ver que M' é de custo máximo em G. De fato, se M' não fosse de custo máximo, então existiria outro emparelhamento, M'', em G com custo maior do que o de M'. Com isso, poderíamos considerar M'' e aumentá-lo para construir um emparelhamento perfeito no grafo modificado. Como todas as arestas que adicionarmos a M'' possuem custo não-negativo, obteríamos um emparelhamento perfeito no grafo modificado com custo maior

do que o de M. Mas, como M é um emparelhamento perfeito de custo máximo no grafo modificado, o novo emparelhamento que obtemos não pode existir e, portanto, M' tem custo máximo.

Usando a formulação anterior e um método de solução baseado nas formas primal e dual do problema, Harold Kuhn (KUHN, 1955) propôs um algoritmo, denominado de *busca húngara*[1], para calcular emparelhamentos máximos de custo máximo em grafos bipartidos. A complexidade do algoritmo de Kuhn é $\mathcal{O}(|V|^2|E| + |V|^3)$, onde V e E são os conjuntos de vértices e arestas do grafo modificado, respetivamente. Muitos outros algoritmos foram propostos depois para o mesmo problema. Uma ampla discussão de algoritmos sequenciais e paralelos para calcular emparelhamentos (máximos) de custo máximo em grafos *bipartidos* pode ser encontrada na dissertação de mestrado em (SAIP, 1993).

5.3 Grafos gerais

Seja $G = (V, E)$ um grafo geral e $c : E \to \mathbb{R}$ uma função de custo associada às arestas de G tal que $c(e) \geq 0$, para toda aresta $e \in E$. Levando em conta o que fizemos na seção anterior, podemos supor que o grafo G é "completo" e que possui um número par de vértices. Caso contrário, podemos aumentar G com arestas de custo zero e um vértice extra. Finalmente, tomando como base o programa linear utilizado para o caso bipartido, o problema do emparelhamento *perfeito* de custo máximo pode ser formulado como o de minimizar

$$\sum_{k=1}^{|E|} c'_k \cdot x_k$$

sujeito a

$$\sum_{k=1}^{|E|} a_{ik} \cdot x_k = 1, \text{ para todo } i = 1, \ldots, |V|$$

e

$$x_k \geq 0, \text{ para todo } k = 1, \ldots, |E|\,.$$

Apesar da semelhança com o caso bipartido, a formulação acima admite soluções básicas ótimas "fracionárias". Para piorar a situação, também não se pode garantir que exista pelo menos uma solução viável básica que não seja fracionária. Logo, a formulação para grafos bipartidos é inadequada para resolver o problema do emparelhamento perfeito de custo máximo com PL.

[1] Uma homenagem aos matemáticos húngaros Denis König e Eugene Egerváry.

Outra grande contribuição de Jack Edmonds para a teoria de emparelhamentos em grafos arbitrários foi a formulação de um programa linear para o problema do emparelhamento (perfeito) de custo máximo que admite soluções ótimas básicas no conjunto $\{0,1\}^{|E|}$ (EDMONDS, 1965a). Mais uma vez, ele notou que os ciclos de cardinalidade ímpar (os "botões") são os obstáculos para estendermos para grafos gerais o programa linear formulado para grafos bipartidos. Como grafos bipartidos não possuem ciclos de cardinalidade ímpar, o problema do emparelhamento pode ser resolvido com a formulação vista na Seção 5.2.

Para lidar com a presença de ciclos de cardinalidade ímpar, Edmonds adicionou variáveis extras à formulação do problema linear. Cada uma dessas variáveis corresponde a um subconjunto de V com cardinalidade ímpar maior do que 1. Mais especificamente, seja

$$S = \{B \subseteq V \mid |\mathrm{B}| \text{ é ímpar e } |\mathrm{B}| > 1\}\,.$$

Então, consideramos uma enumeração, $B_1, \ldots, B_{|S|}$, dos elementos de S e definimos uma variável, $z_i \in \mathbb{R}$, correspondente ao elemento $B_i \in S$, para todo $i = 1, \ldots, |S|$. Note que $|S| = 2^{|V|-1} - |V| = \Theta(2^{|V|})$, pois assumimos que o grafo G tem um número par de vértices[2], $|V|$.

Assim como antes, representamos o emparelhamento por um vetor, $\boldsymbol{x} \in \{0,1\}^{|E|}$, e a soma

$$\sum_{k=1}^{|E|} c_k \cdot x_k$$

é o custo do emparelhamento. Como queremos maximizar o custo, reescrevemos esta soma como

$$\sum_{k=1}^{|E|} c'_k \cdot x_k\,,$$

onde $c'_k = P - c(e_k) \quad \text{e} \quad P = \max\{c(e_k) \mid k = 1, \ldots, |E|\}$ e buscamos um vetor $\boldsymbol{x}$ que a minimize.

Para definir as restrições, consideramos uma enumeração, $v_1, \ldots, v_{|V|}$, dos vértices de V e uma enumeração, $e_1, \ldots, e_{|E|}$, das arestas de E. Em seguida, definimos dois conjuntos de índices, M_1 e M_2, tais que $M_1 = \{1, \ldots, |V|\}$ e $M_2 = \{|V|+1, \ldots, |V|+|S|\}$. Finalmente, definimos a matriz $\boldsymbol{A} = (a_{ij})$ com $|M_1| + |M_2|$ linhas e $|E|$ colunas tal que: (1) para $i \in M_1$ e $j = 1, \ldots, |E|$, temos $a_{ij} = 1$ se o vértice v_i é incidente na aresta e_j e $a_{ij} = 0$ caso contrário; (2) para $i \in M_2$ e $j = 1, \ldots, |E|$, temos $a_{ij} = 1$ se o conjunto

[2]Um número exponencial de restrições não impediu o desenvolvimento de um algoritmo polinomial.

$B_i \in S$ contém a aresta e_j e $a_{ij} = 0$ caso contrário. Agora, escrevemos três conjuntos de restrições:

$$\sum_{k=1}^{|E|} a_{ik} \cdot x_k = 1, \text{ para todo } i \in M_1\,,$$

$$\sum_{k=1}^{|E|} a_{ik} \cdot x_k \leq \left\lfloor \frac{|B_i|}{2} \right\rfloor, \text{ para todo } i \in M_2$$

e

$$x_k \geq 0, \text{ para todo } k = \{1, \ldots, |E|\}.$$

Como podemos constatar, a "novidade" está no segundo conjunto de restrições, que, intuitivamente, obriga o número de arestas do emparelhamento em cada ciclo de cardinalidade ímpar a ser, no máximo, igual ao piso da metade do número de vértices do ciclo. O problema então é

minimizar

$$\sum_{k=1}^{|E|} c'_k \cdot x_k \tag{5.10}$$

sujeito a

$$\sum_{k=1}^{|E|} a_{ik} \cdot x_k = 1, \text{ para todo } i \in M_1\,, \tag{5.11}$$

$$\sum_{k=1}^{|E|} a_{ik} \cdot x_k \leq \left\lfloor \frac{|B_i|}{2} \right\rfloor, \text{ para todo } i \in M_2 \tag{5.12}$$

e

$$x_k \geq 0, \text{ para todo } k = \{1, \ldots, |E|\}. \tag{5.13}$$

Para resolver o problema acima, Edmonds utilizou um método baseado nas formas primal e dual do problema. As variáveis extras que ele definiu aparecem na forma dual abaixo:

maximizar

$$\sum_{i \in M_1} y_i + \sum_{h=1}^{|S|} \left\lfloor \frac{|B_h|}{2} \right\rfloor \cdot z_h \tag{5.14}$$

sujeito a

$$y_i + y_j + \sum_{h=1, \{v_i, v_j\} \subseteq B_h}^{|S|} z_h \geq c'_k, \text{ para toda aresta } e_k = v_i v_j \in E \tag{5.15}$$

e

$$z_h \geq 0, \text{ para todo } h = 1, \ldots, |S|. \tag{5.16}$$

Compare a restrição (5.15) com aquela da definição da forma dual em (5.5). Se considerarmos apenas as M_1 primeiras linhas da matriz $\boldsymbol{A}$, temos que apenas os elementos das linhas i e j da coluna k, onde k é o índice da aresta $e_k = v_i v_j$, possuem valor 1. Os demais possuem valor zero. Se considerarmos as M_2 linhas restantes da matriz $\boldsymbol{A}$, temos que apenas os elementos das linhas h da coluna k, onde k é o índice da aresta $e_k = v_i v_j$ e $\{i, j\} \subseteq B_h$, possuem valor 1. Os demais possuem valor zero. Isto explica porque a soma em (5.5),

$$\sum_{i=1}^{m} y_i a_{ij} \geq c_j \quad \text{para todo } j = 1, \ldots, n$$

se torna

$$y_i + y_j + \sum_{h=1, \{v_i, v_j\} \subseteq B_h}^{|S|} z_h \geq c'_k, \text{ para toda aresta } e_k = v_i v_j \in E\,.$$

5.3.1 O método primal-dual

Edmonds estendeu seu algoritmo de contração de botões, usando um método primal-dual, para calcular o vetor $\boldsymbol{x}$. De forma geral, o algoritmo de Edmonds mantém uma solução (não necessariamente viável) $\boldsymbol{x}$ para a forma primal do problema, mas mantém uma solução viável, $(\boldsymbol{y}, \boldsymbol{z})$, para a forma dual. Em particular, a solução $\boldsymbol{x}$ pode violar a restrição (5.11). Durante a execução do algoritmo, as duas soluções, $\boldsymbol{x}$ e $(\boldsymbol{y}, \boldsymbol{z})$, são ajustadas até que ambas se tornem soluções viáveis ótimas, o que necessariamente deve ocorrer.

