Alocação Eficiente de Tarefas em Grafos Bipartidos: Implementando os Algoritmos de Ford-Fulkerson e Edmonds-Karp

Lucas Prado Ribeiro, Lucas Anderson Ladislau Aguiar

***Abstract*—This article presents and describes a practical application for the Network Flow problem: task allocation using bipartite graphs. The scenario was modeled using a bipartite graph for two sets: the set of employees and the set of tasks. The work implements the Ford-Fulkerson and Edmonds-Karp algorithms to visualize the augmenting paths.**

***Index Terms*—Bipartite Graphs, Network Flow Algorithms, Ford-Fulkerson, Edmonds-Karp, Task Allocation.**

I. Introdução

O Problema do fluxo máximo é um dos tópicos mais estudados na teoria dos grafos e na otimização combinatória, com aplicações que abrangem diversas áreas, como redes de comunicação, logística e alocação de recursos. Esse problema consiste em determinar a quantidade máxima de fluxo que pode ser transportada de uma fonte a um sumidouro em uma rede, onde cada aresta possui uma capacidade máxima. A solução desse problema não apenas fornece insights sobre a eficiência das redes, mas também serve como base para resolver problemas mais complexos.

Alguns problemas combinatórios podem ser facilmente expressos como problemas de fluxo máximo. Um exemplo disso é o emparelhamento máximo em grafos bipartidos, que busca encontrar a maior correspondência possível entre dois conjuntos disjuntos de vértices. Em contextos práticos, isso pode representar a alocação de tarefas a funcionários, onde cada tarefa deve ser atribuída a um único funcionário que possui as habilidades necessárias para completá-la. A interseção entre esses dois problemas se torna evidente quando consideramos a modelagem de alocação de tarefas como um grafo bipartido, onde as arestas representam as capacidades de realização das tarefas pelos funcionários.

Neste artigo, exploraremos a relação entre o problema do fluxo máximo e o emparelhamento máximo em grafos bipartidos, destacando como técnicas de fluxo em rede podem ser aplicadas para resolver problemas de alocação de tarefas. Discutiremos algoritmos clássicos, como Ford-Fulkerson e Edmonds-Karp, que são utilizados para encontrar o fluxo máximo e, consequentemente, o emparelhamento máximo. Além disso, apresentaremos um exemplo prático e casos de uso que ilustram a eficácia dessas abordagens na otimização de processos em diversas aplicações do mundo real.

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received April 19, 2021; revised August 16, 2021.

II. O problema de emparelhamento máximo em grafo bipartido

Às vezes, um grafo tem a propriedade que seu conjunto de vértices pode ser dividido em dois subconjuntos disjuntos, tal que cada aresta conecta um vértice de um destes subconjuntos a um vertice do outro subconjunto. Um grafo simples G é dito **bipartido** [4] se o seu conjunto V de vértices pode ser dividido em dois conjuntos disjuntos V1 e V2 tal que cada aresta do grafo conecta um vértice em V1 e um vértice em V2 (de modo que nenhuma aresta em G conecta dois vértices, seja em V1, seja em V2). Quando essa condição é válida, chamamos o par (V1, V2) de bipartição do conjunto de vértices V de G.

Agora dado um grafo não dirigido $G = (V, E)$, um emparelhamento é um subconjunto de arestas $M \subseteq E$ tal que, para todos os vértices $v \in V$, no máximo uma aresta de M é incidente em v. Dizemos que um vértice $v \in V$ é emparelhado pelo emparelhamento M se alguma aresta em M é incidente em v; caso contrário, v é não emparelhado. Um emparelhamento máximo é um emparelhamento de cardinalidade máxima, isto é, um emparelhamento M tal que, para qualquer emparelhamento M', temos $|M| \geq |M'|$. Nesta seção, restringiremos nossa atenção a determinar emparelhamentos máximos em grafos bipartidos: grafos nos quais o conjunto de vértices pode ser particionado em $V = L \cup R$, onde L e R são disjuntos e todas as arestas em E passam entre L e R. Suporemos ainda que todo vértice em V tem no mínimo uma aresta incidente. [3] A Figura 1 ilustra a noção de emparelhamento em um grafo bipartido.

