UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Tális Breda

ESTUDO DA FAMÍLIA DE EMPARELHAMENTO EM GRAFOS

Florianópolis
2024

Tális Breda

ESTUDO DA FAMÍLIA DE EMPARELHAMENTO EM GRAFOS

Trabalho de Conclusão do Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Rafael de Santiago

Florianópolis
2024

RESUMO

Problemas de emparelhamento em grafos, definidos como uma seleção de conjuntos de arestas sem vértices em comum, possuem ampla relevância em áreas da computação, como visão computacional, e também fora dela, como na biologia. São bastante importantes em tarefas que envolvem correspondência de elementos, como a comparação e correspondência de elementos em conjuntos distintos, por exemplo em análise de dados estruturados. Devido à complexidade computacional frequentemente NP-completa desses problemas, são necessárias abordagens que conciliem precisão e eficiência. Considerando a quantidade de algoritmos e métodos presentes na literatura, torna-se difícil encontrar a maneira mais eficiente de resolver um problema específico. Este trabalho busca levantar, comparar e classificar métodos computacionais, destacando suas aplicações e limitações, com o objetivo de fornecer insights úteis para pesquisadores e profissionais na escolha de soluções eficazes para cenários práticos.

Palavras-chave: Emparelhamento em grafos, grafos bipartidos, visão computacional

ABSTRACT

Graph matching problems, defined as the selection of edge sets with no common vertices, are highly relevant in fields of computer science such as computer vision, as well as in areas outside of it, like biology. They are particularly important for tasks involving element matching, such as comparing and matching elements in distinct sets, for example, in structured data analysis. Due to the often NP-complete computational complexity of these problems, approaches that balance precision and efficiency are necessary. Given the large number of algorithms and methods in the literature, it is challenging to determine the most efficient way to solve a specific problem. This work aims to survey, compare, and classify computational methods, highlighting their applications and limitations, with the goal of providing useful insights for researchers and professionals in selecting effective solutions for practical scenarios.

Keywords: Graph matching, bipartite graphs, computer vision

Conteúdo

1 Introdução

Um emparelhamento em um grafo não-dirigido é definido como um conjunto de arestas sem pontas em comum. Em outras palavras, um emparelhamento é um conjunto M de arestas que satisfaz a seguinte propriedade: o grau de cada vértice no subconjunto M é no máximo 1 (FEOFILOFF, 2019). Os principais problemas relacionados a emparelhamentos geralmente envolvem grafos bipartidos, aqueles que podem ser particionados em dois conjuntos independentes U e V, onde todas as arestas conectam vértices de U com vértices de V.

Os problemas de emparelhamento têm aplicações significativas em várias áreas, como biologia computacional, onde podem ser empregados na diferenciação e classificação de estruturas proteicas (TAYLOR, 2002); redes sociais, para identificação de comunidades ou grupos (FAN, 2012); e visão computacional, para correspondências entre elementos de conjuntos distintos, como pontos em imagens ou vértices em malhas tridimensionais (HALLER et al., 2022).

Resolver problemas de emparelhamento em grafos é um desafio devido à sua complexidade computacional, frequentemente NP-completa, exigindo métodos inovadores para equilibrar precisão e eficiência. Estratégias clássicas, como o Problema de Atribuição Quadrática (QAP), modelam o emparelhamento como uma questão de otimização combinatória, empregando relaxações espectrais e heurísticas para encontrar soluções aproximadas de forma eficiente (YAN; YANG; HANCOCK, 2020). Por outro lado, avanços em transporte ótimo duplamente estocástico, como o algoritmo GOAT, têm demonstrado melhorias em robustez e velocidade, especialmente em grafos maiores (SAAD-ELDIN et al., 2021).

Estudos como esses ilustram a importância de investigar e comparar métodos computacionais que abordem a complexidade inerente ao problema de emparelhamento em grafos, contribuindo para aplicações como aprendizado de máquina e visão computacional. Essa abordagem torna-se essencial para desenvolver soluções adaptadas a diferentes cenários e com impacto direto na melhoria de algoritmos existentes.

Diante desse cenário, este trabalho visa sistematizar os principais problemas e métodos computacionais associados ao emparelhamento em grafos. Ele busca oferecer uma comparação detalhada entre suas aplicações e limitações, contribuindo para o entendimento aprofundado das ferramentas disponíveis e identificando oportunidades para o desenvolvimento de novas soluções. Essa abordagem torna-se ainda mais relevante à luz da importância prática desses algoritmos para resolver problemas reais em grande escala, conforme evidenciado por estudos recentes (SUSSMAN et al., 2021).

