Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Leonardo Cordeiro Portella

Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado: uma análise de algoritmos exatos e não exatos

Rio de Janeiro
2023

Leonardo Cordeiro Portella

Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado: uma análise de algoritmos exatos e não exatos

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Fabiano de Souza Oliveira

Orientador: Prof. Dr. Paulo Eustáquio Duarte Pinto

Rio de Janeiro

2023

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC-A

P843 Portella, Leonardo Cordeiro.
Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado: uma análise de algoritmos exatos e não exatos/ Leonardo Cordeiro Portella – 2023.
71 f. : il.

Orientadores: Fabiano de Souza Oliveira, Paulo Eustáquio Duarte Pinto
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Teoria dos grafos - Teses. 2. Heurística - Teses. 3. Algoritmos de computador - Teses. .I. Oliveira, Fabiano de Souza. II. Pinto, Paulo Eustáquio Duarte. III. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. IV. Título.

CDU 519.17

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

Leonardo Cordeiro Portella

Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado: uma análise de algoritmos exatos e não exatos

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 25 de Janeiro de 2023.

Banca Examinadora:

Prof. Dr. Fabiano de Souza Oliveira (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Paulo Eustáquio Duarte Pinto (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Danilo Artigas da Rocha
Universidade Federal Fluminense - UFF

Prof. Dr. Igor Machado Coelho
Universidade Federal Fluminense - UFF

Prof.ª Dra. Lucila Maria de Souza Bento
Instituto de Matemática e Estatística - UERJ

Rio de Janeiro
2023

AGRADECIMENTOS

Primeiramente agradeço a Deus por sua bondade infinita na forma de Seu filho, o Senhor Jesus Cristo.

Agradeço a minha esposa e filha pelo amor, compreensão, paciência e encorajamento durante o período do projeto.

À minha mãe pelo amor, apoio e o incentivo desde pequeno aos estudos.

Aos meus orientadores Fabiano Oliveira e Paulo Eustáquio pelo incentivo, orientação e ensinamentos.

A todos os membros da banca por aceitarem avaliar este trabalho e pelas excelentes contribuições.

RESUMO

PORTELLA, Leonardo Cordeiro. *Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado*: uma análise de algoritmos exatos e não exatos. 2023. 71 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2023.

O Problema do Carteiro Chinês Ponderado é um problema clássico da Teoria de Grafos e consiste em identificar um percurso que contém cada aresta pelo menos uma vez, iniciando e terminando no mesmo vértice, de forma a minimizar o custo do percurso. A solução clássica inclui a identificação de um emparelhamento entre os vértices de grau ímpar do grafo. Algoritmos exatos para identificar tal emparelhamento são de difícil implementação e auditoria. Neste contexto, algoritmos não exatos, mais simples e geralmente mais rápidos, que retornam um resultado inferior ao ótimo e, alguns, com uma garantia de resultado mínimo comparado com o ótimo, são utilizados pela indústria. Neste trabalho comparamos os principais algoritmos exatos e não exatos. Solucionamos o CPP utilizando estes algoritmos e empregando dados de diversas regiões geográficas. Resultados para os piores casos para os algoritmos não exatos são discutidos. Fundamentos matemáticos para a compreensão do problema e dos algoritmos também estão inclusos.

Palavras-chave: Problema do Carteiro Chinês Ponderado. CPP. Emparelhamento. Heurística. Randomizado.

ABSTRACT

PORTELLA, Leonardo Cordeiro. *Matchings applied to the Weighted Chinese Postman Problem*: an analysis of exact and not exact algorithms. 2023. 71 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2023.

The Weighted Chinese Postman Problem is a classic problem of Graph Theory and consists of identifying a route that contains each end at least once, starting and ending at the same vertex, in order to minimize the cost of the route. Classic solution includes identifying a matching between the vertices of odd degree of the graph. Exact algorithms for identifying such matching are difficult to implement and audit. In this context, algorithms that are not exact, simpler that return a lower result than optimal and, in most cases, with a minimum result guarantee compared to the optimal, are used by the industry. In this work we compared the main exact and not exact algorithms. We solved the CPP using these algorithms and using data from different geographic regions. Results for the worst cases for the not exact algorithms are discussed. Mathematical fundamentals and algorithms for problem understanding are also included in this work.

Keywords: Weighted Chinese Postman Problem. CPP. Matching. Heuristics. Randomized.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

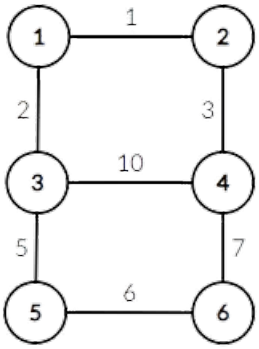
INTRODUÇÃO

Problema

O problema de percorrer todos os trechos de um dado mapa, cada qual interconectando dois pontos geográficos, retornando ao ponto de partida no término de forma a minimizar o custo associado ao trajeto percorrido é conhecido como Problema do Carteiro Chinês (CPP).

Uma instância do CPP é normalmente modelada por um grafo, onde o conjunto de vértices representam tais pontos geográficos e as arestas as interconexões permitidas no percurso, ponderadas pelo custo associado à travessia do trecho entre aqueles pontos. A Figura 1 ilustra um exemplo de grafo de entrada ao problema do CPP, sendo o vértice 1 o início e término desejado, com custo ótimo de 40, correspondendo ao percurso 1,2,4,6,5,3,4,2,1,3,1.

Figura 1 – Exemplo de um grafo de entrada ao problema do CPP.



Quando o grafo que representa o percurso possui todos os vértices com grau par, a solução do CPP é simples, tanto para a versão ponderada ou não, pois existe um circuito Euleriano. Este, naturalmente, é o circuito de custo mínimo, uma vez que todas as arestas precisam ser visitadas ao menos uma vez. Contudo, no caso contrário, a solução clássica consiste em reduzir o problema à obtenção de um emparelhamento ponderado de custo máximo em um grafo auxiliar, que será detalhado adiante.

Para obter este emparelhamento de custo máximo, algoritmos exatos como o algoritmo de Edmonds, com complexidade de tempo $O(n^3)$, onde n representa o número de vértices desse grafo, são muito custosos na prática quando as instâncias são grandes [3] [4].

Note ainda que é senso comum que algoritmos em grafos recaem em uma dificuldade de implementação mais elevada. Alguns autores endereçam as dificuldades no desenvolvimento e implementação de software científico, área na qual pode ser classificado o algoritmo de Edmonds. Segundo Hoare et al. [5], software científico tende a ser desenvolvido sem técnicas de Engenharia de Software, levando a muitas falhas na sua utilização. Elvira et al. [6] acentuam que essas dificuldades vêm da alta complexidade e tamanho, além da necessidade de manutenção intensiva e reutilização desse tipo de software. A

inerente dificuldade de implementação do algoritmo de Edmonds é um entrave para sua consideração em implementações na resolução de problemas reais [3]. Considerando que a maioria dos estudos não aprofundam os detalhes de implementação, a teoria implícita ao algoritmo geralmente é necessária para a correta implementação do software, de acordo com a especificação.

Em resumo, soluções em tempo polinomial são conhecidas para o CPP, mas a complexidade de tempo e a dificuldade de implementação desses algoritmos prejudicam sua utilização prática. Nesse contexto, estudos buscando alternativas que contornam essas dificuldades são desejáveis.

Objetivo

O presente trabalho tem como objetivo contribuir com o aspecto teórico e prático do CPP. Para isso, revisaremos os principais algoritmos não exatos para os problemas de emparelhamento ponderado de custo máximo e mínimo, comparando seus desempenhos em diversos grafos de teste, em instâncias sintéticas quanto oriundas de mapas reais. O problema do emparelhamento ponderado máximo é importante pois o problema do CPP se reduz a esse, como veremos adiante.

Justificativa

O problema de distribuir eficientemente bens e serviços em um local, seja este físico (como uma malha urbana) ou abstrato (como a representação da estrutura de redes de computadores de uma empresa) ganha maior importância à medida que estas estruturas aumentam de tamanho e, consequentemente, de complexidade. Outros fatores também impactam na complexidade, como, por exemplo, a atribuição de algum tipo de bônus ou ônus para as diferentes possibilidades de solução do problema, seja a distância física em um trajeto a ser percorrido ou tempo de envio de um pacote numa rede. Além disso, a competitividade motivada por fins econômicos dá importância à busca pelas melhores soluções.

A dificuldade de implementação e manutenção dos algoritmos eficientes exatos para o CPP é um entrave para as aplicações comerciais em que o código-fonte precisa ser mantido e/ou auditado. Dentro deste contexto, a importância de algoritmos que resolvam eficientemente e de forma simples o CPP é evidente, mesmo que a otimalidade precise ser sacrificada. Em outras palavras, nesse contexto, o uso de soluções não exatas de mais simples implementação são mais convenientes.

Motivação

Apesar do CPP derivar de um problema conhecido como o primeiro em Teoria dos Grafos, ele continua sendo um problema de interesse atual e, portanto, de grande relevância. Um exemplo é o recenseamento que está sendo realizado pelo IBGE no ano de 2023, que é base para outros estudos e pesquisas estatísticas. O Censo é a principal fonte de dados sobre a situação de vida da população nos municípios e localidades. Seus dados podem ser utilizados para a definição de políticas públicas em nível nacional, estadual e municipal bem como auxílio para a tomada de decisões na área de investimentos do setor privado [7].

No recenseamento também são definidas verbas de municípios conforme a densidade populacional. Outras pesquisas do mesmo instituto dependem de informações atualizadas, como a Pesquisa Nacional por Amostra de Domicílios Contínua (PNADC) que, entre outros indicadores, revela níveis de desemprego. Coleta de lixo urbano, entrega de correspondências e distribuição de bens e serviços são outros exemplos de utilização do CPP.

Definições

As definições utilizadas nesse trabalho, em geral, seguem as notações encontradas em [8].

Neste trabalho um *grafo* G consiste dos seguintes conjuntos:

- $V(G) = \{v_1,v_2,...,v_n\}$ representa o *conjunto de vértices*;
- $E(G) = \{e_1,e_2,...,e_m\}$ representa o *conjunto de arestas* onde cada $e_k \in E(G)$ corresponde a um par não-ordenado de vértices (v_i,v_j), com $1 \leq k \leq m$ e $1 \leq i,j \leq n$;
- $A(G) = \{a_1,a_2,...,a_r\}$ representa o *conjunto de arcos* onde cada $a_k \in A(G)$ consiste em um par ordenado de vértices (v_i,v_j), com $1 \leq k \leq r$ e $1 \leq i,j \leq n$;

Quando não houver ambiguidade, um par (u,v) (ordenado ou não) será denotado simplesmente por uv.

Denotamos por $L(G) = A(G) \cup E(G)$ o conjunto de *ligações* do grafo.

Um grafo G é classificado segundo a existência ou não de arestas e arcos da seguinte forma:

- *não-orientado*, se $A(G) = \emptyset$, com representação simplificada $G = (V,E)$;
- *digrafo*, se $E(G) = \emptyset$, com representação simplificada $G = (V,A)$;
- *misto*, se $E(G) \neq \emptyset$ e $A(G) \neq \emptyset$, representado por $G = (V,E,A)$.

Sejam $G = (V,E,A)$ um grafo misto e $R \subseteq V$, $S \subseteq E$ e $T \subseteq A$. Denotamos por:

- $G - R = (V \setminus R, E \setminus \{uv \in E : u \in R \text{ ou } v \in R\}, A \setminus \{uv \in A : u \in R \text{ ou } v \in R\})$;
- $G - S = (V, E \setminus S, A)$;
- $G - T = (V, E, A \setminus T)$.

Para este trabalho, exceto quando dito algo contrário, consideraremos um grafo como não-orientado. Todos os grafos, independente de sua classificação, serão considerados sem arestas paralelas e sem laços. Um grafo orientado é *fortemente conexo* quando houver pelo menos um caminho direcionado entre cada par de vértices do grafo.

Para os problemas que pretendemos abordar nesse trabalho, existirão usualmente *custos* ou *pesos* associados às ligações. Tais custos são representados pela função de custos $c(i,j)$ (ou, ainda, $c(v_i, v_j)$), onde $c(i,j)$ denota o custo associado à ligação entre o par de vértices v_i,v_j. Estes custos das ligações serão usados para definir custos de passeios.

Dado um grafo G, um *passeio* P de G é uma sequência $v_0,v_1,\ldots,v_k$ de vértices tal que, para todo $1 \leq i \leq k, (v_{i-1},v_i) \in L(G)$; tal aresta é dita pertencente ao passeio. Denominamos por $P(i,j)$ o passeio $v_i,\ldots,v_j$, com $0 \leq i < j \leq k$. Uma *trilha* é um

passeio cujas ligações são todas distintas. Um passeio é dito *fechado* se suas extremidades são idênticas, isto é, $v_0 = v_k$. Um *circuito* é uma trilha fechada. Um *caminho* é um passeio cujos vértices são todos distintos. Um *ciclo* é um caminho fechado.

Dado um passeio P de um grafo G, denotamos por $V(P)$, $E(P)$ e $A(P)$ respectivamente o conjunto de vértices, arestas e arcos de G pertencentes a P. O custo $c(P)$ é dado por

$$c(P) = \sum_{(v_i,v_j)\in P} c(i,j)$$

Sejam A e B dois conjuntos. A *diferença simétrica* $A\Delta B$ entre A e B representa o conjunto de elementos que pertencem exclusivamente a A ou B, ou seja, temos que $A\Delta B = (A \cup B) \setminus (A \cap B)$.

O *grau de um vértice* v, denotado por $d(v)$, corresponde à quantidade de arestas incidentes a este. O *grau mínimo de um grafo* G, denotado por $\delta(G)$, corresponde ao grau do vértice de menor grau de G.

Uma *aresta de corte* ou *ponte* é uma aresta $e \in E(G)$ tal que $G \setminus \{e\}$ é desconexo.

Dado um grafo conexo, o CPP consiste em identificar um passeio fechado que contém cada aresta desse grafo pelo menos uma vez, de forma que tal passeio possua o menor custo possível. A definição detalhada do CPP, incluindo exemplos, é apresentada na Seção 2.1.

Para este trabalho consideraremos que os grafos de entrada para o CPP são conexos pois, caso contrário, o problema não teria solução. O escopo deste trabalho se atém apenas a grafos com custo positivo das arestas.

Contribuições

A proposta desta pesquisa foi estudar o CPP com pesos (ou ponderado) para grafos conexos. Discutimos, na perspectiva teórica, as principais abordagens existentes. Na perspectiva prática, comparamos resultados obtidos por algoritmos existentes, tanto exatos quanto não exatos. As comparações entre os algoritmos não exatos foram feitas tanto com instâncias sintéticas quanto oriundas de mapas reais. Os piores casos para os algoritmos não exatos são discutidos.

Estrutura

O restante do trabalho está assim organizado. O Capítulo 1 apresenta os principais conceitos e resultados teóricos, possibilitando a completa compreensão dos temas abordados nos capítulos posteriores. O Capítulo 2 aborda os principais problemas e trabalhos relacionados ao CPP e emparelhamentos ponderados. O Capítulo 3 apresenta os principais resultados obtidos para grafos sintéticos e compara estes resultados entre os algoritmos não exatos e exato. O Capítulo 4 apresenta resultados para casos reais de grafos que representam diversas regiões geográficas e soluciona o CPP para estes grafos, comparando os resultados entre os algoritmos não exatos e exato.

Por fim, são apresentadas as conclusões e sugestões para trabalhos futuros.

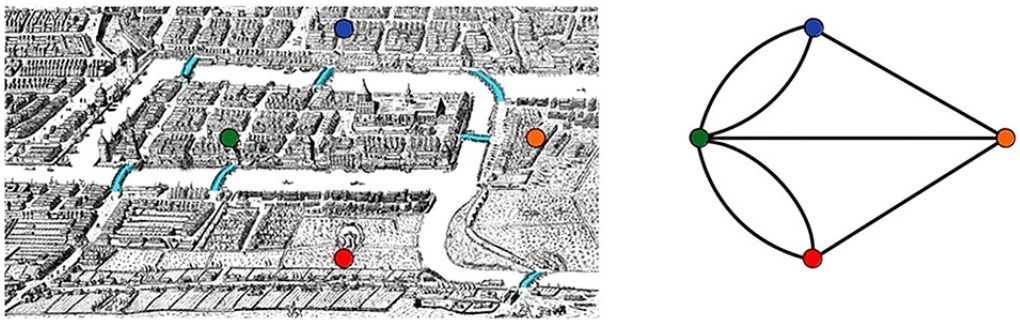
1 FUNDAMENTOS TEÓRICOS

Nesse capítulo discutiremos as principais bases teóricas, fundamentais para a compreensão deste trabalho. Abordaremos as principais definições da teoria de grafos relacionadas ao CPP e emparelhamentos. Na Seção 1.1, apresentaremos o primeiro problema da teoria de grafos. Na Seção 1.2, revisaremos os principais conceitos de emparelhamento. Na Seção 1.3, discutiremos o Problema da Determinação de Caminhos Mínimos.

1.1 Problema do passeio de Euler (PE)

Conhecido como o primeiro problema e possivelmente o registro mais antigo da teoria de grafos, o *Problema do passeio de Euler (PE)*, também conhecido como problema das pontes de Königsberg, foi solucionado pelo matemático e físico suíço Leonhard Euler em 1736 [2].

Figura 2 – Pontes de Königsberg e sua representação em um grafo.



Fonte: https://www.unicamp.br/unicamp/ju/artigos/peter-schulz/ligando-os-pontos-da-pandemia

Na cidade de Königsberg na então Prússia (hoje Kaliningrado, Rússia), existem duas ilhas situadas no Rio Pregel. Estas ilhas são interligadas às margens do rio por meio de sete pontes, conforme disposição na Figura 2. O problema consiste em, partindo e terminando em uma margem ou ilha, determinar um circuito passando por cada uma das pontes exatamente uma vez.

Um *circuito de Euler* ou *circuito euleriano* é um circuito que percorre cada aresta exatamente uma vez, sendo o grafo que contém este circuito denominado *grafo euleriano* [9] ou *grafo unicursal* [10]. Grafos eulerianos possuem uma caracterização trivial de se verificar algoritmicamente (Teorema 1). Agora demonstraremos o Lema 1 que utilizaremos em seguida na demonstração do Teorema 1 de Euler.

Lema 1. *Se um grafo conexo G é tal que $\delta(G) \geq 2$, então G possui um circuito.*

Demonstração. Seja $(v_0, v_1) \in E(G)$. Como $d(v_1) \geq 2$, existe $(v_1, v_2) \in E(G)$, com $v_2 \neq v_0$. Se $(v_0, v_2) \in E(G)$, então há um circuito em G. Caso contrário, como $d(v_2) \geq 2$, existe $(v_2, v_3) \in E(G)$, com $v_3 \neq v_1$. Se $(v_0, v_3) \in E(G)$ ou $(v_1, v_3) \in E(G)$ então há um circuito em G. Podemos repetir esse argumento para todo $i \geq 3$: como $d(v_i) \geq 2$, existe $(v_i, v_{i+1}) \in E(G)$ com $v_{i+1} \notin \{v_0, v_1, \ldots, v_i\}$. De fato, se $v_{i+1} = v_j$ para algum $j = 0,1,\ldots,i-2$, então há o circuito $v_j, v_{j+1}, \ldots, v_i, v_{i+1} = v_j$ em G. Como G é finito, este deve ser o caso para algum $i \geq 3$. □

Teorema 1 (Euler, 1736)**.** *Um grafo conexo G contém um circuito euleriano se e somente se todos os seus vértices possuírem grau par.*

Demonstração. Seja G um grafo conexo que contém um circuito euleriano C, representado por uma sequência de vértices que inicia e termina por $w \in V(G)$. Como C é um circuito euleriano, toda aresta de G aparece exatamente uma vez em C. Seja $v \in V(G)$. Note que a cada vez que o percurso C atinge v por alguma aresta de entrada, ele deve sair de v por alguma aresta de saída. Como cada aresta incidente a v ocorre em C exatamente uma vez, o número de arestas incidentes a v deve ser par. Como v e w são vértices quaisquer, então todo vértice de G tem grau par.

Suponha agora que G é conexo e que todos os vértices de G possuem grau par. Provaremos por indução no número de arestas m de G que G é euleriano.

Se $m = 3$, então $G \cong C_3$ provando o resultado.

Se $m > 3$, então suponha por indução que todo grafo conexo com menos de m arestas tal que todos os seus vértices possuem grau par seja euleriano.

Como $d(v) \geq 2$ todo $v \in V(G)$ e G é conexo, temos pelo Lema 1 que existe um circuito C em G. Se C contém todas as arestas de G o teorema está provado. Se não, seja $H = G \setminus E(C)$. Possivelmente H é desconexo e certamente com menos arestas que G. Cada vértice de H que existia em C teve seu grau diminuído em duas unidades, referentes às arestas de C incidentes no respectivo vértice. Sendo assim, cada componente conexa de H possuirá todos os vértices com grau par e menos de m arestas. Portanto, cada componente de H é euleriana por hipótese de indução. Sejam $C_1, C_2, \ldots, C_k$ os circuitos eulerianos das componentes conexas de H. Veja que C deve possuir um vértice em comum com cada C_i, $i = 1,2,\ldots,k$. Caso contrário, nenhum vértice de C_i seria adjacente a um vértice de C em G e portanto G seria desconexo. Defina agora o seguinte percurso: seja $v \in C$. Inicie o percurso por v percorrendo C em qualquer sentido até voltar a v. Durante o percurso, a cada vértice comum a C e C_i alcançado pelo percurso, percorra por C_i até voltar a tal vértice antes de continuar este percurso em C. Se houver mais de um vértice em comum, faça isso apenas na primeira ocorrência. Claramente seguindo tal percurso, todas as arestas de G serão percorridas exatamente uma vez. Portanto, G é euleriano. □

Ao provar o Teorema 1, Euler verificou que não há solução para a instância do PE original da Figura 2, uma vez que existe o trajeto procurado, nos termos daquele problema, quando e somente quando em cada região ocorrer um número par de pontes.

