UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL DE MATOS FIGUEREDO

APLICAÇÕES PRÁTICAS DOS ALGORITMOS DE FLUXO MÁXIMO

CAMPINA GRANDE - PB

2024

DANIEL DE MATOS FIGUEREDO

APLICAÇÕES PRÁTICAS DOS ALGORITMOS DE FLUXO MÁXIMO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Professor Dr. Rohit Gheyi

CAMPINA GRANDE - PB

2024

DANIEL DE MATOS FIGUEREDO

APLICAÇÕES PRÁTICAS DOS ALGORITMOS DE FLUXO MÁXIMO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Rohit Gheyi

Orientador – UASC/CEEI/UFCG

Franklin de Souza Ramalho

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

Nos desafios enfrentados na resolução de problemas, a mudança de perspectiva pode revelar soluções mais simples. Uma abordagem comum envolve a representação de problemas complexos por meio de Redes de Fluxo, permitindo a aplicação de algoritmos de Fluxo Máximo. Algoritmos de fluxo possuem importância significativa para a resolução de problemas em várias áreas, tendo um papel imprescindível no que se diz respeito à otimização de soluções. Entretanto, existe uma certa dificuldade por parte dos desenvolvedores em saber quando e como utilizá-los. Neste trabalho, o objetivo é mostrar aplicações práticas do uso de algoritmos de Fluxo Máximo para ajudar os profissionais de software a entenderem mais cenários de aplicação. Este artigo se concentra na análise de problemas suscetíveis à modelagem por grafos e na resolução por meio de algoritmos de Fluxo Máximo, demonstrando como esses problemas podem ser formulados em termos de redes de fluxo. Entre os problemas analisados estão problemas de emparelhamento, como a atribuição de *tasks* para *workers* e distribuição balanceada de carros para passageiros em aplicativos de viagem. Além disso, será mostrado uma variação do problema *Baseball Elimination* visando calcular a possibilidade de vitória de competidores em um torneio de xadrez. Ao destacar aplicações práticas de algoritmos como Ford-Fulkerson e Edmonds-Karp, são oferecidos *insights* valiosos, como por exemplo a possibilidade de representar determinadas relações como grafos bipartidos. Desta forma mostrando a utilidade e eficiência desse tipo de abordagem na resolução de uma variedade de problemas do mundo real.

REAL LIFE APPLICATIONS OF MAXIMUM FLOW ALGORITHMS

ABSTRACT

In the challenges faced during problem-solving, a change in perspective can reveal simpler solutions. A common approach involves representing complex problems through Flow Networks, enabling the application of Maximum Flow algorithms. Flow algorithms hold significant importance for problem-solving in various fields, playing an indispensable role in solution optimization. However, developers often struggle with knowing when and how to use them. In this work, the objective is to showcase practical applications of Maximum Flow algorithms to help software professionals understand more application scenarios. This article focuses on the analysis of problems amenable to graph modeling and resolution through Maximum Flow algorithms, demonstrating how these problems can be formulated in terms of flow networks. Among the problems analyzed are matching problems, such as task assignment for workers and balanced distribution of cars for passengers in travel applications. Additionally, a variation of the Baseball Elimination problem will be shown, aiming to calculate the likelihood of competitors winning in a chess tournament. By highlighting practical applications of algorithms like Ford-Fulkerson and Edmonds-Karp, valuable insights are offered, such as the possibility of representing certain relationships as bipartite graphs. Thus, demonstrating the usefulness and efficiency of this type of approach in solving a variety of real-world problems.