1.1 Objetivos

1.1.1 Objetivo geral

Este trabalho tem como objetivo principal levantar, organizar e comparar problemas de emparelhamento em grafos para diversos casos de uso, além de comparar diferentes métodos computacionais presentes na literatura para a solução desses problemas.

1.1.2 Objetivos específicos

- Mapear as variações e características de diferentes problemas de emparelhamento presentes na literatura.
- Investigar e classificar métodos computacionais presentes na literatura de acordo com suas vantagens, limitações e casos de uso.
- Criar um ambiente de testes unificado a fim de reduzir variabilidade nos resultados e proporcionar comparações consistentes
- Implementar e testar os algoritmos descritos, a fim de comparar desempenho e eficiência em diferentes situações
- Organizar os resultados em um formato que facilite o acesso e a compreensão dos tópicos abordados por pesquisadores e profissionais.

1.2 Delimitação do estudo

Os problemas de emparelhamento em grafos podem ser classificados de diversas maneiras, como sendo em grafos bipartidos ou não-bipartidos, ponderados ou não-ponderados, direcionados ou não-direcionados, problemas em multigrafos, problemas em hipergrafos, entre outros. Este trabalho focará em problemas que envolvem grafos simples, isto é, excluindo multigrafos e hipergrafos.

2 Fundamentação teórica

2.1 Definições sobre grafos

Um **grafo** G é uma uma estrutura definida por um par $G = (V, E)$, consistindo de um conjunto finito V de elementos chamados **vértices** (ou **nós** ou **pontos**) e um conjunto (ou família) E de pares não ordenados de vértices, chamados de **arestas** (JUNGNICKEL, 2008). O conjunto V é o conjunto de vértices de $G(VG)$ e E é a família de arestas de $G(EG)$.

No contexto deste projeto, assumimos que as arestas $e = u, v$ são pares não ordenados de vértices distintos. Se uma aresta e conecta os vértices a e b, diz-se que a e b são **adjacentes** ou **vizinhos**, e que a aresta e é **incidente** a a e b (SCHRIJVER, 2004).

O conceito de grafo é amplamente utilizado como uma representação abstrata concisa para estruturas complexas e para codificar relações binárias entre um conjunto de objetos.

2.1.1 Grafo direcionado

Se as relações entre os vértices forem assimétricas, utilizamos um **grafo direcionado** ou **dígrafo**, $D = (V, A)$. Neste caso, A é uma família de pares ordenados de vértices, chamados **arcos** (ou arestas direcionadas). Para um arco $e = (u, v)$, u é chamado o vértice inicial ou **cauda** (tail), e v é o vértice final ou **cabeça** (head) (MANBER, 1989)

2.1.2 Grafo ponderado

Um grafo $G = (V, E)$ é chamado **grafo ponderado** (ou com pesos) se uma **função de peso** (ou função de custo, ou função de comprimento) $w : E \to R$ está associada às arestas. Formalmente: $G = (V, E, w)$, onde $w : E \to R$. Geralmente, o peso $w(e)$ de uma aresta e é um valor real não negativo que representa o custo, comprimento ou capacidade associada àquela aresta (ROSEN; KREHER; STINSON, 1998).

Grafos ponderados podem ser representados por matrizes de adjacência de valor real A, onde a entrada A_{ij} é o peso w_{ij} da aresta (ZASLAVSKIY; VERT, 2009)

2.1.3 Grafo bipartido

Um grafo bipartido é um grafo não direcionado que pode ser facilmente colorido com apenas duas cores (MANBER, 1989)

Formalmente, um grafo $G = (V, E)$ é chamado **grafo bipartido** se o conjunto de vértices V puder ser particionado em dois subconjuntos disjuntos V_1 e V_2, chamados calsses de cor, tais que:

$$V = V_1 \cup V_2 \quad e \quad V_1 \cap V_2 = \emptyset \tag{1}$$

e todas as arestas $e \in E$ conectam um vértice em V_1 a um vértice em V_2. Ou seja, não existem arestas que conectem vértices dentro do mesmo subconjunto (DASGUPTA CHRISTOS H. PAPADIMITRIOU, 2006).