O Algoritmo 1, chamado algoritmo de Fleury, determina um ciclo euleriano. Esse algoritmo possui complexidade de $\Theta(m^2)$ e tem a desvantagem de necessitar a detecção de pontes, cujos algoritmos não são tão diretos. O Algoritmo 2 apresenta um algoritmo de tempo $\Theta(m)$, e de mais simples implementação.

Algoritmo 1: Algoritmo de Fleury.

Complexidade de Tempo: $\Theta(m^2)$.

Entrada: $G = (V,E)$
Saída: Um ciclo euleriano W que começa e termina em v_0
Dados: G é conexo, $d(v)$ é par para todo $v \in V$ e v_0 é um vértice arbitrário de V

```
1  Função Fleury(G, v_0):
2      W ← v_0
3      u ← v_0
4      F ← G
5      enquanto δ_F(u) ≠ 0 faça
6          e ← uv | G \ {(u,v)} seja conexo
7          W ← W + v /* W + v denota o acréscimo de v à sequência W */
8          u ← v
9          F ← F \ {e}
10     retorna W
```

Algoritmo 2: Algoritmo Ciclo Euleriano [1].

Complexidade de Tempo: $\Theta(m)$.

Entrada: $G = (V,E)$ e o vetor de graus $d\colon V(G) \to \mathbb{N}$
Saída: Um ciclo euleriano W que começa e termina em v_0
Dados: G é conexo, $d(v)$ é par para todo $v \in V$ e v_0 é um vértice arbitrário de V

```
1  Função Ciclo_Euleriano(G, d, v_0):
2      W ← ∅ /* Pilha W */
3      P ← ∅ /* Pilha auxiliar P */
4      Empilhe v_0 em P
5      enquanto P ≠ ∅ faça
6          v ← topo da pilha P
7          w ← um vértice qualquer adjacente a w
8          Empilhe w em P
9          E(G) ← E(G) \ (v,w)
10         enquanto P ≠ ∅ e d(topo da pilha) = 0 faça
11             Desempilhe o topo da pilha P e empilhe-o na pilha W
12     retorna W
```

O Algoritmo 2 é baseado na prova construtiva do Teorema 1. Ele empilha vértices de um ciclo e, após empilhar todo um ciclo começa a retirar esses vértices da pilha, interrompendo o processo para considerar outros ciclos que tenham intersecção com o ciclo que está sendo tratado na pilha. O algoritmo utiliza um processo destrutivo, eliminando sucessivamente as arestas dos ciclos considerados.

Euler também contribuiu com o Teorema 2 a seguir, que relaciona a quantidade de vértices com a soma de seus respectivos graus e seu respectivo Teorema 3 que utilizaremos no CPP.

Teorema 2. *A soma dos graus de todos os vértices de um grafo G é igual ao dobro da quantidade de arestas de G.*

Demonstração. Como cada aresta (u,v) contribui com 1 em $d(u)$ e com 1 em $d(v)$, temos que a soma dos graus de todos os vértices contabiliza cada aresta exatamente duas vezes, seguindo o resultado. □

Teorema 3. *O número de vértices de grau ímpar de um grafo G é sempre par.*

Demonstração. Seja n' o número de vértices de G com grau ímpar. Suponha que n' seja ímpar. Logo, o somatório dos graus dos vértices de G é ímpar, uma vez que a soma dos graus de vértices de grau ímpar é ímpar e a soma dos graus de vértices de grau par é par, o que contradiz o Teorema 2. Portanto, o número de vértices de grau ímpar de G deve ser par. □

1.2 Emparelhamento

Como veremos adiante, o CPP admite uma solução via redução ao problema de emparelhamentos máximos. Um *emparelhamento* de um grafo G consiste em um conjunto de arestas de $E(G)$ tal que nenhum par dessas arestas compartilha um vértice [11]. Um emparelhamento M é dito *máximo* se, para todo emparelhamento M', temos que $|M'| \leq |M|$. Um emparelhamento M é dito *maximal* se nenhuma aresta do grafo puder ser adicionada a M sem violar a característica de não compartilhamento de vértices par a par de arestas.

Observe que todo emparelhamento máximo é maximal mas nem todo emparelhamento maximal é máximo. Um caminho ou ciclo $v_1, v_2, \ldots, v_k$ é dito *M-alternante* quando suas arestas encontram-se alternadamente em M e fora de M, isto é, $(v_i, v_{i+1}) \in M \iff (v_{i+1}, v_{i+2}) \notin M$ para todo $1 \leq i \leq k-2$ para caminhos e ciclos, e adicionalmente $(v_{k-1}, v_k) \in M \iff (v_1, v_k) \notin M$ para ciclos. Um vértice v de um emparelhamento M é dito *saturado* se alguma aresta de M incide em v. Vértices que não são incidentes a arestas de M são chamados de *expostos*. Um emparelhamento é dito *perfeito* quando todos os vértices são saturados. Um caminho M-alternante cujos vértices extremos são ambos expostos é dito *M-aumentante* [2]. O *custo de um emparelhamento M* de um grafo ponderado, representado por $c(M)$, é dado pelo somatório dos custos das arestas de M.

Demonstraremos a seguir o Teorema 4 de Berge, utilizado no algoritmo de Edmonds para calcular o emparelhamento perfeito de custo máximo [8]. Antes precisaremos demonstrar os Lemas 2 e 3.

Lema 2 ([2])**.** *Todo caminho M-aumentante possui uma quantidade par de vértices.*

Demonstração. Por definição, os vértices extremos de um caminho M-aumentante são expostos. Seja $P = v_1, v_2, \ldots, v_k$ um caminho M-aumentante onde $k = |V(P)|$. Para $k = 2$ o resultado é trivial. Nos outros casos note que $(v_i, v_{i+1}) \in M$ para todo $2 \leq i < k$ com i par. Como P é M-aumentante, temos que $(v_{k-1}, v_k) \notin M$. Sendo assim, temos que $k-1$ deve ser ímpar e, portanto, k deve ser par. □

Lema 3 ([2])**.** *Sejam G um grafo, M e M' emparelhamentos de G. Seja $H = G[M \Delta M']$. Cada componente conexa de H é um ciclo de comprimento par com arestas alternadamente em M e M' ou um caminho com arestas alternadamente em M e M'.*

Demonstração. Cada vértice de H tem grau 1 ou 2, já que pode ser incidente a no máximo uma aresta de M e a uma aresta de M'. Deste modo temos que cada componente conexa de H é um caminho ou ciclo com arestas alternadamente em M e M'.

Seja $C = v_1, \ldots, v_k, v_1$ um ciclo qualquer de H e $k = |E(C)|$ o comprimento de C. Suponha que k seja ímpar. Sem perda de generalidade, temos que para todo $1 \leq i < k$

com i ímpar vale $(v_i,v_{i+1}) \in M$. Como C é um ciclo M-alternante em M e M', então $(v_k,v_1) \notin M$, $(v_{k-1},v_k) \in M$, $(v_{k-2}, v_{k-1}) \notin M$. Como $k-2$ é ímpar, a última aresta deveria pertencer a M, contradizendo a hipótese. Logo o comprimento de cada ciclo de H deve ser par.

□

Teorema 4 (Berge, 1957)**.** *Seja G um grafo qualquer e M um emparelhamento de G. Então M é emparelhamento máximo se e somente se G não possui caminho M-aumentante.*

Demonstração. Seja M um emparelhamento com cardinalidade máxima de G e suponha que G possui um caminho M-aumentante P. Observe que P começa e termina por vértices expostos e, conforme o Lema 2, possui um número par de vértices. Seja $P = v_0,v_1,...,v_{2k+1}$ este caminho e $M \cap P = \{(v_1,v_2),(v_3,v_4),..., (v_{2k-1},v_{2k})\}$, com $|M \cap P| = k$. Façamos agora $M' = E(P)\Delta M$ e, portanto, $M' \cap P = \{(v_0,v_1),(v_2,v_3),..., (v_{2k},v_{2k+1})\}$ e, consequentemente, $|M' \cap P| = k+1$. Como $M' \setminus E(P) = M \setminus E(P)$, temos que M' possui cardinalidade maior que M e, portanto, M não possui cardinalidade máxima, uma contradição. Logo, se M é emparelhamento máximo então G não possui caminho M-aumentante.

Suponha agora que M não possui cardinalidade máxima em G. Seja M' um emparelhamento de cardinalidade máxima em G. Sendo assim temos que $|M'| > |M|$. Temos que cada componente conexa de $M\Delta M'$ é um ciclo de comprimento par ou caminho, conforme o Lema 3. Como M e M' são emparelhamentos, os ciclos de $M\Delta M'$ possuem a mesma quantidade de arestas de M e M'. Contudo, como $|M'| > |M|$ deve existir pelo menos um caminho P em $M\Delta M'$ com mais arestas em $|M'|$ do que em $|M|$, que começa e termina em $|M'|$, sendo seus vértices inicial e final expostos em relação a M. Sendo assim P é um caminho M-aumentante. Portanto, se M não é máximo então G possui caminho M-aumentante. □

1.2.1 Emparelhamentos em grafos bipartidos

Um grafo G é *bipartido* quando o seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2, tais que toda aresta de G une um vértice de V_1 a outro de V_2 [2]. O emparelhamento de um grafo bipartido é consideravelmente mais fácil de determinar, em comparação com o de grafos gerais. Os Teoremas 4 e 5 caracterizam um grafo bipartido em função do comprimento de seus ciclos.

Teorema 5. *Um grafo $G = (V,E)$ é bipartido se e somente se todo ciclo de G possuir comprimento par.*

Demonstração. Seja G um grafo bipartido, com V(G) particionado em V_1 e V_2. Seja $C = v_1,v_2,\ldots,v_k,v_1$ um ciclo de G. Sem perda de generalidade, sejam $v_1 \in V_1$, $v_2 \in V_2$. Portanto, para todo $1 \leq j \leq k$, $v_j \in V_1$ se j é ímpar e $v_j \in V_2$ se j é par. Como $(v_k,v_1) \in E(G)$, temos que $v_k \in V_2$ e portanto k é par.

Demonstraremos que se todo ciclo de um grafo possuir comprimento par então o grafo é bipartido. Seja G um grafo onde todo ciclo é de comprimento par. Seja $v \in V_1$ um vértice qualquer e $A \subseteq V(G)$ o conjunto dos vértices que estão a uma distância par de v e $B = V(G) \setminus A$. Suponha que $A \cup B$ não seja uma bipartição válida para G, por absurdo. Sendo assim, deve haver pelo menos uma aresta (a,b) entre vértices de uma mesma parte. Note que os caminhos cujos extremos estejam em uma mesma parte possuem comprimento par, enquanto aqueles cujos extremos estejam em partes diferentes possuem comprimento ímpar. Portanto, os caminhos entre v e a, e entre v e b unidos com

a aresta (a,b) formam um ciclo de comprimento ímpar, uma contradição com a hipótese. Portanto, G é bipartido. □

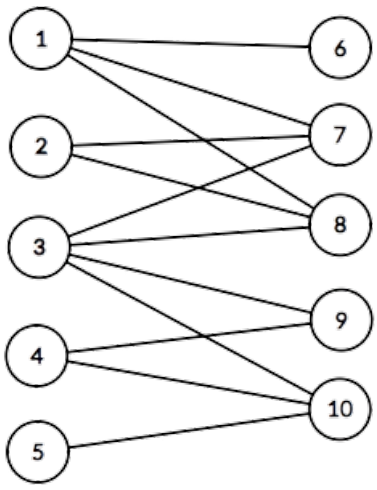
Um grafo que não possui ciclos é *acíclico*. Denomina-se *árvore* um grafo que seja acíclico e conexo [2]. Toda árvore é bipartida por consequência do Teorema 4.

Lema 4. *Se G é uma árvore então existe apenas um caminho entre cada par de vértices.*

Demonstração. Por definição G é conexo e acíclico. Suponha por absurdo que existem dois caminhos entre dois vértices quaisquer. Então a união destes dois caminhos contém um ciclo, uma contradição. Portanto, deve existir apenas um caminho entre cada par de vértices de G. □

Uma abordagem para identificar um emparelhamento de cardinalidade máxima em um grafo bipartido G através do fluxo em redes é encontrada em [12] e consiste em construir um grafo G' a partir de G da seguinte forma. Direcionar as arestas incidentes nos vértices de V_1 para V_2, onde V_1 e V_2 constitui a bipartição de V(G). Em seguida, adicionar uma fonte f e uma aresta direcionada para cada vértice de V_1 a partir de f, além de adicionar um sumidouro s e uma aresta direcionada de cada vértice em V_2 para s. A capacidade de cada aresta é definida como igual a 1. Pode-se argumentar que o valor do fluxo máximo nessa rede G' é igual ao emparelhamento máximo de G. Para calcular o fluxo máximo, pode-se utilizar o Algoritmo de Ford-Fulkerson que possui complexidade de tempo $\Theta(nm)$ [13]. A Figura 3 ilustra um grafo G bipartido e a Figura 4 a rede G' derivada, cujo fluxo máximo F possui custo igual a 5 e arestas correspondentes a F encontram-se em negrito. Note que F é justamente o emparelhamento máximo de G, dado por {(1,6),(2,7),(3,8),(4,9),(5,10)}.

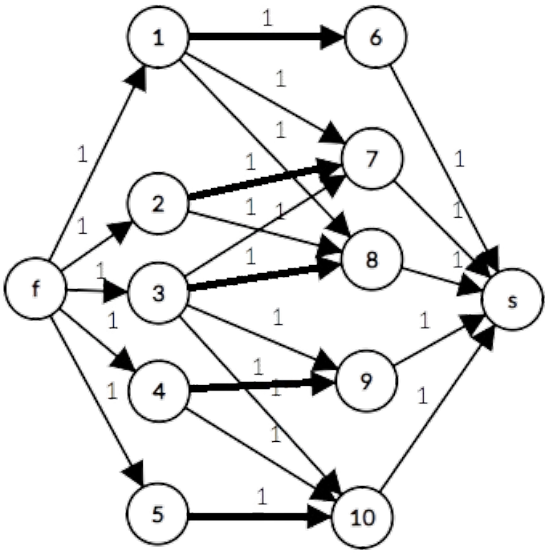
Figura 3 – Grafo bipartido G utilizado como exemplo para o problema do emparelhamento máximo em grafos bipartidos.



Uma *cobertura por vértices* de um grafo $G = (V,E)$ é um subconjunto $C \subseteq V$ onde cada aresta de G possui pelo menos uma extremidade em C. Denominamos uma *cobertura por vértices ponderada* de G uma função cob: $V(G) \rightarrow \mathbb{R}$, que atribui a cada vértice $v_i \in V$ um valor satisfazendo, para cada aresta $(v_i,v_j) \in E$, $cob(v_i)+cob(v_j) \geq c(v_i,v_j)$. Definimos a cobertura por vértices de um grafo G por

$$cob(G) = \sum_{v \in V(G)} cob(v)$$

Figura 4 – Rede G' derivada do grafo G da Figura 3.



Para uma aresta qualquer $(v_1,v_2) \in E$, a diferença $cob(v_1)+cob(v_2)-c(v_1,v_2)$ é denominada *excesso* da aresta e, sendo denotada por $\varepsilon(e)$. Uma *cobertura trivial* é uma cobertura que satisfaz $cob(v) = \max(c(v,v_2) \mid (v,v_2) \in E)$ se $v \in V_2$ e, caso contrário, $cob(v) = 0$.

Teorema 6. *Seja G um grafo bipartido e M um emparelhamento ponderado máximo de G, tal que todas as arestas de M possuem excesso zero. Então $c(M) = cob(G)$.*

Demonstração. Sem perda de generalidade, podemos considerar que $G = (V,E)$, onde temos que $V = V_1 \cup V_2$, é bipartido completo regular pois, se assim não for, podemos acrescentar vértices, para assegurar $|V_1| = |V_2|$ bem como arestas de peso nulo, as quais não influem nos pesos do emparelhamento máximo ou cobertura mínima. Da mesma forma, podemos acrescentar arestas de peso nulo a M de modo a torná-lo perfeito. Sabemos que, para cada aresta $e = (v,w) \in M$, $c(e) = cob(v) + cob(w)$, uma vez que $\varepsilon(e) = 0$. Como M é um emparelhamento perfeito de G, a união dos vértices incidentes às arestas de M é o conjunto V, onde cada vértice aparece exatamente uma vez. Logo, temos que

$$c(M) = \sum_{e \in M} c(e) = \sum_{(v,w) \in M} (cob(v) + cob(w)) = \sum_{v \in V} cob(v) = cob(G)$$

□

Algoritmo 3: Emparelhamento ponderado máximo - Grafos bipartidos [2].

Complexidade de Tempo: $\Theta(n^2m)$.

Entrada: $G = (V,E)$ e a função de custo c: $E(G) \to \mathbb{R}^+$
Saída: Um emparelhamento perfeito M
Dados: G é bipartido, $V = V_1 \cup V_2$, $|V_1| = |V_2|$

```
1  Função Emparelhamento_Bipartido(G,c):
2      para v ∈ V2 faça
3          cob(v) ← max(c(v,v2) | (v,v2) ∈ E)
4      para v ∈ V1 faça
5          cob(v) ← 0
6      para (v1,v2) ∈ E faça
7          ε(v1,v2) ← cob(v1) + cob(v2) − c(v1,v2)
8      E0 ← {(v1,v2) ∈ E | ε(v1,v2) = 0}
9      G0(V0,E0) ← G[E0] /* subgrafo induzido em G, por E0 */
10     enquanto |M| < n/2 faça
11         M ← emparelhamento de cardinalidade máxima de G0
12         D ← Construir_D(M) /* Grafo onde ∀(v1,v2) ∈ E, v1 ∈ V1, v2 ∈ V2, (v1,v2) é
               direcionada de v1 para v2 se (v1,v2) ∈ M e, caso contrário, de v2 para v1. */
13         se |M| < n/2 então
14             V' ← {v ∈ V0 | v é alcançável em D de algum v2 ∈ V0 ∩ V2, v2 exposto}
15             ε ← min {ε(v1,v2) | v1 ∈ V1 \ V' e v2 ∈ V2 ∩ V'}
16             para v ∈ V1 ∩ V' faça
17                 cob(v) ← cob(v) + ε
18             para v ∈ V2 ∩ V' faça
19                 cob(v) ← cob(v) − ε
20             para (v1,v2) ∈ E faça
21                 ε(v1,v2) ← cob(v1) + cob(v2) − c(v1,v2)
22             E0 ← {(v1,v2) ∈ E | ε(v1,v2) = 0}
23             G0(V0,E0) ← G[E0]
24     retorna M
```

O Algoritmo 3 utiliza o conceito do Teorema 6, buscando pela cobertura por vértices de custo mínimo. Para isto, inicia com uma cobertura trivial e calcula o excesso para cada aresta. Em seguida, iterativamente busca diminuir os custos da cobertura de alguns vértices em V_2 e aumentar o custo da cobertura de outros em V_1, gerando um subgrafo induzido nas arestas cujo excesso é zero e obtendo o emparelhamento maximal deste subgrafo.

O grafo D direcionado é utilizado para identificar candidatas ao emparelhamento a partir de um vértice exposto pertencente a V_2. O grafo D é tal que, para cada aresta $(v_1,v_2) \in E, v_1 \in V_1$ e $v_2 \in V_2$, (v_1,v_2) é direcionada de v_1 para v_2 se $(v_1,v_2) \in M$, caso contrário é direcionada de v_2 para v_1. Na Figura 5, mostra-se um grafo bipartido ponderado G, com bipartição $V_1 = \{1,2,3\}$ e $V_2 = \{4,5,6\}$ de G. Inicialmente, *cob* tem os seguintes valores: 0,0,0,1,3,5, para todo $v = 1,\ldots,6$, respectivamente. Assim, $E_0 = \{(1,4),(2,5),(3,6)\}$. O emparelhamento máximo M de G_0 obtido é igual a $\{(1,4),(2,5)\}$. Note que como M não é perfeito, cria-se o digrafo D conforme mostrado na Figura 6, para

o qual $V' = \{3\}$ e $\varepsilon = 1$. Logo, a nova cobertura *cob* é: 1,1,0,0,2,4. E, o novo conjunto $E_0 = \{(1,4),(2,5),(2,6),(3,6)\}$. O novo emparelhamento máximo M de G_0 obtido é igual a $\{(1,4),(2,5),(3,6)\}$, é perfeito e possui custo 8.

Figura 5 – Grafo bipartido ponderado G utilizado como exemplo para o problema do emparelhamento ponderado máximo em grafos bipartido.

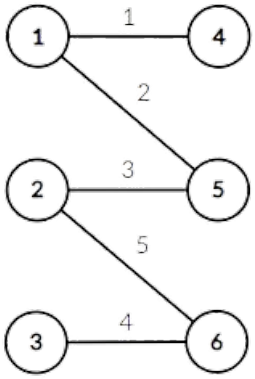
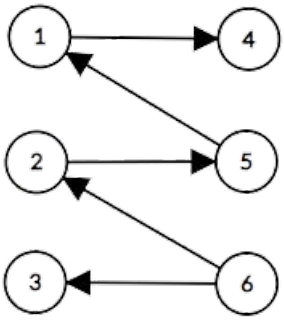


Figura 6 – Digrafo D gerado na primeira iteração do Algoritmo 3 tendo como entrada o grafo da Figura 5.