Aplicações práticas dos Algoritmos de Fluxo Máximo

Daniel de Matos Figueredo
Universidade Federal de Campina Grande
Campina Grande, Brasil
daniel.figueredo@ccc.ufcg.edu.br

Rohit Gheyi
Universidade Federal de Campina Grande
Campina Grande, Brasil
rohit@dsc.ufcg.edu.br

RESUMO

Nos desafios enfrentados na resolução de problemas, a mudança de perspectiva pode revelar soluções mais simples. Uma abordagem comum envolve a representação de problemas complexos por meio de Redes de Fluxo, permitindo a aplicação de algoritmos de Fluxo Máximo. Algoritmos de fluxo possuem importância significativa para a resolução de problemas em várias áreas, tendo um papel imprescindível no que se diz respeito à otimização de soluções. Entretanto, existe uma certa dificuldade por parte dos desenvolvedores em saber quando e como utilizá-los. Neste trabalho, o objetivo é mostrar aplicações práticas do uso de algoritmos de Fluxo Máximo para ajudar os profissionais de software a entenderem mais cenários de aplicação. Este artigo se concentra na análise de problemas suscetíveis à modelagem por grafos e na resolução por meio de algoritmos de Fluxo Máximo, demonstrando como esses problemas podem ser formulados em termos de redes de fluxo. Entre os problemas analisados estão problemas de emparelhamento, como a atribuição de *tasks* para *workers* e distribuição balanceada de carros para passageiros em aplicativos de viagem. Além disso, será mostrado uma variação do problema *Baseball Elimination* visando calcular a possibilidade de vitória de competidores em um torneio de xadrez. Ao destacar aplicações práticas de algoritmos como Ford-Fulkerson e Edmonds-Karp, são oferecidos *insights* valiosos, como por exemplo a possibilidade de representar determinadas relações como grafos bipartidos. Desta forma mostrando a utilidade e eficiência desse tipo de abordagem na resolução de uma variedade de problemas do mundo real.

PALAVRAS-CHAVE

Grafos, Grafos Bipartidos, Redes de Fluxo, Fluxo Máximo, Algoritmos, Modelagem, Complexidade, Aplicações

1 INTRODUÇÃO

A teoria dos grafos é uma ferramenta poderosa e versátil para modelar e resolver uma ampla variedade de problemas de diversas áreas. Nesse contexto existe o problema do Fluxo Máximo, formulado pela primeira vez por Harris e Ross, representando um modelo simplificado do fluxo de tráfego ferroviário na antiga União Soviética [6]. O problema do Fluxo Máximo envolve determinar a quantidade máxima de fluxo que pode ser enviado através de uma rede, respeitando as restrições de capacidade. Além da abordagem original, várias outras situações do mundo real podem ser mapeadas para o problema do Fluxo Máximo.

Neste artigo serão apresentadas três situações em que podemos modelar o problema como uma rede e resolvê-lo com Fluxo Máximo, o primeiro será um problema de emparelhamento simples, em que se deseja fazer uma distribuição de *tasks* entre *workers* de forma a maximizar a quantidade de atribuições. Outra situação exposta será no contexto dos aplicativos de viagem, exemplificando como podemos distribuir os carros entre os passageiros de forma eficiente, mostrando como uma solução utilizando emparelhamento máximo com custo mínimo pode reduzir a espera geral em relação a outras soluções.

A última abordagem analisada será na área dos esportes, onde vamos analisar a possibilidade de vitória de cada participante em um campeonato de xadrez utilizando uma nova abordagem do problema *Baseball Elimination*, que consiste em expandir o cenário original para situações em que existem empates. Dessa forma conseguimos de forma eficiente determinar quais jogadores ainda possuem chance de título a partir das pontuações atuais e do chaveamento do campeonato. O objetivo deste artigo é fornecer um entendimento claro e abrangente dos algoritmos de Fluxo Máximo e sua importância na resolução de problemas.