Observações:

- Um grafo é bipartido se e somente se não contém ciclos de comprimento ímpar (SCHRIJVER, 2004)
- O grafo bipartido completo $K_{m,n}$ possui m vértices em V_1 e n vértices em V_2, e contém todas as arestas possíveis entre V_1 e V_2 (JUNGNICKEL, 2008)

2.2 Problema Geral do Emparelhamento

O problema de **matching** (emparelhamento) em grafos é um problema fundamental na otimização combinatória

Um **matching** M em um grafo não-direcionado $G = (V, E)$ é um subconjunto de arestas $M \subseteq E$ tal que nenhum par de arestas em M compartilha um vértice comum (KLEINBERG; TARDOS, 2005). Em outras palavras, cada nó aparece em no máximo uma aresta de M.

- Um vértice é chamado **coberto** (*matched*) se for incidente a uma aresta em M; caso contrário, é **descoberto** (*unmatched* ou *exposed*) (CORMEN CHARLES E. LEISERSON, 2009)
- Um **matching de cardinalidade máxima** (Maximum Matching) é um matching com o maior número possível de arestas (CORMEN CHARLES E. LEISERSON, 2009). A cardinalidade máxima de um matching é denotada por $v(G)$ (SCHRIJVER, 2004)
- Um **matching perfeito** é um matching que cobre todos os vértices do grafo (KLEINBERG; TARDOS, 2005)
- O **problema de matching ponderado** (Weighted Matching Problem) envolve encontrar um matching para o qual a soma dos pesos das arestas é máxima. Em um grafo ponderado $G = (V, E, w)$, busca-se um $M \subseteq E$ que maximize $w(M)$ (MANDULAK SAYAN GHOSH; SLOTA, 2024)

2.2.1 Matching em grafos bipartidos

O problema de **Matching Bipartido** é o caso clássico de encontrar um matching de cardinalidade máxima em um grafo bipartido $G = (V, E)$, onde $V = X \cup Y$ (KLEINBERG; TARDOS, 2005)

- O matching em grafos bipartidos pode modelar situações de atribuição, como associar empregos (X) a máquinas (Y), ou professores (X) a cursos (Y), onde uma aresta indica uma capacidade de atribuição (KLEINBERG; TARDOS, 2005)
- O problema de matching ponderado em grafos bipartidos é equivalente ao **problema de atribuição (LAWLER, 2011)**, que historicamente motivou o desenvolvimento do **método Húngaro** (SCHRIJVER, 2004)

2.2.2 Formulação geral (emparelhamento e otimização)

Em um contexto mais amplo, o emparelhamento de grafos pode ser formalizado como um problema de otimização que busca maximizar a compatibilidade entre dois grafos G e G' (CAETANO JULIAN J. MCAULEY; SMOLA, 2009).

- O problema de graph matching é frequentemente abordado como um **problema de atribuição quadrática (QAP)**. Essa formulação busca maximizar uma função objetivo que combina termos de compatibilidade unária (nó-nó, $c_{ii'}$) e compatibilidade par a par (aresta-aresta, $d_{ii'jj'}$), sujeito a restrições de atribuição binária ($y_{ii'} \in 0, 1$). O termo quadrático codifica a preservação das relações (arestas) entre os nós (CAETANO JULIAN J. MCAULEY; SMOLA, 2009)
- Para grafos bipartidos, a determinação de um matching máximo pode ser resolvida de maneira eficiente e está intimamente ligada a problemas de *network flow* (fluxo em redes) (KLEINBERG; TARDOS, 2005)

3 Problemas de emparelhamento em grafos bipartidos

3.1 Emparelhamento de cardinalidade máxima

3.1.1 Definição do problema

Dado um grafo não-direcionado $G = (V, E)$, o problema de emparelhamento de cardinalidade máxima (MCM) busca encontrar um emparelhamento $M \subseteq E$ tal que o número de arestas em M seja maximizado,

ou seja, $|M|$ é o maior entre todos os emparelhamentos possíveis em G.

Um caso especial, mas importante, do MCM é o **Emparelhamento perfeito**, que é um emparelhamento onde todos os vértices do grafo são cobertos por exatamente uma aresta do emparelhamento. Para que um emparelhamento perfeito exista, o grafo deve ter um número par de vértices e o emparelhamento máximo deve ter cardinalidade igual a $|V|/2$ (JUNGNICKEL, 2008).

3.1.2 Propriedades

Antes de continuar para os algoritmos e soluções, é necessário entender algumas propriedades importantes

3.1.2.1 Emparelhamento máximo vs maximal

Um emparelhamento máximo é aquele que contém o maior número possível de arestas, enquanto um emparelhamento maximal é aquele que não pode ser aumentado adicionando mais arestas, mas não necessariamente é o maior possível (MANBER, 1989).