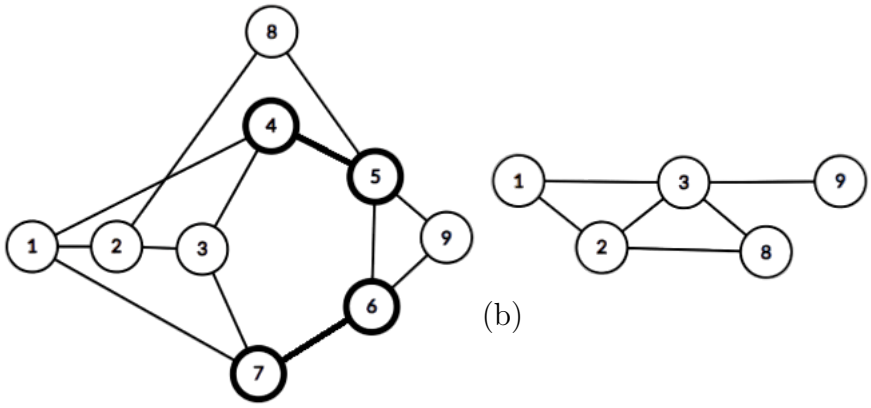


1.2.2 Emparelhamento máximo em grafos não ponderados quaisquer

Relembre que o Teorema 4 relaciona um emparelhamento M em um grafo qualquer G ser máximo com a ausência de caminhos aumentantes.

Sejam G um grafo e C um ciclo ímpar de G com tamanho $2k + 1$ que contém k arestas de M. Seja $c \in V(C)$ o único vértice de C tal que, ou $(c,d) \in M \setminus E(C)$, ou c é exposto. O *encolhimento* de C em G, representado por G/C, é obtido através da remoção de cada aresta $(u,v) \in E(G)$ tal que $u \in V(C)$ e, em seguida, adiciona-se uma aresta (r,c) para cada $(r,s) \in E(G)$ tal que $r \notin V(C), s \in V(C)$. Por fim, remove-se cada vértice $v \in V(C) \setminus \{c\}$.

Figura 7 – Grafo antes e após sofrer encolhimento.



A Figura 7(a) exibe um grafo antes de sofrer encolhimento. Após o encolhimento, todas as arestas incidentes a qualquer vértice do ciclo ímpar são removidas. Em seguida, uma aresta é adicionada entre cada vértice incidente ao ciclo e o vértice 3. O grafo resultante é exibido na Figura 7(b). Arestas e vértices destacados em negrito representam arestas do emparelhamento e vértices saturados.

Lema 5. *Sejam G um grafo, M um emparelhamento em G e C um ciclo ímpar de tamanho $2k+1$ que contém k arestas de M tal que todos os vértices saturados de C encontram-se saturados por arestas pertencentes a $E(C) \cap M$. Construa um novo grafo $G' = G/C$. Então o emparelhamento $M' = M \setminus E(C)$ é máximo em G' se e somente se M é máximo em G.*

Demonstração. Suponha que M não é máximo em G. Então existe um caminho M-aumentante P pelo Teorema 4. Se P é disjunto de C então P também é um caminho aumentante em G' relativo a M' e portanto este não é máximo. Se P é não disjunto de C então pelo menos um vértice x de uma das extremidades de P não pertence a C. Iniciando em x, seja c o primeiro vértice de C encontrado percorrendo P. Então $P(x,c)$ está mapeado em um caminho aumentante relativo a M' e portanto M' não é máximo. Suponha agora que M' não é um emparelhamento máximo em G' e seja N' um emparelhamento de cardinalidade maior que a de M'. Agora expanda C de modo a recuperar G. Note que N' pode ser aumentado em k arestas de C, produzindo o emparelhamento N de G, de forma que $|N| = |N'| + k > |M'| + k = |M|$. Logo, M não é emparelhamento máximo em G. □

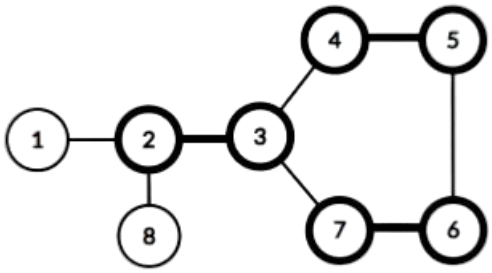
Os grafos da Figura 7 exemplificam o Lema 5, onde ambas contém um emparelhamento que não é máximo, para um grafo e seu grafo encolhido, respectivamente.

Dado um emparelhamento M, um *passeio M-alternante* é similar a um caminho M-alternante, exceto que passeios são considerados ao invés de caminhos, as arestas encontram-se alternadamente em M e fora de M e ambos os vértices extremos são distintos.

Um *passeio M-aumentante* é um passeio M-alternante cujos vértices extremos são expostos. Uma *floração* é um passeio M-alternante $v_0, v_1, \ldots, v_t$, tal que v_0 é exposto, t é ímpar e $v_t = v_i$, para algum $i < t$. O caminho M-alternante $v_0, \ldots, v_i$ de comprimento par é a *haste* e é seguido de um ciclo de comprimento $2k+1$, onde k arestas são emparelhadas,

denominado *flor* e o vértice v_i é a *base* da flor [2] [14]. Note que v_i é saturado se e somente se $i > 0$.

Figura 8 – Exemplo de um passeio M-aumentante contendo uma floração.



Na Figura 8 temos que a sequência de vértices 1,2,3,4,5,6,7,3,2,8 constitui um passeio M-aumentante. A floração é constituída pela flor F, o ciclo ímpar 3,4,5,6,7,3, pela sua haste, o caminho 1,2,3 e sua base, o vértice 3. O vértice 8 não faz parte da floração. Vértices em negrito encontram-se saturados e arestas em negrito fazem parte do emparelhamento.

Teorema 7. *Seja G um grafo, M um emparelhamento em G e F uma flor de uma floração de G. Então o emparelhamento M é máximo se e somente se $M' = M/F$ é máximo em G/F.*

O Teorema 7 é conhecido como Teorema de Edmonds serve de base para o Algoritmo 4.

Algoritmo 4: Emparelhamento em grafos gerais [2].

Complexidade de Tempo: $\Theta(n^2m)$.

Entrada: $G(V,E)$
Saída: Um emparelhamento M máximo

```
1  Função Emparelhamento_Geral(G):
2      repita
3          Redução_Flor(G,M,P)
4          M ← MΔE(P)
5      até P = ∅
6      retorna M
7  Procedimento Redução_Flor(G,M,P):
8      Aumentante(G,M,P)
9      se P contém uma flor F então
10         G' ← G/F
11         M ← M \ E(F)
12         Redução_Flor(G',M',P')
13         P ← Associar_Caminhos(G',M',P', M, G)
14 Procedimento Aumentante(G,M,P):
15     D_f ← Construir_D_f(M)
16     se ∃ caminho P_f em D_f então
17         P ← Obter_Passeio(P_f, D_f, G, M)
```

Em síntese, o Algoritmo 4 funciona da seguinte forma. Até que não exista nenhum caminho aumentante, é realizada a redução na flor identificada no grafo G e é obtido um caminho aumentante P. Após o emparelhamento é aumentado através da diferença simétrica entre M e P.

O procedimento `Redução_Flor` busca um caminho aumentante P no grafo G e, caso este caminho contenha uma flor F, recursivamente busca um caminho aumentante em um novo grafo G' sem a flor. Após, associa o caminho aumentante anterior P ao caminho aumentante obtido P'. Para isto, utiliza-se do o procedimento `Aumentante(G, M, P)`, que verifica se G contém passeio M-aumentante P e constrói um grafo direcionado auxiliar D_f onde, para cada vértice $v \in V(G)$ e $w \in \{x \mid (w,x) \in M, x \neq v\}$ temos que $(v,x) \in E(D_f)$. Se D_f possuir um caminho de v_0 a v_s, tal que, v_0,v_t sejam vértices expostos e $(v_s,v_t) \in E(G)$, o passeio aumentante P será obtido através do procedimento `Obter_Passeio` da seguinte maneira. Escolha vértices $v_1,v_3,\ldots,v_{t-2} \in V$, tal que $(v_0,v_1),(v_2,v_3),\ldots,(v_{t-3},v_{t-2)} \in E(G) - M$, e $(v_1,v_2),(v_3,v_4),\ldots,(v_{t-2},v_{t-1}) \in M$ e em seguida construa o passeio $P = v_0,v_1,\ldots,v_{t-1},v_t$. Após, o procedimento `Redução_Flor` utiliza a função `Associar_Caminhos`, que obtém o caminho P', uma associação entre o caminho M-aumentante do grafo G e o caminho M'-aumentante do grafo G', onde $G' = G \setminus F$ e $M' = M \setminus E(F)$, para a flor F. Para isto, seja b a base de F. Se $b \in P$ faça $P = P'$, caso contrário, seja $u \mid (b,u) \in E(G) \setminus M'$ e $v \mid (b,v) \in F, v \neq u$. Obtenha P' expandindo o vértice b em P, pelo caminho em F M-alternante, de comprimento par entre b e v.

1.2.3 Emparelhamento de custo máximo em grafos ponderados quaisquer

Algoritmos de tempo polinomial para obtenção de emparelhamentos de custo máximo em grafos quaisquer são conhecidos, como o algoritmo de Edmonds, com complexidade de tempo $\Theta(n^3)$, mas estes têm se mostrado de tempo inadequado na prática em problemas de larga escala [3]. Na perspectiva prática, o algoritmo de Edmonds é muito lento [4] e, na perspectiva teórica, sua complexidade de implementação é considerável [3].

Um exemplo das dificuldades mencionadas é dado em [15] onde o autor, após não ter encontrado implementações do algoritmo para emparelhamentos em grafos ponderados quaisquer, decide implementar e, subsequentemente admite ter subestimado o problema. Sua implementação do algoritmo de Edmonds é descrita em [16], a qual utilizaremos para comparar com outros algoritmos no Capítulo 3. Uma implementação em `C++` do algoritmo de Edmonds modificado, de uso não-comercial, chamada de Blossom V pode ser encontrada em [17].

O algoritmo de Edmonds ponderado é de ainda maior dificuldade. O algoritmo de Edmonds ponderado é essencialmente uma adaptação do Algoritmo 4 para o caso de emparelhamento ponderado [11] e também utiliza conceitualmente programação linear. Inicialmente, note que podemos assumir que toda aresta do grafo possui custo não-negativo, visto que arestas de custo negativo não ocorrem em emparelhamentos máximos (afinal bastaria remover uma aresta do emparelhamento para obter outro de custo maior). Pode-se também considerar que o grafo sempre tem quantidade par de vértices, por ser possível adicionar um vértice isolado sem alterar o emparelhamento máximo. Outra premissa é considerar o grafo sempre completo, adicionando uma aresta com peso zero entre cada par de vértices não-adjacentes. A ideia principal do algoritmo é encolher certos tipos de ciclos de cardinalidade ímpar, chamados de flores, conforme o Algoritmo 4, aplicando-se o Lema 5. Sua ideia é buscar um caminho aumentante até que este não exista, sendo portanto o emparelhamento maximal conforme o Teorema 4. Gabow et al., em 2018,

demonstra que, utilizando-se o algoritmo *NCA* (do inglês *Nearest-Common-Ancestor*) dinâmico na estrutura de dados utilizada na busca do algoritmo de Edmonds ponderado, este pode ser executado com complexidade de tempo de $\Theta(nm + n^2 \log n)$ [18] [19]. Contudo, muitos problemas práticos requerem grafos densos (com $m = \Theta(n^2)$), fazendo com que o tempo de execução do algoritmo de Edmonds ponderado seja elevado [20].

1.3 Problema da Determinação de Caminhos Mínimos

O Problema dos Caminhos Mínimos é o de determinar o caminho mínimo entre os vértices desejados [2]. Sua aplicação no CPP dá-se quando o grafo não é euleriano. Nesse caso, reduz-se o problema àquele de encontrar um emparelhamento perfeito de custo mínimo entre cada par de vértices de um grafo auxiliar, criado a partir da determinação dos caminhos mínimos do grafo original.

Para resolver o problema dos caminhos mínimos, considere o Algoritmo 5 de Floyd-Warshall que produz uma matriz com a distância mínima entre cada par de vértices do grafo G dado de entrada. No algoritmo, considere que $d(i,j)$ é a entrada da i-ésima linha e j-ésima coluna da matriz resultante.

A ideia do algoritmo de Floyd-Warshall é para cada $k = 1, 2, \ldots, n$, determinar os caminhos mínimos entre cada par de vértices $i,j \in V(G)$ considerando caminhos no grafo entre i e j cujos vértices intermediários estejam no conjunto $\{v_1, \ldots, v_k\}$.

Algoritmo 5: Algoritmo de Floyd-Warshall.

Complexidade de Tempo: $\Theta(n^3)$.

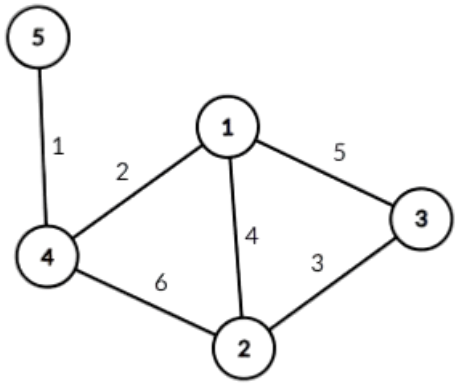
Entrada: Um grafo G ponderado, onde $c(i,j)$ é o custo da aresta (i,j)

Saída: matriz $d(1, \ldots, |V(G)|, 1, \ldots, |V(G)|)$, tal que $d(i,j)$ representa a distância entre os vértices v_i, v_j

```
1  Função Floyd_Warshall(G):
2      para i ← 1,...,n faça
3          se (i,j) ∈ E(G) então
4              d(i,j) ← c(i,j)
5          senão
6              d(i,j) ← ∞
7      para k ← 1,...,n faça
8          para i ← 1,...,n faça
9              para j ← 1,...,n faça
10                 d(i,j) ← min(d(i,j),d(i,k) + d(k,j))
11     retorna d
```

Utilizaremos o grafo da Figura 9 como exemplo do algoritmo de Floyd-Warshall.

Figura 9 – Grafo utilizado no exemplo do algoritmo de Floyd-Warshall.



A matriz de custos d é inicializada conforme a Tabela 1.

Tabela 1 – Matriz de custos do grafo utilizado como exemplo no algoritmo de Floyd-Warshall.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 4 & 5 & 2 & \infty \\ 4 & 0 & 3 & 6 & \infty \\ 5 & 3 & 0 & \infty & \infty \\ 2 & 6 & \infty & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \end{array}$$

Após a atribuição inicial de infinito na diagonal principal, na iteração onde $k = 1$, teremos uma única alteração:

$$d(3,4) = d(4,3) = \min(d(3,4), d(3,1) + d(1,4)) = 7$$

Obteremos então a matriz de custos representada na Tabela 2.

Tabela 2 – Matriz de custos do grafo utilizado como exemplo no algoritmo de Floyd-Warshall – $k = 1$.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 4 & 5 & 2 & \infty \\ 4 & \infty & 3 & 6 & \infty \\ 5 & 3 & \infty & 7 & \infty \\ 2 & 6 & 7 & \infty & 1 \\ \infty & \infty & \infty & 1 & \infty \end{bmatrix} \end{array}$$

Nas iterações onde $k = 2$ e $k = 3$ não ocorrem alterações. Para $k = 4$, teremos as seguintes alterações:

$$c(1,5) = c(5,1) = \min(c(1,5), c(1,4) + c(4,5)) = 3$$
$$c(2,5) = c(5,2) = \min(c(2,5), c(2,4) + c(4,5)) = 7$$
$$c(3,5) = c(5,3) = \min(c(3,5), c(3,4) + c(4,5)) = 8$$

Após essas iterações, obteremos a matriz de custos representada na Tabela 3.

Tabela 3 – Matriz de custos do grafo utilizado como exemplo no algoritmo de Floyd-Warshall – $k = 4$.

	1	2	3	4	5
1	∞	4	5	2	3
2	4	∞	3	6	7
3	5	3	∞	7	8
4	2	6	7	∞	1
5	3	7	8	1	∞

Para $k = 5$ não ocorrem alterações, portanto temos que a matriz de custos mínimos representada na Tabela 3 é a matriz desejada.

2 PROBLEMA DO CARTEIRO CHINÊS E ASPECTOS ALGORÍTMICOS

Neste capítulo, abordaremos os principais trabalhos sobre algoritmos não exatos para o CPP, problema do emparelhamento ponderado de custo máximo e mínimo.

Na Seção 2.1, abordaremos o Problema do Carteiro Chinês e suas variantes. Na Seção 2.2, são descritos os principais algoritmos não exatos para emparelhamentos de custo máximo ou mínimo.

2.1 Problema do Carteiro Chinês (CPP - Chinese Postman Problem)

Considere o cenário onde um carteiro em sua rota diária deve passar por cada rua de uma vila e então retornar ao ponto de origem. Qual rota deverá ser empregada de forma a minimizar a distância percorrida? Esse problema, proposto pela primeira vez pelo matemático chinês Meigu Guan (1962), é o *Problema do Carteiro Chinês* [11] ou *CPP* (do inglês *Chinese Postman Problem*).

O CPP pode ser modelado por um grafo onde o conjunto de vértices representa os pontos de encontro entre as diferentes ruas e as arestas as ruas propriamente ditas. O custo de uma aresta presume-se que seja um número não negativo e pode representar, por exemplo, distâncias físicas ou outras métricas de custo (como tempo de travessia, considerando o trânsito, ou a qualidade do pavimento, considerando evitar custos de manutenção do veículo), que dependem da regra do negócio a ser modelada. Um *circuito do carteiro* é um circuito que contém cada aresta pelo menos uma vez. Dado um grafo conexo, o CPP consiste em identificar um circuito do carteiro de custo mínimo [21].

Observe que, por se tratar de um grafo não-orientado, não existem arcos e portanto o circuito do carteiro é composto apenas por arestas. Apesar de ser permitido, busca-se evitar repetições de arestas de forma a minimizar o custo.

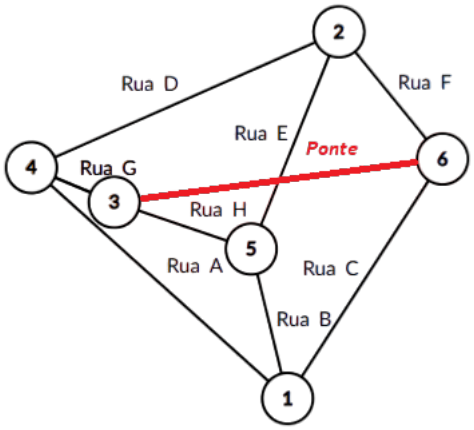
O CPP pode ser resolvido em tempo eficiente, como veremos adiante. Mas ele pode aparecer em outras formas que serão estudadas mais a frente. Nas versões do problema com grafos misto, ele se torna NP-Difícil [22], e portanto não se conhecem soluções exatas em tempo polinomial para estes casos.

O CPP é definido formalmente a seguir.

Problema:	Carteiro Chinês Ponderado - CPP
Entrada:	Um grafo conexo G c: $V(G) \times V(G) \to \mathbb{R}^+$
Questão:	Determinar um circuito do carteiro P tal que $c(P)$ seja mínimo.

A Figura 10 mostra a representação em grafo de um setor urbano onde deseja-se encontrar o circuito do carteiro e, tendo em vista que trata-se de um grafo conexo, o CPP tem solução. Note que as vias se cruzam apenas nos vértices e que um grafo utilizado para representar um setor urbano não é necessariamente planar, devido à existência de pontes e túneis, por exemplo.

Figura 10 – Representação de um setor urbano em um grafo.



Note que, caso exista, um circuito euleriano sempre será uma solução ótima do CPP. Lembre que um grafo conexo é euleriano se e somente se não possui vértices de grau ímpar [8].

Agora suponha que o grafo G conexo, representando a entrada para o CPP, não é euleriano. Considere o circuito do carteiro ótimo em G e substitua cada aresta por uma quantidade de arestas paralelas equivalente ao número de vezes que o percurso passa por esta aresta, cada uma com o mesmo custo da aresta sendo substituída. O resultado será um grafo G' que é euleriano e contém um circuito euleriano, que é o circuito ótimo do carteiro do grafo G original.

Note que o circuito do carteiro ótimo não contém a mesma aresta mais do que duas vezes, caso contrário duas arestas dessas poderiam ser removidas do percurso e um novo circuito do carteiro, de custo menor, poderia ser obtido. Sendo assim, quando o grafo é conexo e não euleriano, o CPP equivale ao problema de minimizar o custo das arestas a serem adicionadas de forma que o grafo resultante seja euleriano e encontrar um respectivo ciclo euleriano [9]. Portanto, o problema se reduz à obtenção de um emparelhamento perfeito entre os vértices de grau ímpar do grafo de custo mínimo.

Para solucionar o CPP, dado um grafo de entrada G, tomamos T como o conjunto de todos os vértices de grau ímpar em G. Note que $|T|$ é par, conforme o Teorema 3. A seguir resolvemos o problema do caminho mínimo para todos os vértices em G. Então, definimos o grafo completo K com conjunto de vértices T cujo peso de cada aresta $(u,v) \in E(K)$ corresponde à distância computada de u a v em G. Finalmente, soluciona-se um problema de emparelhamento perfeito de custo mínimo em K [11]. A computação desse emparelhamento perfeito de custo mínimo é reduzida à computação de um emparelhamento de custo máximo em um grafo G' obtido a partir de G, a ser chamado de *conjugado* a G, definido pelo Lema 6.

Lema 6. *Seja G um grafo completo ponderado, com número par de vértices e função de custo $c\colon E(G) \to \mathbb{R}^+$, e G' o grafo ponderado conjugado de G, com número par de vértices e função de custo $c'\colon E(G) \to \mathbb{R}^+$, tal que $c'(e) = y - c(e)$, onde define-se $y = \max\{c(e) : e \in E(G)\} + 1$. Um emparelhamento máximo de G' é um emparelhamento perfeito mínimo de G.*

Demonstração. Seja M um emparelhamento máximo em G'. Suponhamos, por absurdo, que exista um emparelhamento perfeito M_2 em G com $c(M_2) < c(M)$.