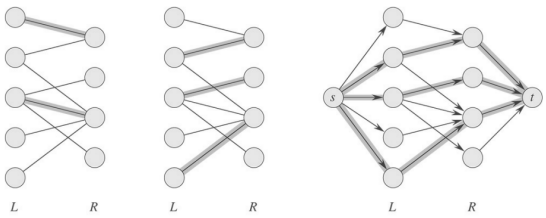

Fig. 1. Um grafo bipartido $G = (V, E)$ com bipartição de vértices $V = L \cup R$.

III. Rede de Fluxo

Primeiro, vamos definir o que é uma rede de fluxo, um fluxo e um fluxo máximo.

Uma rede é um grafo direcionado $G = (V, E)$, com vértices V e arestas E, combinado com uma função c, que atribui a cada aresta $e \in E$ um valor inteiro não negativo, a capacidade

de e. Essa rede é chamada de rede de fluxo se adicionalmente rotularmos dois vértices, um como fonte (s) e outro como dreno (t).

Um fluxo em uma rede de fluxo é uma função f, que novamente atribui a cada aresta e um valor inteiro não negativo, conhecido como fluxo. A função precisa preencher as duas condições seguintes:

1. O fluxo de uma aresta não pode exceder a capacidade:

$$f(e) \leq c(e)$$

2. A soma do fluxo de entrada de um vértice v deve ser igual à soma do fluxo de saída de v, exceto nos vértices de origem (s) e destino (t):

$$\sum_{(v,u)\in E} f((v,u)) = \sum_{(u,v)\in E} f((u,v))$$

O vértice de origem s tem apenas fluxo de saída, e o vértice do dreno t tem apenas fluxo de entrada.

É fácil ver que a seguinte equação é válida:

$$\sum_{(s,u)\in E} f((s,u)) = \sum_{(u,t)\in E} f((u,t))$$

Uma boa analogia para uma rede de fluxo é visualizar as arestas como canos de água. A capacidade de uma aresta é a quantidade máxima de água que pode fluir pelo cano por segundo, e o fluxo de uma aresta é a quantidade de água que atualmente flui pelo cano por segundo. Isso motiva a primeira condição de fluxo: não pode fluir mais água por um cano do que sua capacidade. Os vértices agem como junções, onde a água sai de alguns canos e é distribuída para outros canos. Isso motiva a segunda condição de fluxo: toda a água que entra em um vértice deve ser distribuída para outros canos; ela não pode desaparecer ou aparecer magicamente. A fonte s é a origem de toda a água, e a água só pode escoar para o dreno t.

O valor do fluxo de uma rede é a soma de todos os fluxos que são produzidos na fonte s, ou, de maneira equivalente, a soma de todos os fluxos que são consumidos pelo dreno t. Um fluxo máximo é um fluxo com o valor máximo possível. Encontrar esse fluxo máximo de uma rede de fluxo é o problema que queremos resolver.

IV. Análise de complexidade

A. Algoritmo de Ford-Fulkerson

O algoritmo de Ford-Fulkerson consiste em encontrar um caminho aumentante p na rede residual, e pode ser implementado utilizando diferentes métodos para localizar esses caminhos. No projeto, foi adotada a abordagem de busca em profundidade (DFS) para encontrar esses caminhos. Uma vez encontrado o caminho aumentante, o fluxo f em cada aresta desse caminho é atualizado, respeitando a capacidade residual das arestas envolvidas.

É importante observar que, se dois vértices u e v não estiverem conectados diretamente por uma aresta, o fluxo $f[u, v]$ entre eles deve ser considerado igual a zero. O algoritmo Ford-Fulkerson utilizado no projeto é uma implementação do pseudocódigo clássico Ford-Fulkerson-Method [2], que termina quando não for mais possível encontrar um caminho aumentante na rede residual mostrado no Algorithm 1.