3.1.2.2 Caminhos aumentantes

Um caminho aumentante P em um grafo G é chamado de caminho aumentante se P tem tamanho ímpar, começa e termina em vértices livres (não cobertos por nenhuma aresta do emparelhamento atual) e alterna entre arestas que pertencem a M e arestas que não pertencem a M. A existência de um caminho aumentante indica que o emparelhamento atual não é máximo (SCHRIJVER, 2004)

3.1.3 Algoritmos exatos

Por ser em um ambiente mais restrito, o problema de emparelhamento de cardinalidade máxima tem diversas soluções em tempo polinomial. A seguir listadas algumas delas:

3.1.3.1 Algoritmo do caminho aumentante

O algoritmo de caminho aumentante se baseia em um lema proposto por Claude Berge, que afirma que um grafo G é maximo se e somente se não existe nenhum caminho aumentante em G. Este algoritmo é uma implementação direta deste lema, onde se busca repetidamente por caminhos aumentantes e se aumenta o emparelhamento até que nenhum caminho aumentante possa ser encontrado (LAWLER, 2011).

Começamos com um emparelhamento vazio $M = \emptyset$. Escolhemos um vértice livre u no lado esquerdo do grafo bipartido e tentamos encontrar um caminho aumentante P que começa em u e termina em um vértice livre no lado direito do grafo, respeitando a regra da alternância entre arestas em M e arestas fora de M. Se um caminho aumentante for encontrado, atualizamos o emparelhamento M invertendo as arestas ao longo do caminho P. Repetimos esse processo até que não seja possível encontrar mais caminhos aumentantes. O emparelhamento resultante será o emparelhamento de cardinalidade máxima (HALIM; HALIM, 2013).

Em seguida o algoritmo em pseudo-código:

Listing 1: Algoritmo de caminhos aumentantes (HALIM; HALIM, 2013)

```
// global variables
vi match, vis; // ('vi' is an alias for vector<int>)

int Aug(int l) { // return 1 if an augmenting path is found
    if (vis[l]) return 0; // return 0 otherwise
    vis[l] = 1;
    for (int j = 0; j < (int)AdjList[l].size(); j++) {
        int r = AdjList[l][j]; // edge weight not needed -> vector<vi> AdjList
        if (match[r] == -1 || Aug(match[r])) {
            match[r] = l; return 1; // found 1 matching
    } }
    return 0; // no matching
}
```

```
// inside int main()
    // build unweighted bipartite graph with directed edge left->right set
    int MCBM = 0;
    match.assign(V, -1); // V is the number of vertices in bipartite graph
    for (int l = 0; l < n; l++) { // n = size of the left set
        vis.assign(n, 0); // reset before each recursion
        MCBM += Aug(l);
    }
    printf("Found %d matchings\n", MCBM);
```

Como o algoritmo repete a busca por caminhos aumentantes para cada vértice do lado esquerdo do grafo, o tempo de execução total do algoritmo é $O(VE)$, onde V é o número de vértices e E é o número de arestas no grafo bipartido (HALIM; HALIM, 2013).

3.1.3.2 Redução ao problema de fluxo máximo

O problema de emparelhamento de cardinalidade máxima em grafos bipartidos pode ser eficientemente resolvido através de uma redução ao problema de fluxo máximo em redes. A ideia central é construir uma rede de fluxo a partir do grafo bipartido original, onde cada aresta do grafo bipartido é convertida em uma aresta com capacidade unitária na rede de fluxo. (LAWLER, 2011)

Primeiro, adicionamos um vértice fonte s e um vértice sumidouro t à rede. Conectamos o vértice fonte s a todos os vértices do conjunto esquerdo do grafo bipartido com arestas de capacidade 1. Em seguida, conectamos todos os vértices do conjunto direito do grafo bipartido ao vértice sumidouro t, também com arestas de capacidade 1. As arestas entre os conjuntos esquerdo e direito do grafo bipartido são mantidas na rede de fluxo, cada uma com capacidade 1 (HALIM; HALIM, 2013).

Com isso, temos agora um grafo de fluxo onde o objetivo é encontrar o fluxo máximo do vértice fonte s para o vértice sumidouro t. O valor do fluxo máximo encontrado nesta rede corresponde ao tamanho do emparelhamento máximo no grafo bipartido original. (LAWLER, 2011). Para encontrar o fluxo máximo, podemos utilizar algoritmos clássicos como o de Edmonds-Karp.