Teríamos então:

$$c(M_2) = \sum_{e \in M_2} c(e)$$
$$= \sum_{e \in M_2} (y - c'(e))$$
$$= ny/2 - \sum_{e \in M_2} c'(e)$$

e, de forma similar, temos que

$$c(M) = \sum_{e \in M} c(e)$$
$$= \sum_{e \in M} (y - c'(e))$$
$$= ny/2 - \sum_{e \in M} c'(e)$$

Como $c(M_2) < c'(M)$, temos que

$$ny/2 - \sum_{e \in M_2} c'(e) < ny/2 - \sum_{e \in M} c'(e)$$

o que implica que

$$c'(M_2) > c'(M)$$

Então, o emparelhamento M não seria máximo em G', um absurdo. Logo M é um emparelhamento perfeito de peso mínimo em G. □

Note que é possível generalizar o Lema 6 para que o resultado valha para qualquer grafo G, pois a operação de adicionar um vértice isolado e adicionar arestas de custo zero não alteram o emparelhamento máximo de um grafo.

Outra demonstração do Lema 6 utilizando programação linear é encontrada em [17].

Em resumo, no caso do CPP onde o grafo de entrada G não é euleriano, a solução é dada da seguinte forma: construa um grafo K com os vértices de grau ímpar em G e, para cada $v_i, v_j \in V(K)$ com $v_i \neq v_j$, adicione uma aresta (v_i, v_j) a K com custo igual a distância mínima entre ambos os vértices em G. Encontre um emparelhamento perfeito de custo mínimo de K. Para tal, encontre um emparelhamento máximo no grafo K' conjugado de K. Duplique em G as arestas correspondentes ao emparelhamento máximo de K', obtendo o grafo G', um grafo euleriano. Note que a adição destas arestas é a de menor custo possível. Agora basta determinar o ciclo euleriano de G'.

Para determinar os caminhos mínimos pode-se aplicar o Algoritmo 5 de Floyd-Warshall. Já para encontrar o emparelhamento máximo em qualquer tipo de grafo conexo, pode-se utilizar o Algoritmo de Edmonds para grafos ponderados.

Devido à dificuldade de implementar essa solução exata em tempo polinomial, tendo em vista a complexidade de implementação do algoritmo de Edmonds, temos que normalmente soluções não exatas são utilizadas para identificar o emparelhamento máximo. Tais soluções serão estudadas na Seção 2.2. Antes, porém, abriremos um parênteses para descrever as variações importantes que não serão tratadas neste trabalho. Estas variantes serão apresentadas na próxima subseção.

2.1.1 Principais variações do CPP

O CPP ocorre na prática em diversas variantes. Nessa seção, trataremos de descrever as principais.

2.1.1.1 Problema do Carteiro Chinês Orientado (OCPP)

Considere o seguinte cenário onde um órgão público deseja vistoriar a sinalização de diversas vias rodoviárias em um perímetro urbano. Para isto, é necessário percorrer cada via em seus sentidos de fluxo rodoviário retornando ao final à sede ou filial do órgão. Se a via possuir mais de um sentido neste problema a distância a ser percorrida em ambos os sentidos será igual.

O problema ilustrado pelo exemplo anterior é o *Problema do Carteiro Chinês Orientado (OCPP)* (do inglês, Oriented Chinese Postman Problem). Definimos um *circuito do carteiro orientado* como um circuito que contém cada arco pelo menos uma vez. Dado um grafo orientado G com custos associados aos arcos de G, onde, se há arcos $(u,v), (v,u) \in A(G)$, então estes arcos possuem o mesmo custo, o OCPP consiste em identificar um circuito do carteiro orientado de custo mínimo. Como o nome sugere, trata-se de uma variação do CPP para grafos orientados.

Problema:	Problema do Carteiro Chinês Orientado (OCPP)
Entrada:	Um grafo orientado G, função de custo $c : A(G) \to \mathbb{R}^+$
Questão:	Determinar um passeio fechado P contendo todos os arcos de G tal que $c(P)$ seja mínimo.

Caso o passeio fechado obtido contenha cada arco apenas uma vez, então este será um circuito euleriano e portanto é solução ótima. A condição para a existência de um circuito euleriano num grafo orientado é que, além de ser fortemente conexo, o grafo deve ser *simétrico*, ou seja, o grau de entrada de todo vértice deve ser igual ao seu grau de saída.

O OCPP pode ser resolvido com complexidade de tempo $\Theta(n^3)$ [23].

2.1.1.2 Problema do Carteiro Chinês Misto (MCPP)

Considere o cenário onde um navio precisa entregar mercadorias em diversos portos, partindo de uma central de distribuição e retornando a esta ao término do trabalho. Existem percursos onde há travessias entre dois portos com apenas um sentido de navegação por causa da correnteza. Os demais percursos possíveis são aqueles onde a correnteza é fraca e é possível navegar em ambos os sentidos sem grandes esforços. Se uma travessia possuir mais de um sentido neste problema a distância a ser percorrida em ambos os sentidos será igual. Deseja-se minimizar a distância percorrida em um circuito que solucione o problema.

O problema ilustrado pelo exemplo anterior é o *Problema do Carteiro Chinês Misto - MCPP* (do inglês, Mixed Chinese Postman Problem). Definimos um *circuito do carteiro misto* como um circuito que contém cada ligação pelo menos uma vez. Dado um grafo misto G com custos associados às ligações de G, onde, se há arcos $(u,v), (v,u) \in A(G)$, então estes arcos possuem o mesmo custo, o MCPP consiste em identificar um circuito do carteiro misto de custo mínimo.

Problema:	Problema do Carteiro Chinês Misto (MCPP)
Entrada:	Um grafo misto G, função de custo $c : L(G) \rightarrow \mathbb{R}^+$
Questão:	Determinar um circuito do carteiro misto P tal que $c(P)$ seja mínimo.

Como o nome sugere, trata-se de uma variação do CPP para grafos mistos. Caso o passeio fechado obtido contenha cada ligação apenas uma vez, então este será um circuito euleriano e portanto a solução ótima.

A condição para que um grafo misto tenha um circuito euleriano é que este seja par e balanceado. Um *grafo par* é aquele no qual o número total de arcos e arestas incidentes a cada um de seus vértices é par. Por outro lado, um *grafo balanceado* é aquele no qual, para todo vértice, a diferença entre o número de arcos que saem deste vértice e o número de arcos incidentes a este mesmo vértice deve ser menor ou igual ao número de arestas que ligam esse vértice a um outro vértice qualquer.

Quando o grafo não é balanceado ou não é par, o MCPP é um problema NP-Difícil e normalmente soluções não exatas são utilizadas [24]. Note que o MCPP é um problema mais geral tanto em relação ao OCPP quanto ao CPP.

2.1.1.3 Problema do Carteiro Chinês com Vento (WCPP)

Considere o cenário onde, conforme o CPP, um carteiro precisa cobrir todo um setor urbano de modo a entregar as correspondências, partindo de uma central de distribuição e retornando a esta ao término do trabalho. Contudo, existem ladeiras no percurso e estas possuem custos diferentes no aclive e declive. O problema descrito é um exemplo de aplicação do *Problema do Carteiro Chinês com Vento (WCPP)* (do inglês, Windy Chinese Postman Problem). As ladeiras mencionadas são representadas por arestas e seus custos são possivelmente diferentes entre dois vértices quaisquer de acordo com o sentido percorrido [10] [25]. Deseja-se minimizar o custo do percurso percorrido pelo carteiro, identificando um passeio fechado com custo mínimo.

O WCPP consiste em encontrar um passeio fechado de custo mínimo passando por cada aresta de G pelo menos uma vez e é sabido que este problema é NP-Difícil [10] [25] e normalmente soluções não exatas são utilizadas.

Note que o WCPP não pode ser modelado por duas arestas entre dois pares de vértices ao invés de uma aresta com custos possivelmente diferentes, afinal o WCPP demanda que cada aresta seja percorrida ao menos uma vez, e não é necessário que uma aresta seja percorrida em ambos os sentidos. Nas formulações das variantes do CPP, a existência de uma aresta ou arco obriga a passagem por ela. Assim, ao usar duas arestas com custos diferentes, estaríamos obrigando o percurso a passar por ambas, o que não é o desejado nessa variante.

Problema:	Problema do Carteiro Chinês com Vento (WCPP)
Entrada:	Um grafo G, função de custo $c : \{(u,v),(v,u) \mid (u,v) \in E(G)\} \rightarrow \mathbb{R}^+$
Questão:	Determinar um passeio fechado P tal que $c(P)$ seja mínimo.

Como o nome sugere, trata-se de uma variação do CPP com pesos possivelmente diferentes em diferentes direções entre um par de vértices qualquer. Contudo, devido às suas características, dentre os problemas de carteiro chinês este é o que mais se aproxima da realidade e possui diversas aplicações práticas. Note que caso o passeio fechado obtido

contenha cada ligação apenas uma vez, então este será um circuito euleriano e portanto a solução ótima.

O WCPP claramente é uma generalização do CPP.

2.1.1.4 Problema do Carteiro Rural (RPP)

Considere o seguinte cenário onde um órgão público deseja entrevistar todos os moradores em um perímetro urbano. Para isso, os órgão designa entrevistadores para as diversas vias. Contudo, esses entrevistadores não conseguem alcançar todas as vias necessárias por questão de problema no acesso, seja este físico, como a queda de uma ponte, ou um impedimento de segurança. O órgão então precisa entrevistar moradores de diversas vias que não puderam ser entrevistados nas tentativas anteriores. Para isto, é necessário percorrer cada via que ainda não foi acessada, retornando ao final à sede ou filial do órgão.

O problema ilustrado pelo exemplo anterior é o *Problema do Carteiro Rural (RPP)* (do inglês, Rural Postman Problem), que consiste em identificar um passeio fechado com a quantidade mínima de arestas e contendo todas as arestas solicitadas na entrada do problema.

Denominamos o conjunto de arestas E' como o conjunto de arestas requeridas pelo PCR e que necessariamente devem fazer parte do passeio fechado resultante. Arestas que não pertençam a E' são permitidas porém não são desejáveis, uma vez que elevam a quantidade de arestas da solução. Analogamente, arestas repetidas são permitidas porém não são desejáveis.

O RPP em sua versão com grafos ponderados é um problema mais geral que o CPP, se reduzindo a esse no caso em que $E' = E(G)$.

Problema:	Problema do Carteiro Rural (RPP)
Entrada:	Um grafo G, $E' \subseteq E(G)$.
Questão:	Determinar um passeio fechado P tal que $E' \subseteq P$ e o número de arestas em P seja mínimo.

Suas outras principais versões são conhecidas como: orientada (*Problema do Carteiro Rural Orientado (ORPP)*) e mista (*Problema do Carteiro Rural Misto (MRPP)*). Dentre estes, o MRPP é o mais generalizado e o que mais possui aplicações práticas.

O *problema de roteamento de arcos* consiste em determinar um percurso de custo mínimo entre alguns arcos ou arestas de um grafo, sujeito a possíveis restrições [10]. Embora na literatura haja um número grande de restrições práticas de interesse, diversas delas, em suas versões mais básicas, se reduzem ao RPP.

O RPP é NP-Difícil em todas as versões citadas (RPP, ORPP e MRPP) [10].

2.2 Algoritmos não exatos para emparelhamentos ponderados máximos

A dificuldade de implementação do algoritmo de Edmonds acaba sendo um fator crítico na aplicação do mesmo [26]. Para este trabalho foram procuradas implementações do algoritmo de Edmonds para grafos ponderados, e apenas duas foram encontradas disponibilizadas publicamente, sendo uma delas comercial. Ainda assim, quando implementado, o algoritmo não atende casos reais, representados por grafos maiores, devido à complexidade de tempo do algoritmo. Sendo assim, neste capítulo revisaremos os algoritmos não exatos mais relevantes já desenvolvidos para solucionar emparelhamentos ponderados máximos e mínimos, principalmente para grafos completos.

Para facilitar a comparação dos algoritmos que abordaremos a seguir, utilizaremos o grafo completo K_8 com custos dados na matriz de adjacências da Tabela 4 como exemplo comum a todos os algoritmos. O conjunto de vértices é $\{1,2,\ldots,8\}$. Os custos são inteiros positivos aleatórios e o emparelhamento máximo deste grafo é dado por $M_{\text{máx}} = \{(1,7),(2,8),(3,6),(4,5)\}$, com custo 316.

Tabela 4 – Matriz de adjacências do grafo K_8 ponderado.

	1	2	3	4	5	6	7	8
1	0	28	83	80	32	2	58	59
2	28	0	31	2	16	83	73	99
3	83	31	0	62	80	70	10	7
4	80	2	62	0	89	21	52	84
5	32	16	80	89	0	29	30	79
6	2	83	70	21	29	0	21	35
7	58	73	10	52	30	21	0	51
8	59	99	7	84	79	35	51	0

A medida da qualidade de um algoritmo aproximativo que solucione um emparelhamento ponderado é dada pelo seu *fator de aproximação*. Um algoritmo possui um fator de aproximação f se, para qualquer grafo de entrada G com emparelhamento ótimo M^*, temos que o emparelhamento M retornado pelo algoritmo é tal que $c(M) \geq f \times c(M^*)$.

2.2.1 Greedy

Aplicado a qualquer tipo de grafo, temos o algoritmo *Greedy* com complexidade de tempo $\Theta(m \log n)$ [3] e com fator de aproximação $\frac{1}{2}$. Trata-se da versão mais intuitiva, um algoritmo guloso adicionando sucessivamente as arestas de maior custo ao emparelhamento sempre que possível.

Algoritmo 6: Greedy.
Complexidade de Tempo: $\Theta(m \log n)$.
Entrada: $G = (V,E)$ e a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$
Saída: Um emparelhamento M

```
1 Função Greedy(G,c):
2     E(G) ← {e_k ∈ E(G)} | c(e_k) ≥ c(e_{k−1}) ∀ k ∈ 2,...,m}
3     M ← {}
4     para (u,v) ∈ E(G) faça
5         se u ∈ V(G) e v ∈ V(G) então
6             Adicione (u,v) a M
7         G ← G \ {u, v}
8     retorna M
```

O Algoritmo 6 inicialmente ordena o conjunto das arestas pelo seu custo de forma decrescente e, para cada aresta nessa ordem, se a mesma existir no grafo, então ela é

adicionada ao emparelhamento. A seguir os vértices incidentes a aresta são removidos do grafo.

Executando o algoritmo no grafo da Tabela 4, obteremos o emparelhamento $M = \{(1,3),(2,8),(4,5),(6,7)\}$ com custo igual a 292. Comparando o resultado com o emparelhamento máximo temos

$$\frac{c(M)}{c(M_{\text{máx}})} = \frac{292}{316} = 0{,}92$$

Neste caso o algoritmo obtém um fator de aproximação maior que o mínimo garantido de 0,5.

2.2.2 Vertex Scan

Aplicados a grafos completos ou bipartidos completos $K_{n,n}$ com custos positivos, temos os algoritmos Row/Column-Scan [27] e Vertex Scan [3]. O algoritmo Row/Column-Scan se reduz a duas chamadas do algoritmo Vertex Scan [3] e, portanto, discutiremos este. Aplicado em emparelhamentos ponderados perfeitos de custo mínimo, o algoritmo é guloso e possui complexidade de tempo $\Theta(m)$ [3]. Na verdade, esse algoritmo não é aproximativo, uma vez que não garante qualquer fator de aproximação.

Algoritmo 7: Vertex Scan.

Complexidade de Tempo: $\Theta(m)$.

Entrada: $G = (V,E)$ completo ou bipartido completo e a função de custo $c : V(G) \times V(G) \to \mathbb{R}^+$

Saída: Um emparelhamento M

```
1 Função Vertex_Scan(G,c):
2     M ← ∅
3     enquanto ∃ i ∈ V(G) | d(i) > 1 faça
4         Escolha um vértice i aleatoriamente
5         (i,j) ← aresta de menor custo incidente a i
6         Adicione (i,j) a M
7         G ← G \ {i, j}
8     retorna M
```

O algoritmo originalmente requer que a entrada consista de grafos completos ou bipartidos completos para os quais um emparelhamento perfeito é garantido existir. Para este trabalho foi modificada esta condição e o algoritmo agora será processado enquanto existirem arestas no grafo, selecionando um vértice qualquer incidente a uma destas arestas, e, desta forma, poderá ser aplicado a qualquer grafo. Em síntese, enquanto existir alguma aresta, o algoritmo escolhe um vértice qualquer i e adiciona a aresta de menor custo (i,j) incidente a i ao emparelhamento, removendo todas as arestas vizinhas a i e j bem como os próprios vértices. Note que, se o número de vértices do grafo de entrada for par, o emparelhamento resultante será perfeito.

Utilizando o grafo cuja matriz de custos é dada na Tabela 4 e supondo que nos passos do algoritmo sejam escolhidos, sucessiva e aleatoriamente, os vértices 5,7,4 e 1, teremos como resultado o emparelhamento $M = \{(5,2),(7,3),(4,6),(1,8)\}$ de custo igual a 106. O emparelhamento mínimo é dado por $M_{\text{min}} = \{(1,6),(2,4),(3,8),(5,7)\}$ e possui custo 41. Temos então que

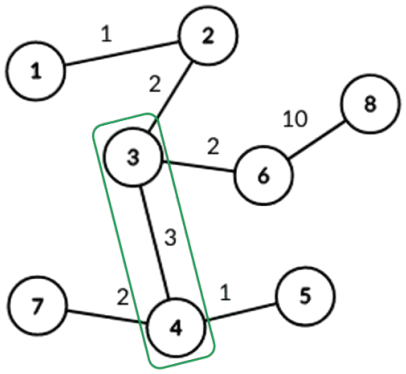
$$\frac{c(M)}{c(M_{min})} = \frac{106}{41} = 2{,}59$$

2.2.3 Linear Approximation Matching - LAM

Robert Preis em [28] descreve o algoritmo LAM (Linear Approximation Matching) que determina um emparelhamento ponderado máximo com fator de aproximação $\frac{1}{2}$, com complexidade linear de tempo [29] em um grafo conexo qualquer.

Dado um grafo G, uma *aresta de custo máximo local* é uma aresta $e \in E(G)$ em que todas as arestas adjacentes a e possuem custo igual ou inferior a $c(e)$.

Figura 11 – Aresta de custo máximo local a partir de (1,2).



Na Figura 11, partindo da aresta (1,2) obtemos (2,3) como candidata, visto que seu custo é maior. Em seguida obtemos (3,4) como a aresta de máximo local, uma vez que não há aresta vizinha com custo maior. Observe que a aresta (6,8), apesar de ter maior custo, não é atingida a partir de (1,2).

Em síntese, dado um grafo G, o Algoritmo 8 sorteia uma aresta qualquer $e \in E(G)$, identifica uma aresta de custo máximo local cujos os vértices não são saturados e, em seguida, move esta aresta de $E(G)$ para o emparelhamento. O algoritmo repete este processo enquanto existirem arestas em G.

Para identificar uma aresta de custo máximo local a partir de uma aresta qualquer $e = (a,b) \in E(G)$, o algoritmo procede conforme a seguir. Caso a e b sejam saturados ou não existam arestas vizinhas aos vértices a e b com custo superior a $c(e)$, move e de $E(G)$ para o emparelhamento. Caso contrário, verifica se existe alguma aresta vizinha ao vértice a, exceto e, com custo superior a $c(e)$ e, se existir, trata esta aresta como nova candidata ao emparelhamento, repetindo este processo até identificar a aresta de custo máximo local, movendo em seguida esta aresta de $E(G)$ para o emparelhamento.

Similarmente, o algoritmo verifica se alguma uma aresta vizinha ao vértice b, exceto e, possui custo superior e, se existir, trata esta aresta como nova candidata ao emparelhamento, repetindo este processo até identificar a aresta de custo máximo local e, em seguida, move esta aresta de $E(G)$ para o emparelhamento. Para tal, o algoritmo utiliza-se de vetores U, R, C_a e C_b onde U contém as arestas que não foram verificadas; R contém arestas definitivamente removidas (observe que a existência de R não altera efetivamente a execução do algoritmo); C_a e C_b contém, para os vértices a e b respectivamente, as arestas verificadas a cada recursão do algoritmo (note que são variáveis locais). Dentre esses, o vetor R é o único que apenas recebe arestas, sendo que os demais vetores podem receber e ceder arestas para os outros.

Observe que, no procedimento `TentarEmparelhamento`, no primeiro laço `enquanto`, a condição ($\exists(a,c) \in U$ ou $\exists(b,d) \in U$) que filtra arestas vértices adjacentes a a ou b e permite que $c = b$ e $d = a$ para contemplar a situação de existir apenas uma aresta $(a,b) \in U$, com a e b não-saturados. Caso não fosse assim, a aresta (a,b) no exemplo anterior não seria movida de U, fazendo com que o algoritmo nunca terminasse.

Algoritmo 8: Linear Approximation Matching - LAM.

Complexidade de Tempo: $\Theta(m)$.

Entrada: $G = (V,E)$ e a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$

Saída: Um emparelhamento M

```
1  Função LAM(G,c):
2      enquanto houver pelo menos uma aresta não verificada em E(G) faça
3          Escolha uma aresta e qualquer
4          TentarEmparelhamento(e)
5      retorna M_Lam
6  Procedimento TentarEmparelhamento(e):
7      enquanto ambos os vértices de e forem expostos e houver alguma aresta adjacente a
        e não verificada faça
8          e_a ← aresta qualquer adjacente a e não verificada
9          se e_a ≠ ∅ então
10             Marque e_a como verificada
11             se c(e_a) > c(e) então
12                 TentarEmparelhamento(e_a)
13         e_b ← aresta adjacente a e não verificada com e_a ≠ e_b
14         se e_b ≠ ∅ então
15             Marque e_b como verificada
16             se c(e_b) > c(e) então
17                 TentarEmparelhamento(e_b)
18     se apenas um vértice de e é exposto então
19         E' ← arestas adjacentes ao vértice exposto
20         para e' ∈ E' faça
21             se ambos os vértices de e' são expostos então
22                 Marque e' como não verificada
23     senão se ambos os vértices de e são expostos então
24         Adicione e a M_Lam
```

Executando o Algoritmo 8 no grafo completo ponderado da Tabela 4, caso as arestas sejam selecionadas na ordem (2,7),(1,3),(4,5),(6,7) obteremos o emparelhamento $M_{\text{LAM}} = \{(2,8),(1,3),(4,5),(6,7)\}$ com custo igual a 292. Comparando o resultado com o emparelhamento máximo temos

$$\frac{c(M_{\text{LAM}})}{c(M_{\text{máx}})} = \frac{292}{316} = 0{,}92$$

Neste caso o algoritmo obtém um fator de aproximação maior que o mínimo garantido de 0,5.