Este documento está estruturado da seguinte forma, na Seção 2 será apresentado os conceitos mais importantes que serão utilizados nas abordagens posteriores, já na Seção 3 são apresentados o escopo dos problemas e suas respectivas soluções, por fim, na Seção 4 apresentaremos as conclusões, trabalhos relacionados e futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados conceitos essenciais para o entendimento do resto do artigo. Na Seção 2.1 serão apresentados conceitos relacionados à teoria dos grafos relevantes no contexto deste documento. Já na Seção 2.2 será abordada a definição de Redes de Fluxo e Fluxo Máximo. Na Seção 2.3 serão explorados alguns dos principais algoritmos utilizados para o cálculo do Fluxo Máximo.

2.1 Grafos

Formalmente, um grafo $G = (V, A)$ é um par ordenado de conjuntos, onde V é um conjunto não vazio de vértices/nós, e A é o conjunto de arestas, sendo cada aresta representada por um par (v, w), onde v e w pertencem a V. Os grafos são comumente empregados em ciência da computação para representar, de maneira abstrata, conjuntos de objetos e suas inter-relações. Por exemplo, uma rede social pode ser modelada como um grafo, onde os vértices correspondem aos usuários e as arestas indicam relações de amizade entre eles.

Grafo dirigido é um tipo de grafo em que as arestas são direcionadas. Em outras palavras, cada aresta a é definida por um par ordenado (v, w), significando que a aresta a vai de v para w. Essas arestas são representadas por setas.

Um *grafo bipartido* é definido por um grafo que pode ser dividido em dois conjuntos disjuntos X e Y, de tal forma que todas as arestas possuem uma extremidade em X e outra em Y.

Um *caminho* em um grafo consiste numa sequência não nula de vértices, de forma que se v e w são vértices consecutivos na sequência, deve existir uma aresta no grafo ligando v à w. Um caminho simples é aquele em que todos os vértices da sequência são diferentes [8].

2.2 Redes de Fluxo

Uma rede de fluxo é um tipo de grafo dirigido em que cada aresta possui uma capacidade e é atribuída um fluxo. Além disso, a rede é caracterizada por dois vértices especiais distintos denominados fonte (*source*) e sumidouro (*sink*) [4]. De maneira mais formal, para uma aresta a, são definidas duas funções: *c(a)* representa a capacidade da aresta, e *f(a)* indica o fluxo através dessa aresta. Essas funções estão sujeitas a algumas restrições [7]:

- $f(a) \geq 0$: O fluxo em cada aresta é não negativo.
- $f(a) \leq c(a)$: O fluxo não pode exceder a capacidade da aresta.
- Para todo vértice v diferente da fonte e do sumidouro, a conservação de fluxo é expressa por:

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

  Isso significa que a soma dos fluxos de todas as arestas que incidem em v é igual à soma dos fluxos de todas as arestas que saem de v. Essa equação reflete a ideia de que o fluxo total que entra em um vértice é igual ao fluxo total que sai dele, garantindo a conservação de fluxo dentro do grafo.
- Para a fonte s e o sumidouro t, a conservação de fluxo é expressa por:

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

  Isso significa que a soma dos fluxos de todas as arestas que saem da fonte é igual à soma dos fluxos de todas as arestas que incidem no sumidouro. Essa equação reflete a condição de que o fluxo que sai da fonte é igual ao fluxo que chega no sumidouro, garantindo a consistência do fluxo ao longo da rede.

A seguir na Figura 1, pode-se observar um exemplo de rede fluxo, o primeiro valor de cada aresta representa o fluxo, que inicialmente é 0 e o segundo representa a capacidade.

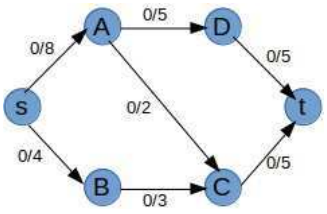

Figura 1: Exemplo de Rede de Fluxo.

O Fluxo Máximo de uma rede é determinado pela maior quantidade de fluxo que pode ser direcionada ao sumidouro, respeitando as restrições previamente mencionadas. A seguir tem-se o exemplo da rede anterior com uma configuração que maximiza o fluxo conforme apresentado na Figura 2.