O algoritmo de Edmonds-Karp é o padrão para resolver o problema de fluxo máximo, utilizando buscas em largura (BFS) para encontrar caminhos aumentantes na rede residual. A seguir, o pseudo-código do algoritmo:

Listing 2: Algoritmo de Edmonds-Karp (HALIM; HALIM, 2013)

```
int res[MAX_V][MAX_V], mf, f, s, t; // global variables
vi p; // p stores the BFS spanning tree from s

void augment(int v, int minEdge) { // traverse BFS spanning tree from s->t
    if (v == s) { f = minEdge; return; } // record minEdge in a global var f
    else if (p[v] != -1) {
        augment(p[v], min(minEdge, res[p[v]][v]));
        res[p[v]][v] -= f; res[v][p[v]] += f;
    }
}

// inside int main(): set up 'res', 's', and 't' with appropriate values
    mf = 0; // mf stands for max_flow

    while (1) { // O(VE^2) (actually O(V^3 E) Edmonds Karp's algorithm
        f = 0;

        // run BFS, compare with the original BFS shown in Section 4.2.2
        vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
        p.assign(MAX_V, -1); // record the BFS spanning tree, from s to t!

        while (!q.empty()) {
            int u = q.front(); q.pop();
            if (u == t) break; // immediately stop BFS if we already reach sink t

            for (int v = 0; v < MAX_V; v++) // note: this part is slow
                if (res[u][v] > 0 && dist[v] == INF)
                    dist[v] = dist[u] + 1, q.push(v), p[v] = u; // 3 lines in 1!
```

```
        }

        augment(t, INF); // find the min edge weight 'f' in this path, if any
        if (f == 0) break; // we cannot send any more flow ('f' = 0), terminate
        mf += f; // we can still send a flow, increase the max flow!
    }
    printf("%d\n", mf);
```

Outra opção para resolver o problema de fluxo máximo é o algoritmo de Dinic, que é mais eficiente em muitos casos práticos, especialmente em grafos densos. O algoritmo de Dinic utiliza uma combinação de buscas em largura (BFS) para construir níveis na rede residual e buscas em profundidade (DFS) para encontrar caminhos aumentantes dentro desses níveis. O tempo de execução do algoritmo de Dinic é $O(E\sqrt{V})$ para grafos gerais, tornando-o uma escolha preferida para muitos problemas de fluxo máximo (HALIM; HALIM, 2013).

Este trabalho não tem uma implementação do algoritmo de Dinic. Ao invés disso, será apresentado o algoritmo de Hopcroft-Karp, que é uma variação especializada do algoritmo de Dinic para o problema de emparelhamento em grafos bipartidos, oferecendo uma solução mais eficiente para este caso específico.

3.1.3.3 Algoritmo de Hopcroft-Karp

O algoritmo de Hopcroft-Karp é um algoritmo eficiente para encontrar o emparelhamento máximo em grafos bipartidos. Ele melhora o desempenho do algoritmo de caminhos aumentantes ao encontrar múltiplos caminhos aumentantes em cada iteração, em vez de apenas um. O tempo de execução do algoritmo de Hopcroft-Karp é $O(E\sqrt{V})$, tornando-o significativamente mais rápido do que o algoritmo de caminhos aumentantes simples, especialmente em grafos grandes (HALIM; HALIM, 2013).

O algoritmo consiste em duas fases principais: a fase de construção de níveis e a fase de busca de caminhos aumentantes. Na fase de construção de níveis, uma busca em largura (BFS) é realizada a partir dos vértices livres no lado esquerdo do grafo bipartido para construir uma camada de níveis que ajuda a identificar os caminhos aumentantes mais curtos. Na fase de busca de caminhos aumentantes, uma busca em profundidade (DFS) é realizada para encontrar todos os caminhos aumentantes possíveis dentro da camada de níveis construída na fase anterior. Esses caminhos são então usados para aumentar o emparelhamento (HALIM; HALIM, 2013).