2.2.4 Path Growing Algorithm - PGA

Contendo uma ideia mais simples que a do algoritmo [8], Drake e Hougardy [30] elaboraram o PGA (Path Growing Algorithm). O PGA é aplicado para obter emparelhamentos ponderados máximos em um grafo. Trata-se de um algoritmo guloso aproximativo com fator de aproximação $\frac{1}{2}$ e complexidade de tempo linear.

A ideia principal do algoritmo é, a partir de um vértice arbitrário do grafo, seguir um caminho de modo que a próxima aresta do caminho tenha o maior custo entre as possíveis para aumentá-lo. Cada aresta desse caminho é adicionada alternadamente a M_1 e M_2, dois emparelhamentos inicialmente vazios. Quando não for possível aumentar o caminho, o processo se reinicia, até que não seja mais possível fazer progresso. O valor retornado pelo algoritmo é o maior custo dentre aqueles de M_1 e M_2.

Algoritmo 9: Path Growing Algorithm - PGA.

Complexidade de Tempo: $\Theta(m)$.

Entrada: $G = (V,E)$ e a função de custo c: $V(G) \times V(G) \rightarrow \mathbb{R}^+$

```
1  Função PGA(G,c):
2      M1 ← ∅
3      M2 ← ∅
4      i ← 1
5      enquanto E ≠ ∅ faça
6          Escolha um vértice x qualquer com grau maior ou igual a 1
7          enquanto d(x) ≥ 1 faça
8              (x,y) ← aresta de maior custo incidente a x
9              Adicione (x,y) a Mi
10             i ← 3 − i /* i varia entre 1 e 2 */
11             Remova x de G
12             x ← y
13     retorna max(c(M1),c(M2))
```

Note que aplicando o Algoritmo 9 é possível chegar numa situação onde o emparelhamento retornado pelo algoritmo não é maximal. Tal condição ocorre, por exemplo, no caso de um C_4, com custos 2,1,1,1 em suas arestas (1,2),(2,3),(3,4),(4,1), respectivamente, no qual o vértice 1 é sorteado inicialmente. Naturalmente o Algoritmo 9 pode ser ampliado em uma fase pós-processamento [31]. Chamaremos tal algoritmo estendido de PGA'. Observe que apenas o emparelhamento M_2 poderá ser ampliado nesta fase pós-processamento pois sua cardinalidade sempre é menor ou igual a cardinalidade de M_1. Os autores afirmam que na prática essas alterações possuem impacto considerável, como observado em [31].

Uma das formas de ampliação é adicionar quaisquer arestas remanescentes do grafo à solução até que esta se torne um emparelhamento maximal. Este algoritmo não exato mantém o fator de aproximação $\frac{1}{2}$.

A seguir, compararemos o resultado da execução do grafo da Tabela 4, com o emparelhamento máximo.

Aplicando o algoritmo *PGA* no grafo, supondo que inicialmente tenha sido sorteado o vértice de número 8, obteremos dois emparelhamentos M_1 e M_2 com custos respectivamente de 279 e 255. O algoritmo PGA retornará o emparelhamento de maior custo dentre os calculados, ou seja, $M_1 = \{(8,2),(6,3),(1,4),(5,7)\}$. Sendo assim, temos que

$$\frac{c(M_1)}{c(M_{\text{máx}})} = \frac{279}{316} = 0{,}88$$

Observe que a cardinalidade de $M_2 = \{(2,6),(3,1),(4,5)\}$ pode ser aumentada adicionando a aresta (7,8) com custo 51 ao emparelhamento. Desta forma, o algoritmo PGA' retornará o emparelhamento M_2 com cardinalidade 4, uma vez que seu custo é superior ao de M_1. Temos então um aumento no custo do resultado.

$$\frac{c(M_2)}{c(M_{\text{máx}})} = \frac{306}{316} = 0{,}97$$

Lembre que o fator de aproximação é igual a 0,5.

2.2.5 Maximal Matching - MM

Outro algoritmo guloso é dado por Drake [31], o MM (Maximal Matching). O algoritmo é guloso, trata de emparelhamentos ponderados de custo máximo em grafos conexos gerais. Possui complexidade de tempo $\Theta(m)$ e não garante qualquer fator de aproximação.

Algoritmo 10: Maximal Matching.
Complexidade de Tempo: $\Theta(m)$.
Entrada: $G = (V,E)$ e a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$

```
1 Função Maximal_Matching(G,c):
2     enquanto E(G) ≠ ∅ faça
3         Escolha um vértice i aleatoriamente
4         (i,j) ← aresta de maior custo incidente a i
5         Adicione (i,j) a M
6         Remova i e j e todas as arestas adjacentes a ambos os vértices de G
7     retorna M
```

Em síntese, enquanto existir algum vértice exposto, o algoritmo escolhe um vértice qualquer i e adiciona a aresta de menor custo (i,j) incidente a i ao emparelhamento, removendo todas as arestas vizinhas a i e j bem como os próprios vértices.

Note que o Algoritmo 10 é semelhante ao Algoritmo 7 mencionado anteriormente. A diferença principal é que um trabalha com emparelhamentos de custo máximo em grafos quaisquer e o outro com emparelhamentos de custo mínimo em grafos completos e bipartidos completos. Além disso, a condição do laço principal do Maximal Matching é a existência de arestas no grafo de entrada enquanto a condição do laço principal do Vertex Scan é a existência de vértices não saturados no grafo de entrada.

No exemplo do grafo cujos custos estão na Tabela 4 e supondo que nos passos do algoritmo sejam escolhidos, sucessiva e aleatoriamente, os vértices 7,1,6 e 4, teremos como resultado o emparelhamento $M = \{(7,2),(1,3),(6,8),(4,5)\}$ de custo igual a 280. Temos então que

$$\frac{c(M)}{c(M_{\text{máx}})} = \frac{280}{316} = 0{,}76$$

Os autores afirmam que o algoritmo Maximal Matching funciona bem na prática e não possui garantias sobre o custo do emparelhamento retornado.

2.2.6 Randomized Matching - RAMA

Pettie introduziu o algoritmo não exato RAMA (Randomized Matching Algorithm) [32] que possui um fator de aproximação esperado de $\frac{2}{3} - \varepsilon$ para o problema de emparelhamentos ponderados máximos.

Algoritmo 11: Randomized Matching - RAMA.

Complexidade de Tempo: $\Theta(m \log \frac{1}{\varepsilon})$.

Entrada: $G = (V,E)$, a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$ e $k \in \mathbb{N}^+$

```
1 Função RAMA(G,c,k):
2     M ← ∅
3     para i ← 1 to k faça
4         Escolha um vértice x aleatoriamente
5         M ← MΔaug(x)   /* aug(x) é um 2-aumento de maior ganho envolvendo x, ou ∅ se
                            inexistente */
6     retorna M
```

Dados um grafo G, sua função de custo c e um inteiro k, o Algoritmo 11 procura realizar pequenos aumentos através de caminhos ou ciclos alternantes com até 2 novas arestas não pertencentes ao emparelhamento, os quais chama de 2-aumento. O algoritmo escolhe um vértice aleatoriamente e verifica se tal vértice está envolvido em um 2-aumento. Caso exista mais de um 2-aumento possível, o algoritmo escolhe o 2-aumento de maior *ganho*, ou seja, aquele cuja diferença simétrica com o emparelhamento corrente produzirá maior aumento no custo do emparelhamento. Note que a diferença simétrica de um emparelhamento com um ciclo ou caminho aumentante também é um emparelhamento. Se a diferença simétrica do 2-aumento com o emparelhamento corrente for maior que o custo do emparelhamento corrente então tal diferença simétrica passa a ser o novo emparelhamento corrente. Esse processo é repetido k vezes, onde k é o dado de entrada.

Utilizando o grafo da Tabela 4, com $k = 5$ e selecionando os vértices na sequência 7,1,3,5,2, obtivemos como resultado o emparelhamento $M_{Rama} = \{(2,7),(1,4),(5,8),(3,6)\}$ de custo igual a 302. Temos então que

$$\frac{c(M_{Rama})}{c(M_{\text{máx}})} = \frac{302}{316} = 0{,}96$$

Neste exemplo o algoritmo obtém um fator de aproximação maior que o mínimo esperado de $\frac{2}{3}$.

Executando o Algoritmo 11 utilizando como entrada G, cuja matriz de custos é dada na Tabela 5, e $k = 6$, caso sejam sorteados os vértices $4, 2, 4, 1, 2, 1$, nesta sequência, o fator de aproximação do emparelhamento resultante é 0,6, inferior a $\frac{2}{3} - \varepsilon$

Observe que o Algoritmo 11 possui fator de aproximação esperado de $\frac{2}{3} - \varepsilon$ e, portanto, não é garantido tal fator de aproximação, conforme o exemplo na Tabela 5. Com o aumento de k, a probabilidade de que o fator de aproximação obtido se afaste de $\frac{2}{3}$ por baixo tende a zero.

2.2.7 Improve Matching

Utilizado em grafos conexos quaisquer, o Improve Matching, busca melhorar o custo máximo de um emparelhamento ponderado maximal já existente. Tem fator de aproxi-

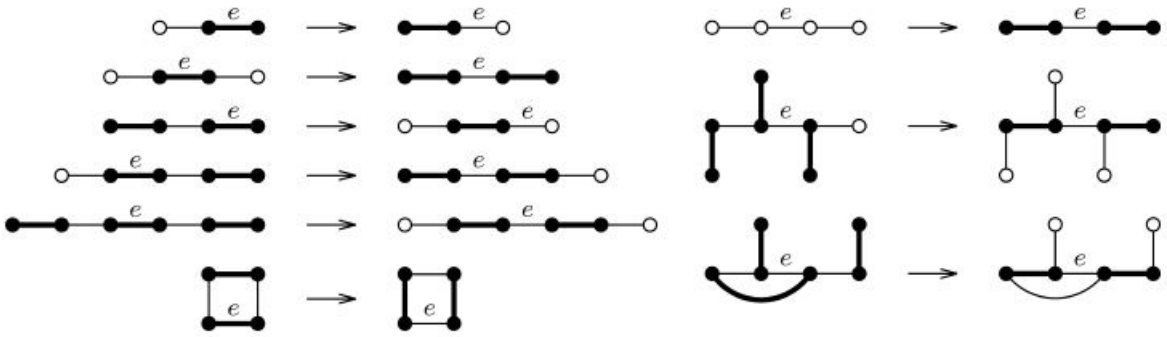
Tabela 5 – Matriz de adjacências do grafo ponderado G.

	1	2	3	4	5	6
1	0	1	2	0	0	0
2	1	0	0	3	0	0
3	2	0	0	0	4	0
4	0	3	0	0	0	5
5	0	0	4	0	0	0
6	0	0	0	5	0	0

mação $\frac{2}{3}$ conforme apresentado em [20]. Através de modificações locais, o algoritmo busca aumentar o custo do emparelhamento inicial.

Um *aumento* é o processo que remove as arestas de um conjunto S de M e adiciona as arestas de S não existentes a M, ou seja, modifica M de forma que $M \leftarrow M \Delta S$. O conjunto $S \setminus M$ é chamado de *conjunto aumentante*. Um *pequeno aumento* é um aumento no qual todas as arestas no aumento são adjacentes a uma aresta $e \in E$, chamada de *centro do pequeno aumento*. Um pequeno aumento pode ter mais de um centro e e, neste caso, qualquer uma destas arestas deverá ser escolhida como centro.

Figura 12 – Exemplos de pequenos aumentos.



A Figura 12 apresenta exemplos de pequenos aumentos. O centro de cada aumento é denotado com e. Arestas em negrito pertencem ao emparelhamento M e as demais não pertencem. Os vértices vazados representam vértices expostos. Os exemplos apresentados à esquerda contém todos os casos onde o centro e pertence ao emparelhamento e à direita encontram-se alguns exemplos onde o centro e não pertence ao emparelhamento.

Seja $\beta > 1$ uma constante. Denote por $M(S)$ as arestas de M que compartilham vértices com alguma aresta de S. Um *β-aumento* é um pequeno aumento onde o conjunto aumentante S é tal que $c(S) \geq \beta \times c(M(S))$. O ganho de um aumento S é definido como $c(S) - c(M(S))$, ou seja, o aumento no custo alcançado por S. Um β-aumento com centro e, com o maior ganho dentre todos os possíveis β-aumentos com centro e, é chamado de *melhor β-aumento com centro e*. Um *bom β-aumento com centro e* é um β-aumento com centro e cujo ganho é, no mínimo, uma fração $\frac{\beta-1}{\beta-\frac{1}{2}}$ do ganho do melhor β-aumento com centro e.

Algoritmo 12: Improve Matching.

Complexidade de Tempo: $\Theta(m)$.

Entrada: $G = (V,E)$, a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$ e um emparelhamento maximal M de G

```
1 Função Improve_Matching(G,c,M):
2     M' ← M
3     para e ∈ M faça
4         se ∃ β-aumento com centro e em M' então
5             Aumente M' por um bom β-aumento com centro e
6     retorna M'
```

A ideia do Algoritmo 12 é, a partir de um emparelhamento M maximal, aumentar o custo de M através de pequenos β-aumentos cujos centros são, respectivamente, as arestas de M. Novas arestas adicionadas ao emparelhamento através de pequenos β-aumentos não serão consideradas como aumentos. O algoritmo visita cada aresta $e \in M$ uma vez e verifica se existe algum β-aumento com centro e em M'. Neste caso o algoritmo aumenta M' por um bom β-aumento com centro e.

A *captação* de uma aresta $a \in E \setminus M$ adjacente a e é $cap_e(a) := c(a) - c(M(a) \setminus \{e\})$, onde $M(a)$ denota as arestas de M que compartilham vértice com a aresta a. Ou seja, a captação de uma aresta a é dada pelo custo de a menos o custo de todas as arestas diferentes de e em M que são adjacentes a a.

Algoritmo 13: Bom β-aumento.

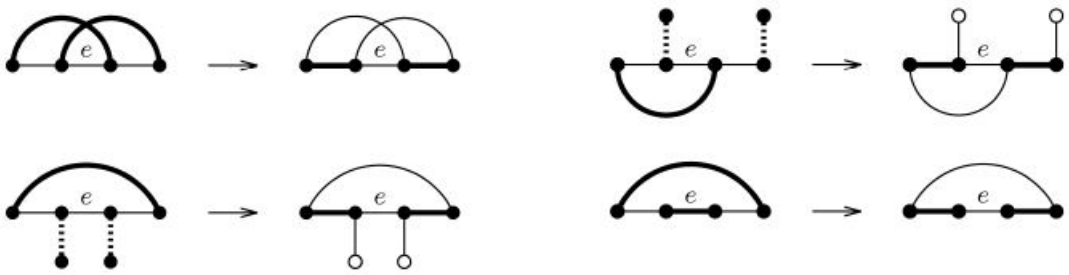
Complexidade de Tempo: $\Theta(d(u) + d(v))$, onde $e = (u,v)$.

Entrada: $G = (V,E)$, a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$, um emparelhamento M de G, $e \in E(G)$

```
1 Função Bom_β_aumento(G,c,e,M):
2     A_1 ← melhor β-aumento com centro e que contém no máximo uma aresta ∉ M
3     A_2 ← melhor β-aumento com centro e que contém um ciclo
4     F ← {f ∈ E \ M | f é adjacente a e e cap_e(f) ≥ 1/2 × c(e)}
5     A_3 ← Max_Allowable(F, e)
6     F ← {f ∈ E \ M | f é adjacente a e}
7     A_4 ← Max_Allowable(F, e)
8     retorna β-aumento de maior custo em {A_1,A_2,A_3,A_4}
```

Utilizado no Algoritmo 12 para obter um bom β-aumento, o Algoritmo 13 inicialmente busca β-aumentos com no máximo uma aresta não pertencente ao emparelhamento. Para isto basta verificar as arestas adjacentes aos vértices de e.

Figura 13 – Todos os β-aumentos com centro em e que contém um ciclo.



A Figura 13 contém todos os β-aumentos com centro em e que contém um ciclo e, para encontrar o melhor com complexidade de tempo linear, deve-se utilizar marcas nos vértices da seguinte forma. Seja $M(a)$ o conjunto de todas as arestas no emparelhamento incidentes à uma aresta qualquer $a \in E(G)$. Partindo de um vértice qualquer v_1 de e, marque todos os vértices das arestas incidentes a v_1, exceto e, diferentes de v_1. Em seguida, partindo do outro vértice v_2 incidente a e, para cada aresta a adjacente a v_2, exceto e, e para cada aresta a_M em $M(a)$, se o vértice de a_M que não incide em a estiver marcado então um ciclo de tamanho 4 foi identificado, caso contrário, se existir uma aresta adjacente a a com vértice marcado então um ciclo de tamanho 5 foi identificado.

Seja a função $excedente_e(a) := c(a) - \beta \ \times c(M(a) \setminus \{e\})$ o *excedente* de uma aresta a adjacente a e com $a \in \{E(G) \setminus M\}$.

Algoritmo 14: Max_Allowable.

Complexidade de Tempo: $\Theta(d(u) + d(v))$, onde $e = (u,v)$.

Entrada: $G = (V,E)$, a função de custo c: $V(G) \times V(G) \to \mathbb{R}^+$, um emparelhamento M de G, $e = \{x,y\} \in E(G)$, $F \subseteq E(G)$

Função `Max_Allowable(`G,c,M,e,F`):`

$F_x \leftarrow \{f \in F \mid x \in f\}$

$F_y \leftarrow \{f \in F \mid y \in f\}$

Escolha $f_1, f_2 \in F_x$ de modo que $excedente_e(f_1) \geq excedente_e(f_2) \geq excedente_e(f) \ \forall \ f \in F_x \setminus \{f_1,f_2\}$

Escolha $f_3, f_4 \in F_y$ de modo que $excedente_e(f_3) \geq excedente_e(f_4) \geq excedente_e(f) \ \forall \ f \in F_y \setminus \{f_3,f_4\}$

se $e \in M$ **então**

  $z = c(e)$

senão

  $z = 0$

$X_1 \leftarrow \{f \in F_y \mid f, f_1$ não adjacentes e $excedente_e(f) + excedente_e(f_1) \geq \beta \times z\}$

$X_2 \leftarrow \{f \in F_y \mid f, f_2$ não adjacentes e $excedente_e(f) + excedente_e(f_2) \geq \beta \times z\}$

$X_3 \leftarrow \{f \in F_y \mid f, f_3$ não adjacentes e $excedente_e(f) + excedente_e(f_3) \geq \beta \times z\}$

$X_4 \leftarrow \{f \in F_y \mid f, f_1$ não adjacentes e $excedente_e(f) + excedente_e(f_4) \geq \beta \times z\}$

retorna β-aumento de maior captação contido em $\{X_1 \cup X_2 \cup X_3 \cup X_4\}$

Utilizado no Algoritmo 13 para obter um β-aumento que contém a aresta com o maior ganho, o Algoritmo 14 verifica todas as arestas incidentes a e e identifica se existe um β-aumento com centro em e contendo esta aresta. A razão pela qual é necessário considerar as duas arestas de maior excedente em cada aresta incidente a e é que uma destas pode ser adjacente a uma outra adjacente ao lado oposto de e.

2.2.8 $\frac{3}{4}$-emparelhamento

Hanke e Hougardy [33] apresentam um algoritmo com fator de aproximação $\frac{3}{4}-\varepsilon$ para problemas de emparelhamentos perfeitos de custo máximo.

Algoritmo 15: $\frac{3}{4}$-emparelhamento.

Complexidade de Tempo: $\Theta(m \log n)$.

Entrada: $G = (V,E)$, a função de custo c: $V(G) \times V(G) \rightarrow \mathbb{R}^+$, um emparelhamento maximal M de G

```
1  Função 3_4_emparelhamento(G,c,M):
2      ALG ← ∅
3      enquanto M ≠ ∅ faça
4          para e ∈ M faça
5              para i ∈ {0,1} faça
6                  A_i ← braços com i arestas não pertencentes a M, a partir e em ambas as
                       orientações de e
7              para arm_a ∈ A_1 faça
8                  para arm_b ∈ {braços de A_1 com o vértice inicial diferente do vértice inicial
                           de arm_a} faça
9                      A_2 ← A_2 ∪ {arm_a ∪ arm_b}
10                     para ext ∈ {até duas extensões de maior rank de arm_a} faça
11                         A_3 ← A_3 ∪ {arm_a ∪ arm_b ∪ ext}
12         A ← M−aumento de ganho máximo em A_i(e⃗), i ∈ {0,...,3}
13         ALG ← ALG ∪ {A}
14         V(G) ← V(G) \ V(A)
15         M ← M \ E(A)
16     M' = M aumentado por todos os aumentos em ALG
17     retorna M'
```

Denota-se por A_i o conjunto de todos os $M-$aumentos que contém pelo menos i arestas que não pertençam a M, com $i \geq 0$. O Algoritmo 15 considera um *braço* de um vértice v um caminho alternante de comprimento 1 ou 2, iniciando em v e no qual a primeira aresta não pertence ao emparelhamento. Um $M-$aumento é um ciclo ou caminho alternante cuja diferença simétrica com um emparelhamento M obtém como resultado um conjunto que também é um emparelhamento.

Seja um braço s. Denote por $M(s)$ as arestas de M que também pertencem a s. O *ganho* de s, denotado por $g(s)$, é definido como a soma dos custos das arestas de $S \setminus M(s)$, subtraído da soma dos custos das arestas de $M(s)$.