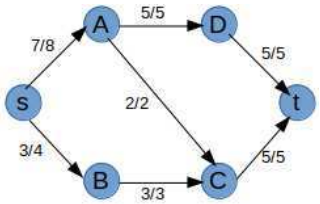

Figura 2: Exemplo de Rede de Fluxo com uma configuração que maximiza o fluxo.

2.3 Algoritmos

Nesta seção serão explorados alguns dos principais algoritmos para resolver o problema do Fluxo Máximo, Na Seção 2.3.1 será apresentado o algoritmo de Ford-Fulkerson [3] e na Seção 2.3.2 será descrito o algoritmo de Edmonds-Karp [2]. Na Seção 2.3.3 será mostrado o algoritmo para calcular o Fluxo Máximo com o Custo Mínimo.

2.3.1 Ford-Fulkerson. O algoritmo de Ford-Fulkerson utiliza o conceito de capacidade residual, que consiste da diferença entre a capacidade da aresta e o fluxo desta, assim pode-se construir o grafo residual de uma rede, um grafo semelhante ao original, porém utilizando as capacidades residuais. Além disso, são criadas arestas reversas, de forma que se existe uma aresta *(v, w)*, é criada uma aresta *(w, v)*, de capacidade *0* e *fluxo f((w, v)) = - f((v, w))*. Logo, se a capacidade residual das arestas reversas é positiva quer dizer que é possível remover unidades do fluxo da aresta original.

O algoritmo consiste em encontrar caminhos aumentativos no grafo residual até que não seja mais possível, obtendo assim o Fluxo Máximo, como pode ser observado no Algoritmo 1. Primeiramente com base na rede original constrói-se o grafo residual, a partir daí tenta-se encontrar um caminho simples da fonte até o sumidouro, tal que todas as arestas do caminho possuam capacidade residual positiva, esse é o chamado caminho aumentativo. Após isso checa-se a aresta do caminho que possui menor capacidade residual, sendo x essa capacidade, podemos aumentar o fluxo de todas as arestas do caminho em x, consequentemente diminuindo em x o fluxo das arestas reversas, além disso o fluxo total da rede também será aumentado em x, ou seja, sendo F o fluxo atual da rede, $F = F + x$ ao fim deste passo.

Isso é repetido até que não seja encontrado um caminho aumentativo. Logo, F representará o Fluxo Máximo dessa rede [11]. Para encontrar o caminho entre a fonte e o sumidouro pode-se usar tanto uma busca em largura (BFS), quanto uma busca em profundidade (DFS). Como a complexidade dos dois algoritmos é $O(|E|)$, sendo $|E|$ o número de arestas e a cada vez que é encontrado um caminho de aumento o fluxo é incrementado em pelo menos uma unidade, a complexidade total do Ford-Fulkerson é $O(|E|\cdot F)$, isso quer dizer que a complexidade depende do Fluxo Máximo da rede.

2.3.2 Edmonds-Karp. O algoritmo de Edmonds-Karp é uma implementação idêntica ao algoritmo de Ford-Fulkerson, porém nele usa-se necessariamente uma busca em largura para encontrar os caminhos aumentativos. Fazendo isso a complexidade deixa de ser em função do fluxo e o algoritmo passa a ter complexidade na ordem de $O(V\cdot|E|^2)$.

Algoritmo 1 Ford-Fulkerson

```
1:  function FordFulkerson(graph, source, sink)
2:      max_flow ← 0
3:      while exist_augmenting_path(graph, source, sink) do
4:          min_residual_capacity ← ∞
5:          for all (v, w) ∈ augmenting_path do
6:              if c((v, w)) - f((v,w)) < min_residual_capacity then
7:                  min_residual_capacity ← c((v, w)) - f((v,w))
8:          for all (v, w) ∈ augmenting_path do
9:              f((v, w)) ← f((v, w)) + min_residual_capacity
10:             f((w, v)) ← f((w, v)) - min_residual_capacity
11:         max_flow ← max_flow + min_residual_capacity
12:     return max_flow
```

