

UNIVERSIDADE FEDERAL DO ABC
CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO
PROJETO DE GRADUAÇÃO EM COMPUTAÇÃO III

Sobre k -fatores em grafos

Agosto/2026

Resumo

O problema do k -fator consiste em identificar, em um grafo G , um subgrafo gerador k -regular, caso ele exista. Neste projeto de graduação, foram explorados especificamente os k -fatores simples. Esse subconjunto de k -fatores não apenas apresenta uma formulação mais adequada para determinados problemas, como também se mostra mais apropriado para diversas aplicações, tanto no âmbito da Teoria dos Grafos quanto em áreas práticas, como a Computação Gráfica. Dessa forma, a computação eficiente dessa estrutura revela-se essencial. O objetivo principal deste trabalho foi entender melhor essa estrutura, através da implementação na linguagem de programação C++, o algoritmo proposto por Meijer, Núñez-Rodríguez e Rappaport [7], que computa k -fatores simples de maneira eficiente.

1 Introdução

Grafos são estruturas matemáticas que representam relações par a par entre elementos, amplamente utilizadas para modelar problemas de diversas áreas, como Ciência da Computação, Engenharia, Biologia, Logística e Sociologia, por exemplo. Isso porque permitem representar redes de transporte, relações entre pessoas, conexões em circuitos, mapeamento de relações complexas entre sequências genéticas de maneira eficiente, dentre outras. De forma simples, um grafo é composto por vértices que representam os elementos e arestas, que conectam pares de vértices, modelando uma relação entre eles.

Dentro do estudo da Teoria dos Grafos, a investigação de propriedades estruturais é frequentemente motivada pela busca de algoritmos eficientes, ou seja, aqueles em que o tempo de execução é limitado por uma função polinomial do tamanho do grafo. No estudo da Ciência da Computação, problemas que admitem tais soluções compoem a classe P. Contudo, diversos problemas estruturais permanecem até o momento computacionalmente intratáveis, significando que não se conhece uma maneira de encontrar soluções polinomiais para todas as suas instâncias através de um algoritmo. Muitos destes problemas situam-se na classe NP, que abrange problemas para os quais, apesar de não conhecermos algoritmos rápidos para solucionar, conseguimos verificar a validade de uma resposta de maneira eficiente (chamado pela literatura de certificado). Pertencem a essa classe os chamados problemas NP-completos, que são os problemas mais difíceis em NP. Eles possuem a propriedade de que, caso um algoritmo polinomial seja encontrado para um deles, todos os outros problemas NP-completo poderão ser resolvidos eficientemente. Por fim, o termo NP-difícil descreve problemas em que sabemos que são pelo menos tão difíceis quanto os problemas NP-completos, mas que não necessariamente possuem a característica de verificação rápida, sendo frequentemente presentes em contextos de otimização.

A aplicação dessas definições aos problemas estudados na Teoria dos Grafos revela dois

tipos distintos de problemas. Aqueles em que suas propriedades estruturais facilitam a computação e aqueles que ainda desafiam a eficiência algorítmica. Exemplos proeminentes de problemas NP-completos incluem o **Problema do Ciclo Hamiltoniano**, que busca determinar a existência de um ciclo que visite cada vértice de um grafo exatamente uma vez. Já o **Problema do Caixeiro Viajante (TSP)** é uma extensão de otimização NP-difícil que propõe encontrar um ciclo hamiltoniano de custo mínimo em um grafo ponderado; ambos carecem de algoritmos eficientes conhecidos e são fundamentais para o estudo de complexidade. Em contrapartida, o estudo estrutural permite identificar problemas que, apesar de sua aparência complexa, possuem solução algorítmica robusta pertencente à classe P, Esse é o caso do **Problema do Emparelhamento Máximo**, que propõe identificar o maior conjunto possível de pares conectados, de modo que nenhum elemento possa fazer par com mais de um elemento nesse conjunto.

Embora problemas variem drasticamente em dificuldade computacional, os algoritmos que propoem solucioná-los frequentemente compartilham e utilizam estruturas e conceitos fundamentais. Tais estruturas, como emparelhamentos, ciclos ou subgrafos com propriedades específicas, atuam como ferramentas essenciais na análise de problemas complexos, implementados como etapas intermediárias ou até como condições necessárias para esses problemas. Por exemplo, a solução do Problema do Emparelhamento Máximo atua como o componente central na resolução de diversos desafios estruturais mais abrangentes. Muitos problemas robustos, como a busca por subgrafos que incluam todos os vértices da rede sob restrições específicas de vizinhança em cada ponto, podem ter sua resolução transposta para o domínio dos emparelhamentos simples. De forma semelhante, a identificação de conjuntos de ciclos que percorrem todos os elementos de um grafo, etapa primordial em algoritmos que buscam soluções aproximadas para instâncias do *TSP* como aqueles propostos por Papadimitriou e Yannakakis [9], fundamentam-se na capacidade de processar essas estruturas básicas de forma eficiente.

É neste contexto que surgem os chamados k -fatores. Um k -fator em um grafo é um subgrafo que inclui todos os vértices do grafo original (chamado de subgrafo gerador) de modo que a cada vértice deste subgrafo incide exatamente k arestas. Essa propriedade de regularidade, onde todos os vértices possuem o mesmo grau, é justamente o que torna k -fatores tão relevantes para resolver problemas estruturais complexos. Vale notar que emparelhamentos, mencionados anteriormente como ferramentas essenciais, são um caso especial de k -fatores onde $k = 1$, isto é, cada vértice conecta-se a exatamente um outro vértice.

A teoria dos k -fatores foi profundamente desenvolvida por diversos autores, como Tutte [13], que estabeleceu condições necessárias e suficientes para a existência de k -fatores em grafos localmente finitos. Esses resultados forneceram uma base sólida para o desenvolvimento de algoritmos e aplicações envolvendo tais estruturas. Além disso, Mulder [8], em sua análise do trabalho de Petersen [10], aponta-o como sendo o primeiro artigo, na história da mate-

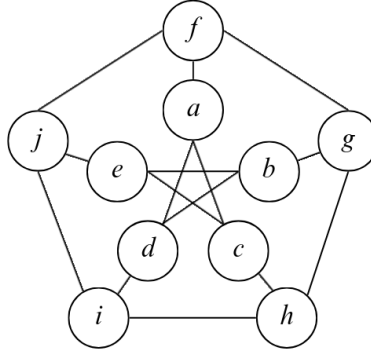


Figura 1: Grafo de Petersen, cujo conjunto de arestas $\{af, bg, ch, di, ej\}$ ilustra a formação de um 1-fator, enquanto os conjuntos de arestas $\{fg, gh, hi, ij, jf\}$ e $\{ad, db, be, ec, ca\}$ compõem um 2-fator.

mática, a conter resultados fundamentais explicitamente na Teoria dos Grafos. A Figura 1 ilustra o que é chamado *Grafo de Petersen*, que se tornou um dos exemplos mais importantes da Teoria dos Grafos devido às suas diversas propriedades estruturais e ao seu papel como contraexemplo em múltiplos resultados. Neste trabalho, Petersen introduziu o conceito de fatores em um grafo e forneceu condições suficientes e necessárias para a existência dessa estrutura, bem como algumas propriedades importantes.

Trabalhos mais recentes como o de Eppstein e Goppi [6] utilizam k -fatores no campo da computação gráfica, onde sua estrutura auxilia na tarefa de renderização, permitindo uma organização eficiente dos dados geométricos por meio da *stripification* (técnica que organiza polígonos em uma malha contínua) e compressão de grandes conjuntos de triângulos, o que é essencial para alcançar desempenho otimizado em renderizações rápidas. O mesmo procedimento pode ser utilizado para tetraedros e quadriláteros [4]. Isso evidencia a relevância e extensão da aplicação dessa área de estudo em diferentes tópicos, ao longo dos anos.

Desta forma, o principal objetivo deste projeto foi compreender melhor a teoria existente acerca dos k -fatores, a partir da implementação de um algoritmo capaz de identificá-los em grafos caso eles existam, na linguagem de programação C++. É válido ressaltar ainda que a estrutura de k -fatores e os algoritmos auxiliares utilizados para encontrá-los não foram compreendidos nas disciplinas da grade curricular da graduação, apresentando um desafio considerável para este projeto. O escopo de análise foi reduzido para os chamados k -fatores simples, também chamados de não degenerados, visto que apresentam uma estrutura mais apropriada e útil na resolução de diversos problemas computacionais.

Para alcançar esse objetivo, utilizou-se uma técnica baseada na construção de *gadgets*, estruturas que facilitam a identificação em tempo polinomial dos k -fatores em grafos, empregando um algoritmo auxiliar de emparelhamento para esse fim. A abordagem seguida foi fundamentada principalmente no artigo *An algorithm for computing simple k -factors* de Meijer, Núñez-Rodríguez e Rappaport [7] que serve como base para este trabalho. Na Seção 2

são apresentados os conceitos essenciais sobre grafos, imprescindíveis para compreender os termos utilizados ao longo deste trabalho. A Seção 3 dedica-se ao problema do emparelhamento em grafos e suas propriedades fundamentais. A compreensão dessa estrutura é essencial, pois ela compõe a base teórica do método proposto por Meijer *et al.* [7]. Além dos conceitos fundamentais, detalhamos o Algoritmo de Edmonds [5], escolhido no presente trabalho como a ferramenta computacional para solucionar a etapa de emparelhamento exigida por essa abordagem. Analogamente, na Seção 4 são apresentados o problema do k -fator e as estratégias utilizadas para tentar solucioná-lo. A Seção 5 apresenta o algoritmo proposto no artigo e as estratégias utilizadas para reproduzi-lo em C++. Por fim, a Seção 6 retoma os principais resultados obtidos, discute os desafios encontrados durante o desenvolvimento da implementação e aponta limitações do trabalho.

2 Conceitos básicos

A seguir serão apresentados os conceitos e definições fundamentais sobre as principais estruturas abordadas neste trabalho, a partir do livro *Graph Theory* de Bondy e Murty [3].

Um **grafo (simples)** é definido por um par (V, E) , onde V é um conjunto finito de elementos, chamados **vértices**, e E é um conjunto de pares não ordenados dos elementos de V , chamados **arestas**. Para um grafo G qualquer, usaremos $V(G)$ para denotar o conjunto de vértices e $E(G)$ para denotar o conjunto de arestas. Dessa forma, não há necessidade de definir o par ordenado para todo grafo. O número total de elementos de $V(G)$ é denotado por $v(G)$, isto é, $v(G) = |V(G)|$, esse valor representa a **ordem** de G . De forma análoga, denota-se $e(G) = |E(G)|$, a quantidade de arestas de G , também chamado de **tamanho** de G .

Denotamos a aresta formada pelo par $u, v \in V(G)$ por uv (ou vu), representando o par não ordenado $\{u, v\} \in E(G)$. Os vértices u e v são **extremos** da aresta uv , ela **incide** em u e em v , e os vértices u e v são **vizinhos** ou **adjacentes**. Denotamos por $N_G(u)$ a **vizinhança** do vértice u , que é definida pelo conjunto de vizinhos de u no grafo G . Duas arestas distintas $e_1, e_2 \in E(G)$ são chamadas vizinhas ou adjacentes se incidirem sobre um mesmo vértice. A Figura 2 apresenta um exemplo de grafo simples.