Algorithm 1 Ford-Fulkerson Algorithm.

FORD-FULKERSON(G, s, t)
  $f \leftarrow 0$ **// fluxo inicial**
  $G_f \leftarrow G$ **// grafo residual**

  while existem caminhos aumentantes p de s a t em G_f **do**
    $c_p \leftarrow$ CAPACIDADE(p) **// capacidade do caminho**
    $f \leftarrow f + c_p$ **// atualiza o fluxo**
    ATUALIZAR(G_f, p, c_p) **// atualiza o grafo residual**
  end while

  return f **// fluxo máximo**

O tempo de execução do algoritmo depende da maneira como o caminho aumentante é determinado. Em alguns casos, o algoritmo pode nunca terminar (se as arestas tiverem capacidades irracionais) ou pode demorar muito tempo para encontrar o valor de fluxo máximo. Para garantir que o algoritmo execute em tempo polinomial, deve-se utilizar a Busca em Largura para determinar o caminho aumentante.

Se as capacidades das arestas consistirem em números inteiros, no pior caso o caminho será aumentado de uma em uma unidade a cada iteração. O tempo de execução do algoritmo será $O(E \cdot |f^*|)$, onde f^* corresponde ao valor do fluxo máximo encontrado pelo algoritmo.

A implementação do algoritmo deve utilizar uma lista de adjacências para representar a rede $G = (V, E)$. Assim, o tempo para encontrar um caminho na rede residual será $O(E)$. Como o laço `while` na linha 4 é executado em tempo $O(E)$, e será executado no máximo $|f^*|$ vezes (o valor do fluxo aumenta pelo menos uma unidade a cada iteração), o tempo de execução total será $O(E \cdot |f^*|)$.

B. Algoritmo de Edmonds-Karp

O algoritmo de Edmonds-Karp consegue melhorar o limite do algoritmo de Ford-Fulkerson escolhendo sempre a distância do caminho mais curto desde s até t na rede residual utilizando **BFS** para encontrar caminhos aumentado.

A complexidade do algoritmo pode ser analisada independentemente do fluxo máximo. O algoritmo é executado em tempo $O(VE^2)$, mesmo quando as capacidades são irracionais. A intuição por trás disso é que, a cada vez que encontramos um caminho aumentante, uma das arestas é saturada, e a distância dessa aresta até o vértice s será maior se ela voltar a aparecer em um caminho aumentante mais tarde. O comprimento dos caminhos simples é limitado por V.

C. Vantagens da Utilização dos Algoritmos

A escolha dos algoritmos Ford-Fulkerson e Edmonds-Karp para o problema de alocação de tarefas em grafos bipartidos se baseia na maneira eficiente como o problema foi modelado, levando em conta a restrição de que cada tarefa só pode ser concluída por funcionários específicos, conforme determinado

por uma lista pré-definida. Essa abordagem estruturada garante que a alocação de tarefas seja feita de forma correta e otimizada, respeitando as limitações impostas pelo problema.

Em contrapartida, uma abordagem aleatória, que tenta alocar tarefas aos funcionários de maneira não determinística, não considera essas restrições de maneira adequada. Como resultado, essa abordagem pode gerar soluções subótimas, nas quais tarefas permanecem não alocadas ou a carga de trabalho é distribuída de forma ineficiente ou até incorreta. Além disso, o fato de que não há garantia de que a alocação respeitará a lista de funcionários aptos para cada tarefa pode comprometer ainda mais a eficiência do sistema.

V. Avaliação Experimental

Nesta seção, são apresentados os resultados da avaliação experimental comparativa entre os algoritmos Edmonds-Karp e Ford-Fulkerson, implementados para resolver o problema de alocação de tarefas em grafos bipartidos. O objetivo é mensurar a eficiência dos algoritmos em termos de tempo de execução para diferentes tamanhos de entrada.