A seguir, o pseudo-código do algoritmo de Hopcroft-Karp:

Listing 3: Algoritmo de Hopcroft-Karp (implementação própria)

```
ALGORITMO Hopcroft-Karp(G, U, V):
    Para cada u em U: PairU[u] = NIL
    Para cada v em V: PairV[v] = NIL
    Matching = 0

    Enquanto BFS() for verdadeiro:
        Para cada u em U:
            Se PairU[u] == NIL:
                Se DFS(u) for verdadeiro:
                    Matching = Matching + 1

    Retornar Matching

------------------------------------------------------

FUNCAO BFS():
    Fila Q = vazia
    Para cada u em U:
        Se PairU[u] == NIL:
            Dist[u] = 0
            Q.push(u)
        SenAo:
            Dist[u] = INFINITO

    Dist[NIL] = INFINITO

```

```
    Enquanto Q nAo estiver vazia:
        u = Q.pop()

        // Se a distAncia atual for menor que a distancia para o NIL,
        // continuamos procurando. Se for maior, ja achamos um caminho mais curto antes.
        Se Dist[u] < Dist[NIL]:
            Para cada v adjacente a u:
                // Se v ja tem par, verificamos a distancia desse par
                Se Dist[PairV[v]] == INFINITO:
                    Dist[PairV[v]] = Dist[u] + 1
                    Q.push(PairV[v])

    // Retorna verdadeiro se alcancamos o NIL (ou seja, achamos um caminho aumentante livre)
    Retornar Dist[NIL] != INFINITO

--------------------------------------------------------

FUNCAO DFS(u):
    Se u != NIL:
        Para cada v adjacente a u:
            // So seguimos se o vizinho estiver na proxima "camada" valida (distancia + 1)
            Se Dist[PairV[v]] == Dist[u] + 1:
                Se DFS(PairV[v]) for verdadeiro:
                    PairV[v] = u
                    PairU[u] = v
                    Retornar VERDADEIRO

        // Otimizacao: Se nao achou caminho por u, marca como infinito para nao tentar de
            novo nesta fase
        Dist[u] = INFINITO
        Retornar FALSO

    Retornar VERDADEIRO
```

3.2 Problema de atribuição (Assignment Problem)

Talvez chamar de Problema de emparelhamento ponderado?

3.2.1 Descrição do problema

O problema de atribuição consiste em encontrar uma combinação ótima de atribuições entre dois conjuntos disjuntos, minimizando o custo total associado a essas atribuições. Exemplo: Considere N trabalhadores e N tarefas, onde cada trabalhador pode ser designado a exatamente uma tarefa, e cada tarefa deve ser atribuída a exatamente um trabalhador. O custo de atribuir o trabalhador i à tarefa j é representado por uma matriz de custos $W = [w_{ij}]$ (LAWLER, 2011). O objetivo é encontrar um conjunto de atribuições que minimize o custo total.

3.2.2 Propriedades

3.2.3 Algoritmos exatos

3.2.3.1 Método Húngaro

O método Húngaro é um algoritmo eficiente para resolver o problema de atribuição em tempo polinomial. Ele foi desenvolvido por Harold Kuhn em 1955 e é baseado no trabalho de Dénes Kőnig e Jenő Egerváry sobre emparelhamentos em grafos bipartidos (JUNGNICKEL, 2008).

O método se baseia em manter um conjunto de potenciais para os vértices dos dois conjuntos do grafo bipartido, sendo u_i o potencial do vértice i no conjunto esquerdo e v_j o potencial do vértice j no conjunto direito. A regra que o algoritmo mantém é que para cada aresta (i, j), a soma dos potenciais deve ser maior ou igual ao custo da aresta, ou seja, $u_i + v_j \geq w_{ij}$.

Cria-se um subgrafo de igualdade $H_{u,v}$ contendo apenas as arestas onde a soma dos potenciais é igual ao custo da aresta, ou seja, $u_i + v_j = w_{ij}$. O algoritmo então tenta encontrar um emparelhamento

máximo neste subgrafo de igualdade. Se o emparelhamento encontrado cobre todos os vértices do conjunto esquerdo, então ele é ótimo para o problema de atribuição original. Caso contrário, o algoritmo ajusta os potenciais para criar novas arestas de igualdade e repete o processo até encontrar um emparelhamento máximo que cubra todos os vértices do conjunto esquerdo (JUNGNICKEL, 2008).

Em seguida o pseudo-código do método Húngaro:

Algorithm 1: Hungarian Algorithm (JUNGNICKEL, 2008)

```
(1)  Procedure HUNGARIAN(n, w; mate)
(2)      for v ∈ V do mate(v) ← 0
(3)      for i = 1 to n do u_i ← max{w_ij : j = 1, ..., n}; v_i ← 0
(4)      nrex ← n
(5)      while nrex ≠ 0 do
(6)          for i = 1 to n do m(i) ← false; p(i) ← 0; δ_i ← ∞
(7)          aug ← false; Q ← {i ∈ S : mate(i) = 0}
(8)          repeat
(9)              remove an arbitrary vertex i from Q; m(i) ← true; j ← 1
(10)             while aug = false and j ≤ n do
(11)                 if mate(i) ≠ j' then
(12)                     if u_i + v_j − w_ij < δ_j then
(13)                         δ_j ← u_i + v_j − w_ij; p(j) ← i
(14)                         if δ_j = 0 then
(15)                             if mate(j') = 0 then
(16)                                 AUGMENT(mate, p, j'; mate)
(17)                                 aug ← true; nrex ← nrex − 1
(18)                             else
(19)                                 Q ← Q ∪ {mate(j')}
(20)                             end
(21)                         end
(22)                     end
(23)                 end
(24)                 j ← j + 1
(25)             end
(26)             if aug = false and Q = ∅ then
(27)                 J ← {i ∈ S : m(i) = true}; K ← {j' ∈ T : δ_j = 0}
(28)                 δ ← min{δ_j : j' ∈ T \ K}
(29)                 for i ∈ J do u_i ← u_i − δ
(30)                 for j' ∈ K do v_j ← v_j + δ
(31)                 for j' ∈ T \ K do δ_j ← δ_j − δ
(32)                 X ← {j' ∈ T \ K : δ_j = 0}
(33)                 if mate(j') ≠ 0 for all j' ∈ X then
(34)                     for j' ∈ X do Q ← Q ∪ {mate(j')}
(35)                 else
(36)                     choose j' ∈ X with mate(j') = 0
(37)                     AUGMENT(mate, p, j'; mate)
(38)                     aug ← true; nrex ← nrex − 1
(39)                 end
(40)             end
(41)         until aug = true
(42)     end
```

Algorithm 2: Augment Procedure (JUNGNICKEL, 2008)

```
(1) Procedure AUGMENT(mate, p, j'; mate)
(2)     repeat
(3)         i ← p(j); mate(j') ← i; next ← mate(i); mate(i) ← j'
(4)         if next ≠ 0 then
(5)             j' ← next
(6)         end
(7)     until next = 0
```

3.2.3.2 Jonker-Volgenant

O algoritmo de Jonker-Volgenant é uma evolução otimizada para a resolução do problema de atribuição linear, proposto por Roy Jonker e Anton Volgenant em 1987. Ele foi desenvolvido para superar o desempenho prático do método Húngaro, especialmente em grafos densos, combinando a teoria de dualidade com estratégias de inicialização e busca mais eficientes (JONKER; VOLGENANT, 1987).

Assim como o método Húngaro, este algoritmo baseia-se na manutenção de potenciais duais (u_i e v_j) e respeita as condições de folga complementar. No entanto, sua principal distinção conceitual é o uso da estratégia de Caminho Aumentante Mais Curto (Shortest Augmenting Path). Ao invés de construir emparelhamentos máximos em fases distintas, o algoritmo foca em encontrar o caminho de custo mínimo que conecta uma linha não atribuída a uma coluna livre.

O processo inicia com heurísticas avançadas (conhecidas como redução de colunas e transferência de redução) para resolver rapidamente as atribuições triviais. Para os vértices restantes, o algoritmo realiza uma busca similar ao algoritmo de Dijkstra: ele explora o grafo para encontrar o caminho aumentante mais curto, atualizando os potenciais duais simultaneamente durante a busca. Isso garante que, a cada iteração, o emparelhamento seja aumentado pelo menor custo possível até que a solução ótima completa seja atingida.

3.2.3.3 Redução para problema de fluxo de custo mínimo

Similarmente ao que foi feito para o problema de emparelhamento de cardinalidade máxima, o problema de atribuição pode ser resolvido através de uma redução ao problema de fluxo de custo mínimo. Nesta abordagem, o grafo bipartido é transformado em uma rede de fluxo onde cada aresta tem uma capacidade unitária e um custo associado, que corresponde ao custo da atribuição original (LAWLER, 2011).

Para resolver

3.2.3.4 Algoritmo simplex

3.3 Problema de atribuição gargalo

O problema de atribuição gargalo, também conhecido como **Problema de Emparelhamento Min-Max** ou **Bottleneck Assignment problem**, é uma variação do problema de atribuição tradicional. O objetivo deste problema é encontrar, em um grafo bipartido ponderado, um emparelhamento de cardinalidade máxima (maior número possível de arestas) no qual o mínimo dos pesos das arestas do emparelhamento seja maximizado (LAWLER, 2011).