Para um vértice $v \in V(G)$, o número de arestas de G que incidem em v é chamado de **grau** de v , denotado por $d_G(v)$ (Figura 3). O maior grau dentre todos os vértices de G é chamado de **grau máximo** e denotado $\Delta(G)$ e, de maneira equivalente, o menor grau dentre todos os vértices é denominado **grau mínimo**, denotado por $\delta(G)$.

Chamamos de **passeio** em G uma sequência de vértices $W = (v_0, v_1, \dots, v_k)$ tal que $v_i v_{i+1} \in E(G)$, para $0 \leq i < k$. Chamamos v_0 de **início** e v_k de **término** de W , estes vértices são denominados **extremos** de W , enquanto os vértices $v_1, \dots, v_{k-1}$ são **vértices internos** de W . Um passeio W é dito **fechado** se $v_0 = v_k$. Se W não repete arestas, chamamos W de

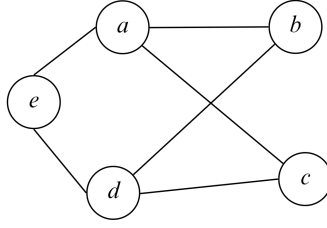


Figura 2: Exemplo de grafo G , onde o conjunto de vértices é $V(G) = \{a, b, c, d, e\}$ e o conjunto de arestas é $E(G) = \{ab, ac, ae, bd, de\}$. Note que a ordem do grafo é $v(G) = 5$ e o tamanho é $e(G) = 6$. O vértice a é vizinho de c e as arestas ac e ab são adjacentes.

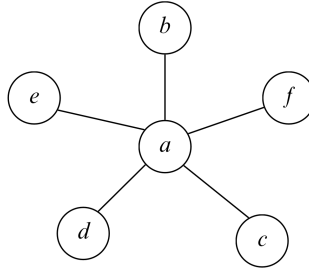


Figura 3: Exemplo de um grafo estrela G , onde o grau do vértice a , o grau máximo e o grau mínimo de G é, respectivamente $d_G(a) = 5$, $\Delta(G) = 5$ e $\delta(G) = 1$.

trilha. Se W é um passeio fechado que não repete arestas, chamamos W de **trilha fechada**. Se W não repete vértices, chamamos W de **caminho**. Se W é um passeio fechado com pelo menos três arestas e que não repete vértices internos, chamamos W de **ciclo**. A Figura 4 exemplifica esses conceitos.

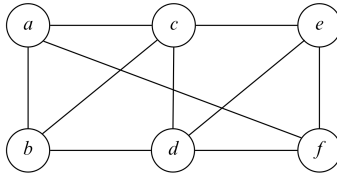


Figura 4: No grafo da figura temos o passeio (a, c, d, e) ; o passeio fechado (a, c, d, c, a) ; a trilha (a, c, d, e, c, b) ; a trilha fechada (c, b, d, f, e, d, c) ; o caminho (a, c, d, b) e o ciclo (a, c, b, a) , por exemplo.

Dados dois grafos G e H , dizemos que H é **subgrafo** de G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$, e denotamos essa relação por $H \subseteq G$. Dizemos também que G é **supergrafo** de H , pois G **contém** H . Se $V(H) = V(G)$, dizemos ainda que H é um **subgrafo gerador** de G (Figura 5). Passeios em G também podem ser considerados como subgrafos de G . Um passeio $W = (v_0, \dots, v_k)$ pode ser considerado como o subgrafo em que $V(W) = \{v_0, \dots, v_k\}$ e $E(W) = \{v_0v_1, \dots, v_{k-1}v_k\}$. Dado um conjunto de vértices $S \subseteq G$ de um grafo G , denotamos por $G[S]$ o **subgrafo induzido** de G , que é o subgrafo H de G , de modo que $V(H) = S$ e

$E(H)$ é o conjunto de arestas de G que possuem ambas as extremidades em S . Analogamente, se F é um subconjunto de arestas de G , então o subgrafo induzido de G por F , denotado $G[F]$ é o subgrafo H de G , de modo que $E(H) = F$ e $V(H)$ é o conjunto de vértices de G que são extremidades de alguma aresta em F .

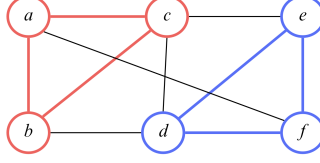


Figura 5: No grafo, temos os subgrafos H_1 , com $E(H_1) = \{ab, bc, ca\}$ e H_2 , com $E(H_2) = \{ed, df, fe\}$. Além disso, H_1 e H_2 formam um grafo gerador H_g de G , com arestas $E(H_g) = \{ab, bc, ca, ed, df, fe\}$.

É comum realizar operações que modificam grafos, uma forma de gerar um subgrafo de G é removendo elementos de G . Dado $S \subseteq V(G)$, denotamos $G - S$ o grafo gerado ao remover S de G . Formalmente, $G - S = G[V(G) \setminus S]$, o subgrafo induzido pelos vértices de G após a remoção de S . De forma análoga, é possível remover arestas para gerar um novo grafo, dado $F \subseteq E(G)$, o grafo gerado ao remover F de G é denotado $G - F$. Formalmente, $G - F = G[E(G) \setminus F]$, o subgrafo induzido pelas arestas após a remoção. $G - F$ é tal que $V(G - F) = V(G)$ e $E(G - F) = E(G) \setminus F$. Analogamente, uma forma de gerar supergrafos de G é adicionar elementos a G . Dados $S \not\subseteq V(G)$ e $F \not\subseteq E(G)$, cujos elementos são pares de elementos de $V(G)$, denotamos por $G + S$ o grafo gerado a partir da adição de S a G e $G + F$ o grafo gerado pela adição de F a G . Formalmente temos que $G + S$ é tal que $V(G + S) = V(G) \cup S$ e $E(G + S) = E(G)$. Analogamente, $G + F$ é tal que $V(G + F) = V(G)$ e $E(G + F) = E(G) \cup F$. É usual utilizar um abuso de notação para quando S e F consistirem de apenas um elemento. Para subgrafos, utilizamos $G - v = G - \{v\}$ para $v \in V(G)$ e $G - e = G - \{e\}$ para $e \in E(G)$. Analogamente, para supergrafos, utilizamos $G + v = G + \{v\}$ para $v \notin V(G)$ e $G + e = G + \{e\}$ para $e \notin E(G)$, com $e = xy$ e $x, y \in V(G)$.

Um grafo G é dito **conexo** se existir uv -caminho para todo par de vértices $u, v \in V(G)$. Caso contrário, dizemos que G é **desconexo**. Uma **componente conexa** de um grafo G é um subgrafo conexo induzido por um conjunto de vértices S , tal que não existe arestas entre S e $V(G) \setminus S$ em G . Desta forma, um grafo é dito conexo se possuir apenas uma componente conexa, e desconexo caso possuir duas ou mais componentes conexas, como apresentado na Figura 6. São chamadas de **componentes ímpares** as componentes conexas que possuem um número ímpar de vértices. O número de componentes ímpares de um grafo é denotado por $o(G)$.

Um grafo G é chamado **regular** se, para algum inteiro k , $d_G(v) = k$ para todo vértice $v \in V(G)$. Quando queremos enfatizar o valor de k , denotamos esse grafo como k -regular. Um **grafo completo** com n vértices, denotado por K_n , é um grafo no qual cada vértice se conecta

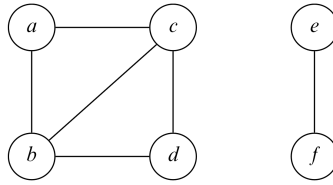


Figura 6: Grafo com duas componentes conexas, formadas pelos conjuntos de vértices $\{a, b, c, d\}$ e $\{e, f\}$.

a todos os outros vértices. Isto significa que todo vértice u está conectado a $V(G) \setminus \{u\}$. Consequentemente, um grafo completo com n vértices é um grafo $(n-1)$ -regular. A Figura 7 apresenta um grafo regular e um grafo completo, respectivamente.

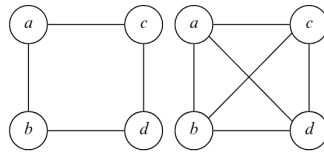


Figura 7: Exemplos de um grafo 2-regular à esquerda e um grafo completo K_4 à direita.

Um grafo G é **bipartido** se conseguimos particionar $V(G)$ em dois conjuntos independentes X e Y , de modo que toda aresta $uv \in E(G)$ possui suas extremidades $u \in X$ e $v \in Y$. Formalmente, G é bipartido se existem conjuntos X e Y de vértices, de modo que $X \cup Y = V(G)$ e $X \cap Y = \emptyset$. Dizemos também que G é (X, Y) -bipartido para os conjuntos de bipartição X e Y . A Figura 8 ilustra um exemplo de grafo bipartido.

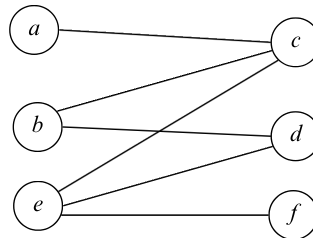


Figura 8: Grafo bipartido com os conjuntos de vértices $X = \{a, b, e\}$ e $Y = \{c, d, f\}$.

3 Emparelhamento

Um **emparelhamento** M em um grafo G é um subconjunto de arestas que são **não adjacentes** duas a duas em G , como exemplificado pela Figura 9. Essa estrutura é amplamente utilizada para modelar problemas como designação de tarefas, atribuição de candidatos a vagas de emprego, sistemas de recomendação, análises de imagens e problemas de visão computacional, dentre outros.

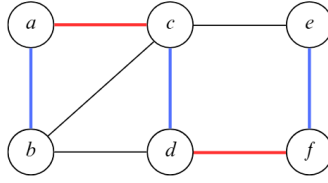


Figura 9: Exemplo de dois emparelhamentos $M_1 = \{ac, df\}$ e $M_2 = \{ab, cd, ef\}$.

Dizemos que as arestas pertencentes ao emparelhamento estão **emparelhadas** por M . Um vértice $v \in V(G)$ é dito **coberto** ou **saturado** por M se v é incidente a alguma aresta de M . Consequentemente, como as arestas de um emparelhamento são não adjacentes, todo vértice coberto por M incide a exatamente uma aresta em M . Um vértice que não é coberto por M é chamado de **M -insaturado**, ou **livre**.

Um dos principais problemas no estudo de emparelhamentos é o de encontrar um **emparelhamento máximo**. Um emparelhamento M^* é chamado de **máximo** quando possui o maior número possível de arestas de G . Um emparelhamento é chamado **maximal** quando a ele não é possível acrescentar uma aresta a M sem que isso viole a própria definição de emparelhamento. Denotamos o tamanho de um emparelhamento máximo como $\alpha'(G)$. Desse modo, um emparelhamento máximo é também maximal na propriedade de inclusão de arestas. O Problema 1 define formalmente o problema de encontrar um emparelhamento máximo.

Problema 1 (Emparelhamento máximo).

***Entrada:** $\langle G \rangle$, onde G é um grafo qualquer.*

***Saída:** Um emparelhamento que possua o maior número possível de arestas.*

Os principais conceitos que possibilitam encontrar um emparelhamento máximo são os de caminhos M -alternantes e caminhos M -aumentador, utilizados diretamente no algoritmo que encontra emparelhamentos máximos em grafos. Denomina-se **caminho M -alternante** um caminho no grafo G em que as arestas pertencem **alternadamente** a M e a $E(G) \setminus M$. Na Figura 9 temos o caminho M -alternante $P_1 = (ba, ac, cd, df, fe)$, considerando o emparelhamento M_1 e $P_2 = (ce, ef, fd)$ se considerarmos o emparelhamento M_2 , por exemplo.