2.3.3 Fluxo Máximo com Custo Mínimo. O algoritmo para calcular o Fluxo Máximo com o custo mínimo também é baseado no algoritmo de Ford-Fulkerson. A ideia principal continua sendo encontrar caminhos aumentativos no grafo residual. As mudanças consistem na adição da função de custo *a*, dessa forma, semelhante ao que ocorre com a capacidade, o custo das arestas reversas será o inverso das arestas originais, ou seja para cada aresta *(v, w)*, $a((w, v)) = -a((v, w))$. Além disso, para encontrar um caminho aumentativo, passa-se a procurar o menor caminho, porém diferente do Edmonds-Karp o menor caminho é em relação à soma dos custos do caminho.

Outra mudança é que ao encontrar a aresta do caminho aumentativo que possui menor capacidade residual *x*, além de aumentar o fluxo de cada aresta em *x*, precisamos também atualizar o custo total *C*, para cada aresta *(v, w)* do caminho aumentativo, $C = C + a((v, w)) \cdot x$. Logo, quando não existem mais caminhos aumentativos no grafo residual tem-se em *C* o custo do fluxo como pode-se observar no Algoritmo 2.

A complexidade do algoritmo depende da estratégia utilizada para encontrar os caminhos aumentativos, utilizando o algoritmo de Bellman-Ford [1] por exemplo, para encontrar os caminhos com menor custo, a complexidade será $O(F \cdot |V| \cdot |E|)$. Isso se dá pelo fato de que a complexidade do Bellman-Ford é $O(|V| \cdot |E|)$ para encontrar um menor caminho e para cada caminho aumentativo o fluxo será aumentado em pelo menos uma unidade.

Algoritmo 2 Fluxo Máximo com Custo Mínimo

```
1:  function MinCostFlow(graph, source, sink)
2:      max_flow ← 0
3:      cost ← 0
4:      while exist_augmenting_path(graph, source, sink) do
5:          min_residual_capacity ← ∞
6:          for all (v, w) ∈ augmenting_path do
7:              if c((v, w)) - f((v,w)) < min_residual_capacity then
8:                  min_residual_capacity ← c((v, w)) - f((v,w))
9:          for all (v, w) ∈ augmenting_path do
10:             f((v, w)) ← f((v, w)) + min_residual_capacity
11:             f((w, v)) ← f((w, v)) - min_residual_capacity
12:             cost ← cost + a((v, w)) × min_residual_capacity
13:         max_flow ← max_flow + min_residual_capacity
14:     return max_flow, cost
```

3 APLICAÇÕES

Esta seção examina aplicações práticas em que o problema pode ser modelado como um grafo, permitindo sua resolução por meio de algoritmos de Fluxo Máximo. Na Seção 3.1 será analisada a aplicação do emparelhamento máximo em um grafo bipartido e na Seção 3.2 como problemas reais podem ser mapeados para esse tipo de abordagem, mostrando como maximizar a associação entre trabalhadores e tarefas e uma sugestão de como distribuir de forma balanceada carros entre passageiros em aplicativos de viagem. Na Seção 3.3 será analisada uma aplicação no campo dos esportes, utilizando uma extensão do problema conhecido como *Baseball Elimination* para calcular quais jogadores possuem chance de título em um campeonato de xadrez de forma eficiente.

3.1 Emparelhamento Máximo

Um emparelhamento (*matching*) em um grafo não dirigido *G(V, E)* consiste em um conjunto de arestas E' ($E' \subseteq E$), tal que todo vértice v ($v \in V$), é terminal de no máximo uma aresta de *E'*. Abaixo pode-se ver um exemplo de emparelhamento válido e não válido, respectivamente. Nos exemplos as arestas escolhidas para o emparelhamento estão em vermelho na Figura 3.