5.3.1.1 Inicialização

O algoritmo começa com um emparelhamento, M, arbitrário em G, que satisfaz, pelo menos, (5.12) e (5.13). Cada vértice $v_i \in V$ tem sua variável y_i, denominada *potencial de* v_i, associada com ele e todas as variáveis z_h, com $h = 1, \ldots, |S|$, iguais a zero. O valor dos potenciais são escolhidos de tal forma que as restrições (5.15) sejam válidas. Note que (5.16) são válidas. Além disso, duas outras condições, denominadas de condições de folga complementares,

$$x_k > 0 \Longrightarrow \pi_k = 0, \text{ para todo } k = 1, \ldots, |E|\,, \tag{5.17}$$

onde

$$\pi_k = y_i + y_j - c_k + \sum_{h=1,\{v_i,v_j\}\subseteq B_h}^{|S|} z_h$$

é o *custo reduzido* da aresta e_k, e

$$z_h > 0 \Longrightarrow \sum_{h=1,\{v_i,v_j\}\subseteq B_h}^{|S|} z_h = \left\lfloor \frac{|B_h|}{2} \right\rfloor \text{ para todo } B_h \in S\,, \tag{5.18}$$

devem ser satisfeitas. A condição (5.17) nos diz que toda aresta e_k com $x_k > 0$ deve ter custo reduzido igual a zero, enquanto a condição (5.18) nos diz que um número máximo de arestas de B_h devem pertencer ao emparelhamento. Mais adiante, o propósito dessas condições de folga complementares se tornará evidente. O algoritmo opera em fases. Em cada fase, dois vértices expostos se tornam ocupados e, portanto, não mais violam a restrição (5.11). Logo, após $\mathcal{O}(|V|)$ fases, um emparelhamento perfeito de custo máximo é encontrado.

5.3.1.2 Fases

Como no algoritmo para emparelhamento de cardinalidade máxima de Edmonds, o presente algoritmo constrói árvores alternantes. As árvores são construídas em paralelo (e não uma por vez). Cada árvore alternante, T_r, tem raiz em um vértice exposto, r. As arestas de T_r são todas "justas", ou seja, possuem custo reduzido igual a zero. Além disso, todo caminho do vértice raiz, r, até um vértice folha alterna arestas do emparelhamento com arestas que não pertencem ao emparelhamento. Os vértices de T_r são rotulados como pares ou ímpares: um vértice u é rotulado *par* (resp. *ímpar*) quando o caminho de u a r tiver comprimento par (resp. ímpar). Vértices que não fazem parte de nenhuma árvore são ocupados e denominados de não-rotulados. Seguindo (MEHLHORN; SCHÄFER, 2002), usaremos a notação u^+, u^- e $u^\emptyset$ para denotar um vértice u par, ímpar e não-rotulado, respectivamente.

Uma árvore alternante cresce a partir de um vértice par, $u^+ \in T_r$. Quando existe uma aresta justa, uv, conectando u a um vértice $v^\emptyset$, a aresta uv e a aresta do emparelhamento, vw, que deve existir, pois v é não-rotulado (e, portanto, ocupado), são acrescentadas a T_r. Em seguida, os vértices v e w são rotulados como ímpar e par, respectivamente. Quando uma aresta, uv, justa, com $u^+ \in T_r$, $v^+ \in T'_r$ e $T_r \neq T'_r$, existe, um caminho aumentante de r para r' é descoberto. Este caminho consiste da concatenação do caminho de r a u com a aresta uv e o caminho de v a r'. O emparelhamento atual é então aumentado com

relação ao novo caminho aumentante. Depois disso, todos os vértices de T_r e T'_r estão ocupados. Com isso, as árvores T_r e T'_r podem ser destruídas e seus vértices considerados não-rotulados.

O último caso a ser considerado ocorre quando uma aresta, uv, justa e incidente em dois vértices pares, $u^+ \in T_r$ e $v^+ \in T_r$, de uma mesma árvore existe. Este caso configura a presença de um botão. Então, como esperado, seguimos os caminhos de u e v até a raiz da árvore, buscando encontrar o primeiro ancestral comum de u e v, digamos w. Por construção de T_r, o vértice w deve ser par. O subconjunto B, de V, é o botão definido por todos os vértices do ciclo $w, \ldots, u, v, \ldots, w$. Assim como no algoritmo para calcular emparelhamentos de cardinalidade máxima, o botão é contraído, formando um único pseudo-vértice *par*, b_w. Em seguida, todas as arestas com dois vértices em B são removidas do grafo e cada aresta cd, com $c \in B$ e $d \notin B$, é substituída pela aresta cb_w. A partir daí, o algoritmo segue tentando aumentar a árvore atual e as demais árvores existentes.

O que foi descrito até então pode ser visto como o algoritmo de Edmonds para calcular emparelhamentos de cardinalidade máxima no grafo formado por arestas justas. Então, o único problema que temos é que o algoritmo pode "parar" por falta de arestas justas. Neste momento, o algoritmo faz um ajuste das variáveis da forma dual, como descrito a seguir.

5.3.1.3 Ajuste dual

A solução dual, $(\boldsymbol{y}, \boldsymbol{z})$, é ajustada de tal forma que o valor da soma (5.14) da forma dual cresça. Além disso, a solução $(\boldsymbol{y}, \boldsymbol{z})$ deve permanecer viável e satisfazer as condições de folga complementares (5.17) e (5.18). Mais especificamente, todas as variáveis y_i, para cada vértice $i = 1, \ldots, |V|$, e as variáveis z_h, correspondentes a cada botão *não-trivial* (isto é, com mais de um vértice) e que não esteja contido em outro botão, são atualizadas como segue:

$$y_i \leftarrow y_i + \sigma\delta \quad \text{e} \quad z_h \leftarrow z_h - 2\sigma\delta$$

onde δ é um valor positivo e σ, denominado *indicador de status*, é igual a 1 ou -1 para botões (triviais ou não) rotulados ímpares e pares, respectivamente, ou igual a 0, caso contrário.

Sejam $T_{r_1}, \ldots, T_{r_l}$ as árvores alternantes atuais. Então, o valor δ é definido como igual

a

$$\delta = \min\{\delta_1, \delta_2, \delta_3\}$$

com

$$\begin{aligned}
\delta_1 &= \min_{k \in E}\{\pi_k \mid e_k = uv,\ u^+ \in T_{r_i} \text{ e } v^\emptyset \text{ não está em nenhuma árvore}\}, \\
\delta_2 &= \min_{k \in E}\left\{\frac{\pi_k}{2} \mid u^+ \in T_{r_i} \text{ e } v^+ \in T_{r_j}\right\}, \\
\delta_3 &= \min_{B_h \in S}\left\{\frac{z_h}{2} \mid B_h^- \in T_{r_i}\right\},
\end{aligned}$$

onde T_{r_i} e T_{r_j} denotam qualquer árvore alternante. Por convenção, o mínimo de um conjunto vazio é definido como sendo ∞. Quando $\delta = \infty$, o programa linear dual é ilimitado e, pelo Teorema da Dualidade, o programa primal não possui nenhuma solução viável. Como nosso grafo sempre admite um emparelhamento perfeito, isso não ocorrerá aqui.

O valor escolhido para δ garante que uma aresta se tornará justa ou que o valor de alguma variável z_h será zero, onde $B_h \in S$ contém os vértices de um botão contraído para um vértice rotulado ímpar. Devido à restrição (5.16), a variável z_h não deve participar de nenhum ajuste futuro. Além disso, o botão correspondente a B_h é expandido e os vértices resultantes da expansão são adicionados à árvore contendo o vértice para o qual o botão foi contraído.

5.3.1.4 Complexidade de tempo

A complexidade de tempo obtida por Edmonds para o algoritmo que acabamos de descrever é $\mathcal{O}(|V|^4)$. Entretanto, esta complexidade pode ser reduzida com o uso apropriado de algumas estruturas de dados. Harold Gabow apresentou um algoritmo com complexidade de tempo $\mathcal{O}(|V| \cdot (|E| + |V| \log |V|))$, que permanece como o melhor algoritmo conhecido até o presente momento (GABOW, 1990). Há também várias implementações eficientes do algoritmo descrito aqui, assim como de algumas variantes (EDMONDS; JOHNSON; LOCKHART, 1969; COOK; ROHE, 1999; MEHLHORN; SCHÄFER, 2002; KOLMOGOROV, 2009).

5.4 Considerações finais

Neste trabalho, nós não implementamos nenhum algoritmo para calcular emparelhamentos em grafos ponderados. Ao invés disso, nós usamos implementações disponíveis

gratuitamente na biblioteca LEMON[3]. Esta biblioteca possui funções que nos permitem calcular emparelhamentos perfeitos de custo máximo e emparelhamentos de custo máximo nos grafos duais das triangulações das superfícies que consideramos no trabalho. Como tais grafos sempre admitem um emparelhamento perfeito, o problema sempre admitirá solução.

A decisão por não implementar o algoritmo aqui descrito se deve meramente a restrições do cronograma. Inicialmente, a nossa intenção era justamente implementar um algoritmo, como forma de sedimentar os conceitos estudados. No entanto, vimos que tal implementação exigiria bastante tempo e, quando percebemos isso, o tempo que restava para finalizar o trabalho estava curto e a implementação poderia comprometer a escrita da monografia.

[3] http://lemon.cs.elte.hu

6 Resultados

Este capítulo apresenta, compara e discute os resultados da aplicação da abordagem gulosa e da abordagem baseada em emparelhamento em grafos para converter triangulações de superfícies em $\mathbb{E}^3$ em quadrilaterizações. A Seção 6.1 descreve como os resultados foram gerados e comparados. A Seção 6.2 apresenta uma métrica para avaliar a qualidade de forma de cada quadrilátero de uma quadrilaterização de superfície. Esta métrica foi utilizada para comparar os resultados. A Seção 6.3 fornece várias estatísticas obtidas dos resultados gerados pelas abordagens de conversão utilizadas e com respeito à métrica definida na seção anterior. Por último, a Seção 6.4 oferece uma interpretação dos dados estatísticos calculados, efetuando também uma comparação de desempenho entre os algoritmos.

6.1 Considerações iniciais

Um dos dois objetivos gerais deste trabalho (veja a Seção 1.2) é a comparação, experimental, da abordagem baseada em emparelhamentos em grafos com a abordagem gulosa de Luiz Velho (VELHO, 2000) para converter triangulações de superfícies em $\mathbb{E}^3$ em quadrilaterizações. Para realizar esta comparação, nós consideramos um conjunto de superfícies bastante representativo do tipo de superfície encontrado em trabalhos da área de Processamento Gráfico. Além disso, nós implementamos o algoritmo `TriQuad()`, descrito na Seção 3.5, que utiliza a abordagem gulosa desenvolvida por Luiz Velho (VELHO, 2000).