Quando um caminho M -alternante possui suas extremidades com vértices não cobertos por M , denominamos esse caminho por **caminho M -aumentador** P . A relevância desse caminho reside em sua estrutura, visto que por sua construção, obrigatoriamente ele começa e termina com arestas pertencentes a $E(G) \setminus M$. Dessa forma, um caminho M -aumentador P possui um número ímpar de arestas, apresentando exatamente uma aresta desemparelhada a mais do que emparelhadas. Essa propriedade permite ampliar o emparelhamento atual por meio da operação de diferença simétrica, que representa o conjunto de arestas que pertencem apenas a um ou outro grafo, mas não a ambos. Formalmente, definimos **diferença simétrica**

de dois conjuntos E_1 e E_2 como $E_1 \Delta E_2 = (E_1 \cup E_2) / (E_1 \cap E_2)$. Dessa forma, conforme exemplificado na Figura 10, podemos aplicar esta operação para os conjuntos M e $E(P)$ e inverter o estado das arestas ao longo de P , trocando arestas de M por arestas de $E(G) \setminus M$ e vice-versa, produzindo um novo emparelhamento M' , tal que $|M'| = |M| + 1$.

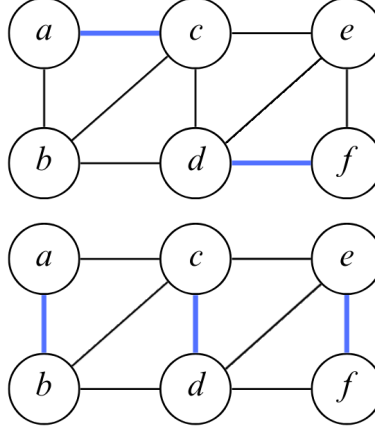


Figura 10: Na parte superior apresentamos um emparelhamento M em azul. Na parte inferior apresentamos um novo emparelhamento, criado a partir da diferença simétrica entre M e o caminho M -aumentador $P = (ba, ac, cd, df, fe)$.

O Teorema 1 estabelece uma condição suficiente e necessária para que um emparelhamento seja máximo, relacionando-o à inexistência de um caminho M -aumentador.

Teorema 1 (Teorema de Berge). *Um emparelhamento M em um grafo G é máximo se, e somente se, G não possuir um caminho M -aumentador.*

Utilizando-o, é simples encontrar um emparelhamento máximo pois é suficiente, partindo de um emparelhamento arbitrário, procurar iterativamente caminhos M -aumentadores e aplicar sobre eles a operação de diferença simétrica, até que não seja mais possível encontrar um caminho M -aumentador.

Um outro tipo de emparelhamento importante é o emparelhamento perfeito. Um emparelhamento M é chamado de **perfeito** se saturar todos os vértices de G . Um grafo é dito **emparelhável** se possuir um emparelhamento perfeito. A Figura 9 apresenta o emparelhamento perfeito $M = \{ab, cd, ef\}$. Como cada aresta do emparelhamento conecta exatamente dois vértices, uma condição necessária para que G admita um emparelhamento perfeito é que seu número de vértices seja par. Assim, se um grafo possui uma quantidade ímpar de vértices, conclui-se imediatamente que ele não é emparelhável. Todo emparelhamento perfeito é também um emparelhamento máximo.

Se M é um emparelhamento em G , pela construção de M cada componente ímpar de G deve possuir pelo menos um vértice que não está coberto por M , visto que um emparelhamento agrupa pares de vértices. Portanto, para um grafo arbitrário, o número total

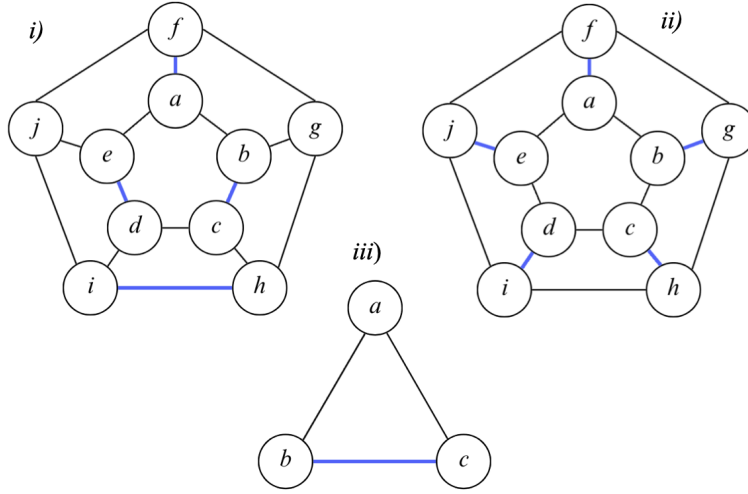


Figura 11: Exemplos dos tipos de emparelhamento. i) Emparelhamento Maximal; ii) Emparelhamento Perfeito; iii) Emparelhamento Máximo.

de vértices não cobertos por um emparelhamento deverá ser sempre, no mínimo, o número de componentes ímpares em G , denotado $o(G)$. Em outras palavras, se U é o conjunto de vértices não emparelhados das componentes ímpares de G , temos que $|U| \geq o(G)$. Essa inequação pode ser estendida para quaisquer subgrafos induzidos de G . Seja $S \subseteq V(G)$ um subconjunto de vértices e M um emparelhamento em G . Considere uma componente ímpar H de $G - S$, ou seja, obtida ao remover S de G . Se todo vértice de H for coberto por M , então temos que pelo menos um vértice de H tem que estar emparelhado com um vértice de S , pois não é possível emparelhar com um vértice interno de H . Como há apenas $|S|$ vértices disponíveis para emparelhar vértices sobrando de $G - S$ a S , temos pelo menos $o(G - S) - |S|$ componentes ímpares de G que possuem vértices não cobertos por M . Essa observação leva à Inequação (1):

$$|U| \geq o(G - S) - |S|. \quad (1)$$

É possível observar isso facilmente no grafo estrela da Figura 3. Tome $S = \{a\}$. Com isso, o subgrafo $G - S$ se constitui de 5 vértices isolados $\{b, c, d, e, f\}$, onde cada um é também uma componente ímpar. Utilizando a Inequação (1), temos $|U| = o(G - S) - |S| = 5 - 1 = 4$, indicando que, nesse grafo, sempre haverá 4 vértices não cobertos por M . Note que, como as arestas de M cobrem exatamente 2 vértices, $|U| = v(G) - 2|M|$ representa o número de vértices em G que não são cobertos por M . Sendo assim, caso tenhamos um subconjunto B que forneça a igualdade na Inequação (1), ou seja $|U| = o(G - B) - |B| = v(G) - 2|M|$, então o conjunto B certifica que M não cobriu o menor número possível de vértices, o que o torna um emparelhamento máximo. Sendo assim, é possível utilizar B como um certificado de otimalidade de M . Esse conjunto B é chamado de **barreira** de G . Um grafo que possui

um emparelhamento perfeito possui $B = \emptyset$ ou B composto por vértices isolados.

O Teorema 2 sintetiza os resultados fundamentais de Tutte [12] e Berge [2], garantindo a existência dessa estrutura certificadora para qualquer grafo arbitrário.

Teorema 2 (Teorema de Tutte-Berge). *Qualquer grafo G possui ao menos uma barreira $B \subseteq V(G)$ que certifica a otimalidade de seus emparelhamentos máximos.*

Com isso, segue o Teorema 3 que apresenta uma condição suficiente e necessária para a existência de um emparelhamento perfeito em um grafo arbitrário.

Teorema 3 (Teorema de Tutte). *Um grafo G possui um emparelhamento perfeito se e somente se $o(G - S) \leq |S|$, para todo $S \subseteq V(G)$.*

Demonstração. Suponha que G possui um emparelhamento perfeito M . Vamos mostrar que $o(G - S) \leq |S|$ para todo $S \subseteq V(G)$. Seja $S \subseteq V(G)$ qualquer conjunto de vértices de G e U o conjunto de vértices de G que não são cobertos por M , logo $U = \emptyset$ e $|U| = 0$. Portanto, utilizando a Inequação (1), temos que $o(G - S) \leq |S|$.

Suponha agora que $o(G - S) \leq |S|$ para todo $S \subseteq V(G)$, vamos mostrar que G tem um emparelhamento perfeito. Para fins de contradição, suponha que G não tem um emparelhamento perfeito. Seja um emparelhamento máximo M^* de G , com U o conjunto de vértices de G não cobertos por M^* . Pelo Teorema 2, temos que todo grafo possui uma barreira, ou seja, um subconjunto $B \subseteq V(G)$, em que $o(G - B) - |B| = |U|$. Como M^* não é perfeito, então existem vértices de G que não estão emparelhados, logo $|U| \geq 1$, portanto $o(G - B) = |U| + |B| \geq |B| + 1$, contradizendo a suposição para o subconjunto $S := B$. Sendo assim, temos que se a condição for respeitada para todo subconjunto S de $V(G)$, G possui um emparelhamento perfeito. $\square$

3.1 O Algoritmo de Edmonds

Um dos principais algoritmos para encontrar emparelhamentos máximos em grafos gerais é o proposto por Edmonds [5]. Frequentemente chamado de algoritmo de *blossoms*, este foi o primeiro algoritmo em tempo polinomial para resolver o problema em grafos que não possuem necessariamente estrutura bipartida. Embora não seja o algoritmo mais eficiente conhecido para encontrar emparelhamentos máximos em grafos gerais, ele é suficiente para os propósitos deste trabalho e possui uma descrição conceitualmente mais acessível. Por esse motivo, optamos por apresentar a abordagem de Edmonds, ainda que o trabalho de Meijer *et al.* [7] utilize originalmente o algoritmo de Micali e Vazirani [11] como rotina auxiliar.

A fundamentação teórica do algoritmo se baseia no Teorema 1, segundo o qual um emparelhamento M em um grafo G é máximo se, e somente se, não existem caminhos M -aumentadores em G . Em grafos bipartidos, a busca por tais caminhos pode ser realizada

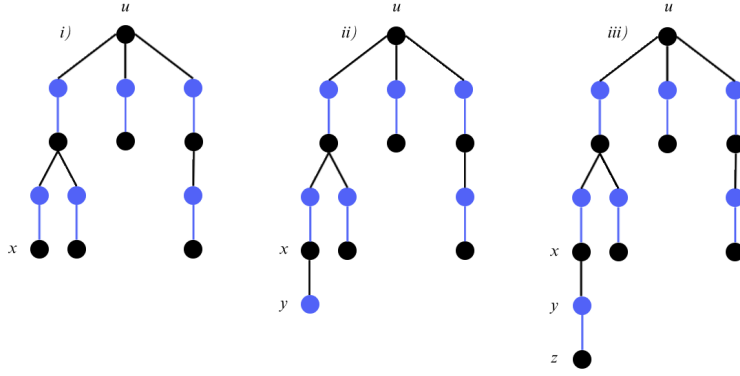


Figura 12: Exemplo de busca por caminho M -aumentador, em que *i)* apresenta uma árvore M -alternante; *ii)* ilustra a possibilidade em que um caminho M -aumentador foi encontrado e *iii)* a possibilidade em que a árvore M -alternante é expandida.

diretamente por meio de árvores alternantes. Em grafos gerais, entretanto, a presença de ciclos ímpares pode impedir que essa busca prossiga da mesma forma. A principal contribuição do algoritmo de Edmonds é justamente tratar esses ciclos ímpares, chamados de *blossoms*, por meio de operações de contração.