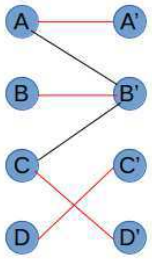

Figura 3: Exemplo de emparelhamento válido em grafo não dirigido.

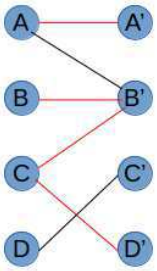

Figura 4: Exemplo de emparelhamento inválido em grafo não dirigido.

No segundo exemplo na Figura 4, o emparelhamento não é válido pois mais de uma aresta selecionada incide nos vértices *C* e *B'*. O Emparelhamento Máximo consiste naquele que possui maior cardinalidade, ou seja, para obter o emparelhamento máximo deve-se selecionar o maior número de arestas do grafo de forma que no máximo uma aresta incida em cada vértice do grafo. Caso o

grafo seja bipartido, o problema pode ser modelado como uma rede de fluxo e ser resolvido com algum dos algoritmos vistos anteriormente.

3.1.1 Modelagem. Para resolver o problema do emparelhamento máximo em um grafo bipartido *(G, E)* basta criar uma rede de fluxo que representa esse grafo aplicando os seguintes passos:

- Divide-se o grafo em duas partições X e Y, de forma que todas as arestas conectam nós de partições diferentes.
- Para cada vértice v da partição X, é criada uma aresta *(s, v)*, sendo s a fonte da rede, essa aresta terá capacidade 1.
- Para cada aresta *(v, w)* no grafo original, que liga um vértice $v \in X$ e um vértice $w \in Y$, cria-se uma aresta direcionada *(v, w)*, essa aresta pode ter qualquer capacidade positiva, nos exemplos terá capacidade 1.
- Para cada vértice w da partição Y, é criada uma aresta *(w, t)* com capacidade 1, sendo t o sumidouro da rede.

Como o grafo utilizado no exemplo anterior é bipartido, este pode ser utilizado como exemplo, na imagem abaixo pode-se observá-lo após sua modelagem como uma rede de fluxo apresentado na Figura 5.

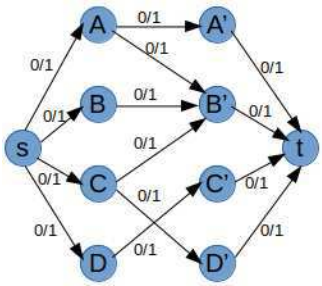

Figura 5: Representação de grafo bipartido como rede de fluxo.

Ao executar algum dos algoritmos de Fluxo Máximo temos que a quantidade de arestas do emparelhamento máximo é igual ao valor do Fluxo Máximo. Para saber as arestas que fazem parte do emparelhamento deve-se checar quais arestas da camada intermediária da rede estão saturadas, ou seja aquelas arestas que ligam dois vértices das duas partições que não sejam a fonte ou o sumidouro e possuem o fluxo igual à 1, mais formalmente uma aresta *(v, w)* fará parte do emparelhamento máximo, caso:

- $v \neq s$ e $w \neq t$
- $f((v, w)) = 1$

Executando algum dos algoritmos no exemplo exposto, temos a seguinte configuração, com as arestas vermelhas sendo aquelas pertencentes ao emparelhamento máximo apresentado na Figura 6.

3.1.2 Prova. A prova de corretude do algoritmo pode ser dividida em duas partes: (i) provar porque o emparelhamento gerado pelo algoritmo é válido, e (ii) porque ele é máximo. Para provar a primeira afirmação basta utilizar as regras de conservação de fluxo, como para todo vértice v da partição X temos que *c((s, v)) = 1*, logo $f((s, v)) \leq 1$, assim pela regra da conservação do fluxo existe

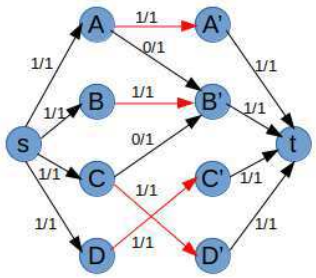

Figura 6: Rede após o cálculo do emparelhamento máximo.

no máximo uma aresta *(v, w)*, tal que *f((v, w)) = 1*, dessa forma no máximo uma aresta que sai de v estará no emparelhamento.