Em síntese, o Algoritmo 15 busca por braços com tamanho máximo 2 para cada vértice de cada aresta orientada do emparelhamento inicial, em ambos os sentidos e, portanto, verifica ambos os vértices de cada aresta do emparelhamento original.

O vértice v mais distante do braço pode ainda ser estendido por um outro braço a partir de v, chamado de *extensão*. Se existir, uma extensão será adicionada somente se a quantidade de arestas em cada braço que não pertencem ao emparelhamento for igual. Note que a extensão e o braço somente serão utilizados pelo algoritmo se o ganho for positivo. A extensão pode ocorrer em ambos os braços. Por este motivo o algoritmo verifica cada aresta do emparelhamento nos dois sentidos.

O conjunto A_3 é composto por uma aresta não emparelhada do braço esquerdo, uma aresta não emparelhada do braço direito e uma aresta não emparelhada da extensão. Os conjuntos A_0, A_1,A_2 também são verificados e o que possuir maior ganho é selecionado.

Para garantir a complexidade final de $\Theta(m \log n)$, o Algoritmo 15 restringirá a busca no conjunto A_2 e consequentemente no conjunto A_3. Para tanto, emprega-se a definição da função rank tal que, sendo g_{max} o ganho do braço de maior ganho dentre todos para um mesmo vértice, o *rank* $r(s)$ é conforme a seguir.

$$r(s) = 0, \text{ se } c(s) \leq \frac{\alpha \times g_{max}}{n}$$

$$r(s) = i > 0 \text{ caso contrário, onde } i \text{ é o menor inteiro tal que}$$

$$c(s) \leq \beta^i \times \frac{\alpha \times g_{max}}{n}$$

onde $\alpha > 0$ e $\beta > 1$ são constantes.

Primeiro, armazena-se numa lista todos os braços de cada vértice $v \in V(G)$ ordenados pelo ganho. Cada vértice de v possui uma quantidade de braços igual ao seu grau e, conforme o Teorema 2, temos que a quantidade de braços em um grafo é $2m$. Após, o algoritmo identifica o caminho ou ciclo de maior ganho para posterior diferença simétrica com o emparelhamento original M. O rank é utilizado como critério de escolha para possíveis extensões.

Executando o algoritmo no grafo da Tabela 4, obteremos o emparelhamento $M = \{(1,3),(2,8),(4,5)\}$ com custo igual a 271. Comparando o resultado com o emparelhamento máximo temos

$$\frac{c(M)}{c(M_{\text{máx}})} = \frac{271}{316} = 0{,}86$$

Neste caso o algoritmo obtém um fator de aproximação maior que o mínimo garantido de 0,75.

Hanke e Hougardy [33] sugerem como estender o Algoritmo 15 para um algoritmo não exato com fator de aproximação $\frac{4}{5} - \varepsilon$. Contudo, conforme Duan e Pettie [34], os detalhes de tal algoritmo não são fundamentados. Por este motivo tal algoritmo não será considerado neste trabalho.

Existe ainda um algoritmo com aproximação $1 - \varepsilon$ [34]. Contudo, o mesmo apresenta complexidade de implementação análoga à do algoritmo de Edmonds original, utilizando florações e reduções de flores, além de ser baseado em programação linear. Por esta razão tal algoritmo também não será considerado.

3 RESULTADOS EM GRAFOS SINTÉTICOS

Neste capítulo apresentaremos um resultado empírico em grafos sintéticos empregando os algoritmos do Capítulo 2. Na Seção 3.1, discutimos os piores casos de cada algoritmo. Na Seção 3.2, descrevemos como foram conduzidos os experimentos com o algoritmo exato. Na Seção 3.3 apresentamos resultados obtidos na execução de implementações tanto do algoritmo exato quanto dos algoritmos não exatos, comparando inclusive os piores casos. Todas as implementações, incluindo as instâncias de teste, estão disponíveis publicamente em [35].

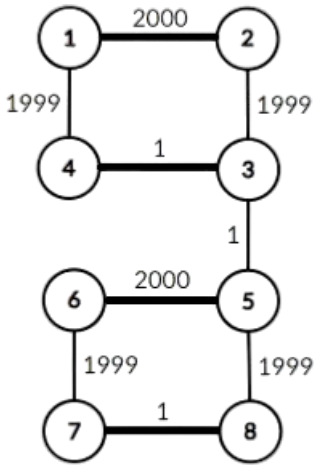
3.1 Piores casos dos algoritmos não exatos

Nessa seção, buscaremos apresentar grafos para os quais o fator de aproximação do emparelhamento resultante aos vários algoritmos do Capítulo 2 seja próximo do mínimo garantido pelo respectivo algoritmo.

3.1.1 Greedy

O pior caso do Algoritmo 6 ocorre em grafos onde para cada aresta e escolhida a ser adicionada ao emparelhamento, duas outras arestas e_1 e e_2 adjacentes a e não podem mais ser adicionadas ao emparelhamento e $c(e_1) \leq c(e_2) < c(e)$. Quanto mais os valores de $c(e_1)$ e $c(e_2)$ tendem ao valor de $c(e)$ mais o fator de aproximação se aproxima do mínimo. As demais arestas adjacentes a e,e_1,e_2, devem ter custo mínimo.

Figura 14 – Exemplo de um grafo com fator de aproximação próximo do limite mínimo para o Algoritmo 6.



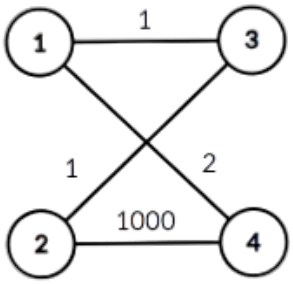
Na Figura 14 as arestas em negrito pertencem ao emparelhamento obtido pelo Algoritmo 6. O emparelhamento de custo máximo é dado pelas arestas {(1,4),(2,3),(5,8),(6,7)}.

3.1.2 Vertex Scan

O pior caso do Algoritmo 7 ocorre quando um vértice selecionado leva o algoritmo a excluir uma aresta que pertence ao emparelhamento ótimo.

Os grafos G e G_{con} que utilizaremos como exemplo são construídos da seguinte forma: G é um grafo bipartido $K_{4,4}$, com $V(G) = v_i \mid 1 \leq i \leq 4$ e $E(G) = \{(1,3),(1,4),(2,3),(2,4)\}$. Os custos são $c(1,3) = 1000$, $c(1,4) = 999$, $c(2,3) = 1000$, $c(2,4) = 1$. Construímos G_{con}, um grafo obtido a partir de G com alteração dos custos tais que $c_{con}(1,3) = 1$, $c_{con}(1,4) = 2$, $c_{con}(2,3) = 1$, $c_{con}(2,4) = 1000$. Temos que o emparelhamento maximal mínimo ponderado de G equivale ao emparelhamento máximo ponderado de G_{con} conjugado a G, conforme o Lema 6.

Figura 15 – Grafo G_{con}, conjugado a G, com fator de aproximação próximo de zero para o Algoritmo 7.



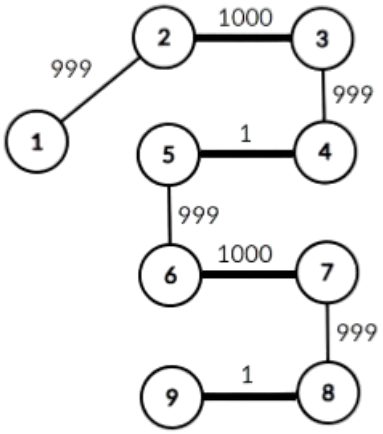
Utilizando o grafo G como parâmetro de entrada para o Algoritmo 7, caso seja sorteado o vértice 1, o emparelhamento obtido é $\{(1,4),(2,3)\}$ e, aplicado ao grafo da Figura 15, o fator de aproximação é próximo a zero. O emparelhamento de custo máximo de G_{con} é dado pelas arestas $\{(1,3),(2,4)\}$.

Note que é possível generalizar um ou mais grafos estrela conexos para que qualquer grafo G possa ser utilizado como entrada do Algoritmo 7, pois a operação de adicionar arestas de custo infinito não altera o emparelhamento mínimo de um grafo. A probabilidade da aresta do emparelhamento ótimo ser escolhida para ser adicionada ao emparelhamento diminui conforme existem diversas arestas adjacentes com custo inferior. Este efeito é multiplicado pela quantidade de subgrafos estrela.

3.1.3 LAM

O pior caso do Algoritmo 8 ocorre quando as arestas de máximo local excluem duas outras arestas pertencentes ao emparelhamento de custo máximo.

Figura 16 – Exemplo de um grafo com fator de aproximação próximo do limite para o Algoritmo 8.

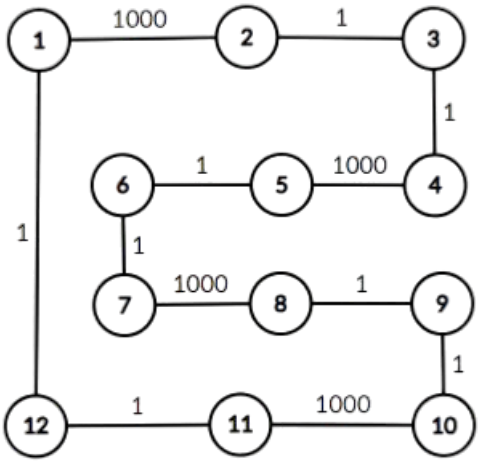


Na Figura 8 arestas em negrito pertencem ao emparelhamento obtido pelo algoritmo. O emparelhamento de custo máximo é dado pelas arestas {(1,2),(3,4),(5,6),(7,8)}.

3.1.4 PGA

O pior caso do Algoritmo 9 ocorre quando metade das arestas do emparelhamento de custo máximo são adicionadas ao emparelhamento M_1 do algoritmo e a outra metade é adicionada ao emparelhamento M_2 do algoritmo.

Figura 17 – Exemplo de um grafo com fator de aproximação próximo do limite para o Algoritmo 9.



Na Figura 17, os emparelhamentos M_1 e M_2 obtidos pelo algoritmo, a partir da escolha do vértice inicial 1, são respectivamente {(1,2),(3,4),(5,6),(7,8),(9,10),(11,12)} e {(2,3),(4,5),(6,7),(8,9),(10,11),(12,1)}. O emparelhamento de custo máximo é dado pelas arestas {(1,2),(4,5),(7,8),(10,11)}.

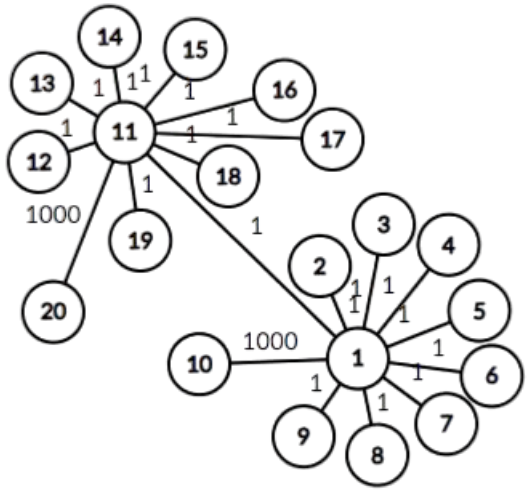
3.1.5 Maximal Matching

O pior caso do Algoritmo 10 ocorre em grafos formados por subgrafos do tipo estrela, com uma aresta conectando seus centros de modo a ser conexo. A probabilidade da aresta

de maior custo ser escolhida para ser adicionada ao emparelhamento diminui conforme existem diversas arestas adjacentes com custo inferior. Este efeito é multiplicado pela quantidade de subgrafos estrela.

O grafo G que utilizaremos como exemplo do Algoritmo 10 é construído da seguinte forma: $V(G) = v_i \mid 1 \leq i \leq 20$ e $E(G) = (v_1, v_i) \mid 2 \leq i \leq 10 \cup (v_{11}, v_i) \mid 12 \leq i \leq 20$. Os custos são tais que, para todo $e \in E(G)$, $c(e) = 1$, se e é incidente a 10 ou 20, caso contrário $c(e) = 1000$.

Figura 18 – Grafo G, com fator de aproximação próximo de zero para o Algoritmo 10.



Note que seria possível adaptar o exemplo na Figura 18 para ter uma única estrela, com igual conclusão. O exemplo apresentado mostra a possibilidade de aumentar o número de estrelas sem aumentar o tamanho de cada uma, limitando o grau máximo do grafo.

Utilizando o grafo G como parâmetro de entrada para o Algoritmo 10, caso não seja sorteado nenhum dos vértices {1,10,11,20}, o fator de aproximação do emparelhamento obtido aplicado ao grafo da Figura 18 é próximo a zero e caso apenas um desses vértices seja sorteado, o fator de aproximação é próximo de $\frac{1}{2}$. O emparelhamento de custo máximo de G_{con} é dado pelas arestas {(1,10),(11,20)}, únicas com peso 1000. Todas as demais arestas possuem peso 1.

3.1.6 RAMA

O pior caso do Algoritmo 11 é de difícil identificação por se tratar de um algoritmo randomizado. Contudo, note que se k for insuficiente o emparelhamento pode ser arbitrariamente distante do ótimo.

A Tabela 6 contém os resultados da execução do Algoritmo 11 em grafos completos e não-completos, em comparação com o parâmetro de entrada k. Para cada par n, k, são feitas 100 execuções. Todos os grafos utilizados possuem emparelhamento perfeito. Os resultados obtidos ilustram o comportamento do Algoritmo 11 em função dos parâmetros n e k. Não há garantias de um valor mínimo para k em função da cardinalidade do grafo que garanta um emparelhamento perfeito ou ao menos maximal, mesmo em um grafo completo. Contudo, observou-se, com base nos dados da Tabela 6, que um valor de k 4 vezes maior que o número de vértices do grafo obteve um emparelhamento perfeito em pelo menos 80% das execuções.

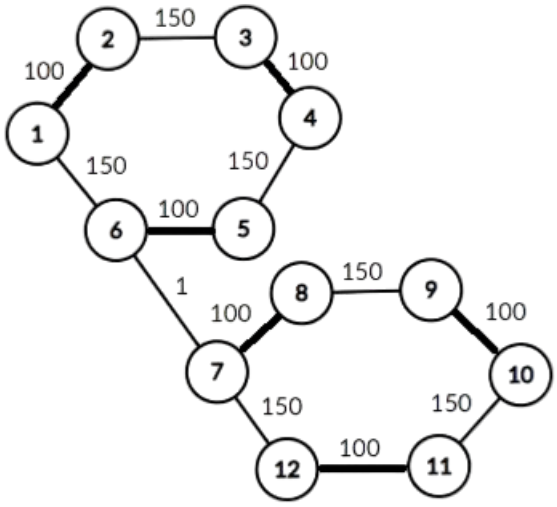
Tabela 6 – Resultados preliminares da execução do algoritmo RAMA.

n	k	Tempo execução médio (h:min:seg)	Erro médio (%)	Emparelhamentos	
				perfeitos (%)	não perfeitos (%)
100	100	00:00:01	4,52	76	24
100	200	00:00:01	3,27	79	21
100	300	00:00:01	2,95	82	18
100	400	00:00:01	3,05	80	20
200	200	00:00:01	5,69	76	24
200	400	00:00:01	3,43	96	4
200	600	00:00:01	3,12	100	0
200	800	00:00:01	3,19	100	0
500	500	00:00:02	2,99	82	18
500	1000	00:00:04	2,50	90	10
500	1500	00:00:07	2,42	97	3
500	2000	00:00:09	2,44	99	1

3.1.7 Improve Matching

O pior caso ocorre quando não existe β-aumento com centro nas arestas do emparelhamento inicial, por exemplo ciclos $C_k \mid k \in 6{,}8, \ldots, 2i$ com emparelhamento maximal inicial M e função de custos c conforme a seguir. Seja $x > 0 \mid x \in \mathbb{R}^+$, $c(e) = x$ se $e \in M$, $c(e) = \frac{2}{3}x$ caso contrário.

Figura 19 – Exemplo de um grafo com fator de aproximação próximo do limite para o Algoritmo 12.



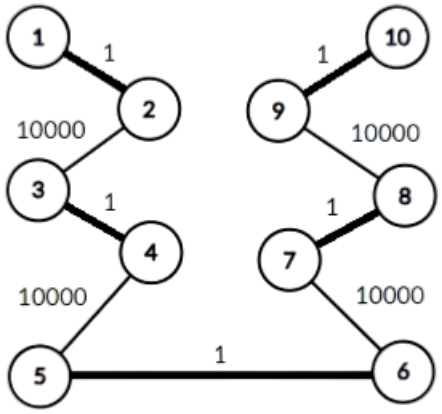
Na Figura 19 arestas em negrito pertencem ao emparelhamento inicial, idêntico ao obtido pelo algoritmo. Tal emparelhamento é {(1,2),(3,4),(5,6),(7,8),(9,10),(11,12)} e possui fator de aproximação $\frac{2}{3}$. O emparelhamento de custo máximo é dado pelas arestas {(1,6),(2,3),(4,5),(7,12),(8,9),(10,11)}. Note que seria possível adaptar o exemplo na Figura 19 para ter um único ciclo de cardinalidade 6, com igual conclusão. O exemplo apresentado mostra a possibilidade de aumentar o número de ciclos sem aumentar o ta-

manho de cada um.

3.1.8 $\frac{3}{4}$-emparelhamento

O pior caso ocorre quando temos um caminho onde 4 arestas que pertençam ao emparelhamento ótimo não pertencem ao emparelhamento inicial. Isso ocorre pois, ao identificar o caminho de maior ganho com tamanho 7, por exemplo o caminho entre os vértices 1 e 8 da Figura 20, o algoritmo remove todos os vértices deste caminho e consequentemente uma aresta não emparelhada pertencente ao emparelhamento ótimo. Note que, quanto maior a diferença entre os custos das arestas do emparelhamento ótimo e as arestas do emparelhamento inicial, mais o fator de aproximação tende ao seu limite mínimo.

Figura 20 – Exemplo de um grafo com fator de aproximação próximo do limite para o Algoritmo 15.



Na Figura 20 arestas em negrito pertencem ao emparelhamento inicial. Tal emparelhamento é {(1,2),(3,4),(5,6),(7,8),(9,10)} e possui fator de aproximação 0,001. O emparelhamento de custo máximo é dado pelas arestas $\{(2,3),(4,5),(6,7),(8,9)\}$. O emparelhamento obtido pelo algoritmo é dado pelas arestas $\{(2,3),(4,5),(6,7),(9,10)\}$, com fator de aproximação próximo de $\frac{3}{4}$.

3.2 Utilização e testes dos algoritmos exatos

Aqui descreveremos como utilizamos os algoritmos exatos obtidos em [15] [17] e como verificamos o resultado obtido por estes.

Seja A uma árvore com função de custos $c_a(e) = 1, e \in E(A)$, M um emparelhamento máximo de A e $G_p = (V,E)$ um grafo e $p \in \{10,40,70,100\}$. Cada G_p utilizado nos testes desta seção foi gerado da seguinte forma. Faça $V(G_p) = V(A)$ e cada possível aresta $e \in E(G_p)$ de fato existe com probabilidade independente p, se $e \notin E(A)$, senão, 100. Seja a função de custos c onde, se $e \in E(A) \cap M$, então $|V(A)| < c(e) \leq 2|V(A)|$, senão, $1 \leq c(e) \leq |V(A)|$. Note que para cada G_p construído conhecemos o emparelhamento máximo M. Utilizamos os grafos descritos para comparar com os algoritmos exatos obtidos em [15] [17], com três valores de cardinalidades: $500, 1000, 1500$.

Implementado na linguagem `Python`, o algoritmo obtido em [15] foi incorporado ao projeto desta dissertação, implementado na mesma linguagem, e obteve um emparelhamento de custo igual ao emparelhamento de custo máximo em todos os casos testados. Sua utilização é realizada através de uma chamada à função `maxWeightMatching` com

parâmetros de entrada `edges`, uma lista de tuplas, cada qual contendo, respectivamente, dois vértices de uma aresta e o custo da mesma, e `maxcardinality`, parâmetro que indica se o emparelhamento retornado deve ser de máxima cardinalidade. Esta função retorna uma lista onde cada item $i \neq -1$ corresponde a um vértice u e aponta para o outro vértice v tal que (u,v) pertence ao emparelhamento obtido pelo algoritmo.

Implementado na linguagem `C++`, o algoritmo obtido em [17] é de uso não comercial. Seu projeto, com 34 arquivos, foi compilado e o arquivo executável obtido é executado pelo comando a seguir, onde `p1` é o caminho completo do arquivo executável, `p2` é o caminho completo do grafo de entrada e `p3` é o caminho completo do arquivo de saída.

```
p1\Executavel.exe -e p2\grafo.txt -w p3\Result.TXT
```

O parâmetro de execução `e` indica que a entrada é um grafo no formato de lista de adjacências e o parâmetro de execução `w` indica que o resultado deve ser salvo em um arquivo. A execução deste comando no projeto desta dissertação se deu através da biblioteca `Python Subprocess` utilizando a função `Run`. Em casos com grafos de cardinalidade a partir de 500 o algoritmo apresentou erro `0xC0000005`: violação de acesso ao gravar no local. Nos demais casos testados o algoritmo obteve um emparelhamento de custo igual ao emparelhamento de custo máximo.

3.3 Resultados comparativos

Aqui apresentaremos resultados comparativos preliminares obtidos na execução dos algoritmos anteriores, considerando duas situações. A primeira é para grafos aleatórios, com densidades variadas. A segunda é para os piores casos de cada algoritmo, conforme a caracterização feita anteriormente.

Todos os resultados apresentados nos Capítulos 3 e 4 foram realizados em um computador com as seguintes configurações de hardware e software. Processador Intel Core i5-10210U (1.60GHz – 4 núcleos), memória RAM de 8,00 GB, sistema operacional Windows 10 Home (64 bits) e Microsoft Visual Studio Community 2022.