A comparação que realizamos envolveu o algoritmo `TriQuad()` e mais três algoritmos de conversão baseados em emparelhamento em grafos. Mais especificamente, as conversões baseadas em emparelhamento em grafos utilizam (1) o algoritmo para calcular emparelhamentos perfeitos do Capítulo 4, (2) o algoritmo para calcular o emparelhamento perfeito de custo máximo do Capítulo 4 e (3) o algoritmo para calcular o emparelhamento de custo máximo do Capítulo 4, que é o mesmo de (2). Lembre-se de que quando o al-

goritmo `TriQuad()` ou o emparelhamento de custo máximo são usados na conversão, a subdivisão resultante pode conter triângulos isolados. Se este for o caso, um passo de subdivisão dos triângulos isolados e quadriláteros deve ser executado para obtermos uma quadrilaterização.

Para comparar os quatro algoritmos, nós usamos uma métrica definida em (DANIELS; NONATO; SIQUEIRA; LIZIER; SILVA, 2011) e que determina a "qualidade" do quadrilátero formado pelo emparelhamento de dois triângulos adjacentes. Esta métrica combina duas medidas: planaridade e ortogonalidade. A primeira delas diz respeito a quão planar é o quadrilátero resultante, enquanto a segunda diz respeito a quão próximos de 90^o são os ângulos do quadrilátero. Quanto mais planar e mais ortogonal for um quadrilátero, maior é a sua qualidade de forma. Com esta métrica, nós comparamos os resultados das quadrilaterizações dos quatro algoritmos mencionados anteriormente.

É importante ressaltar que o custo associado a uma aresta do grafo dual ponderado nas abordagens (2) e (3) é o valor da métrica de qualidade para o quadrilátero formado pela união dos dois triângulos incidentes na aresta. Isto é, as abordagens (2) e (3) maximizam a função objetivo do programa linear usando a própria métrica utilizada na comparação dos resultados. Em contraste, a abordagem gulosa utiliza a heurística da aresta mais longa para escolher quais triângulos emparelhar. Logo, a nossa comparação revela o quanto a heurística da aresta mais longa é efetiva em relação à métrica de planaridade e distorção. Para conhecer um pouco mais sobre a eficácia da abordagem gulosa, nós também a utilizamos com o valor de medida de planaridade e distorção (ao invés do comprimento da aresta).

Finalmente, nós comparamos todas as abordagens com respeito ao (1) número de quadriláteros gerados, (2) número de *doublets* e (3) número de triângulos isolados produzidos durante a geração da quadrilaterização. Obviamente, o número de quadriláteros gerados é o mesmo para as abordagens baseadas em emparelhamentos perfeitos e, muito provavelmente, menor do que o número de quadriláteros gerados pela abordagem gulosa e pela abordagem baseada em emparelhamento de custo máximo, pois aquelas não sofrem do problema de triângulos isolados. Por outro lado, todas elas podem gerar *doublets*, que são pares de quadriláteros "degenerados" que compartilham duas arestas consecutivas. Cada quadrilátero do par (um *doublet*) tem a forma de um triângulo e um ângulo próximo a 180^o. Esses quadriláteros são extremamente indesejáveis em aplicações numéricas, por exemplo.

6.2 Métrica de planaridade e ortogonalidade

A métrica de qualidade de forma de quadrilátero usada neste trabalho foi definida em (DANIELS; NONATO; SIQUEIRA; LIZIER; SILVA, 2011) e é, formalmente, descrita através de uma função, $m : \mathcal{Q} \to [0, 1] \subset \mathbb{R}$, onde $\mathcal{Q}$ é o conjunto de todos os quadriláteros de uma quadrilaterização. Mais epecificamente, a cada quadrilátero $q \in \mathcal{Q}$, a função m associa um valor real contido no intervalo $[0, 1]$:

$$m(q) = \frac{p(q) + o(q)}{2},$$

onde

$$p : \mathcal{Q} \to [0, 1] \subset \mathbb{R} \quad \text{e} \quad o : \mathcal{Q} \to [0, 1] \subset \mathbb{R}$$

são funções que especificam o grau de planaridade e o de ortogonalidade do quadrilátero, q.

Para definir as funções p e o, suponha que o quadrilátero q tenha sido gerado pelo emparelhamento dos triângulos, t_α e t_β, com aresta comum $e_{\alpha\beta}$ (veja Figura 6.1). Então, temos

$$p(q) = \frac{1 + \boldsymbol{n}_\alpha \cdot \boldsymbol{n}_\beta}{2},$$

onde $\boldsymbol{n}_\alpha$ e $\boldsymbol{n}_\beta$ são as normais unitárias dos triângulos t_α e t_β, respectivamente, e $\boldsymbol{n}_\alpha \cdot \boldsymbol{n}_\beta$ é o produto escalar dessas duas normais, um valor no intervalo $[-1, 1] \subset \mathbb{R}$. Por sua vez, temos

$$o(q) = \frac{\text{sen}(\alpha_j) + \text{sen}(\beta_j) + \text{sen}(\alpha_i + \beta_i) + \text{sen}(\alpha_k + \beta_k)}{4},$$

onde α_i, α_j, α_k, β_i, β_j e β_k são os ângulos de t_α e t_β, respectivamente, como mostrado na Figura 6.1.

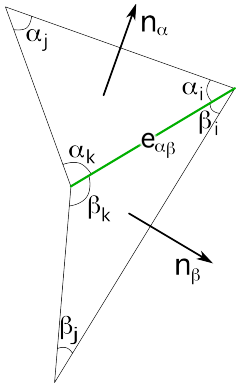


Figura 6.1: . Elementos da métrica de qualidade de forma de quadriláteros.

Intuitivamente, quanto mais parecidas forem as direções das normais de t_α e t_β, mais

próximo de 1 será o valor de $p(q)$. Em outras palavras, quando q é bastante "plano", o valor de $p(q)$ será grande, indicando que este quadrilátero é "desejável" quanto à planaridade. Por sua vez, $o(q)$ é a média aritmética entre os senos dos quatro ângulos internos de q. Em linhas gerais, este valor cresce à medida em que a forma de q se aproxima à de um retângulo.

6.3 Comparações

Como dissemos na Seção 6.1, nós consideramos uma base de dados formada por triangulações de superfícies comumente encontradas em trabalhos de Processamento Geométrico. Como de hábito em Processamento Geométrico, chamamos de *modelo* cada superfície e sua triangulação associada. No nosso experimento, utilizamos 12 (doze) modelos, os quais foram obtidos do repositório, de acesso público e gratuito, do Projeto AIM@SHAPE[1]. A Tabela 6.1 mostra o nome e fornece a característica de Euler-Poincaré de cada modelo.

Tabela 6.1: Nome e característica de Euler-Poincaré dos modelos usados nos experimentos.

Modelo	**# Vértices**	**# Faces**	**# Arestas**	**# Genus**
Armadillo	171889	343774	515661	0
Bimba	14839	29674	44511	0
Botijo	20000	40016	60024	5
Cow	4315	8626	12939	0
Dinosaur	56194	112384	168576	0
Eros	197230	394456	591684	0
Fandisk	6475	12946	19419	0
Fertility	19996	40000	60000	4
Gargoyle	97130	194256	291384	0
Knot	31545	63090	94635	1
Santa	75444	150880	226320	3
Teeth	116038	232072	348108	0

As Figuras 6.2-6.10 apresentam uma projeção perspectiva de quatro modelos da Tabela 6.1.

Nós executamos quatro procedimentos de conversão das triangulações em quadrilaterizações:

- conversão utilizando o algoritmo `TriQuad()`;

[1] `http://www.aimatshape.net/`

- conversão utilizando emparelhamento perfeito;
- conversão utilizando emparelhamento perfeito de custo máximo e
- conversão utilizando emparelhamento de custo máximo.

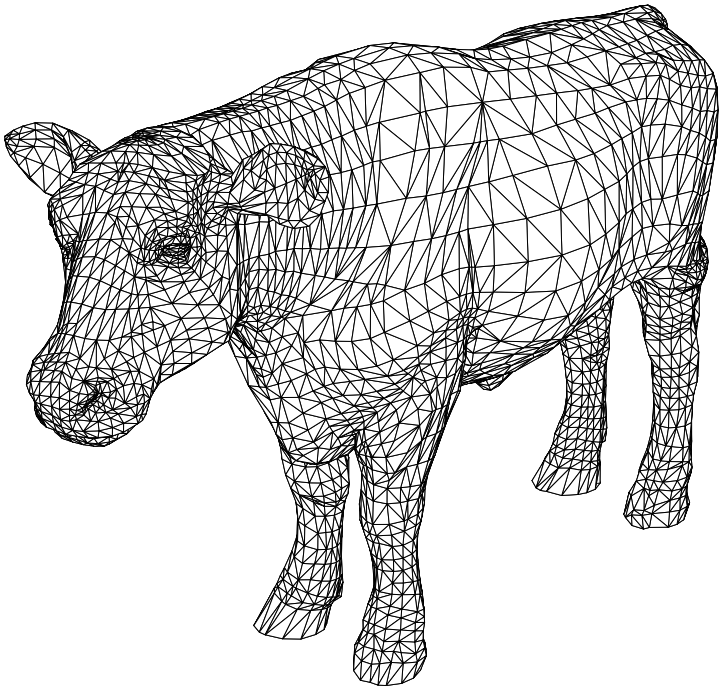


Figura 6.2: O modelo Cow.

Em seguida, geramos medidas de resumo, com respeito à métrica de planaridade e ortogonalidade, para as quadrilaterizações obtidas a partir de cada modelo por cada um dos procedimentos acima. Mais especificamente, calculamos o menor e o maior valor da métrica para todos os quadriláteros, assim como a média e o desvio padrão da média, além do número de *doublets* da quadrilaterização gerada e o tempo de execução do procedimento, em segundos.

Os resultados obtidos são exibidos em cinco tabelas dadas seguir, uma para cada procedimento baseado em emparelhamento em grafos e duas para o procedimento baseado na abordagem gulosa. A razão para termos duas tabelas baseadas na abordagem gulosa é que, em uma delas, a abordagem gulosa é usada com a heurística da aresta mais longa, como originalmente foi proposto por Luiz Velho (VELHO, 2000). Na outra tabela, temos as medidas da abordagem gulosa usada com a métrica de planaridade e ortogonalidade. Denominamos esta segunda abordagem de *TriQuad-M()* a seguir, para diferenciá-la da

original, *TriQuad()*. Desta forma, podemos fazer uma comparação mais justa entre a abordagem gulosa e as abordagens baseadas em emparelhamentos com custo, pois, em todos os procedimentos baseados em grafos ponderados, o custo de uma aresta é o valor da função m (isto é, a métrica) para o quadrilátero formado pelo emparelhamento dos triângulos que compartilham a aresta.