Dado um emparelhamento M , uma **árvore M -alternante** enraizada em um vértice u não saturado por M é uma árvore T tal que, para qualquer vértice $v \in V(T)$, o único caminho de u até v em T é um caminho M -alternante. Durante a busca por um caminho M -aumentador, o algoritmo expande iterativamente essa árvore a partir da raiz u .

Para facilitar a descrição da busca, os vértices de T são classificados de acordo com a paridade de sua distância até a raiz. Vértices a uma distância par são chamados de **pares**, enquanto vértices a uma distância ímpar são chamados de **ímpares** [3]. Denotaremos por $P(T)$ o conjunto de vértices pares e por $I(T)$ o conjunto de vértices ímpares, de modo que $(P(T), I(T))$ forma uma bipartição de T .

O processo de expansão da árvore é ilustrado na Figura 12, em que os vértices pares estão representados na cor preta e os vértices ímpares na cor azul. A expansão ocorre pela inspeção de arestas que ligam um vértice par $x \in V(T)$ a algum vértice y . Se y for um vértice não saturado por M , então o caminho da raiz u até x , acrescido da aresta xy , forma um caminho M -aumentador. Por outro lado, se y for saturado por M , existe uma aresta $zy \in M$; nesse caso, a árvore é expandida pela adição das arestas xy e zy .

A busca tradicional falha em grafos não bipartidos quando o algoritmo encontra uma aresta conectando dois vértices pares pertencentes à mesma árvore alternante T . A união dessa aresta com os caminhos que conectam esses vértices à raiz forma um ciclo de comprimento ímpar. Essa estrutura é chamada de *blossom*. A Figura 13 ilustra essa situação, suprimindo o restante do grafo.

Formalmente, um **blossom** B é um ciclo de comprimento ímpar $2k + 1$ no qual k arestas

pertencem ao emparelhamento M , deixando exatamente um vértice não saturado no interior do ciclo em relação às arestas de B . Esse vértice é chamado de base do *blossom* e está conectado à raiz da árvore alternante por um caminho M -alternante de comprimento par, chamado *stem*.

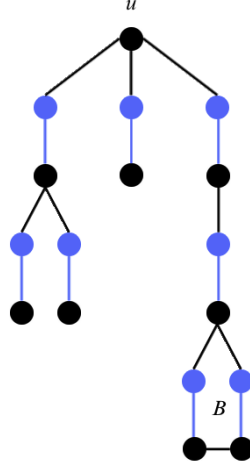


Figura 13: Um ciclo ímpar na árvore alternante de busca forma uma *blossom* B .

Para contornar essas estruturas, o Algoritmo de Edmonds introduz a operação de **contração** de *blossoms* (Figura 14). Quando um *blossom* B é identificado, todos os vértices que compõem esse ciclo ímpar são fundidos em um único pseudovértice, normalmente associado à base do *blossom*. As arestas que ligavam vértices externos a algum vértice de B passam a ser incidentes ao pseudovértice. Já as arestas internas tornam-se laços e são removidas, enquanto eventuais arestas múltiplas resultantes da contração são preservadas ou simplificadas conforme a representação do grafo adotada. O emparelhamento também é atualizado, originando um emparelhamento M/B no grafo contraído G/B .

A propriedade fundamental dessa operação, demonstrada por Edmonds [5], é que o grafo original G possui um caminho M -aumentador se, e somente se, o grafo contraído G/B possui um caminho M/B -aumentador. Dessa forma, a busca pode prosseguir no grafo contraído.

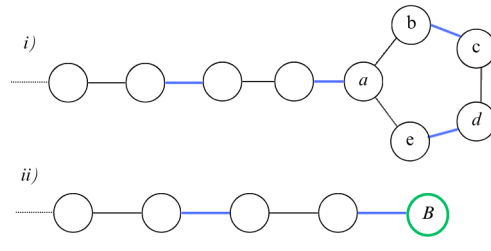


Figura 14: Processo de contração da *blossom* acima em um pseudovértice B abaixo.

Se um caminho M/B -aumentador é encontrado no grafo contraído e esse caminho passa por pseudovértices, o algoritmo realiza o procedimento de **descontração**, cujo objetivo é

reconstruir um caminho M -aumentador correspondente no grafo original. Seja b um pseudo-vértice que representa um *blossom* B . Para cada ocorrência de b no caminho encontrado no grafo contraído, substitui-se esse pseudovértice por um caminho apropriado no interior de B , preservando a alternância entre arestas que pertencem e que não pertencem ao emparelhamento.

Quando b aparece como extremo do caminho aumentador no grafo contraído, a expansão deve terminar na base do *blossom*. Por outro lado, se b aparece como vértice interno do caminho, então o caminho entra em b por uma aresta ub e sai por uma aresta bv . No grafo original, isso significa que existem vértices $x, y \in V(B)$ tais que $ux \in E(G)$ e $yv \in E(G)$. Como B é um ciclo ímpar, existem dois caminhos internos em B ligando x a y : um de comprimento par e outro de comprimento ímpar. O algoritmo escolhe aquele que preserva a alternância em relação a M .

Após substituir todos os pseudovértices pelos caminhos alternantes correspondentes no grafo original, obtém-se um caminho M -aumentador P em G . O emparelhamento atual é então atualizado pela operação de diferença simétrica

$$M' = M \Delta E(P),$$

que aumenta o tamanho do emparelhamento em uma unidade.

Com base nos procedimentos descritos acima, o Algoritmo 1 apresenta a estrutura geral do método de Edmonds. O algoritmo mantém um emparelhamento M e, enquanto existir um caminho M -aumentador, atualiza M pela diferença simétrica com as arestas desse caminho.

Algoritmo 1 Algoritmo de Edmonds (*Blossom Algorithm*)

Entrada: Um grafo $G = (V, E)$

Saída: Um emparelhamento máximo M

1: $M \leftarrow \emptyset$

2: **repita**

3: Encontre um caminho M -aumentador P utilizando o Algoritmo 2

4: **se** $P \neq \emptyset$ **então**

5: $M \leftarrow M \Delta E(P)$

6: **até** $P = \emptyset$

7: **devolva** M

A rotina responsável por encontrar um caminho M -aumentador é descrita no Algoritmo 2. Ela constrói uma floresta alternante a partir dos vértices não saturados por M e trata os três casos possíveis durante a exploração: extensão da floresta, detecção de caminho aumentador e contração de *blossoms*.

O Algoritmo 1 termina quando a busca não encontra mais nenhum caminho M -aumentador. Pelo Teorema 1, a inexistência de caminhos M -aumentadores certifica que o emparelhamento

Algoritmo 2 Busca de um caminho M -aumentador

Entrada: Um grafo $G = (V, E)$ e um emparelhamento M

Saída: Um caminho M -aumentador ou $\emptyset$

- 1: Inicialize uma floresta alternante F com todos os vértices M -insaturados como raízes.
 - 2: **enquanto** existir uma aresta não explorada incidente a um vértice rotulado como *par*
 faça
 - 3: Seja uv essa aresta.
 - 4: **se** v não possui rótulo **então**
 - 5: Rotule v como *ímpar*
 - 6: Seja $vw \in M$
 - 7: Rotule w como *par*
 - 8: Adicione uv e vw à floresta
 - 9: **senão se** v é rotulado como *par* e pertence a outra árvore **então**
 - 10: **devolva** o caminho alternante obtido unindo as duas árvores
 - 11: **senão se** v é rotulado como *par* e pertence à mesma árvore **então**
 - 12: Identifique o *blossom* B
 - 13: Contraia B , obtendo G/B e M/B
 - 14: Encontre recursivamente um caminho aumentador P em G/B
 - 15: **se** $P \neq \emptyset$ **então**
 - 16: Expanda P para um caminho aumentador em G
 - 17: **devolva** P
 - 18: **devolva** $\emptyset$
-

obtido é máximo.

Uma análise do Algoritmo de Edmonds deve considerar que cada fase consiste na busca por um caminho M -aumentador. Durante essa busca, podem ser identificados *blossoms*, os quais são contraídos, permitindo que a busca prossiga no grafo contraído. Embora múltiplas contrações possam ocorrer em uma mesma fase, o custo total da construção da floresta alternante, da exploração das arestas e das contrações é limitado por $O(v(G) \cdot e(G))$.

Sempre que um caminho M -aumentador é encontrado, aplica-se a diferença simétrica, aumentando o tamanho do emparelhamento em uma unidade e iniciando uma nova fase. Como um emparelhamento em um grafo com $v(G)$ vértices possui no máximo $\lfloor v(G)/2 \rfloor$ arestas, podem ocorrer no máximo $O(v(G))$ aumentos. Dessa forma, o algoritmo executa no máximo $O(v(G))$ fases, cada uma com custo $O(v(G) \cdot e(G))$, resultando em uma complexidade de tempo $O(v(G)^2 \cdot e(G))$.

4 Fatores de Grafos

Define-se um **fator** de um grafo G como um subgrafo gerador de G , isto é, um subgrafo F tal que $V(F) = V(G)$. Embora a definição seja ampla, o interesse acadêmico concentra-se usualmente em fatores que satisfazem condições específicas de regularidade sobre os graus de seus vértices. Seja $f : V(G) \rightarrow \mathbb{Z}^+$ uma função que atribui um inteiro positivo a cada vértice

de G . Um **f -fator** é um fator F em que o grau de cada vértice v em F é exatamente $f(v)$:

$$d_F(v) = f(v), \quad \forall v \in V(G). \quad (2)$$

Um caso particular é o **k -fator**, no qual a função f é constante, ou seja, $f(v) = k$ para todo $v \in V(G)$. Em termos práticos, um k -fator é um subgrafo gerador k -regular. A Figura 15 ilustra essas estruturas.

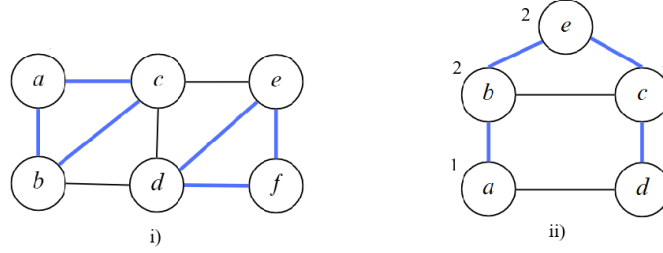


Figura 15: i) Um grafo G contendo um 2-fator ($k = 2$). ii) Um grafo G contendo um f -fator, onde os rótulos nos vértices denotam os valores de $f(v)$.

Estruturalmente, um 1-fator é um subgrafo cujas arestas formam um **emparelhamento perfeito**, enquanto um 2-fator é constituído por uma união de **ciclos disjuntos** nos vértices que cobrem todo o conjunto $V(G)$. A complexidade de encontrar tais estruturas varia, mas muitos problemas podem ser simplificados via reduções. Tutte [14] demonstrou em seu trabalho que o problema de decidir a existência de um f -fator em um grafo pode ser reduzido ao problema de encontrar um 1-fator em um grafo derivado G' , permitindo que algoritmos de emparelhamento sejam aplicados a problemas mais gerais de f -fatores. Vale ressaltar que a definição de fator se aplica a grafos simples ou multigrafos sendo assim, para fins deste trabalho, denominamos um **k -fator simples** aquele que não possui arestas paralelas ou laços.