3.3.1 Resultados para grafos aleatórios

Foram utilizadas quatro classes de grafos. Para cada grafo $G = (V,E)$ utilizado nos testes dessa seção, temos que $m \leq \frac{n \times (n-1)}{2}$ e $1 \leq c(e) \leq m$, para todo $e \in E(G)$. Cada grafo foi criado com 1000 vértices. Além disso, os valores médios foram obtidos pela execução de 100 vezes para cada situação listada. Os grafos foram gerados como grafos aleatórios da seguinte forma:

- Tipo 1: toda possível aresta de G de fato existe com probabilidade independente de 10%;
- Tipo 2: toda possível aresta de G de fato existe com probabilidade independente de 40%;
- Tipo 3: toda possível aresta de G de fato existe com probabilidade independente de 70%;
- Tipo 4: o grafo é completo;

Note que os grafos do Tipo 4 são os mais importantes para o CPP, afinal o grafo auxiliar gerado pelas distâncias mínimas entre todos os pares de vértices ímpares do grafo original sempre será completo.

Uma vez que o Algoritmo 7, Vertex Scan, é um algoritmo para emparelhamento ponderado de custo mínimo, para cada grafo G dos Tipos 1 a 4, utilizamos o grafo conjugado G' de G para encontrar as arestas do emparelhamento máximo em G, que são problemas equivalentes como mostrado no Lema 6.

Os resultados das execuções dos diversos algoritmos estão mostrados nas Tabelas 7 e 8. Na Tabela 7, apresentamos, para cada algoritmo utilizado, o tipo de grafo, o tempo de execução e o fator de aproximação médio do custo do emparelhamento obtido. Observe que o último algoritmo considerado nesta tabela é o algoritmo exato, que obtém o emparelhamento de custo máximo que serve de comparação para os demais algoritmos.

Observa-se pelos dados desta tabela que, para todos os tipos de grafos, no quesito fator de aproximação, todos os algoritmos não exatos, com exceção de um caso, apresentam fatores de aproximação superiores a 90%. Os seis primeiros algoritmos têm um desempenho semelhante com fator de aproximação médio superior a 98%, o que é bastante alto. O algoritmo Improve Matching apresenta o menor desempenho, com fator médio de cerca de 92%. Com relação aos tempos de execução, verifica-se que todos os tempos são bastante baixos, situando-se na casa de alguns segundos e que esses tempos crescem com o tamanho dos grafos. Não há discrepâncias relevantes no tempo de execução entre os diversos tipos de algoritmos.

A Tabela 8 realça a diferença de tempo médios de execução mais baixos das diversas soluções não exatas, em relação ao algoritmo exato. São apresentados e comparados os tempos de execução mais baixos obtidos pelos diversos algoritmos não exatos, para cada tipo de grafo. Apresenta-se, também, na coluna Razão, a proporção entre esse tempo e o do algoritmo exato.

Observa-se que, para todos os tipos de grafos, o algoritmo que obteve o menor tempo foi o RAMA. Em todos os casos esse tempo foi pelo menos 40 vezes inferior ao do algoritmo exato, ressaltando uma grande discrepância entre os tempos de execução das soluções não exatas com a exata. Essa razão decresce com o aumento do tamanho do grafo, o que sugere que as soluções não exatas são mais sensíveis à densidade do grafo.

3.3.2 Resultados para os piores casos

A Tabela 9 ilustra os tempos de execução e os fatores de aproximação obtidos pela execução de seis dos sete métodos não exatos considerados. O algoritmo RAMA foi deixado de lado pela dificuldade de identificar o pior caso de um algoritmo complexo e também randomizado.

Para as comparações foram criados grafos que procurassem realizar o menor fator de aproximação garantido pelo algoritmo correspondente. Além disso, foram considerados grafos de três tamanhos, em relação ao número de vértices: 500, 1000 e 1500. Entretanto, em alguns casos só se conseguiu criar grafos com tamanhos próximos a essas referências. A geração de tais grafos foi feita de acordo com o que foi apresentado no início deste capítulo. Os tempos de execução e fatores de aproximação são valores médios de 50 execuções.

Uma análise inicial dos resultados mostrados na Tabela 9 é a de que foram obtidos fatores de aproximação bem próximos dos valores mínimos esperados. Aqui devemos lembrar que dois dos métodos: Maximal Matching e Vertex Scan, não têm garantia de aproximação e foram os que apresentaram os piores fatores, o que atesta que as ideias usadas para obter os piores casos funcionou bem. Com relação aos tempos de execução, observa-se que os tempos são pequenos e isso se deve ao fato dos piores casos envolverem grafos com baixa densidade.

Finalmente, concluímos que, embora os métodos não exatos possam ter um desem-

Tabela 7 – Resultados preliminares - grafos com $n = 1000$.

Algoritmo	Tipo de grafo	Tempo execução médio (h:min:seg)	Erro médio
RAMA	1	00:00:09	4,06
	2	00:00:40	1,26
	3	00:01:19	0,76
	4	00:02:05	0,55
PGA'	1	00:00:47	3,74
	2	00:02:41	1,26
	3	00:05:18	0,70
	4	00:07:53	0,48
Maximal Matching	1	00:00:25	3,50
	2	00:01:39	1,19
	3	00:02:53	0,78
	4	00:03:54	0,52
LAM	1	00:00:14	4,24
	2	00:00:56	2,86
	3	00:01:42	2,11
	4	00:02:40	1,79
Vertex Scan	1	00:00:24	3,48
	2	00:01:41	1,22
	3	00:02:51	0,77
	4	00:04:05	0,61
Greedy	1	00:00:23	3,00
	2	00:01:33	1,00
	3	00:02:44	0,80
	4	00:03:56	0,39
Improved	1	00:00:56	10,20
	2	00:03:22	8,20
	3	00:06:06	6,69
	4	00:08:16	7,50
$\frac{3}{4}$-emparelhamento	1	00:00:49	5,79
	2	00:03:48	5,53
	3	00:06:34	2,64
	4	00:08:42	1,79
Edmonds Joris	1	00:11:26	0,00
	2	00:39:53	0,00
	3	01:05:06	0,00
	4	01:26:18	0,00

penho médio muito bom, tanto em termos de tempos de execução quanto de fatores de aproximação, os piores casos de fatores de aproximação podem acontecer na prática, ficando-se longe das soluções ótimas. Desta forma, a utilização desses métodos deve ser feita de forma cuidadosa e merece ainda um maior aprofundamento na continuação deste trabalho.

Tabela 8 – Comparativos dos melhores tempos com o algoritmo exato.

Tipo de grafo	Menor tempo médio	Algoritmo	Tempo algoritmo exato	Razão
1	00:00:09	RAMA	00:11:26	76
2	00:00:40	RAMA	00:39:53	59
3	00:01:19	RAMA	01:05:06	49
4	00:02:05	RAMA	01:26:18	41

Tabela 9 – Resultados preliminares - piores casos.

Algoritmo	n	Tempo execução médio (h:min:seg)	Fator de aproximação médio	Erro médio (%)
PGA'	498	00:00:01	0,500	50,00
	1002	00:00:02	0,500	50,00
	1500	00:00:05	0,500	50,00
Maximal Matching	500	00:00:01	0,001	99,90
	1000	00:00:01	0,001	99,90
	1500	00:00:01	0,001	99,90
LAM	501	00:00:01	0,500	50,00
	1001	00:00:01	0,500	50,00
	1501	00:00:01	0,500	50,00
Vertex Scan	500	00:00:01	0,001	99,90
	1000	00:00:01	0,051	94,90
	1500	00:00:01	0,014	98,60
Greedy	500	00:00:01	0,500	50,00
	1000	00:00:01	0,500	50,00
	1500	00:00:03	0,500	50,00
Improved	498	00:00:01	0,667	33,30
	996	00:00:01	0,667	33,30
	1500	00:00:01	0,667	33,30
$\frac{3}{4}$-emparelhamento	500	00:00:01	0,750	25,00
	1000	00:00:01	0,750	25,00
	1500	00:00:03	0,750	25,00

4 EXPERIMENTOS COM CASOS REAIS

Neste capítulo abordaremos os principais resultados obtidos com casos reais conforme descreveremos mais adiante.

4.1 Impacto da complexidade cúbica

O experimento a seguir visa responder ao seguinte questionamento. Qual o impacto da complexidade de tempo cúbica em computadores com hardwares e softwares atuais? Este questionamento é de interesse pois, como veremos a frente, a complexidade cúbica está presente nos algoritmos exatos que envolvem a solução do CPP de maneira exata. Para tal, foi desenvolvido um algoritmo simples, com um parâmetro de entrada n e três laços. Dentro do laço mais interno é realizada uma conta aritmética simples, com o intuito de evitar que o otimizador do compilador suprima o laço. O algoritmo foi desenvolvido na linguagem `Python`.

Algoritmo 16: Experimento com Complexidade de Tempo Cúbica

Complexidade de Tempo: $\Theta(n^3)$.

Entrada: $n \in \mathbb{N}$

```
1 Função Experimento_O3(n):
2     a ← 0
3     para i ∈ 1...n faça
4         para j ∈ 1...n faça
5             para k ∈ 1...n faça
6                 a ← a + √k
```

A Tabela 10 ilustra os tempos de execução do Algoritmo 16 para valores de n entre 500 e 5000. Para n igual a 3000 o tempo de execução já ultrapassava 1 hora.

Tabela 10 – Execuções do Algoritmo 16 com complexidade de tempo $\Theta(n^3)$.

n	Tempo execução (h:min:seg)
500	00:00:28
1000	00:03:50
1500	00:13:22
2000	00:29:09
2500	00:53:31
3000	01:29:28
3500	02:17:40
4000	03:14:26
4500	04:37:18
5000	06:34:51

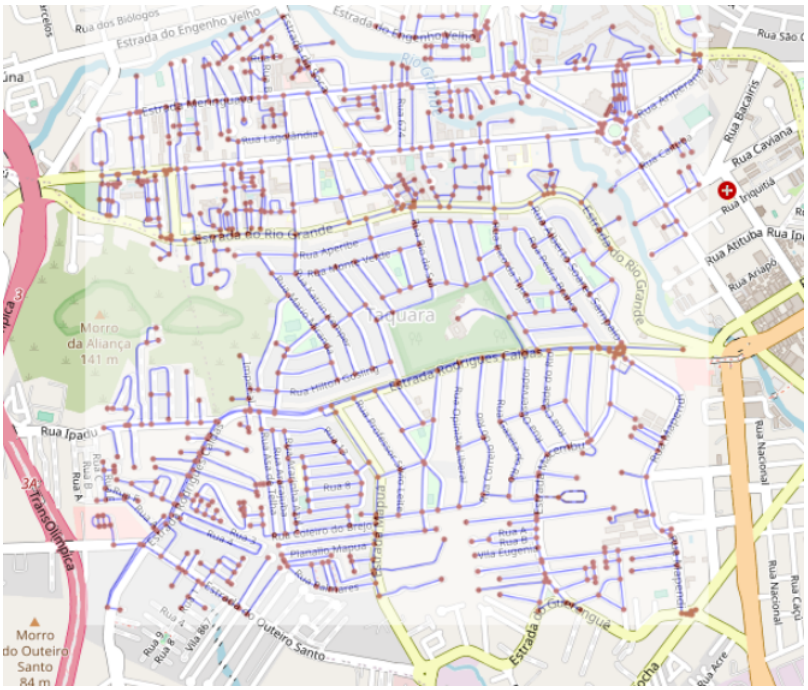
Figura 21 – Gráfico das execuções de um algoritmo simples com complexidade de tempo $\Theta(n^3)$.

A Figura 21 ilustra os dados da Tabela 10. Para esse experimento, a função $f(n)$ que representa aproximadamente tal função cúbica é $f(n) = 0{,}00000023\ n^3$. Para tal função, para valores de $n \geq 22.420$, $f(n)$ ultrapassaria 2.592.000 segundos (equivalente a 1 mês). Conforme será mostrado na Seção 4.2 deste Capítulo, o grafo do estado do Rio de Janeiro possui 108.258 vértices, quase 5 vezes maior que o valor de n esperado para a duração de 1 mês. Além disso, o algoritmo deste experimento é significativamente mais simples que os demais algoritmos abordados neste trabalho e, portanto, é esperado que o tempo de execução dos demais algoritmos seja ainda maior.

4.2 Grafos a partir de localidades geográficas

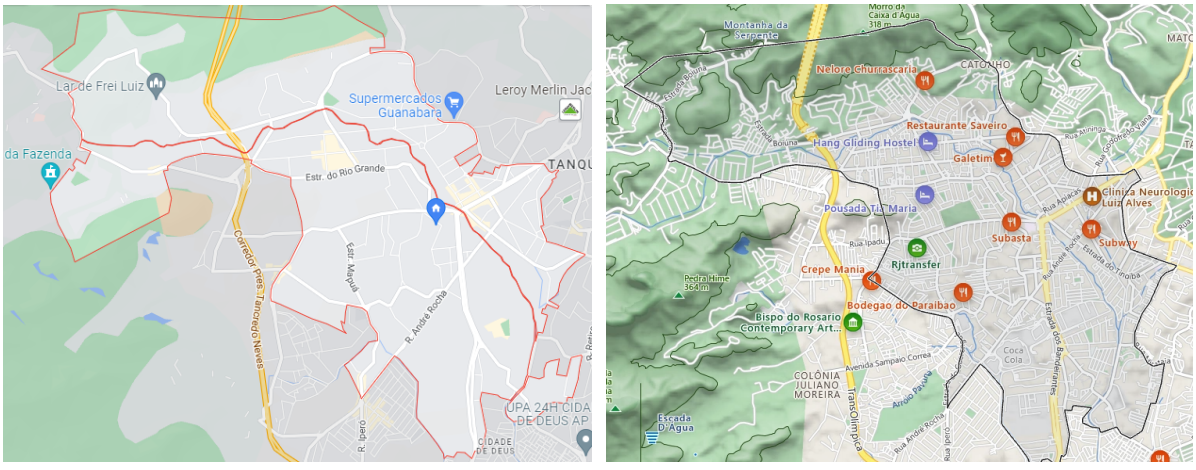
Os grafos deste capítulo foram obtidos através do Open Street Maps, utilizando-se a biblioteca `Osmnx` do `Python`.

Figura 22 – Grafo obtido pela biblioteca Osmnx do bairro Taquara.



Na Figura 22 temos a plotagem do grafo obtido, ao solicitar-se à biblioteca `Osmnx`, o bairro Taquara, no município do Rio de Janeiro, sobreposto com uma pequena parte do mapa do próprio bairro. Note que as imagens das ruas (linhas em azul) derivadas dos dados obtidos coincidem com as ruas quando desenhadas a partir do serviço de mapas.

Figura 23 – Mapas do bairro da Taquara.



(a) (b)

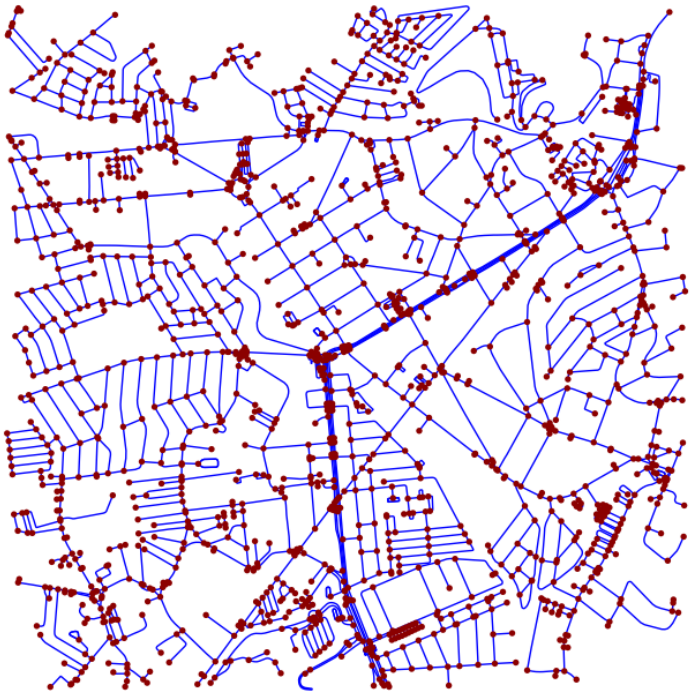
Na Figura 23(a) temos o mapa obtido pelo Google Maps do bairro Taquara, no município do Rio de Janeiro e na Figura 23(b) temos o mapa da mesma região obtido pelo Microsoft Bing. Ambos parecem coincidir em uma inspeção visual.

Observe que os mapas da Figuras 23(a) e (b) não parecem coincidir visualmente com o mapa da mesma região representado na Figura 22. Isto se deve, provavelmente, aos limites territoriais do bairro estarem diferentes de acordo com a ferramenta utilizada. O

Google Maps e o Microsoft Bing parecem possuir limites territoriais para o bairro da Taquara diferentes do Open Street Maps.

Optaremos, no caso do bairro da Taquara, por grafos que correspondam ao raio do mapa a partir de um determinado ponto, ao invés de solicitar à biblioteca o bairro por completo.

Figura 24 – Grafo correspondente a região do bairro Taquara, no município do Rio de Janeiro.

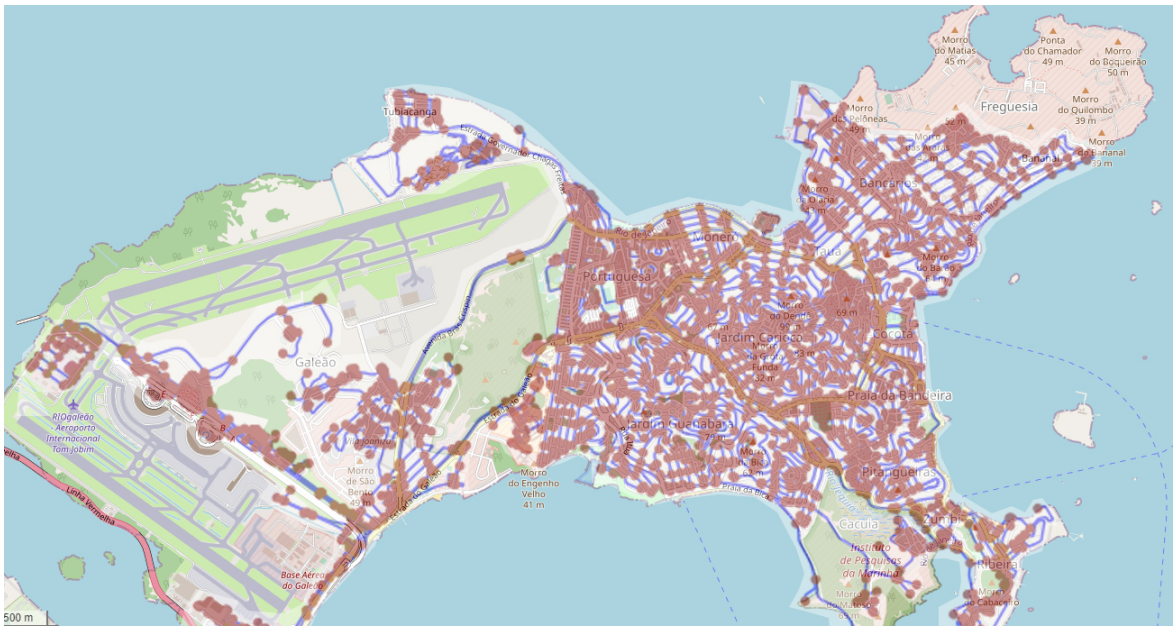


Na Figura 24 temos a representação de uma região com raio do mapa de até 1500 metros no bairro Taquara, no município do Rio de Janeiro, a partir do ponto central do mapa. Essa consulta resulta em um grafo com 1.618 vértices representando as intercessões entre as vias e 2.089 arestas representando o comprimento do percurso, em metros, entre cada par de vértices. O grafo possui 1.338 vértices de grau ímpar. Os dados foram obtidos através do Open Street Maps. Note que o raio do mapa entre dois pontos não é necessariamente a menor distância em linha reta entre dois pontos e, portanto, o grafo obtido a partir da distância do percurso a partir de um ponto não necessariamente será uma circunferência.

Figura 25 – Sobreposição do grafo representando uma região no bairro Taquara e o mapa da mesma região do Open Street Maps.

Na Figura 25 temos a sobreposição do grafo da Figura 24 com o mapa obtido pelo *www.openstreetmap.org*. Observa-se visualmente que ambos parecem coincidir com relação a vértices e arestas do grafo comparados com interseções e vias do mapa.

Figura 26 – Sobreposição do grafo representando a região de Ilha do Governador e o mapa da região do Open Street Maps.

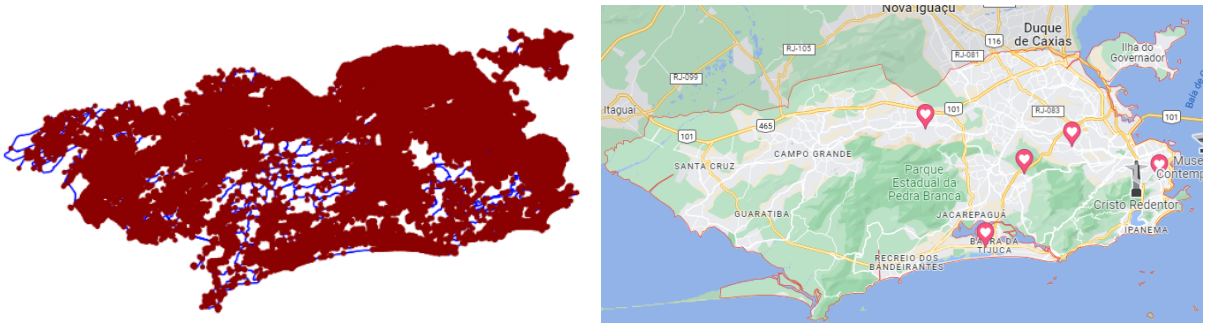


Na Figura 26 temos a sobreposição do grafo da região da Ilha do Governador, com 3.510 vértices representando as intercessões entre as vias e 4.934 arestas representando comprimento do percurso, em metros, entre cada par de vértices, com o mapa obtido pelo

www.openstreetmap.org. O grafo possui 2.928 vértices de grau ímpar. Os dados foram obtidos através do Open Street Maps.