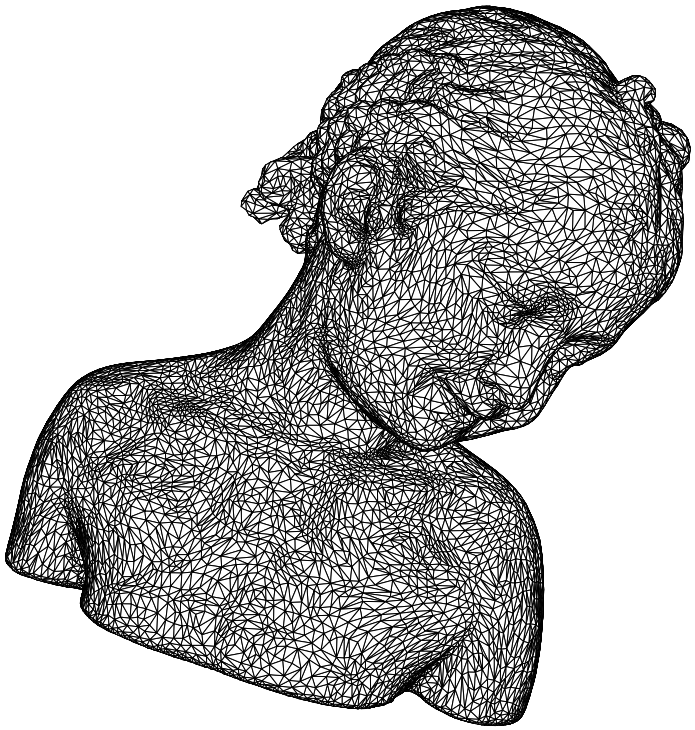


Figura 6.3: O modelo Bimba.

6.4 Análise dos resultados

A Tabela 6.2 e a Tabela 6.3 apresentam as medidas de resumo para os procedimentos de conversão baseados em emparelhamento perfeito e emparelhamento perfeito de custo máximo. Como esperado, o emparelhamento perfeito de custo máximo elevou o menor valor da métrica de qualidade, assim como a média dos valores sobre todos os quadriláteros, para todos os modelos. Entretanto, a melhoria é mais significativa quando comparamos o número de *doublets*. O emparelhamento perfeito de custo máximo reduz em pelo menos quatro vezes o número de *doublets* das quadrilaterizações geradas a partir de todos os modelos. As Figuras 6.4 e 6.5 comparam os desempenhos de todas as abordagens quanto à média dos valores da métrica de qualidade e quanto ao número de *doublets*, respectivamente.

Tabela 6.2: Conversão usando emparelhamento perfeito. Para cada modelo (linha), a primeira coluna contém o nome do modelo; a segunda (terceira) coluna contém o menor (maior) valor da função m; a quarta coluna contém a média de m com respeito a todos os quadriláteros; a quinta coluna contém o desvio padrão dos valores de m com respeito à média, a sexta coluna é o número de *doublets* e a sétima coluna é o tempo de execução.

Modelo	**Min**	**Max**	**Média**	$\boldsymbol{\sigma}$	**# Doublets**	**T (seg)**
Armadillo	0,618	1	0,916	0,050	19	153,304
Bimba	0,150	1	0,862	0,098	266	0,430
Botijo	0,606	1	0,903	0,063	130	1,596
Cow	0,086	1	0,860	0,114	75	0,156
Dinosaur	0,581	1	0,910	0,071	119	1,002
Eros	0,539	1	0,924	0,037	1	173,774
Fandisk	0,440	1	0,905	0,101	13	0,093
Fertility	0,116	1	0,871	0,092	520	0,476
Gargoyle	0,104	1	0,902	0,070	80	44,813
Knot	0,623	1	0,887	0,081	16	2,487
Santa	0,619	1	0,925	0,061	5	2,668
Teeth	0,606	1	0,924	0,068	14	2,616

Tabela 6.3: Conversão usando emparelhamento perfeito de custo máximo. Para cada modelo (linha), a primeira coluna contém o nome do modelo; a segunda (terceira) coluna contém o menor (maior) valor da função m; a quarta coluna contém a média de m com respeito a todos os quadriláteros; a quinta coluna contém o desvio padrão dos valores de m com respeito à média, a sexta coluna é o número de *doublets* e a sétima coluna é o tempo de execução.

Modelo	**Min**	**Max**	**Média**	$\boldsymbol{\sigma}$	**# Doublets**	**T (seg)**
Armadillo	0,672	1	0,959	0,035	3	14,599
Bimba	0,144	1	0,917	0,069	72	0,982
Botijo	0,698	1	0,939	0,047	10	1,474
Cow	0,387	1	0,946	0,074	9	0,253
Dinosaur	0,627	1	0,960	0,038	11	4,663
Eros	0,598	1	0,945	0,027	0	19,041
Fandisk	0,574	1	0,993	0,020	0	0,251
Fertility	0,375	1	0,921	0,065	116	1,389
Gargoyle	0,136	1	0,946	0,046	22	8,959
Knot	0,652	1	0,950	0,049	8	2,583
Santa	0,708	1	0,971	0,037	0	5,049
Teeth	0,645	1	0,973	0,035	4	8,339

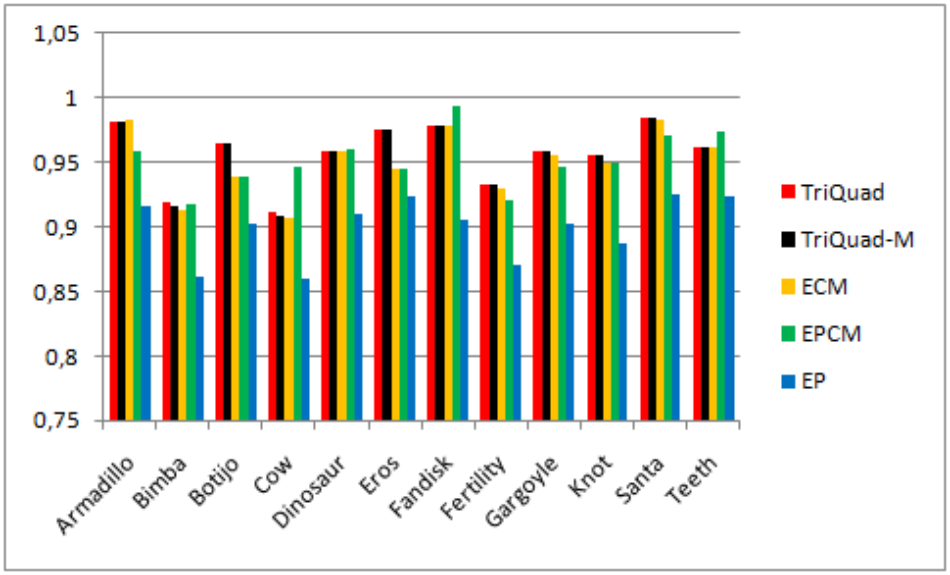


Figura 6.4: Gráfico comparativo das abordagens, quanto à média dos valores da métrica m.

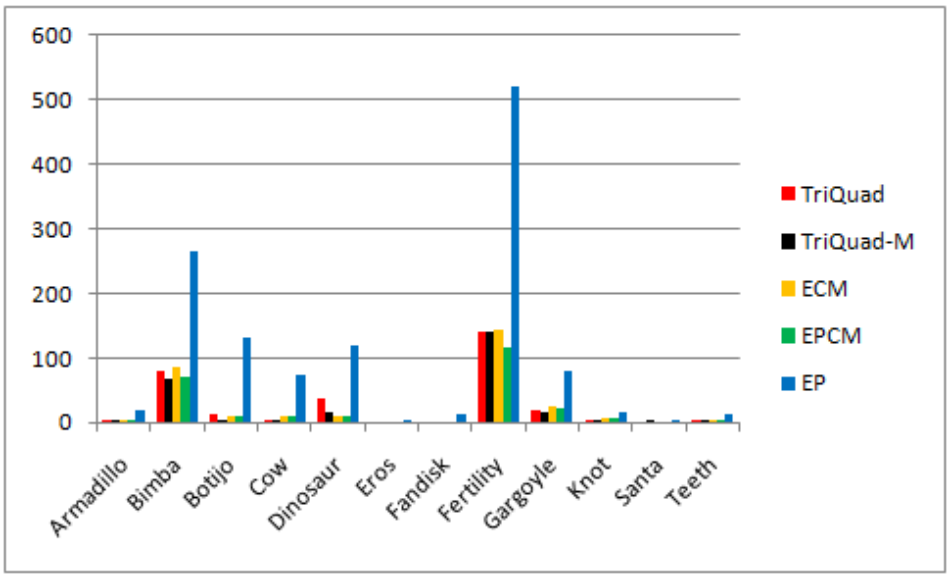


Figura 6.5: Gráfico comparativo das abordagens, quanto ao número de *doublets*.

É interessante notar, também, que o tempo de execução do procedimento baseado em emparelhamento perfeito de custo máximo é muito menor, em todos os modelos com grande número de vértices, do que o do procedimento baseado em emparelhamento perfeito. Obviamente, isto pode parecer estranho, mas se deve ao fato de termos utilizado o algoritmo do Capítulo 4 para calcular o emparelhamento perfeito, que tem complexidade de tempo $\mathcal{O}(|V|^3)$, e o algoritmo da biblioteca `LEMON` calcular o emparelhamento perfeito de custo máximo, que tem complexidade de tempo $\mathcal{O}(|V| \cdot (|E| + |V| \log |V|))$. Se tivéssemos usado o algoritmo em (DIKS; STANCZYK, 2010) para calcular os emparelhamentos perfeitos, a situação seria completamente diferente. A Figura 6.6 exibe os valores de tempo de execução obtidos para cada um dos algoritmos utilizados no trabalho.