A. Metodologia

A avaliação experimental foi conduzida utilizando dois programas distintos: um baseado no algoritmo Edmonds-Karp e outro no Ford-Fulkerson. O problema foi modelado de forma que temos um conjunto de tarefas que são alocadas de acordo com a aptidão dos funcionários para cumpri-las. As execuções foram realizadas em uma máquina com especificações padrão, e os testes seguiram os parâmetros estabelecidos nos arquivos de entrada `.txt`, que especificam o número de trabalhadores e tarefas a serem alocados em cada caso de teste.

Para cada instância de teste, os programas foram executados seis vezes, e o tempo de execução médio foi registrado para minimizar o impacto de flutuações ocasionais no desempenho. O fluxo da execução foi automatizado por meio de um script em Python, que realiza a execução dos binários com os dados de entrada e coleta os tempos de execução. O script também é capaz de gerar gráficos que comparam os tempos de execução médios entre os algoritmos. O grafo final de atribuições pode ser visualizado de forma ilustrativa através do uso da biblioteca GraphViz.

B. Resultados

Os tempos médios de execução foram registrados para diversos tamanhos de entrada (número de trabalhadores e tarefas). Estes dados foram utilizados para gerar gráficos de desempenho, comparando diretamente os algoritmos Edmonds-Karp e Ford-Fulkerson. Gráficos específicos para cada algoritmo foram gerados, evidenciando o comportamento de cada um conforme o número de tarefas e trabalhadores aumenta conforme mostrado nas Figuras 2 e 3.

A Tabela I resume os tempos médios de execução obtidos para cada conjunto de testes. Observa-se que o algoritmo Ford-Fulkerson apresenta uma performance superior em relação ao Edmonds-Karp, particularmente à medida que o número de nós aumenta, demonstrando que as otimizações aplicadas resultaram em uma redução significativa do tempo de execução.

TABLE I
Resultados dos Testes

Nº de Funcionários	Nº de Tarefas	Tempo Médio (s)	Algoritmo
50	48	0.00066	Karp
50	48	0.00037	Ford
100	100	0.00367	Karp
100	100	0.00325	Ford
200	196	0.01986	Karp
200	196	0.01522	Ford
300	302	0.06847	Karp
300	302	0.05094	Ford
400	398	0.15653	Karp
400	398	0.11512	Ford
500	498	0.42787	Karp
500	498	0.21266	Ford
600	602	0.67784	Karp
600	602	0.38153	Ford
700	702	1.07518	Karp
700	702	0.68378	Ford
800	796	1.48254	Karp
800	796	0.87927	Ford
900	902	1.65518	Karp
900	902	1.27416	Ford

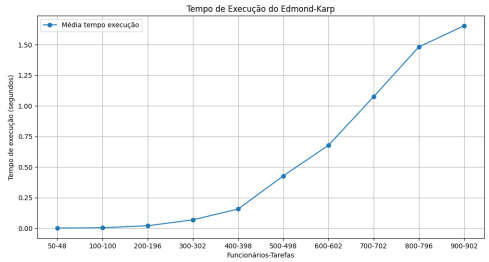

Fig. 2. Tempo de execução do algoritmo Edmonds-Karp.

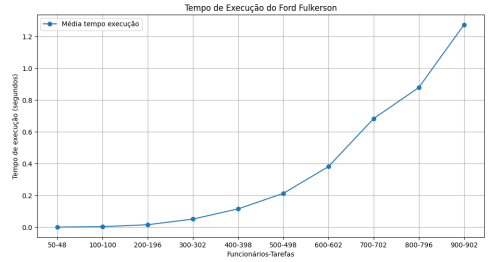

Fig. 3. Tempo de execução do algoritmo Ford-Fulkerson.