4.1 Fatorização e Resultados Clássicos

Além da existência de um único fator, podemos investigar a partição do conjunto de arestas de um grafo em diversos fatores. Segundo Akiyama e Kano [1], um grafo é dito **k -fatorável** se seu conjunto de arestas $E(G)$ puder ser decomposto em k -fatores disjuntos nas arestas, conforme expresso a seguir:

$$E(G) = F_1 \cup F_2 \cup \dots \cup F_m, \quad \text{onde } F_i \cap F_j = \emptyset \text{ para } i \neq j. \quad (3)$$

Esse processo de decomposição é denominado **k -fatorização**. Historicamente, os primeiros resultados significativos na área são atribuídos a Petersen [10] em seu trabalho *Die Theorie Der Reguaren Graphs*, de 1891. Motivado por problemas de fatoração de polinômios de grau 3 (representados por grafos 3-regulares em seu trabalho), Petersen estabeleceu as

bases da teoria ao estudar grafos regulares. Seus resultados fundamentais são apresentados a seguir:

Teorema 4 (Teorema de 2-fatoração de Petersen). *Para todo inteiro $r \geq 1$, todo grafo $2r$ -regular é 2-fatorável. Consequentemente, para cada inteiro k tal que $1 \leq k \leq r$, todo grafo $2r$ -regular possui um $2k$ -fator.*

Teorema 5 (Teorema de Petersen). *Todo grafo 3-regular sem arestas de corte (bridges) possui um emparelhamento perfeito (1-fator).*

Enquanto o Teorema 4 garante que grafos de grau par sempre podem ser decompostos em ciclos (2-fatores), o Teorema 5 estabelece uma condição suficiente para a existência de emparelhamentos em grafos cúbicos, utilizando propriedades de conectividade para mitigar impedimentos estruturais.

4.2 O Teorema dos Fatores Regulares de Tutte

Uma condição necessária e suficiente da existência de um k -fator em um grafo foi estabelecida por Tutte [13]. O Teorema 6 fornece um critério matemático para garantir que o grafo possua uma estrutura tal que seja suficiente para possuir um k -fator, de forma que não haja conjunto de vértices que isole componentes a ponto de impedir sua formação. A função $\delta(S, T)$ determina a folga estrutural que o grafo precisa possuir para garantir a existência de um k -fator, de modo que, para quaisquer conjuntos disjuntos de vértices S e T , essa folga deve ser maior ou igual a zero. Além disso, as principais componentes do Teorema 6 são: $k|S|$, que representa a capacidade total de conexões que os vértices do conjunto S podem realizar com as demais componentes; $\sum_{x \in T} d_{G-S}(x)$ é a soma dos graus das arestas em T , removendo as arestas que ligavam a S , fator que serve para contabilizar as arestas disponíveis para os vértices de T que não são originadas em S (tanto arestas internas a T quanto para as componentes isoladas $G - (S \cup T)$); $k|T|$ representa qual a demanda total que o conjunto T possui de arestas para a existência do k -fator regular; $q(S, T)$ identifica a quantidade de componentes k -ímpares de $G - (S \cup T)$ indicando a quantidade de vértices que, por questão de paridade, não conseguem satisfazer o grau k sem conectar arestas para S (equivalente às componentes ímpares do Teorema 3).

Teorema 6 (Teorema de Fatores Regulares). *Seja $k \geq 1$ um inteiro e G um grafo. G possui um k -fator se, e somente se, para todos os subconjuntos disjuntos $S, T \subseteq V(G)$:*

$$\delta(S, T) = k|S| + \sum_{x \in T} d_{G-S}(x) - k|T| - q(S, T) \geq 0, \quad (4)$$

onde $q(S, T)$ denota o número de componentes C de $G - (S \cup T)$, denominadas **componentes**

k-ímpares, ou seja, que satisfazem a seguinte condição de paridade:

$$k|C| + e_G(C, T) \equiv 1 \pmod{2}. \quad (5)$$

em que $e_G(C, T)$ representa o número de arestas que conecta T à componente C .

Para fins de exemplo, apresentaremos dois casos, sendo um de sucesso e um de falha deste teorema. Primeiramente, considere o grafo *i*) da Figura 16, para o qual queremos verificar a existência de um 1-fator, isto é, de um emparelhamento perfeito.

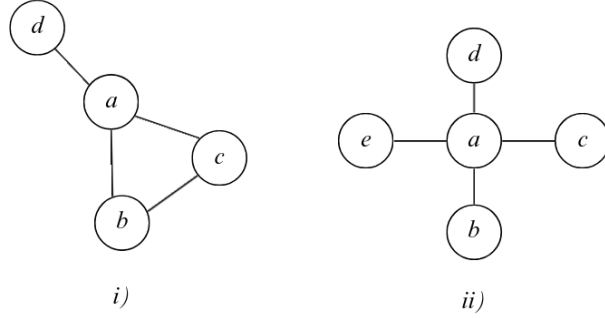


Figura 16: *i*) Grafo pipa, que possui um 1-fator e *ii*) grafo $K_{1,4}$, que não possui um 2-fator.

Selecionando $S = \{a\}$ e $T = \{b\}$, temos que $k|S| = k|T| = 1$. Como o vértice a pertence a S , ele é removido na construção de $G - S$; desse modo, a única aresta incidente a b em $G - S$ é a aresta bc , e portanto

$$\sum_{x \in T} d_{G-S}(x) = d_{G-S}(b) = 1.$$

Além disso, o grafo $G - (S \cup T)$ possui duas componentes: $C_1 = \{c\}$ e $C_2 = \{d\}$. Para C_1 , temos

$$k|C_1| + e_G(C_1, T) = 1 + 1 = 2,$$

pois existe a aresta bc ligando C_1 a T ; logo, C_1 não é uma componente k -ímpar. Para C_2 , por outro lado, temos

$$k|C_2| + e_G(C_2, T) = 1 + 0 = 1,$$

pois não existe aresta ligando d a b ; assim, C_2 é uma componente k -ímpar. Portanto, $q(S, T) = 1$. Substituindo esses valores na desigualdade 4, obtemos

$$\delta(S, T) = 1 + 1 - 1 - 1 = 0.$$

Logo, para essa escolha de S e T , a desigualdade é satisfeita. Repetindo a verificação para todos os pares disjuntos $S, T \subseteq V(G)$, obtém-se $\delta(S, T) \geq 0$ em todos os casos. Portanto,

pelo Teorema 6, o grafo *i*) possui um 1-fator. De fato, esse resultado é compatível com a existência do emparelhamento perfeito formado pelas arestas ad e bc .

Considere agora o grafo *ii*) da Figura 16. Nesse caso, queremos verificar se o grafo possui um 2-fator. Para isso, tomemos $S = \emptyset$ e $T = \{a\}$. Como $k = 2$, temos

$$k|S| = 2 \cdot 0 = 0 \quad \text{e} \quad k|T| = 2 \cdot 1 = 2.$$

As componentes de $G - (S \cup T)$ são

$$C_1 = \{b\}, \quad C_2 = \{c\}, \quad C_3 = \{d\}, \quad C_4 = \{e\}.$$

Cada uma dessas componentes é formada por um único vértice e possui exatamente uma aresta ligando-a ao conjunto T , isto é, ao vértice a . Assim, temos que

$$\sum_{x \in T} d_{G-S}(x) = d_{G-S}(a) = 4.$$

Como todas as componentes possuem uma aresta conectando-as a $T = \{a\}$, logo elas possuem paridade 1, que é ímpar; sendo assim

$$q(S, T) = 4.$$

Substituindo esses valores na desigualdade 4, obtemos

$$\delta(S, T) = k|S| + \sum_{x \in T} d_{G-S}(x) - k|T| - q(S, T) = 0 + 4 - 2 - 4 = -2.$$

Como $\delta(S, T) < 0$, a desigualdade do Teorema 6 não é satisfeita para essa escolha de conjuntos disjuntos S e T . Portanto, o grafo *ii*) não possui um 2-fator.

A intuição por trás desse teorema reside nas componentes k -ímpares. Se houver componentes demais exigindo conexões com o conjunto S , é impossível que a estrutura do grafo comporte um k -fator. É uma restrição de paridade, visto que para que os vértices de uma componente conexa C atinjam o grau k no fator, a soma das incidências internas de C com as arestas que partem para T deve ser par. Caso a Equação 5 seja satisfeita, há uma sobra ímpar de incidências que não pode ser resolvida apenas dentro de $C \cup T$, exigindo uma conexão com o conjunto S . Portanto, as componentes k -ímpares representam exigências estruturais que o conjunto S deve ser capaz de suprir. Se o custo, imposto por T e pelas componentes k -ímpares, exceder a capacidade de conexão de S , o grafo falha em possuir um k -fator.

Embora o Teorema 6 forneça uma caracterização estrutural para a existência de k -fatores, o interesse está na formulação algorítmica desse conceito. Em outras palavras, dado um grafo G e um inteiro positivo k , deseja-se determinar se existe um subgrafo gerador k -regular e,

quando existir, construí-lo. Essa tarefa é formalizada no Problema 2.

Problema 2 (k -fator).

Entrada: Um grafo G e um inteiro positivo k .

Saída: Um k -fator simples em G , caso exista; caso contrário, uma indicação de inexistência.

Na literatura, diversas estratégias buscam resolver este problema. Além das transformações de Tutte, Diaz-Gutierrez e Gopi [4] propuseram o método de **substituição de templates**. Através da criação de um grafo inflado, onde vértices do grafo são substituídos por estruturas específicas, chamadas de (*gadgets*), demonstra-se que a busca por um 2-fator no grafo original equivale à busca por um emparelhamento perfeito no grafo inflado. Uma abordagem similar é adotada por Umans [15], reforçando a robustez das técnicas de redução para a solução de problemas de fatoração. Na seção a seguir, apresentaremos a implementação do algoritmo proposto por Meijer *et al.* [7], com enfoque primeiramente nas considerações e implicações teóricas e, posteriormente na Seção 5.1, a ideia geral da nossa implementação em C++.

5 Desenvolvimento

Um k -fator F de G é **simples** se F não contém arestas múltiplas entre um mesmo par de vértices e não há laços, isto é, arestas incidentes a um mesmo vértice em ambas as extremidades. Nesse contexto em seu artigo *An algorithm for computing simple k -factors*, Meijer, Núñez-Rodríguez e Rappaport [7] introduzem um algoritmo eficiente para computar um k -fator simples. Utilizando generalizações das abordagens de *templates* propostas por Umans [15] e Diaz-Gutierrez e Gopi [4], o método propõe transformações estruturais em um grafo geral G por meio de componentes denominados *gadgets*. A ideia é criar um grafo G' chamado **inflado** a partir do grafo original G de modo que G' seja um grafo composto por *gadgets* originados a partir dos vértices de G . O grafo G' é tal que, a existência de um k -fator em G implica em um emparelhamento perfeito em G' e vice versa. A seguir, detalharemos essa discussão.

O conjunto de *gadgets* é definido como $\Gamma_{k,d_G(v)}$, em que $k \in \mathbb{Z}$, $2 \leq k \leq d$ é o valor do fator que se busca encontrar e $d_G(v)$ é o grau de cada vértice substituído pelo *gadget* no grafo original G .