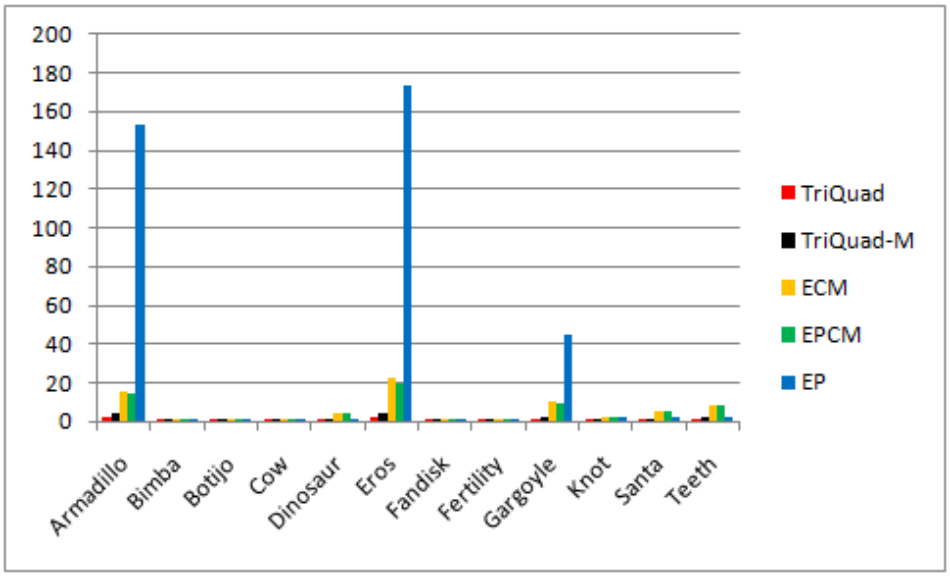


Figura 6.6: Gráfico comparativo das abordagens, quanto ao tempo de execução em segundos.

Note que, nos procedimentos baseados em emparelhamentos perfeitos, o número de quadriláteros da quadrilaterização obtida de um modelo é sempre a metade do número do triângulos e, portanto, fixo (para cada modelo). Este não é o caso das quadrilaterizações obtidas com a abordagem gulosa ou com o emparelhamento de custo máximo, pois elas, quase sempre, requerem um passo extra de subdivisão de triângulos e quadriláteros para eliminar os triângulos isolados e gerar uma subdivisão com quadriláteros apenas. Como o número de triângulos isolados pode variar entre as duas abordagens (e com o uso de diferentes heurísticas, no caso da abordagem gulosa), o número de triângulos isolados e o número de quadriláteros da quadrilaterização final são boas indicações da eficácia dos procedimentos de conversão. Quanto menores forem esses números, melhor a quadrilaterização.

A Tabela 6.4 apresenta as medidas de resumo para o procedimento de conversão baseado na abordagem gulosa com a heurística da aresta mais longa. Examinando a tabela, podemos concluir que a média dos valores da métrica de qualidade é ligeiramente superior à média da Tabela 6.3. Isto mostra que a heurística da aresta mais longa é, de fato, efetiva quando se analisa a qualidade de forma dos quadriláteros resultantes. No entanto, o número de quadriláteros da quadrilaterização final é mais do que quatro vezes o número de quadriláteros das quadrilaterizações correspondentes geradas com o emparelhamento perfeito de custo máximo. Isto se deve à existência de triângulos isolados, que obrigam a execução do passo de subdivisão extra. Este passo divide cada face triangular em três e cada face quadrangular em quatro. Basta haver *um* triângulo isolado para que este passo seja executado.

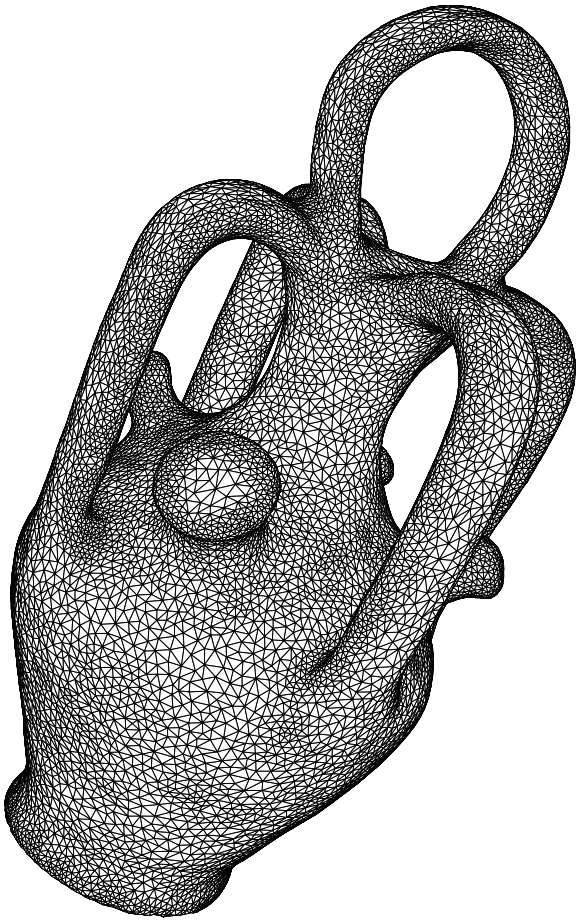


Figura 6.7: O modelo Botijo.

Tabela 6.4: Conversão usando a abordagem gulosa com a heurística da aresta mais longa. Para cada modelo (linha), a primeira coluna contém o nome do modelo; a segunda (terceira) coluna contém o menor (maior) valor da função m; a quarta coluna contém a média de m com respeito a todos os quadriláteros; a quinta coluna contém o desvio padrão dos valores de m com respeito à média, a sexta coluna é o número de *doublets*; a sétima coluna é o número de quadriláteros da quadrilaterização final; a oitava coluna é o número de triângulos isolados (antes do passo extra de subdivisão) e a nona coluna é o tempo de execução.

Modelo	**Min**	**Max**	**Média**	$\boldsymbol{\sigma}$	**# D**	**# Q**	**# I**	**T (seg)**
Armadillo	0,585	1	0,982	0,019	5	714654	27106	2,200
Bimba	0,107	1	0,919	0,067	81	62794	3446	0,156
Botijo	0,713	1	0,964	0,029	12	85060	5028	0,219
Cow	0,110	1	0,911	0,084	5	17782	530	0,031
Dinosaur	0,585	1	0,959	0,030	38	234896	10128	0,655
Eros	0,480	1	0,976	0,019	0	846572	57660	2,512
Fandisk	0,637	1	0,979	0,056	0	25952	60	0,062
Fertility	0,177	1	0,933	0,056	140	84812	4812	0,234
Gargoyle	0,140	1	0,958	0,041	18	410198	21686	1,201
Knot	0,701	1	0,955	0,041	2	132692	6512	0,376
Santa	0,758	1	0,985	0,020	0	308102	6342	0,921
Teeth	0,669	1	0,961	0,031	2	477480	13336	1,449

A Figura 6.8 compara os resultados obtidos por todas as abordagens quanto ao número final de quadriláteros. A Figura 6.9 compara os resultados obtidos apenas pelos algoritmos `TriQuad()`, `TriQuad-M()` e emparelhamento de custo máximo quanto ao número de triângulos isolados restantes antes do passo de subdivisão, já que os outros dois algoritmos não geram triângulos isolados. Perceba que o número de quadriláteros obtido pelo emparelhamento perfeito de custo máximo é ligeiramente menor do que os números obtidos pelos algoritmos `TriQuad()` e `TriQuad-M()`. Isto ocorre porque o número de triângulos isolados gerado pelo emparelhamento de custo máximo é drasticamente inferior (na maioria dos casos, quase nulo), comparado às variações do `TriQuad()`. Embora o passo extra de subdivisão seja, em geral, necessário em ambos os casos, este passo produz um número maior de quadriláteros à medida em que o número de triângulos isolados aumenta.

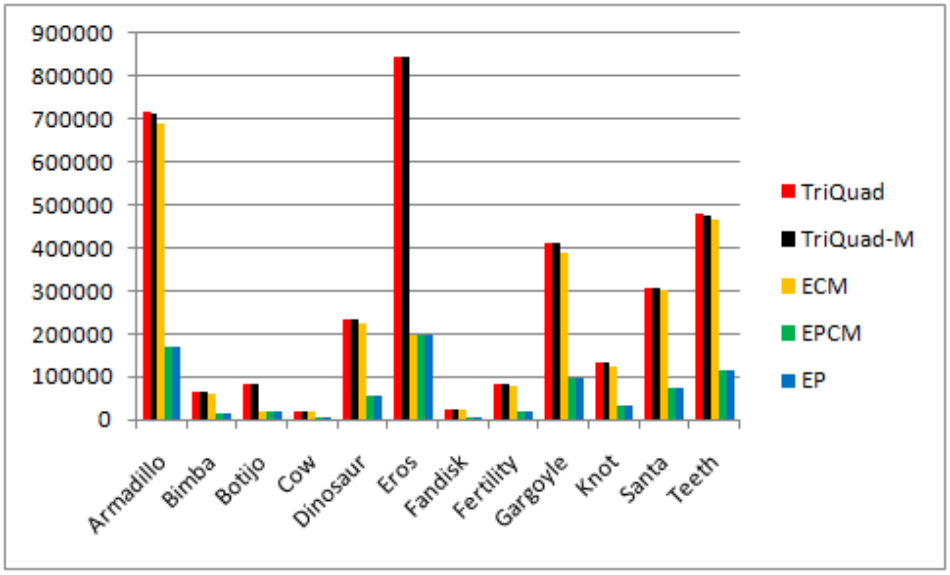


Figura 6.8: Gráfico comparativo das abordagens, quanto ao número final de quadriláteros.

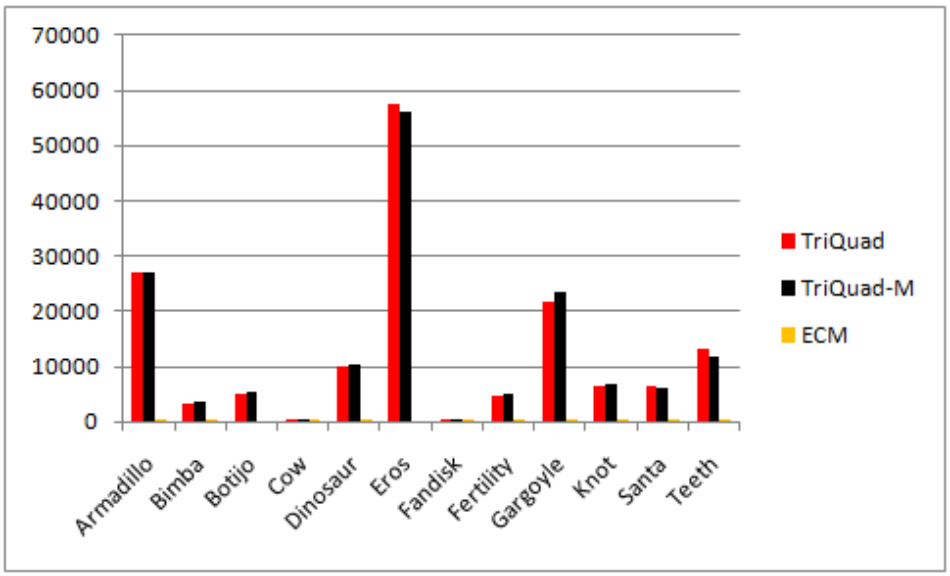


Figura 6.9: Gráfico comparativo das abordagens `TriQuad()`, `TriQuad-M()` e emparelhamento de custo máximo, quanto ao número de triângulos isolados (antes do passo extra de subdivisão).