Observa-se visualmente que ambos parecem coincidir com relação a vértices e arestas do grafo comparados com interseções e vias do mapa.

Figura 27 – Grafo do município do Rio de Janeiro e seu respectivo mapa.



(a) (b)

Na Figura 27(a) temos a representação do município do Rio de Janeiro, em um grafo com 108.258 vértices representando as intercessões entre as vias e 145.897 arestas representando a distância entre cada par de vértices. O grafo possui 91.972 vértices de grau ímpar. Os dados foram obtidos através do Open Street Maps. Na Figura 27(b) temos o mapa da mesma região mapa obtido pelo Google Maps.

Note que, diferente do ocorrido com as Figuras 23(a) e (b), as Figuras 27(a) e (b) parecem coincidir visualmente.

4.3 Resultados dos experimentos

Nesta seção apresentamos os resultados dos experimentos com grafos em função do raio do mapa e o grafo da Ilha do Governador. Não iremos realizar experimentos com o grafo do município do Rio de Janeiro pois, conforme será possível constatar a partir dos dados que apresentaremos a seguir, o tempo de execução é maior que aquele considerado limitante para essa pesquisa.

Tabela 11 – Quantidade de vértices total e de grau ímpar em função do raio do mapa.

Raio do mapa (metros)	Vértices	T_1 (h:min:seg)	Vértices ímpares	T_2 (h:min:seg)
500	219	00:00:05	144	00:00:48
1.000	701	00:00:49	540	00:11:29
1.500	1.618	00:10:37	12538	03:46:07
2.000	2.709	00:53:10	2.306	15:55:09
2.500	3.752	02:00:17	3.232	43:49:59

A Tabela 11 ilustra a quantidade de vértices de um grafo G obtido em função do raio do mapa de um determinado ponto da região no bairro da Taquara. A coluna T_1 indica o tempo para a obtenção do grafo auxiliar G', cujos vértices são os vértices de grau ímpar de G e cujas arestas correspondem à distância mínima entre os respectivos

vértices em G. A coluna *Vértices ímpares* indica a quantidade de vértices ímpares em G e, consequentemente, a quantidade de vértices em G'. A coluna T_2 ilustra o tempo de execução do algoritmo exato para obtenção do emparelhamento perfeito de custo mínimo do grafo G'. A quantidade de vértices de grau ímpar constituem aproximadamente 85% dos vértices do respectivo grafo G.

Tabela 12 – Tempo de execução dos algoritmos não exatos em grafos por função do raio do mapa.

Raio do mapa (metros) / Algoritmo	500	1000	1500	2000	2500
RAMA	00:00:01	00:00:07	00:00:43	00:04:24	00:18:01
PGA'	00:00:01	00:00:40	00:10:23	00:54:46	02:21:19
Maximal Matching	00:00:01	00:00:21	00:05:16	00:25:42	01:16:36
LAM	00:00:01	00:00:12	00:02:48	00:12:54	00:37:08
Vertex Scan	00:00:01	00:00:20	00:05:19	00:25:36	01:22:12
Greedy	00:00:01	00:00:11	00:02:52	00:13:33	00:46:35
Improved	00:00:05	00:01:06	00:05:26	00:15:26	00:40:02
$\frac{3}{4}$-emparelhamento	00:00:02	00:00:59	00:05:44	00:17:55	00:49:58

A Tabela 12 ilustra o tempo de execução médio dos algoritmos não exatos para os grafos de grau ímpar detalhados na Tabela 11. O erro médio é inferior a 4% para todos os algoritmos não exatos para todos os grafos da Tabela 12. Observa-se que o algoritmo RAMA foi o algoritmo não exato com o menor tempo de execução em todos os casos.

Tabela 13 – Quantidade de vértices total e de grau ímpar - Ilha do Governador.

Vértices	T_1 (h:min:seg)	Vértices ímpares	T_2 (h:min:seg)
3510	01:22:17	2928	31:39:38

A Tabela 13 ilustra a quantidade de vértices de um grafo G da Ilha do Governador, obtido pela biblioteca `Osmnx`. A coluna T_1 indica o tempo para a obtenção do grafo auxiliar G', cujos vértices são os vértices de grau ímpar de G e cujas arestas correspondem à distância mínima entre os respectivos vértices em G. A coluna Vértices ímpares indica a quantidade de vértices ímpares em G e, consequentemente, a quantidade de vértices em G'. A coluna T_2 ilustra o tempo de execução do algoritmo exato para obtenção do emparelhamento perfeito de custo mínimo do grafo G'. A quantidade de vértices de grau ímpar é aproximadamente 83% dos vértices do grafo.

Tabela 14 – Tempo de execução dos algoritmos não exatos no grafo da Ilha do Governador.

RAMA	PGA'	Maximal Matching	LAM	Vertex Scan	Greedy	Improved	$\frac{3}{4}$-emp.
00:15:47	01:02:30	00:30:55	00:16:54	00:29:32	00:26:53	00:25:17	00:37:43

A Tabela 14 ilustra o tempo de execução médio dos algoritmos não exatos para o grafo de grau ímpar da Ilha do Governador, detalhado na Tabela 14. O erro médio é inferior a 3% para todos os algoritmos não exatos. Observa-se que o algoritmo RAMA foi o algoritmo não exato com o menor tempo de execução.

Tabela 15 – Tempo de execução do CPP utilizando algoritmo exato.

Raio do mapa (metros)	T_1 (h:min:seg)	T_2 (h:min:seg)	T_3 (h:min:seg)	$Total$ (h:min:seg)
500	00:00:05	00:00:48	00:00:01	00:00:54
1.000	00:00:49	00:11:29	00:00:16	00:12:34
1.500	00:10:37	03:46:07	00:05:24	04:02:08
2.000	00:53:10	15:55:09	00:33:18	17:21:37
2.500	02:00:17	43:49:59	01:20:15	47:10:31

A Tabela 15 ilustra o tempo de execução do CPP para os grafos obtidos em função do raio do mapa máximo a partir de um determinado ponto da região no bairro da Taquara. A coluna T_1 indica o tempo para geração do grafo de graus ímpares. A coluna T_2 indica o tempo de execução do algoritmo exato. A coluna T_3 indica o tempo de execução para a duplicação das arestas entre o caminho mínimo de cada par de vértices de grau ímpar e a obtenção do circuito euleriano. A coluna Total é a soma das colunas T_1, T_2 e T_3.

Tabela 16 – Tempo de execução do CPP utilizando algoritmo exato - Ilha do Governador.

T_1 (h:min:seg)	T_2 (h:min:seg)	T_3 (h:min:seg)	$Total$ (h:min:seg)
01:22:17	31:39:38	01:02:21	34:04:16

A Tabela 16 ilustra o tempo de execução do CPP para um grafo da Ilha do Governador, obtido pela biblioteca `Osmnx`. A coluna T_1 indica o tempo para geração do grafo de graus ímpares. A coluna T_2 indica o tempo de execução do algoritmo exato. A coluna T_3 indica o tempo de execução para a duplicação das arestas entre o caminho mínimo de cada par de vértices de grau ímpar e a obtenção do circuito euleriano. A coluna Total é a soma das colunas T_1, T_2 e T_3.

A Tabela 17 ilustra o tempo de execução do CPP para os grafos obtidos em função do raio do mapa a partir de um determinado ponto da região no bairro da Taquara, utilizando-se algoritmos não exatos para a obtenção do emparelhamento perfeito de custo mínimo. Como esperado pelos resultados anteriores, o algoritmo RAMA foi o que obteve melhor desempenho quanto ao tempo de execução.

A Tabela 18 ilustra o tempo de execução do CPP para o grafo obtido da Ilha do Governador, utilizando-se algoritmos não exatos para a obtenção do emparelhamento perfeito de custo mínimo. O algoritmo RAMA foi o que obteve melhor desempenho quanto ao tempo de execução.

Para grafos acima de 1.500 metros de raio do mapa, ao gerar o emparelhamento perfeito de custo mínimo utilizando o algoritmo exato, ocorreu erro de limite de recursão e, portanto, foi necessário aumentar o limite de recursão do `Python` através do método `sys.setrecursionlimit`. Após, a execução foi normalizada.

Tabela 17 – Tempo de execução do CPP utilizando algoritmos não exatos.

Raio do mapa (metros) / Algoritmo	500	1000	1500	2000	2500
RAMA	00:00:07	00:01:12	00:16:44	01:30:52	03:38:33
PGA'	00:00:07	00:01:45	00:26:24	02:21:14	05:41:51
Maximal Matching	00:00:07	00:01:26	00:21:17	01:52:10	04:37:08
LAM	00:00:07	00:01:17	00:18:49	01:39:22	03:57:40
Vertex Scan	00:00:07	00:01:25	00:21:20	01:52:04	04:42:44
Greedy	00:00:07	00:01:16	00:18:53	01:40:01	04:07:07
Improved	00:00:11	00:02:11	00:21:27	01:41:54	04:00:34
$\frac{3}{4}$-emparelhamento	00:00:07	00:01:16	00:18:59	01:44:23	04:10:30

Tabela 18 – Tempo de execução do CPP utilizando algoritmos não exatos - Ilha do Governador.

RAMA	PGA'	Maximal Matching	LAM	Vertex Scan	Greedy	Improved	$\frac{3}{4}$-emp.
02:40:25	03:27:08	02:55:33	02:41:32	02:54:10	02:51:31	02:49:55	03:02:21

CONCLUSÃO

Nesta dissertação, trabalhamos com o Problema do Carteiro Chinês, que visa determinar um percurso passando por todos os trajetos entre pontos geográficos definidos, retornando ao ponto de partida no término, de forma a minimizar o custo do trajeto percorrido.

Trata-se de um problema de enorme interesse prático para empresas envolvidas na logística de entregas e distribuição de produtos, além de aplicação em outros problemas de natureza teórica.

Uma instância do CPP é normalmente modelada por um grafo, onde o conjunto de vértices representam tais pontos geográficos e as arestas as interconexões permitidas no percurso, ponderadas pelo custo associado à travessia entre aqueles pontos.

A solução clássica consiste em reduzir o problema à obtenção de um emparelhamento ponderado de custo máximo em um grafo auxiliar. O algoritmo de Edmonds, com complexidade de tempo $O(n^3)$, onde n representa o número de vértices desse grafo, é a referência básica em termos de uma solução exata.

Na parte inicial do trabalho procuramos apresentar uma abordagem bem compreensível de todo o fundamento teórico relacionado ao problema CPP, até chegarmos ao algoritmo de Edmonds.

Entretanto, o uso desse algoritmo básico tem se demonstrado muito custoso na prática em problemas de instâncias grandes [3] [4]. Além disso, a inerente dificuldade de implementação do algoritmo de Edmonds é um problema para sua consideração em implementações na resolução de problemas reais [3].

Motivados pelo fato de que o algoritmo exato é de difícil implementação e auditoria, em especial para o problema de emparelhamento ponderado de custo máximo, e que o CPP é importante para a indústria em inúmeras aplicações práticas onde soluções não exatas são admissíveis, especialmente se houver alguma garantia envolvida (como um fator de aproximação da solução ótima), a proposta deste trabalho foi estudar e discutir as principais abordagens não exatas existentes para o CPP e apresentar uma comparação experimental entre as mesmas, visando guiar seu uso na prática. As soluções alternativas são de várias naturezas, dentre as quais heurísticas, algoritmos aproximativos e algoritmos randomizados. Isto está desenvolvido na Seção 2.2, onde discutimos as principais soluções não exatas com complexidade de implementação menor que a dos algoritmos exatos. Abordamos o algoritmo não exato Greedy, guloso, com fator de aproximação $\frac{1}{2}$ e mostramos como construir o pior caso para este. Descrevemos também o algoritmo não exato Vertex Scan, que não garante qualquer fator de aproximação. Abordamos o algoritmo não exato LAM, que garante um fator de aproximação $\frac{1}{2}$, bem como o algoritmo não exato PGA, com mesmo fator de aproximação. Em seguida abordamos o algoritmo não exato Maximal Matching que não garante qualquer fator de aproximação mas que possui bom desempenho na prática segundo testes de comparação do autor. A seguir, descrevemos o algoritmo não exato RAMA, com fator de aproximação esperado de $\frac{2}{3}$. Trabalhamos também o algoritmo não exato Improve Matching que garante um fator de aproximação de $\frac{2}{3}$. Por fim, descrevemos também o algoritmo $\frac{3}{4}$-emparelhamento, com

fator de aproximação $\frac{3}{4}$.

Após a descrição das soluções aproximadas, realizamos experimentações onde comparamos os resultados obtidos pelos diversos algoritmos não exatos estudados com o algoritmo exato de Edmonds para quatro situações: grafos ponderados completos aleatórios com 1000 vértices, e não completos cuja probabilidade percentual de existência das arestas seja de 10%, 40% e 70%. A implementação do algoritmo de Edmonds foi obtida de [16], enquanto todos os algoritmos não exatos foram implementados no contexto desse trabalho. Caracterizamos, também, para cada solução aproximada, as entradas de pior caso e também comparamos essas instâncias de pior caso.

Considerando os resultados com grafos artificiais, verificamos que há soluções aproximadas muito boas, destacando-se no quesito tempo de execução o Algoritmo 11, RAMA, que obteve os melhores resultados em todos os tipos de grafos testados, cuja garantia teórica é fator de aproximação esperado de $\frac{2}{3}$. O algoritmo não exato com melhor garantia até o momento é o Algoritmo 15, $\frac{3}{4}$-emparelhamento, com fator de aproximação garantido de $\frac{3}{4}$, dentro os algoritmos de mais fácil implementação.

Em seguida, realizamos experimentações com grafos baseados em dados reais obtidos do Open Street Maps. Tais grafos correspondem a representação de ruas de porções de mapa centradas em um determinado ponto do bairro Taquara, no Rio de Janeiro, e considerando uma variedade de raios no entorno (500, 1000, 1500, 2000 e 2500 metros). Além destes, também foi experimentado um grafo correspondente a região da Ilha do Governador, do mesmo município. Para estes grafos foi mensurado o tempo de geração do grafo auxiliar a partir dos vértices ímpares, o tempo de execução do algoritmo exato e dos algoritmos não exatos. Após trabalhamos o CPP completo e tabelamos o tempo decorrido para cada grafo nos diversos algoritmos estudados.

Considerando o resultado com grafos baseados em dados reais, percebemos que o resultado se aproxima bastante do resultado com grafos artificiais, o que indica que o modelos artificiais funcionaram bem. Verificamos que há soluções aproximadas muito boas inclusive em casos reais, destacando-se novamente no quesito tempo de execução o Algoritmo 11, RAMA, que obteve os melhores resultados em todos os grafos.

Em geral o algoritmo com melhor resultado no quesito tempo de execução, tanto para grafos artificiais como reais, foi o Algoritmo 11, RAMA, com fator de aproximação esperado de $\frac{2}{3}$, seguido do Algoritmo 8, LAM, com fator de aproximação garantido de $\frac{1}{2}$. Os resultados de ambos os algoritmos foram bastantes próximos em questão de tempo de execução bem como nos erros médios obtidos, sendo que o segundo garante um emparelhamento com pelo menos 50% do custo do emparelhamento ótimo. As implementações realizadas dos algoritmos mencionados estão disponíveis em [35].

Como trabalhos futuros, sugerimos o estudo de meta-heurísticas tais como o algoritmo genético e o recozimento simulado, comparando as dificuldades percebidas e a aproximação do resultado obtido com o ótimo. Também seriam interessantes estudos utilizando aprendizado profundo. O estudo presente voltou-se, nos casos reais, a áreas urbanas de uma metrópole. Áreas urbanas de cidades de menor porte e áreas rurais podem também ser alvo de estudos posteriores. O Problema do Carteiro Rural e o Problema do Carteiro Chinês com Vento são variantes interessantes que merecem novos estudos com foco prático.

REFERÊNCIAS

[1] SEDGEWICK, R. *Algorithms in C++ Part 5: Graph Algorithms*. 3. ed. [S.l.]: Addison Wesley, 2002. v. 5. ISBN 978-0201361186.

[2] SZWARCFITER, J. L. *Teoria Computacional de Grafos - Os Algoritmos*. 1.a. ed. Rio de Janeiro, Brasil: Elsevier, 2018.

[3] AVIS, D. A survey of heuristics for the weighted matching problem. *Networks*, Wiley Online Library, v. 13, n. 4, p. 475–493, 1983.

[4] UEHARA, R.; CHEN, Z.-Z. Parallel approximation algorithms for maximum weighted matching in general graphs. In: SPRINGER. *IFIP International Conference on Theoretical Computer Science*. [S.l.], 2000. p. 84–98.

[5] HOARE, T. et al. The verified software initiative: A manifesto. In: *Theories of Programming: The Life and Works of Tony Hoare*. [S.l.: s.n.], 2021. p. 81–92.

[6] ARVANITOU, E.-M. et al. Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 172, p. 110848, 2021.

[7] IBGE. *Manual do Recenseador*. 1st. ed. [S.l.]: IBGE, 2022. Https://censo2022.ibge.gov.br/sobre/treinamento/manuais.html.

[8] WEST, D. B. *Introduction to Graph Theory*. 2.a. ed. Urbana, Estados Unidos: Pearson Education, 2001.

[9] BONDY, U. M. A. *Graph Theory*. 1. ed. London, England: Springer London, 2008.

[10] DROR E. BENAVENT, D. M. *Arc routing: theory, solutions, and applications / edited by Moshe Dror*. 1. ed. Nova York, Estado Unidos: Springer, 2000.

[11] PLUMMER, L. L. e M. *Matching Theory*. 29. ed. Amsterdam, Holanda: North-Holland, 1986.

[12] GIBBONS, A. *Algorithmic graph theory*. [S.l.]: Cambridge University Press, 1985.

[13] KYI, M. T.; NAING, L. L. Application of ford-fulkerson algorithm to maximum flow in water distribution pipeline network. *International Journal of Scientific and Research Publications*, v. 8, n. 12, p. 306–310, 2018.

[14] FIGUEIREDO, C. d.; SZWARCFITER, J. Emparelhamento em grafos, algoritmos e complexidade. *Instituto de Matemática, UFRJ*, 1999.

[15] RANTWIJK, J. van. *Maximum Weighted Matching*. 2013. Disponível em: <http://jorisvr.nl/article/maximum-matching>.

[16] GALIL, Z. Efficient algorithms for finding maximum matching in graphs. Association for Computing Machinery, New York, Estados Unidos, v. 18, n. 1, p. 23–38, 1986. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/6462.6502>.

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

[18] 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*. [S.l.: s.n.], 1990. p. 434–443.

[19] GABOW, H. N. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms (TALG)*, ACM New York, NY, USA, v. 14, n. 3, p. 1–80, 2018.

[20] DRAKE, D. E.; HOUGARDY, S. Improved linear time approximation algorithms for weighted matchings. In: *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*. [S.l.]: Springer, 2003. p. 14–23.

[21] EDMONDS, J.; JOHNSON, E. L. Matching euler tours and the chinese postman. *Mathematical Programming 5 - IBM Watson Research Center*, p. 88–124, 1973.

[22] SHERAFAT, H. Uma heurística para o problema do carteiro chinês misto. In: Associação Brasileira de Engenharia e Ciências Mecânicas – ABCM. Ilhéus/BA, Brasil, 2013.

[23] BEVERN, R. van et al. Chapter 2: The complexity of arc routing problems. In: *Arc routing: Problems, methods, and applications*. [S.l.]: SIAM, 2015. p. 19–52.

[24] FILHO, M. G.; JUNQUEIRA, R. de Ávila R. Um algoritmo para auxiliar na escolha de métodos de solução para o problema do carteiro chinês: proposta e aplicação em uma grande cidade do interior paulista. *XII SIMPEP – Bauru, SP, Brasil*, p. 88–124, 2005.

[25] SöKMEN, D. Özlem Çomakli. An overview of chinese postman problem. In: 3^{rd} International Conference on Advanced Engeneering Technologies. Erzurum, Turquia, 2019.

[26] VINKEMEIER, D. E. D.; HOUGARDY, S. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms (TALG)*, ACM New York, NY, USA, v. 1, n. 1, p. 107–122, 2005.

[27] KURTZBERG, J. M. On approximation methods for the assignment problem. *Journal of the ACM (JACM)*, ACM New York, Estados Unidos, v. 9, n. 4, p. 419–439, 1962.

[28] PREIS, R. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In: SPRINGER. *Annual Symposium on Theoretical Aspects of Computer Science*. [S.l.], 1999. p. 259–269.

[29] PREIS, R. Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching in general graphs. In: SPRINGER. *Annual Symposium on Theoretical Aspects of Computer Science*. [S.l.], 1999. p. 259–269.

[30] DRAKE, D. E.; HOUGARDY, S. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, Elsevier, v. 85, n. 4, p. 211–213, 2003.

[31] DRAKE, D. E.; HOUGARDY, S. Linear time local improvements for weighted matchings in graphs. In: SPRINGER. *International Workshop on Experimental and Efficient Algorithms*. [S.l.], 2003. p. 107–119.

[32] PETTIE, S.; SANDERS, P. A simpler linear time 2/3- ε approximation for maximum weight matching. *Information processing letters*, Elsevier, v. 91, n. 6, p. 271–276, 2004.

[33] HANKE, S.; HOUGARDY, S. *New approximation algorithms for the weighted matching problem*. [S.l.]: Citeseer, 2010.

[34] DUAN, R.; PETTIE, S. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, ACM New York, Estados Unidos, v. 61, n. 1, p. 1–23, 2014.

[35] PORTELLA, L. C. *Emparelhamentos aplicados ao Problema do Carteiro Chinês Ponderado: uma análise de algoritmos exatos e não exatos*. [S.l.]: GitHub, 2023. <https://github.com/LeonardoPortella/Emparelhamentos>.