A arquitetura de cada componente em $\Gamma_{k,d_G(v)}$ compreende três conjuntos disjuntos de vértices: $d_G(v)$ **vértices externos** (*outer vertices*), $d_G(v)$ **vértices internos** (*inner vertices*) e k **vértices de núcleo** (*core vertices*). A conectividade interna do *gadget* obedece à seguinte regra de construção: arestas de núcleo conectam cada vértice de núcleo a todos os vértices internos; arestas internas ligam cada vértice interno a exatamente um vértice externo; e arestas externas conectam os vértices externos de *gadgets* distintos, preservando a vizinhança

original de cada vértice $v \in V(G)$. Vértices do *gadget* pertencentes a um mesmo conjunto não possuem conexões entre si. A Figura 17 exemplifica a formação de um *gadget* para um vértice com $d_G(v) = 4$ e fator $k = 2$.

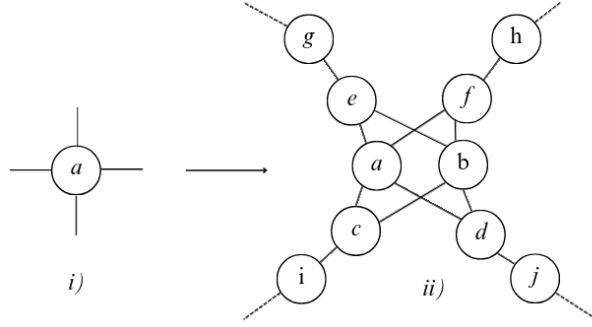


Figura 17: Transformação de um vértice a com $d_G(a) = 4$ (i) em um *gadget* $\Gamma_{2,4}$ (ii), com $\{a, b\}$ vértices de núcleo, $\{c, d, e, f\}$ vértices internos e $\{g, h, i, j\}$ vértices externos, conectados pelas arestas $\{ac, ad, ae, af, bc, bd, be, bf\}$ de núcleo, $\{ci, dj, eg, fh\}$ arestas internas e com as arestas externas tracejadas, conectando com outros *gadgets* vizinhos.

A partir dessa regra de construção, substitui-se cada vértice $v \in V(G)$ por uma instância correspondente de $\Gamma_{k,d_G(v)}$ resultando na construção do chamado **grafo inflado** G' . A Figura 18 ilustra um exemplo de um grafo inflado.

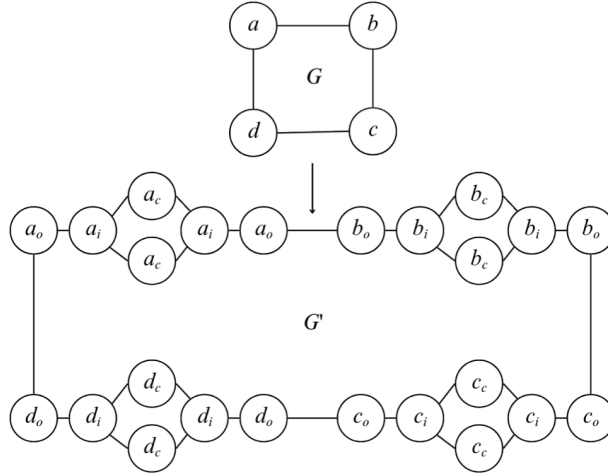


Figura 18: Exemplo da conversão do grafo G acima para o grafo inflado G' abaixo, em que para cada $v \in V(G)$, temos os vértices de núcleo, internos e externos denotados por v'_c , v'_i e v'_o respectivamente, para cada $v' \in G'$.

Cada *gadget* possui k vértices de núcleo, $d_G(v)$ vértices internos e $d_G(v)$ vértices externos. Logo, ao construir o **grafo inflado** G' , temos que cada vértice $v \in V(G)$ dá origem a $k + 2d_G(v)$ vértices e, por conseguinte, $v(G)$ vértices originam um total de $v(G') = kv(G) + 4e(G)$ vértices em G' . A Tabela 1 apresenta um resumo das principais características do grafo inflado G' em termos dos parâmetros do grafo original G .

Tabela 1: Resumo das propriedades do grafo inflado G' .

Componente	Quantidade em G'
Vértices de núcleo	$kv(G)$
Vértices internos	$2e(G)$
Vértices externos	$2e(G)$
Total de vértices	$kv(G) + 4e(G)$

O método proposto por Meijer *et al.* [7] para encontrar um k -fator simples em G , caso ele exista, é descrito pelo algoritmo 3.

Algoritmo 3 Computar k -fator simples

Entrada: Um grafo G

Saída: Um k -fator simples F , caso exista, ou $\emptyset$ caso contrário

- 1: **se** G possui vértice de grau menor que k **então**
 - 2: **devolva** $\emptyset$
 - 3: Seja G' o grafo inflado a partir de G
 - 4: Seja M um emparelhamento máximo em G'
 - 5: **se** M não é um emparelhamento perfeito **então**
 - 6: **devolva** $\emptyset$
 - 7: Seja F um grafo com todos os vértices de G , sem arestas
 - 8: Adicione ao grafo F todas as arestas de G que correspondem às arestas externas de M
 - 9: **devolva** F
-

A primeira verificação do Algoritmo 3 descarta grafos que certamente não possuem um k -fator. Tal explicação pode ser compreendida formalmente pelo Lema 7.

Lema 7 (Condição necessária). *Se G possui um k -fator, então $\delta(G) \geq k$.*

Demonstração. Considere F um k -fator de G . Para todo $v \in V(G)$, temos $d_F(v) = k$ por definição de k -fator. Como F é subgrafo de G , segue que $d_G(v) \geq d_F(v) = k$. Logo, o menor grau de G satisfaz $\delta(G) \geq k$. $\square$

A abordagem de Meijer *et al.* [7] sugere, a partir do Teorema 8, uma relação entre a existência do k -fator em G e um emparelhamento perfeito encontrado em seu grafo inflado G' .

Teorema 8. *Seja G um grafo com grau mínimo $\delta(G) \geq k$. O grafo G possui um k -fator se, e somente se, seu grafo inflado correspondente G' possui um emparelhamento perfeito.*

Demonstração. Primeiramente, mostraremos que se G possui um k -fator simples, então o grafo inflado G' possui um emparelhamento perfeito. Suponha que F seja um k -fator simples de G . Para cada vértice $v \in V(G)$, seja v' a cópia do *gadget* $\Gamma_{k,d_G(v)}$ correspondente a v em G' . Como F é um k -fator, temos $d_F(v) = k$. Portanto, dentre as $d_G(v)$ arestas incidentes a v em G , exatamente k pertencem a F . Para cada uma dessas k arestas de F incidentes a v , incluímos no emparelhamento de G' a aresta externa correspondente, isto é, a aresta

que conecta o vértice externo de v' ao vértice externo do *gadget* associado ao outro extremo da aresta em G . Como F é simples, essas arestas correspondem a arestas distintas de G e, portanto, selecionam vértices externos distintos em cada *gadget*. Restam, em v' , exatamente $d_G(v) - k$ vértices externos não emparelhados por arestas externas. Cada um desses vértices externos é então emparelhado com seu vértice interno correspondente. Após essa etapa, restam exatamente k vértices internos não emparelhados no *gadget*. Como cada vértice de núcleo é adjacente a todos os vértices internos, podemos emparelhar esses k vértices internos restantes com os k vértices de núcleo de v' . Repetindo essa construção para todos os vértices $v \in V(G)$, todos os vértices de G' são emparelhados exatamente uma vez. Logo, G' possui um emparelhamento perfeito.

Vamos agora mostrar que se G' possui um emparelhamento perfeito, então G possui um k -fator simples. Suponha que M seja um emparelhamento perfeito de G' . Considere um vértice $v \in V(G)$ de grau $d_G(v)$ e seja v' a cópia do *gadget* $\Gamma_{k,d_G(v)}$ correspondente a v . O *gadget* v' possui exatamente k vértices de núcleo. Como esses vértices só são adjacentes a vértices internos, todos os vértices de núcleo devem ser emparelhados, em M , com k vértices internos distintos de v' . Assim, restam $d_G(v) - k$ vértices internos que não foram emparelhados com vértices de núcleo. Cada um desses vértices internos só pode ser emparelhado com seu vértice externo correspondente, pois suas únicas adjacências são o vértice externo correspondente e os vértices de núcleo. Consequentemente, $d_G(v) - k$ vértices externos de v' são emparelhados internamente no *gadget*, restando exatamente k vértices externos de v' que não são emparelhados dentro do próprio *gadget*. Como M é um emparelhamento perfeito, esses k vértices externos restantes devem ser emparelhados por arestas externas com vértices externos de outros *gadgets*. Cada uma dessas arestas externas corresponde a uma aresta do grafo original G incidente a v . Definimos F como o subgrafo de G formado por todas as arestas correspondentes às arestas externas de M . Pelo argumento acima, para cada vértice $v \in V(G)$, exatamente k arestas de F incidem em v . Logo, F é um subgrafo gerador k -regular de G . Além disso, como M é um emparelhamento, cada aresta externa é utilizada no máximo uma vez; portanto, nenhuma aresta de G é repetida em F . Concluímos que F é um k -fator simples de G . $\square$

A demonstração anterior estabelece a equivalência entre a existência de um k -fator em G e a existência de um emparelhamento perfeito em G' . Resta observar, de maneira mais concreta, como as arestas desse emparelhamento permitem recuperar o fator no grafo original.

A Figura 19 ilustra esse mecanismo para um *gadget* $\Gamma_{2,4}$. Como $k = 2$, os dois vértices de núcleo a e b precisam ser emparelhados com dois vértices internos; no exemplo, pelas arestas $\{ac, bd\}$. Os dois vértices internos restantes, e e f , são então emparelhados com seus respectivos vértices externos, pelas arestas $\{eg, fh\}$. Dessa forma, os vértices externos i e j permanecem impossibilitados de serem cobertos por arestas internas ao *gadget* e, portanto

devem ser emparelhados com vértices externos de outros *gadgets*, para que o emparelhamento seja perfeito.

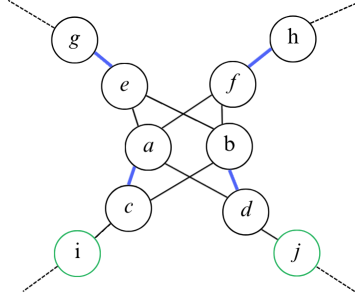


Figura 19: Exemplo de emparelhamento perfeito em um *gadget* $\Gamma_{2,4}$, em que $k = 2$ vértices externos precisam ser emparelhados com outros *gadgets*

Esse comportamento é justamente o que permite recuperar o k -fator. Cada vértice externo de um *gadget* associado a $v \in V(G)$ representa uma das arestas incidentes a v no grafo original. Portanto, as arestas externas que pertencem ao emparelhamento perfeito de G' identificam diretamente quais arestas de G devem ser selecionadas.

Como, conforme demonstrado anteriormente, exatamente k vértices externos de cada *gadget* são emparelhados com vértices externos de outros *gadgets*, a seleção dessas arestas produz em G exatamente k arestas incidentes a cada vértice. Logo, para o subgrafo F obtido,

$$d_F(v) = k, \quad \forall v \in V(G).$$

Além disso, como F preserva todos os vértices do grafo original, $V(F) = V(G)$. Assim, F é um subgrafo gerador k -regular de G , isto é, um k -fator. A Figura 20 exemplifica essa relação entre as k arestas externas selecionadas em cada *gadget* e as k arestas incidentes ao vértice correspondente no grafo original G .