É importante ressaltar que o número de *doublets* é, proporcionalmente, menor nas quadrilaterizações geradas com a abordagem gulosa e a heurística da aresta mais longa. Além disso, o tempo de execução do procedimento baseado na abordagem gulosa é bem menor do que o dos procedimentos baseados em grafos. Isto é uma simples conseqüência do fato do algoritmo `TriQuad()` ter complexidade $\mathcal{O}(|E| \cdot \log |E|)$ e $|E|$ é $\Theta(|V|)$ (para os nossos grafos).

Tabela 6.5: Conversão usando a abordagem gulosa com a métrica m. Para cada modelo (linha), a primeira coluna contém o nome do modelo; a segunda (terceira) coluna contém o menor (maior) valor da função m; a quarta coluna contém a média de m com respeito a todos os quadriláteros; a quinta coluna contém o desvio padrão dos valores de m com respeito à média, a sexta coluna é o número de *doublets*; a sétima coluna é o número de quadriláteros da quadrilaterização final; a oitava coluna é o número de triângulos isolados (antes do passo extra de subdivisão) e a nona coluna é o tempo de execução.

Modelo	**Min**	**Max**	**Média**	$\boldsymbol{\sigma}$	**# D**	**# Q**	**# I**	**T (seg)**
Armadillo	0,475	1	0,982	0,019	1	714442	26894	3,806
Bimba	0,107	1	0,916	0,071	67	63108	3760	0,297
Botijo	0,606	1	0,965	0,029	5	85274	5242	0,421
Cow	0,110	1	0,909	0,086	5	17772	520	0,078
Dinosaur	0,574	1	0,959	0,030	15	235098	10330	0,201
Eros	0,523	1	0,976	0,019	0	845072	56160	4,478
Fandisk	0,602	1	0,979	0,056	0	25960	68	0,125
Fertility	0,177	1	0,933	0,058	140	85124	5124	0,406
Gargoyle	0,094	1	0,958	0,042	17	412096	23584	2,216
Knot	0,684	1	0,955	0,041	2	132854	6674	0,687
Santa	0,746	1	0,985	0,020	1	308016	6256	1,622
Teeth	0,656	1	0,961	0,031	1	476000	11856	2,574

A Tabela 6.5 apresenta as medidas de resumo para a abordagem gulosa combinada com a métrica de planaridade e ortogonalidade. Como podemos constatar, a média dos valores da métrica aplicada aos quadriláteros gerados a partir de qualquer modelo é praticamente a mesma obtida com a heurística da aresta mais longa. O número de quadriláteros também é bastante semelhante e o número de *doublets* é ligeiramente inferior quando o valor da métrica é utilizado no lugar do comprimento da aresta. Então, as tabelas nos levam a crer que o ganho obtido com o uso do valor da métrica, no lugar da heurística, não foi significativo.

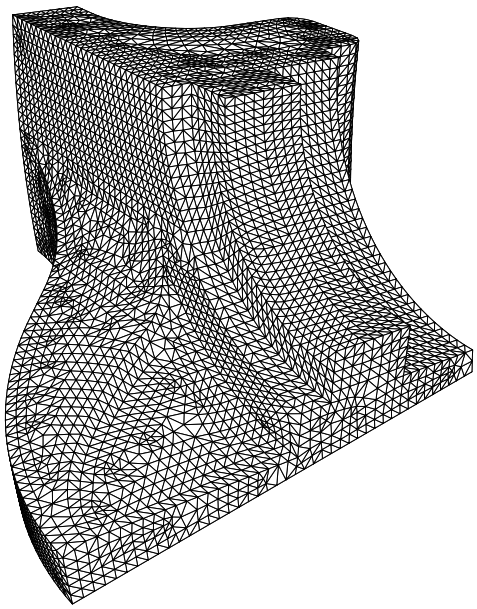


Figura 6.10: O modelo Fandisk.

A Figura 6.11 revela um aspecto importante da qualidade das quadrilaterizações geradas com a abordagem gulosa, que não é capturado pelas tabelas acima. Examinando a figura, podemos notar que as extremidades das patas da "vaca" não foram preservadas quando a abordagem gulosa foi utilizada, assim como não foi preservada a ortogonalidade das cadeias de arestas da triangulação original. A razão para isso reside no passo extra de subdivisão, que basicamente "inverte" as arestas emparelhadas com as não-emparelhadas (veja a Figura 1.3) após subdividir os triângulos isolados e os quadriláteros existentes. De certa forma, esta inversão tem um efeito similar ao de executar uma "rotação" de 45º na direção das arestas da quadrilaterização, com respeito à direção anterior ao processo de subdivisão.

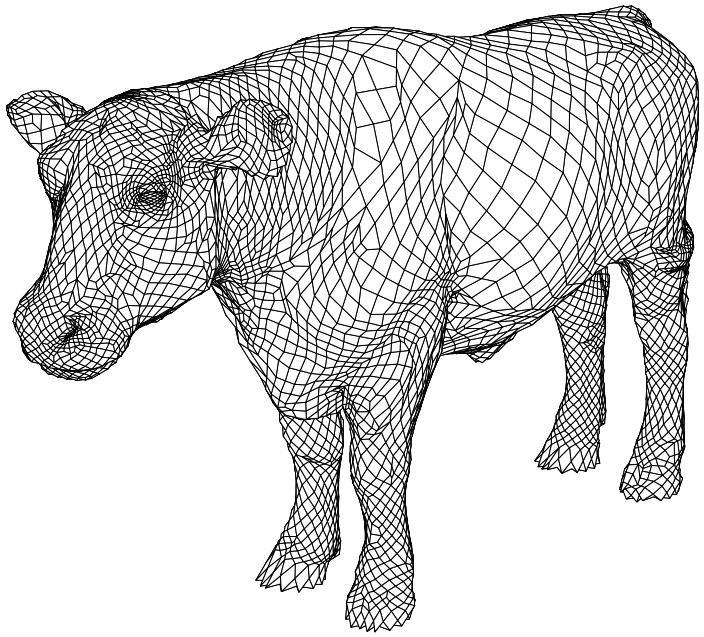
A Tabela 6.6 apresenta as medidas de resumo para o procedimento de conversão baseado em emparelhamento de custo máximo (seguido, se necessário, pelo passo extra de subdivisão para eliminar triângulos isolados). Dentre estas medidas, aquelas relacionadas à métrica m, assim como o número de *doublets*, foram semelhantes àquelas registradas pelas abordagens gulosas. No entanto, o número final de quadriláteros já foi bastante reduzido em relação ao das abordagens gulosas, porque o emparelhamento de custo máximo consegue deixar um menor número de triângulos isolados antes do passo extra de subdivisão. Por sinal, para dois dos modelos usados, Botijo e Eros, a execução do emparelhamento de custo máximo já foi necessária para encontrar um emparelhamento perfeito, obtendo-se quadrilaterizações com metade do número de faces dos modelos originais.

Tabela 6.6: Conversão usando emparelhamento de custo máximo. Para cada modelo (linha), a primeira coluna contém o nome do modelo; a segunda (terceira) coluna contém o menor (maior) valor da função m; a quarta coluna contém a média de m com respeito a todos os quadriláteros; a quinta coluna contém o desvio padrão dos valores de m com respeito à média, a sexta coluna é o número de *doublets*; a sétima coluna é o número de quadriláteros da quadrilaterização final; a oitava coluna é o número de triângulos isolados (antes do passo extra de subdivisão) e a nona coluna é o tempo de execução.

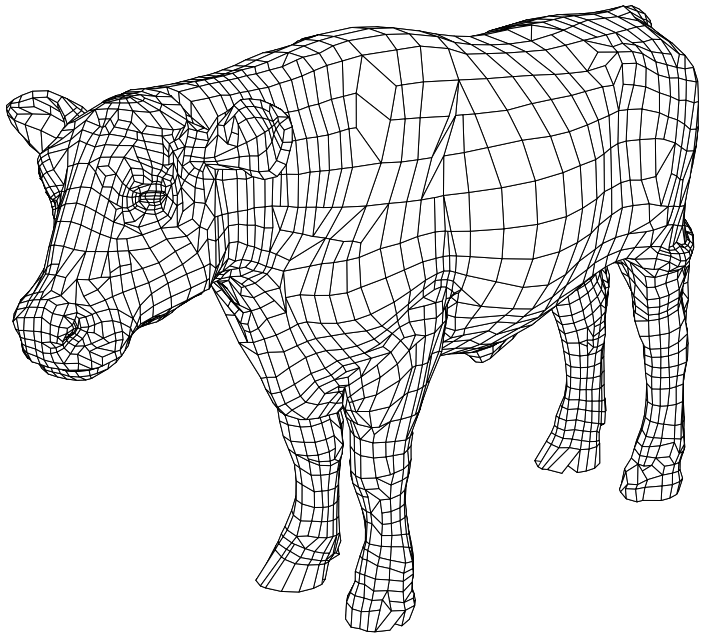
Modelo	**Min**	**Max**	**Média**	$\boldsymbol{\sigma}$	**# D**	**# Q**	**# I**	**T (seg)**
Armadillo	0,475	1	0,983	0,020	3	687624	76	15,834
Bimba	0,124	1	0,913	0,071	85	59354	6	1,201
Botijo	0,698	1	0,939	0,047	10	20008	0	1,669
Cow	0,096	1	0,907	0,085	9	17282	30	0,219
Dinosaur	0,574	1	0,958	0,029	10	224780	12	4,680
Eros	0,598	1	0,945	0,027	0	197228	0	22,526
Fandisk	0,640	1	0,979	0,056	0	25898	6	0,281
Fertility	0,224	1	0,929	0,059	143	80002	2	1,669
Gargoyle	0,140	1	0,955	0,043	25	388520	8	10,624
Knot	0,704	1	0,950	0,043	7	126194	14	2,543
Santa	0,732	1	0,983	0,021	0	301904	144	5,179
Teeth	0,691	1	0,961	0,030	4	464274	130	8,284

Numa comparação entre os desempenhos dos algoritmos de emparelhamento de custo máximo e emparelhamento perfeito de custo máximo, também notamos semelhança entre os valores registrados pela métrica, mas com leve vantagem para o emparelhamento perfeito de custo máximo. Isto porque o passo extra de subdivisão executado após o emparelhamento de custo máximo não leva mais em conta as métricas de qualidade. Além disso, na maioria dos casos, o emparelhamento de custo máximo ainda gera um número final de quadriláteros mais de quatro vezes maior que o gerado pelo emparelhamento perfeito de custo máximo, em virtude do mesmo passo extra de subdivisão.