Por exemplo. Considere N trabalhadores e N estações de trabalho. w_{ij} representa a eficiência do trabalhador i na estação de trabalho j. A eficiência total da produção é limitada pela eficiência do trabalhador menos eficiente. O objetivo é atribuir trabalhadores às estações de trabalho de forma a maximizar a eficiência mínima entre todas as atribuições. Para resolver este problema, podemos utilizar o método de threshold (LAWLER, 2011).

3.3.1 Problema do emparelhamento estável (Stable Marriage Problem)

Referências

CAETANO JULIAN J. MCAULEY, L. C. Q. V. L. T. S.; SMOLA, A. J. Learning graph matching. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, IEEE Computer Society, United States, v. 31, n. X, December 2009. ISSN 0162-8828.

CORMEN CHARLES E. LEISERSON, R. L. R. C. S. T. H. *Introduction to Algorithms*. 3rd. ed. Cambdrige, MA: The MIT Press, 2009. ISBN 978-0-262-53305-8.

DASGUPTA CHRISTOS H. PAPADIMITRIOU, U. V. V. S. *Algorithms*. 1st. ed. New York, NY: McGraw-Hill Education, 2006. ISBN 978-0073523408.

FAN, W. Graph pattern matching revised for social network analysis. In: *Proceedings of the 15th International Conference on Database Theory*. New York, NY, USA: Association for Computing Machinery, 2012. (ICDT '12), p. 8–21. ISBN 9781450307918. Disponível em: ⟨https://doi.org/10.1145/2274576.2274578⟩.

FEOFILOFF, P. *Emparelhamento em Grafos Bipartidos*. 2019. IME-USP. Disponível em: ⟨https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/matching-bipartite.html⟩.

HALIM, S.; HALIM, F. *Competitive Programming 3: The New Lower Bound of Programming Contests*. 3rd. ed. [S.l.]: Lulu.com, 2013. Self-published by the authors. ISBN 978-1482852483.

HALLER, S. et al. *A Comparative Study of Graph Matching Algorithms in Computer Vision*. 2022. Disponível em: ⟨https://arxiv.org/abs/2207.00291⟩.

JONKER, R.; VOLGENANT, A. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, v. 38, n. 4, p. 325–340, 1987. Disponível em: ⟨https://doi.org/10.1007/BF02278710⟩.

JUNGNICKEL, D. *Graphs, Networks and Algorithms*. 3rd. ed. Berlin, Germany: Springer, 2008. ISBN 978-354727798.

KLEINBERG, J.; TARDOS Éva. *Algorithm Design*. 1st. ed. Boston, MA: Pearson, 2005. ISBN 978-0321295354.

LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids*. New York, NY: Dover Publications, 2011. Reprint edition. ISBN 978-0486414539.

MANBER, U. *Introduction to Algorithms: A Creative Approach*. 1st. ed. Reading, MA: Addison-Wesley, 1989. ISBN 978-0201120370.

MANDULAK SAYAN GHOSH, S. M. F. M. H. M.; SLOTA, G. Efficient weighted graph matching on gpus. *SC24*, IEEE Computer Society, United States, November 2024. ISSN 0162-8828.

ROSEN, K. H.; KREHER, D. L.; STINSON, D. R. *Discrete Mathematics and Its Applications*. 1st. ed. Boca Raton, FL: CRC Press, 1998. ISBN 978-0849339882.

SAAD-ELDIN, A. et al. *Graph Matching via Optimal Transport*. 2021. Disponível em: ⟨https://arxiv.org/abs/2111.05366⟩.

SCHRIJVER, A. *Combinatorial Optimization: Polyhedra and Efficiency*. 1st. ed. Berlin, Germany: Springer, 2004. ISBN 3540204563.

SUSSMAN, D. L. et al. *Overview of Graph Matching Challenges and Approaches*. 2021. Acessado em: 26 nov. 2024. Disponível em: ⟨https://www.ll.mit.edu/⟩.

TAYLOR, W. R. Protein structure comparison using bipartite graph matching and its application to protein structure classification. *Molecular and Cellular Proteomics*, American Society for Biochemistry and Molecular Biology, United States, v. 1, n. 4, April 2002. ISSN 1535-9476.

YAN, J.; YANG, S.; HANCOCK, E. Learning for graph matching and related combinatorial optimization problems. In: BESSIERE, C. (Ed.). *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2020. p. 4988–4996. Survey track. Disponível em: ⟨https://doi.org/10.24963/ijcai.2020/694⟩.

ZASLAVSKIY, F. B. M.; VERT, J.-P. A path following algorithm for the graph matching problem. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, IEEE Computer Society, United States, v. 31, n. 12, December 2009. ISSN 0162-8828.