A complexidade do Algoritmo 3 pode ser compreendida ao analisarmos suas duas etapas principais. A primeira etapa consiste na construção do grafo inflado G' a partir do grafo original G . Para construir G' , percorremos cada vértice de G e, para cada um, visitamos sua lista de adjacência. Como a soma dos graus de todos os vértices é igual a $2e(G)$, o trabalho total desta etapa é $O(v(G) + e(G))$, que pode ser simplificado para $O(e(G))$ em grafos densos onde $e(G) \geq v(G)$.

Na segunda etapa, aplicamos o Algoritmo 1 sobre o grafo inflado G' para encontrar um emparelhamento máximo. Como visto na Seção 2, a complexidade do algoritmo de Edmonds é $O(v(G')^2 \cdot e(G'))$. No entanto, para obter a complexidade em termos do grafo original G , precisamos expressar $v(G')$ e $e(G')$ em função de $v(G)$ e $e(G)$. Quanto ao número de vértices $v(G)$, o grafo inflado G' possui, por construção, $v(G') = kv(G) + 4e(G)$ vértices. Como k é uma constante fornecida como entrada do problema, esta expressão pode ser simplificada

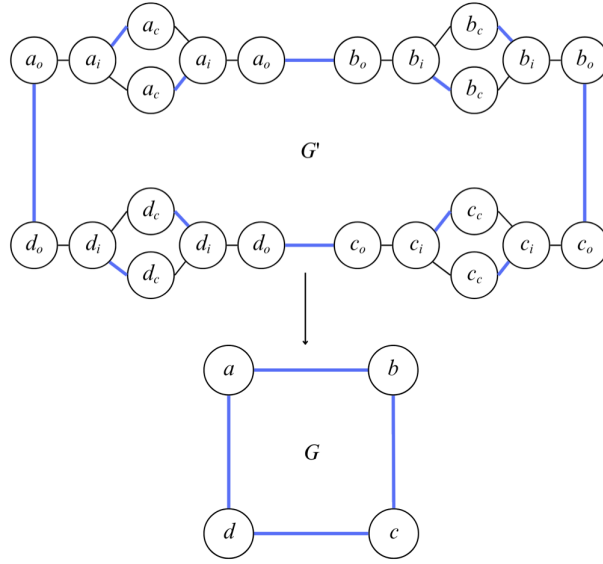


Figura 20: Recuperação de um 2-fator de G a partir das arestas externas pertencentes a um emparelhamento perfeito do grafo inflado G' .

para $O(v(G) + e(G))$. Quanto ao número de arestas $e(G')$, o grafo G' possui três tipos de conexões: as arestas de núcleo (que conectam vértices de núcleo a internos), as arestas internas (que conectam vértices internos a externos) e as arestas externas (que conectam vértices externos entre *gadgets* distintos). A quantidade total de arestas é proporcional a $e(G)$, visto que cada *gadget* deve respeitar o grau do vértice $v \in V(G)$ que o originou, sendo assim, $e(G') = O(e(G))$. Substituindo esses valores na complexidade do algoritmo de Edmonds, obtemos $O((v(G) + e(G))^2 \cdot e(G))$. Para grafos densos, onde $e(G) \approx v(G)^2$, esta expressão reduz-se a $O(v(G)^2 \cdot e(G))$. Para grafos esparsos, onde $e(G) \approx v(G)$, a complexidade é $O(v(G)^3)$. Em ambos os casos, a complexidade é polinomial. Logo, é possível perceber que a complexidade do tempo está diretamente correlacionada ao algoritmo de emparelhamento utilizado.

5.1 Implementação

A implementação deste algoritmo foi desenvolvida utilizando C++ e está disponível no repositório do projeto no *Github*¹. A organização do código fundamenta-se no paradigma de **Programação Orientada a Objetos**, o que possibilita uma melhor separação de responsabilidades, manutenção e compreensão do código. A estrutura possui três classes principais: a classe **Grafo** que representa um grafo e contém toda a lógica para a construção do grafo inflado que o Algoritmo 3 necessita; a classe **Matching**, que encapsula o algoritmo de emparelhamento máximo de Edmonds, desenvolvido por terceiros, disponibilizado neste outro

¹https://github.com/peserra/k_factor_algorithm

repositorio no *Github*², e por fim a classe **GrafoAdapter**, que funciona como um adaptador entre as duas classes anteriores, permitindo uma integração entre elas com alterações mínimas. É importante informar que as classes escritas em inglês assim estão pois seu código original estava em inglês e, com a intenção de fazer o mínimo de alterações possível no código de terceiros, foram mantidas desta forma.

Conforme previamente mencionado na seção anterior, o problema de encontrar um k -fator simples em um grafo geral G é equivalente ao de encontrar um emparelhamento perfeito M em seu grafo inflado G' . Dessa forma, o procedimento pode ser dividido nas etapas: verificação da condição necessária, determinada pelo Lema 7; construção do grafo inflado e construção do k -fator a partir do emparelhamento encontrado.

A primeira etapa consiste em satisfazer o Lema 7, verificando se $\delta(G) \geq k$. Para isso, o método `criaListaGraus()`, pertencente à classe **Grafo**, percorre os vértices $v \in V(G)$, armazenando em uma lista o grau de cada um e atualizando a propriedade `menorGrau`. Caso o valor de `menorGrau` seja menor que k , o algoritmo retorna imediatamente, indicando que o k -fator não existe.

A segunda consiste na construção correta do *grafo inflado* G' . Nessa etapa, cada vértice v do grafo original é substituído pelo seu *gadget* equivalente respeitando as regras de construção propostas. Como discutido na seção anterior, cada *gadget* $\Gamma_{k,d_G(v)}$ é composto por $v(\Gamma_{k,d_G(v)}) = k + 2d_G(v)$ vértices, dessa forma G' deve possuir um total de $v(G') = kv(G) + 4e(G)$ vértices. Portanto, inicialmente, criamos uma instância de grafo que possui $v(G')$ vértices inicialmente desconectados. O Algoritmo 3 requer um mapeamento eficiente entre o vértice original e seu *gadget*, portanto pensamos em utilizar os $v(\Gamma_{k,d_G(v)})$ vértices como índices delimitadores de cada *gadget*. Dessa forma, o vértice $v = 0$ de G dá origem aos $0, \dots, k + 2d_G(v)$ vértices de G' e assim por diante, até o último vértice de G' , cujo índice é $kv(G) + 4e(G)$. Para auxiliar o mapeamento criamos também uma tabela T que guarda os índices iniciais de cada *gadget* sob o índice do vértice $v \in V(G)$. Essa tabela é construída a partir do método `populaTabelaIndicesIniciais()`, cujo pseudocódigo é dado pelo Algoritmo 4, que percorre os vértices de G em ordem e acumula o número de vértices gerados pelos *gadgets* anteriores, seguindo a Equação 6. Desse modo, cada elemento T_i indica o índice do primeiro vértice do *gadget* correspondente ao i -ésimo vértice de G no grafo inflado. Formalmente:

$$T_i = ik + 2 \sum_{j=0}^{i-1} d_G(u_j) \quad (6)$$

²<https://github.com/dilsonpereira/Minimum-Cost-Perfect-Matching>

Algoritmo 4 Construir tabela de índices iniciais

Entrada: Um grafo G **Saída:** Uma tabela de índices iniciais T

- 1: Seja $S \leftarrow 0$ o acumulador do total de vértices
 - 2: Seja $I_c \leftarrow 0$ o índice de conversão
 - 3: **para** *todo* $vrtice\ v \in V(G)$ **faça**
 - 4: Seja i o índice de v
 - 5: $I_c \leftarrow ik + S$
 - 6: $S \leftarrow S + 2|N_G(v)|$
 - 7: Adicione I_c em T_i na tabela de índices iniciais T
-

A Figura 21 ilustra a construção do grafo inflado G' a partir de uma instância de G , enquanto a Tabela 2 apresenta a tabela de índices iniciais T associada a essa construção.

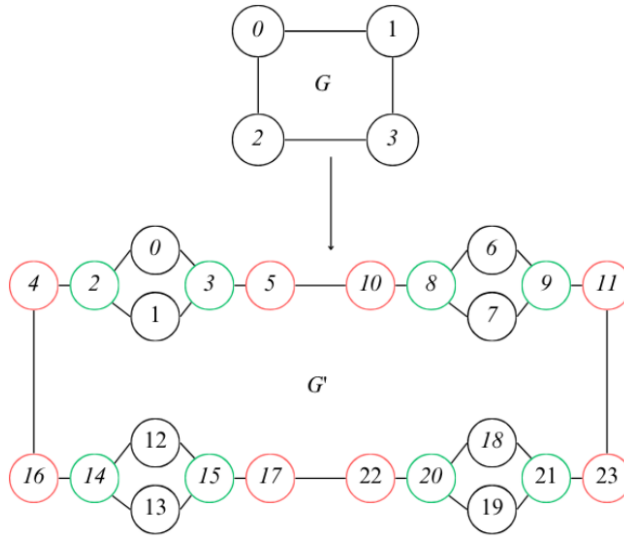


Figura 21: Representação do procedimento de transformar o grafo G em G' , tomando $k = 2$, onde cada vértice de G (acima) cria um conjunto de vértices em G' (abaixo) com os respectivos rótulos.

Tabela 2: Representação da tabela de índices iniciais, que relaciona o índice do vértice v do grafo original G com o índice do primeiro vértice do conjunto de vértices originados por v em G' .

Índice em G	Índice do vértice inicial em G'
0	0
1	6
2	12
3	18

O Algoritmo 5 representa o pseudocódigo da lógica utilizada na implementação para criar o grafo inflado G' . Utilizando a tabela T , o método `preencheArestasGrafoInflado()` determina os intervalos correspondentes a cada classe de vértices (núcleo, internos e externos)

dentro de cada *gadget* originado por v , de modo que os vértices de núcleo ocupam o intervalo $[T_i, T_i + k - 1]$, indicados pelo calculo de I_n ; Os vértices internos ocupam o intervalo $[T_i + k, T_i + k + d_G(v) - 1]$, indicados pelo calculo de I_i e os vértices externos ocupam o intervalo restante, até o número indicado pelo calculo de I_u . No laço principal, o Algoritmo 5 identifica para qual classe o vértice $j \in \Gamma_{k, d_{G'}(j)}$ atual pertence e chama o método correspondente para adicionar as arestas (no pseudocódigo essa chamada de funções foi suprimida para simplificar). Para os vértices de núcleo, o método `adicionaArestasCore()` conecta-os a todos os vértices internos do mesmo *gadget*, conforme a primeira regra de construção. Para os vértices internos, o método `adicionaArestasInner()` conecta cada vértice interno ao seu correspondente vértice externo, conectando o vértice j ao vértice com índice $j + d_{G'}(j)$, conforme a segunda regra de construção. Para os vértices externos, é necessário levar em conta a vizinhança original de v , portanto o método `adicionaArestasOuter()` conecta os vértices externos de *gadgets* calculando o índice relativo I_r do vértice externo atual, buscando na vizinhança de v o vizinho com índice correspondente, e buscando o primeiro vértice sem conexões no *gadget* originado por este vizinho, o que garante a conexão única entre vértices externos respeitando a vizinhança original de v .