Portanto, as comparações isoladas do algoritmo de emparelhamento perfeito de custo máximo com os demais utilizados nos permitem concluir que o primeiro mostrou-se o mais apropriado para resolver o problema-alvo abordado neste trabalho. Ele registrou valores excelentes tanto em relação às medidas de qualidade quanto ao número final de quadriláteros dos produtos finais, além de proporcionar implementações com tempo de execução razoável, como a que utilizamos aqui.



(a)



(b)

Figura 6.11: Quadrilaterizações do modelo Cow geradas com (a) a abordagem gulosa combinada com a heurística da aresta mais longa e com (b) emparelhamento perfeito de custo máximo.

7 Conclusão

Este capítulo conclui a presente monografia. A Seção 7.1 apresenta um pequeno resumo dos aspectos mais importantes do trabalho. A Seção 7.2 revela as principais dificuldades enfrentadas ao longo do desenvolvimento do trabalho. A Seção 7.3 destaca algumas possíveis extensões para trabalhos futuros de conclusão de curso e descreve alguns problemas de pesquisa.

7.1 Considerações finais

Neste trabalho, foram propostas três abordagens baseadas em grafos para a resolução de um problema computacional importante para a área de Processamento Gráfico: o de converter uma triangulação de uma superfície $S \subset \mathbb{E}^3$, compacta e com fronteira vazia, em uma quadrilaterização da mesma. O algoritmo estudado de uma das abordagens foi implementado pelo autor deste trabalho, enquanto os algoritmos das outras duas abordagens foram apenas estudados e um código público e gratuito foi usado nas avaliações experimentais.

Tendo como base um banco de modelos bem representativos do tipo de superfície utilizado em trabalhos da área de Processamento Gráfico, avaliamos as três abordagens propostas, comparando-as entre si e com uma abordagem gulosa proposta por Luiz Velho (VELHO, 2000). Pudemos verificar que uma das abordagens propostas aqui, a do emparelhamento perfeito de custo máximo, obteve desempenho comparável ao da abordagem gulosa sem a necessidade de refinar a subdivisão resultante para eliminar triângulos isolados. Este passo extra de refinamento da subdivisão acarreta a geração de um número excessivo de quadriláteros (mais de quatro vezes maior de acordo com nossos experimentos).

7.2 Dificuldades encontradas

Encontramos duas grandes dificuldades ao longo do projeto. A maior delas foi a implementação do algoritmo de emparelhamento de cardinalidade máxima. A ideia inicial era implementar o algoritmo de Micali e Vazirani (MICALI; VAZIRANI, 1980), cuja complexidade de tempo é assintoticamente inferior à do algoritmo de Gabow utilizado. Porém, quando a implementação foi concluída, verificamos que esta não encontrava uma solução ótima em alguns casos e, como o algoritmo era bastante complicado, não conseguimos corrigir os erros que havíamos cometido. Para não correr o risco de gastar mais tempo, migramos para o algoritmo de Gabow, que nos proporcionou uma implementação mais simples.

A outra dificuldade ocorreu na integração dos códigos-fonte do projeto à biblioteca `LEMON`, para a execução dos algoritmos de emparelhamento de custo máximo e de emparelhamento perfeito de custo máximo. Mas, apesar do surgimento de alguns problemas inesperados, fomos capazes de utilizar a biblioteca. Finalmente, vimos que o tempo exigido para cumprir todas as etapas da proposta incial deste trabalho foi extremamente mal dimensionado. Muitas das tarefas propostas inicialmente, tais como uma implementação própria de algoritmos para emparelhamentos (perfeitos) de custo máximo, não poderia mesmo ter sido contemplada, juntamente com as demais partes do trabalho, em um único semestre.

7.3 Trabalhos futuros

As extensões mais imediatas deste trabalho, que poderiam muito bem serem realizadas como trabalhos de final de curso, dizem respeito ao estudo e implementação do algoritmo em (DIKS; STANCZYK, 2010) para calcular emparelhamentos perfeitos em grafos 3-regulares e sem arestas de corte (que possui a menor complexidade de tempo entre todos os algoritmos conhecidos) e o estudo e implementação de algoritmos para calcular emparelhamentos perfeitos de custo máximo, com base no trabalho em (KOLMOGOROV, 2009).

Uma outra possibilidade para trabalho de final de curso é estender a avaliação experimental que realizamos aqui para incluir o algoritmo "crawler", descrito em (TARINI; PIETRONI; CIGNONI; PANOZZO; PUPPO, 2010), o qual substitui o passo de subdivisão extra da abordagem gulosa por uma busca em largura para emparelhar, de cada vez, dois

triângulos isolados. Esta busca também leva em consideração a qualidade de forma dos quadriláteros que são re-emparelhados durante a busca.

No contexto deste trabalho, existem dois problemas de pesquisa. Esses problemas não fazem parte do escopo do trabalho, mas poderão ser objetos de estudos de pós-graduação. O primeiro deles diz respeito ao problema do emparelhamento perfeito em grafos 3-regulares e sem arestas de corte. No artigo (BIEDL; BOSE; DEMAINE; LUBIW, 2001), há também um algoritmo de tempo linear em $|V|$ para calcular emparelhamentos perfeitos em grafos 3-regulares, sem arestas de corte e *planares*. Como imersões em superfícies são, de certa forma, uma generalização de imersões planares para espaços mais complexos (isto é, as superfícies), seria também bastante interessante investigar a possibilidade de estender o algoritmo em (BIEDL; BOSE; DEMAINE; LUBIW, 2001) para superfícies e de tal forma que a complexidade de tempo do algoritmo seja $o(|V| \cdot \log^2 |V|)$, pois o melhor algoritmo conhecido tem complexidade $\mathcal{O}(|V| \cdot \log^2 |V|)$.

Referências

ALFORS, L.; SARIO, L. *Riemann Surfaces*. Princeton, NJ, USA: Princeton University Press, 1960.

ALLMAN, D. A quadrilateral finite element including vertex rotations for plane elasticity analysis. *International Journal for Numerical Methods in Engineering*, v. 26, p. 717–730, 1988.

ATALAY, F. B.; RAMASWAMI, S.; XU, D. Quadrilateral meshes with bounded minimum angle. In: *Proceedings of the 17th International Meshing Roundtable (IMR)*. [S.l.: s.n.], 2008. p. 73–91.

BALINSKI, M. L. Labelling to obtain a maximum matching. In: *Combinatorial Mathematics and Its Applications*. Chapel Hill, NC, USA: University of North Carolina Press, 1967. p. 585–602.

BERBERICH, E.; FOGEL, E.; HALPERIN, D.; MEHLHORN, K.; WEIN, R. Sweeping and maintaining two-dimensional arrangements on surfaces: a first step. In: *Proceedings of the 15th Annual European Symposium on Algorithms (ESA'07)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. p. 645–656.

BERBERICH, E.; KERBER, M. Exact arrangements on tori and dupin cyclides. In: *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM'08)*. New York, NY, USA: ACM, 2008. p. 59–66.

BERN, M.; EPPSTEIN, D. Mesh generation and optimal triangulation. In: HWANG, F.; DU, D.-Z. (Ed.). *Computing in Euclidean Geometry*. [S.l.]: World Scientific, 1992.

BERN, M.; EPPSTEIN, D. Quadrilateral meshing by circle packing. In: *Proceedings of the 6th International Meshing Roundtable (IMR)*. [S.l.: s.n.], 1997. p. 7–19.

BERN, M.; PLASSMANN, P. Mesh generation. In: SACK, J.-R.; URRUTIA, J. (Ed.). *Handbook of Computational Geometry*. [S.l.]: Elsevier Science, 2000.

BIEDL, T. C.; BOSE, P.; DEMAINE, E. D.; LUBIW, A. Efficient algorithms for petersen's matching theorem. *Journal of Algorithms*, Academic Press, Inc., Duluth, MN, USA, v. 38, n. 1, p. 110–134, 2001.

BLACKER, T. Paving: a new approach to automated quadrilateral mesh generation. *International Journal For Numerical Methods in Engineering*, v. 32, n. 4, p. 811–847, 1991.

BLOCH, E. *A first course in geometric topology and differential geometry*. Boston, MA, USA: Birkhäuser, 1997.

BLUM, N. A new approach to maximum matching in general graphs. In: *Proceedings of the 17th international colloquium on Automata, languages and programming (ICALP)*. New York, NY, USA: Springer-Verlag, 1990. p. 586–597.

BOISSONNAT, J.-D.; OUDOT, S. Provably good sampling and meshing of surfaces. *Graphical Models*, Academic Press Professional, Inc., San Diego, CA, USA, v. 67, n. 5, p. 405–451, 2005.

BOMMES, D.; ZIMMER, H.; KOBBELT, L. Mixed-integer quadrangulation. *ACM Transactions on Graphics*, ACM, New York, NY, USA, v. 28, n. 3, p. 1–10, 2009.

BONDY, J. A.; MURTY, U. S. R. *Graph Theory*. New York, NY, USA: Springer-Verlag, 2007.

CARVALHO, P. C. P.; VELHO, L.; GOMES, J. de M. *Spatial decompositions: theory and practice*. Rio de Janeiro, RJ, Brasil, 1992.

CHAZELLE, B. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, v. 6, n. 1, p. 485–524, 1991.

CHENG, S.-W.; DEY, T. K.; RAMOS, E. A.; RAY, T. Sampling and meshing a surface with guaranteed topology and geometry. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 37, n. 4, p. 1199–1227, 2007.

COOK, W.; ROHE, A. Computing minimum-weight perfect matching. *Journal on Computing*, INFORMS, Linthicum, MD, USA, v. 11, n. 2, p. 138–148, 1999.