C. Análise

Os resultados indicam que, embora o Edmonds-Karp seja conhecido pela sua robustez, o Ford-Fulkerson mostrou-se mais eficiente nessa situação de resolução do problema de alocação de tarefas em grafos bipartidos. É válido ressaltar que, apesar de o Ford-Fulkerson ter sido mais rápido neste problema, ele pode apresentar situações em que não seja a opção mais eficiente, enfrentando alguns gargalos, conforme explicado anteriormente.

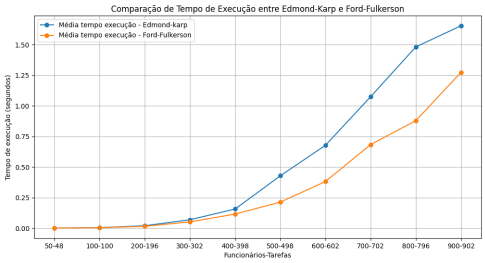

Fig. 4. Comparação de tempo de execução entre o algoritmo Edmonds-Karp e Ford-Fulkerson.

D. Visualização e Armazenamento dos Resultados

As visualizações geradas pelos gráficos demonstram de forma clara as pequenas diferenças de desempenho entre os algoritmos, e podem ser vistas na Figura 4. Tanto o gráfico da Figura 2 quanto o da Figura 3 demonstram comportamentos semelhantes, apresentando um baixo tempo de execução nos casos menores e um aumento gradual conforme o número de tarefas e funcionários cresce..

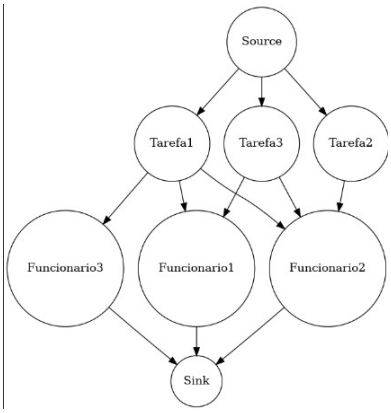

Fig. 5. Exemplo de visualição gráfica da alocação de tarefas

O projeto completo se encontra nesse link do github https://github.com/Lucasx10/L_Prado_L_Anderson_FinalProject_AA_RR_2024

VI. Conclusão

Este artigo apresentou uma comparação entre algoritmos de fluxo de rede para o problema de alocação de tarefas em grafos bipartidos, utilizando os algoritmos de Ford-Fulkerson e Edmonds-Karp para encontrar o fluxo máximo. Os resultados da avaliação experimental destacaram que, embora ambos os algoritmos sejam eficazes para resolver o problema, a implementação do Ford-Fulkerson mostrou-se mais rápida que Edmonds-Karp, principalmente à medida que a complexidade dos grafos aumentava.

A principal vantagem dos algoritmos baseados em fluxo de rede, como os que utilizamos, é que eles garantem uma solução ótima para o problema de emparelhamento máximo, ao contrário de abordagens aleatórias, que podem resultar em alocações subótimas e ineficientes. Algoritmos randômicos não possuem garantia de encontrar o emparelhamento máximo, podendo levar a uma má distribuição das tarefas, especialmente em cenários onde há restrições rígidas de capacidades ou habilidades específicas. Em contraste, o uso de algoritmos de fluxo como Ford-Fulkerson e Edmonds-Karp asseguram a máxima utilização dos recursos disponíveis, minimizando o tempo de execução e otimizando a alocação.

Portanto, a escolha de algoritmos de fluxo para resolver esse tipo de problema não apenas proporciona eficiência computacional, mas também oferece garantias de solução ótima, algo que é essencial em contextos de alocação de recursos críticos, como o gerenciamento de equipes e tarefas em projetos complexos.

References

[1] J. Edmonds and R. M. Karp, 'Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.
[2] L. R. Ford and D. R. Fulkerson, 'Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Algoritmos: teoria e prática*, 2nd ed. Rio de Janeiro, Brazil: Editora Campus, 2002.
[4] K. H. Rosen, *Mathematics Discrete and Its Applications*, 6th ed. New York, NY, USA: McGraw-Hill, 2012.