A terceira etapa consiste em executar o algoritmo de emparelhamento para verificar se G' possui um emparelhamento perfeito. Uma distinção importante entre o algoritmo proposto por Meijer *et al.* [7] e a implementação que realizamos é a escolha do algoritmo de emparelhamento. O artigo propõe o uso do algoritmo de Micali e Vazirani [11] para encontrar o emparelhamento perfeito no grafo inflado. No entanto, optamos pela utilização do algoritmo de Edmonds [5], apresentado na Seção 3.1 e descrito pelo Algoritmo 1, cuja complexidade polinomial é suficiente para o escopo deste projeto. Utilizamos uma implementação desenvolvida por terceiros e, para viabilizar a integração entre as estruturas desenvolvidas por nós, utilizamos do padrão de projeto **Adapter**, implementando a classe `GrafoAdapter`, que encapsula a classe `Grafo` e expõe a interface esperada pela classe `Matching` da biblioteca externa, permitindo que o algoritmo de Edmonds opere sobre o grafo inflado sem modificações em seu código original. O método `SolveMaximumMatching()` da classe `Matching` retorna uma lista de índices das arestas que compõe o emparelhamento máximo para a instância de grafo fornecida. Se a propriedade `perfect` da classe for falsa, o indicando que existem vértices descobertos pelo emparelhamento, o Algoritmo 3 retorna imediatamente, visto que pelo Lema 8, a inexistência de um emparelhamento perfeito em G' implica na inexistência de um k -fator simples em G . Caso o emparelhamento seja perfeito, extraímos o k -fator iterando as arestas que pertencem ao emparelhamento, utilizando o método `verticePertenceOuter()` da classe `Grafo`, que identifica se alguma das extremidades da aresta pertence à classe de vértices externos. Quando uma aresta externa é encontrada, o método `getVerticeOriginal()` utiliza uma busca binária na tabela de índices iniciais T para identificar qual vértice de G originou o vértice em G' , o que possibilita a reconstituição do k -fator F . Ao fim do método

Algoritmo 5 Criar Grafo Inflado G'

Entrada: Um grafo G e um número inteiro k .

Saída: Um grafo inflado G'

```
1:  $n' \leftarrow kv(G) + 4e(G)$  ▷ Número total de vértices de  $G'$ 
2: Inicialize  $G'$  com  $n'$  vértices e sem arestas
3: Crie a lista  $T$  com os índices iniciais de cada gadget ▷ Algoritmo 4
4: para cada vértice  $v_i \in V(G)$ , com índice  $i$  faça
5:   seja  $I_b = T_i$  ▷ Primeiro índice do gadget de  $v_i$ 
6:   seja  $I_n = I_b + k - 1$  ▷ Último índice dos vértices de núcleo
7:   seja  $I_i = I_b + k + d_G(v_i) - 1$  ▷ Último índice dos vértices internos
8:   seja  $I_u = I_b + k + 2d_G(v_i) - 1$  ▷ Último índice do gadget
9:   para  $j$  de  $I_b$  até  $I_u$  faça
10:    se  $j \leq I_n$  então ▷  $j$  é um vértice de núcleo
11:      para  $v_{interno}$  de  $I_n + 1$  até  $I_i$  faça
12:        adicione a aresta  $j, v_{interno}$  em  $G'$ 
13:      senão se  $j \leq I_i$  então ▷  $j$  é um vértice interno
14:         $v_{externo} = j + d_G(v_i)$ 
15:        adicione a aresta  $j, v_{externo}$  em  $G'$ 
16:      senão ▷  $j$  é um vértice externo
17:         $I_r = j - I_i - 1$  ▷ Índice relativo do vértice externo  $j$ 
18:         $v_d = N_{I_r}(v_i)$  ▷ Vizinho de  $v_i$  correspondente ao índice relativo  $I_r$ 
19:        se  $i < v_d$  então ▷ Evita criar arestas duplicadas
20:           $U_d = T_{v_d} + k + 2d_G(v_d) - 1$  ▷ Último vértice externo do gadget de destino
21:          seja  $D_{outer}$  o número de vértices externos ainda disponíveis no destino
22:           $I_d = U_d - D_{outer} + 1$  ▷ Próximo vértice externo disponível no destino
23:          adicione a aresta  $j, I_d$  em  $G'$ 
24:           $D_{outer} = D_{outer} - 1$ 
25: devolva  $G'$ 
```

`achaKFatorSimples()`, que representa o Algoritmo 3 de Meijer *et al.* [7], o k -fator F é devolvido, encerrando o procedimento.

A implementação foi validada por meio de testes automatizados envolvendo diferentes classes de grafos. Com o objetivo de respeitar as limitações impostas pelo artigo em relação ao tamanho do grafo e à quantidade de arestas, foi desenvolvido um *script* em Python capaz de gerar instâncias aleatórias para avaliar a implementação proposta. Esses testes consideraram a busca por 2-fatores e 3-fatores em grafos conexos com 10, 50 e 100 vértices.

Ao todo, foram gerados 600 casos de teste, disponibilizados no repositório do projeto no *GitHub*. A execução desses testes produz um arquivo em formato *Markdown*, contendo os tempos médios de execução do algoritmo para cada quantidade de vértices considerada. Os resultados obtidos são apresentados na Tabela 3.

Tabela 3: Resultado dos testes em grafos aleatórios, buscando fatores com $k = 2$ e $k = 3$.

Número de vértices em G	Tempo de execução médio (s)
10	4
50	12
100	33,5

Além dos testes randomizados, também foram avaliados, ao longo do desenvolvimento do projeto, casos em que a existência de um k -fator é matematicamente impossível. Um exemplo é a busca por um 3-fator em um grafo K_5 , caso que viola o Lema do Aperto de Mãos, pois exigiria que a soma dos graus do subgrafo resultante fosse ímpar.

Os resultados obtidos indicam que a implementação é capaz de identificar corretamente tanto instâncias que admitem um k -fator quanto instâncias nas quais tal estrutura não existe. Dessa forma, os testes contribuem para validar a corretude da implementação e a robustez da solução desenvolvida. Por fim, ressalta-se que o custo computacional do algoritmo é dominado pela etapa de emparelhamento máximo, realizada neste trabalho por meio do algoritmo de Edmonds.

6 Conclusão

Este trabalho apresentou uma implementação do algoritmo para encontrar k -fatores simples em um grafo geral proposto por Meijer *et al.* [7]. Utilizando de conceitos de teoria dos grafos e com padrões de projeto bem definidos, como o uso da Programação Orientada a Objetos, foi possível implementar o algoritmo na linguagem C++ de maneira eficiente.

O principal desafio deste trabalho não esteve apenas na implementação do algoritmo apresentado, mas também na compreensão da estrutura teórica associada ao problema do k -fator e de suas implicações algorítmicas. Ressalta-se que esse conteúdo, assim como outros tópicos

abordados ao longo do trabalho, não é usualmente explorado nas disciplinas obrigatórias da graduação, o que exigiu um estudo teórico adicional. Além disso, a linguagem de programação escolhida também representou um desafio significativo. Embora C++ seja amplamente documentada e adequada para a implementação de algoritmos eficientes, sua sintaxe e seus recursos exigem maior cuidado em relação à organização do código, gerenciamento de tipos e estruturação das classes. Dessa forma, desenvolver a implementação seguindo, tanto quanto possível, os princípios da Programação Orientada a Objetos em C++ constituiu uma etapa importante do trabalho.

É importante ressaltar que, em seu artigo original, Meijer *et al.* [7] apresentam duas implementações distintas de *gadgets* originados a partir de vértice $v \in G$, visto que, para grandes valores de k , quando $k \geq \deg_G(v)/2$, ou grafos muito densos, o processo padrão de construção dessas estruturas pode resultar em um grafo inflado grande demais, que tornaria o processo mais custoso. Os autores propõem, portanto, uma implementação alternativa de *gadgets* para estas situações que não possuem vértices ou arestas internas. Tal implementação não foi contemplada neste trabalho. Além disso, é importante ressaltar que, apesar do Teorema dos Fatores Regulares 6 apresentar um certificado importante para quando um grafo pode ou não possuir um k -fator, o algoritmo proposto por Meijer *et al.* [7] o utiliza como forma de verificação, se valendo apenas do Teorema 8 para garantir a inexistência deste fator em G . Uma expansão possível do algoritmo proposto pelos autores seria relacionar as duas abordagens, apresentando qual seriam os conjuntos de vértices S e T que oferecem essa barreira e impedem o grafo de possuir um k -fator.

Este trabalho evidencia a importância do desenvolvimento de algoritmos eficientes para problemas fundamentais em Teoria dos Grafos. A obtenção de um k -fator pode atuar como etapa essencial na resolução de problemas mais amplos, tanto em contextos teóricos quanto em aplicações práticas. Entre esses contextos, destacam-se a busca por ciclos Hamiltonianos, procedimentos de stripificação e compressão de malhas em Computação Gráfica, problemas de alocação de recursos, entre outros.

Dessa forma, algoritmos capazes de computar k -fatores de maneira eficiente possuem relevância significativa. A abordagem estudada neste trabalho, proposta pelos autores e posteriormente implementada, apresenta uma solução para a computação de k -fatores simples a partir da construção de um grafo inflado e da redução do problema à busca por um emparelhamento perfeito.

Referências

- [1] J. Akiyama and M. Kano. Book of factors and factorizations of graphs. june. 2007. doi: 10.1007/978-3-642-21919-1.

- [2] C. Berge. Sur le couplage maximum d'un graphe. *CR Acad. Sci. Paris*, 247(258-259): 2–29, 1958.
- [3] J. A. Bondy and U. S. R. Murty. Graph theory. In *Graduate Texts in Mathematics*, 2008. URL <https://api.semanticscholar.org/CorpusID:195946075>.
- [4] P. Diaz-Gutierrez and M. Gopi. Quadrilateral and tetrahedral mesh stripification using 2-factor partitioning of the dual graph. *The Visual Computer*, 21:689–697, 2005. doi: 10.1007/s00371-005-0336-9.
- [5] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965. doi: 10.6028/jres.069b.013.
- [6] D. Eppstein and M. Gopi. Single-strip triangulation of manifolds with arbitrary topology. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04*, page 455–456, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138857. doi: 10.1145/997817.997888.
- [7] H. Meijer, Y. Núñez-Rodríguez, and D. Rappaport. An algorithm for computing simple k -factors. *Information Processing Letters*, 109(12):620–625, 2009. ISSN 0020-0190. doi: 10.1016/j.ipl.2009.02.013.
- [8] H. M. Mulder. Julius Petersen's theory of regular graphs. *Discrete Mathematics*, 100(1):157–175, 1992. ISSN 0012-365X. doi: 10.1016/0012-365X(92)90639-W.
- [9] C. H. Papadimitriou and M. Yannakakis. The Traveling Salesman Problem with Distances One and Two. *Mathematics of Operations Research*, 18(1):1–11, 1993. doi: 10.1287/moor.18.1.1.
- [10] J. Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891. URL <https://api.semanticscholar.org/CorpusID:123779343>.
- [11] P. A. Peterson and M. C. Loui. The general maximum matching algorithm of micali and vazirani. *Algorithmica*, 3(1):511–533, 1988. doi: 10.1007/BF01762129.
- [12] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947. doi: 10.1112/jlms/s1-22.2.107.
- [13] W. T. Tutte. The factors of graphs. *Canadian Journal of Mathematics*, 4:314–328, 1952. doi: 10.4153/CJM-1952-028-2.
- [14] W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of mathematics*, 6:347–352, 1954. doi: 10.4153/CJM-1954-033-3.

- [15] C. Umans. An algorithm for finding hamiltonian cycles in grid graphs without holes. 1996. URL <https://api.semanticscholar.org/CorpusID:117785391>.