COPPERSMITH, D.; WINOGRAD, S. Matrix multiplication via arithmetic progressions. In: *Proceedings of the 19th Annual ACM Conference on Theory of Computing (STOC'87)*. New York, NY, USA: ACM, 1987. p. 1–6.

DANIELS, J.; NONATO, L. G.; SIQUEIRA, M.; LIZIER, M.; SILVA, C. T. Template-based quadrilateral meshing. *Computer & Graphics*, 2011. (accepted for publication).

DANTZIG, G. *Linear Programming and Extensions*. Princeton, NJ, USA: Princeton University Press, 1963.

DIKS, K.; STANCZYK, P. Perfect matching for biconnected cubic graphs in $\mathcal{O}(n \log^2 n)$ time. In: *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'10)*. Germany: Springer-Verlag, 2010. p. 321–333.

DONG, S. K.; KIRCHER, S.; GARLAND, M. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design*, Elsevier, Amsterdam, The Netherlands, v. 22, n. 5, p. 392–423, 2005.

EDMONDS, J. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, v. 69, p. 125–130, 1965.

EDMONDS, J. Paths, trees, and flowers. *Canadian Journal of Mathematics*, v. 17, p. 449–467, 1965.

EDMONDS, J.; JOHNSON, E. L.; LOCKHART, S. C. *Blossom I: a computer code for the matching problem*. Yorktown Heights, NY, USA, 1969.

FOGEL, E.; SETTER, O.; HALPERIN, D. Arrangements of geodesic arcs on the sphere. In: *Proceedings of the 24th ACM-Eurographics Annual Symposium on Computational Geometry (SCG'08)*. New York, NY, USA: ACM, 2008. p. 218–219.

FREMUTH-PAEGER, C.; JUNGNICKEL, D. Balanced network flows viii: a revised theory of phase-ordered algorithms and the $\mathcal{O}(\sqrt{n}m\log(n^2/m)/\log n)$ bound for the nonbipartite cardinality matching problem. *Networks*, v. 41, n. 3, p. 137–142, 2003.

GABOW, H. N. *An efficient implementation of Edmond's maximum matching algorithm*. [S.l.], 1972.

GABOW, H. N. An efficient implementation of edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, v. 23, n. 2, p. 221–234, 1976.

GABOW, H. N. Data structures for weighted matching and nearest common ancestors with linking. In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms (SODA'90)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990. p. 434–443.

GABOW, H. N.; GALIL, Z.; SPENCER, T. H. Efficient implementation of graph algorithms using contraction. *Journal of the ACM*, ACM, New York, NY, USA, v. 36, n. 3, p. 540–572, 1989.

GABOW, H. N.; TARJAN, R. E. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, v. 38, n. 4, p. 815–853, 1991.

GALIL, Z.; MICALI, S.; GABOW, H. N. An $\mathcal{O}(|e|\cdot|v|\cdot\log|v|)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 15, n. 1, p. 120–130, 1986.

GASS, S. I. *Linear Programming*. Second. New York, NY, USA: McGraw-Hill, Inc., 1964.

GEVERS, T.; SMEULDERS, A. W. M. Combining region splitting and edge detection through guided delaunay image subdivision. In: *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*. Washington, DC, USA: IEEE Computer Society, 1997. p. 1021–1026. ISBN 0-8186-7822-4.

GOLDBERG, A. V.; KARZANOV, A. V. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, v. 100, n. 3, p. 537–568, 2004.

GROSS, J. L.; TUCKER, T. W. *Topological graph theory*. Mineola, NY, USA: Dover Publications, Inc., 2001.

GUIBAS, L.; STOLFI, J. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, v. 4, n. 2, p. 74–123, 1985.

HENLE, M. *A combinatorial introduction to topology*. Mineola, NY, USA: Dover Publications, Inc., 1994.

JOE, B. Quadrilateral mesh generation in polygonal regions. *Computer-Aided Design*, v. 27, n. 3, p. 209–222, 1995.

JOHNSON, C. *Numerical solution of partial differential equations by the finite element method*. Mineola, NY, USA: Dover Publicaions, Inc., 2009.

KÄLBERER, F.; NIESER, M.; POLTHIER, K. Quadcover: surface parametrization using branched coverings. *Computer Graphics Forum*, Blackwell Publishing Ltd, United Kingdom, v. 26, n. 3, p. 375–384, 2007.

KOLMOGOROV, V. Blossom v: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, Springer, Germany, v. 1, n. 1, p. 43–67, 2009.

KUHN, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, v. 2, p. 83–97, 1955.

LAI, M.-J. Scattered data interpolation and approximation using bivariate c^1 piecewise cubic polynomials. *Computer Aided Geometric Design*, v. 13, n. 1, p. 81–88, 1996.

LAI, Y.-K.; KOBBELT, L.; HU, S.-M. An incremental approach to feature aligned quad dominant remeshing. In: *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling (SPM'08)*. New York, NY, USA: ACM, 2008. p. 137–145.

LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids*. New York, NY, USA: Holt, Rinehart and Winston, 1976.

LOVÁSZ, L.; PLUMMER, M. D. *Matching Theory*. Amsterdam, The Netherlands: Elsevier Science Publishers B.V., 1986. (Annals of Discrete Mathematics, v. 29).

MALANTHARA, A.; GERSTLE, W. Comparative study of unstructured meshes made of triangles and quadrilaterals. In: *Proceedings of the 6th International Meshing Roundtable (IMR)*. Park City, Utah, USA: [s.n.], 1997. p. 437–447.

MARINOV, M.; KOBBELT, L. Direct anisotropic quad-dominant remeshing. In: *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications (PG'04)*. Washington, DC, USA: IEEE Computer Society, 2004. p. 207–216.

MEHLHORN, K.; SCHÄFER, G. Implementation of $\mathcal{O}(nm \log n)$ weighted matchings in general graphs: the power of data structures. *Journal of Experimental Algorithmics*, ACM, New York, NY, USA, v. 7, p. 4, 2002.

MEYER, M.; KIRBY, R. M.; WHITAKER, R. Topology, accuracy, and quality of isosurface meshes using dynamic particles. *IEEE Transactions on Visualization and Computer Graphics*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 13, n. 6, p. 1704–1711, 2007.

MICALI, S.; VAZIRANI, V. V. An $\mathcal{O}(|e| \cdot \sqrt{|V|})$ algorithm for finding maximum matchings in general graphs. In: *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1980. p. 17–27.

MUCHA, M.; SANKOWSKI, P. Maximum matchings via gaussian elimination. In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*. Washington, DC, USA: IEEE Computer Society, 2004. p. 248–255.

NING, P.; BLOOMENTHAL, J. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 13, n. 6, p. 33–41, 1993.

OWEN, S.; STATEN, M.; CANNAN, S.; SAIGAL, S. Q-morph: an indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering*, v. 9, n. 44, p. 1317–1340, 1999.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY, USA: Dover Publications, Inc., 1998.

PAV, S. E.; WALKINGTON, N. J. Delaunay refinement by corner lopping. In: *Proceedings of the 14th International Meshing Roundtable*. [S.l.: s.n.], 2005. p. 165–181.

RADÓ, T. Über den begriff der riemannschen fläche. *Acta Litterrarum ac Scientiarum*, v. 2, p. 101–121, 1925.

RAMASWAMI, S.; RAMOS, P.; TOUSSAINT, G. Converting triangulations to quadrangulations. *Computational Geometry Theory: and Applications*, v. 9, p. 257–276, 1998.

RAMASWAMI, S.; SIQUEIRA, M.; SUNDARAM, T.; GALLIER, J.; GEE, J. Constrained quadrilateral meshes of bounded size. *International Journal of Computational Geometry & Applications*, v. 15, n. 1, p. 55–98, 2005.

RAY, N.; LI, W. C.; LéVY, B.; SHEFFER, A.; ALLIEZ, P. Periodic global parameterization. *ACM Transactions on Graphics*, ACM, New York, NY, USA, v. 25, n. 4, p. 1460–1485, 2006.

SAIP, H. A. B. *Algoritmos para Emparelhamento em Grafos Bipartidos*. Dissertação (Mestrado) — Departamento de Computação, Universidade de Campinas, 1993.

SCHUMAKER, L. L. Triangulations in cagd. *IEEE Computer Graphics and Applications*, v. 13, n. 1, p. 47–52, 1993.

SUGIHARA, K. Laguerre voronoi diagram on the sphere. *Journal for Geometry and Graphics*, v. 6, n. 1, p. 69–81, 2002.

TARINI, M.; PIETRONI, N.; CIGNONI, P.; PANOZZO, D.; PUPPO, E. Practical quad mesh simplification. *Computer Graphics Forum*, Blackwell Publishing Ltd, United Kingdom, v. 29, n. 2, p. 407–418, 2010.

TONG, Y.; ALLIEZ, P.; COHEN-STEINER, D.; DESBRUN, M. Designing quadrangulations with discrete harmonic forms. In: *Proceedings of the 4th Eurographics Symposium on Geometry Processing (SGP'06)*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006. p. 201–210.

TRUDEAU, R. *Introduction to graph theory*. Mineola, NY, USA: Dover Publications, Inc., 1993.

VELHO, L. Quadrilateral meshing using 4-8 clustering. In: *Proceedings of the Symposium on Mesh Generation and Self-Adaptivity (CILANCE 2000)*. [S.l.: s.n.], 2000. p. 61–64.

VELHO, L.; FIGUEIREDO, L. H. de; GOMES, J. A unified approach for hierarchical adaptive tesselation of surfaces. *ACM Transactions on Graphics*, ACM, New York, NY, USA, v. 18, n. 4, p. 329–360, 1999.

VISWANATH, N.; SHIMADA, K.; ITOH, T. Quadrilateral meshing with anisotropy and directionality control via close packing of rectangular cells. In: *Proceedings of the 9th International Meshing Roundtable (IMR)*. New Orleans, Louisiana, USA: [s.n.], 2000. p. 217–225.

VLASSOPOULOS, V. Adaptive polygonization of parametric surfaces. *The Visual Computer*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 6, n. 5, p. 291–298, 1990.

WATT, A. *3D Computer Graphics*. Essex, England: Addison-Wesley Publishing Company, Inc., 1999.

WITZGALL, D.; ZAHN, C. T. J. Modification of edmonds' algorithm for maximum matching of graphs. *Journal of Research of the National Bureau of Standards*, v. 69B, p. 91–98, 1965.

ZHANG, Y.; BAJAJ, C. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering*, v. 195, n. 9-12, p. 942–960, 2006.