Podemos usar a mesma lógica para os vértices da partição Y, como para todo vértice w da partição Y temos que *c((w, t)) = 1*, logo $f((w, t)) \leq 1$, assim pela regra da conservação do fluxo existe no máximo uma aresta *(v, w)*, tal que *f((v, w)) = 1*, dessa forma no máximo uma aresta que chega em em w estará no emparelhamento.

Para mostrar que esse emparelhamento é máximo pode-se utilizar uma demonstração por contradição. Digamos que exista outro emparelhamento válido *M'*, tal que sua cardinalidade seja maior que o emparelhamento M encontrado pelo algoritmo de Fluxo Máximo, como *M'* é um emparelhamento válido, significa que os vértices de cada partição são tocados por no máximo uma aresta de *M'*. Assim, considerando a rede inicialmente com o fluxo zerado, poderíamos para cada aresta $(v, w) \in M'$, encontrar um caminho aumentativo na rede, fazendo as seguinte atribuições:

- $f((s, v)) = 1$
- $f((v, w)) = 1$
- $f((w, t)) = 1$

Logo, seria possível encontrar um fluxo *F*, tal que *|M'| = F*, sendo assim o fluxo utilizado para gerar M não seria máximo.

3.2 Emparelhamento Máximo em problemas reais

Diversos problemas do mundo real podem ser mapeados para o problema do emparelhamento máximo, como por exemplo no campo da saúde, onde essa abordagem pode ser utilizada no pareamento entre doadores e receptores no contexto da doação de órgãos [9]. Muitos problemas que envolvem maximizar o casamento de objetos de grupos distintos podem ser modelados como uma rede e resolvidos com algoritmos de fluxo.

3.2.1 Atribuição de tasks entre workers. É comum encontrarmos no dia a dia, principalmente em ambientes empresariais, situações em que há um conjunto de atividades para serem realizadas por trabalhadores, onde cada trabalhador é apto a realizar um subconjunto dessas tarefas, um desafio encontrado é como podemos distribuir os trabalhadores entre essas tarefas de forma a maximizar o número de atribuições. Esse é um cenário em podemos aplicar diretamente o emparelhamento máximo.

Pode-se criar um grafo bipartido, com uma partição representando os trabalhadores e outra partição representando as tarefas e para cada tarefa w na qual o trabalhador v seja apto a realizar é

criada uma aresta *(v, w)*. Após isso, pode-se criar a rede de fluxo que representa esse grafo, visando o cálculo do emparelhamento máximo. Para exemplificar, considera-se o seguinte cenário ilustrativo em que se deseja alocar 3 funcionários para realizar 3 *tasks*, cada funcionário tem um conjunto de habilidades e cada tarefa possui um conjunto de requisitos, como mostrado na Figura 7.

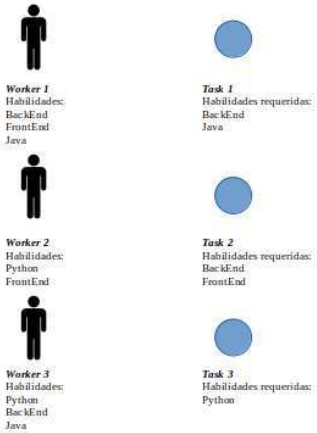

Figura 7: Representação de trabalhadores e tarefas.

Dessa forma, pode-se criar a rede e calcular o emparelhamento máximo visando encontrar um pareamento em que cada trabalhador fique responsável por uma tarefa, esse processo pode ser observado nas Figuras 8 e 9.

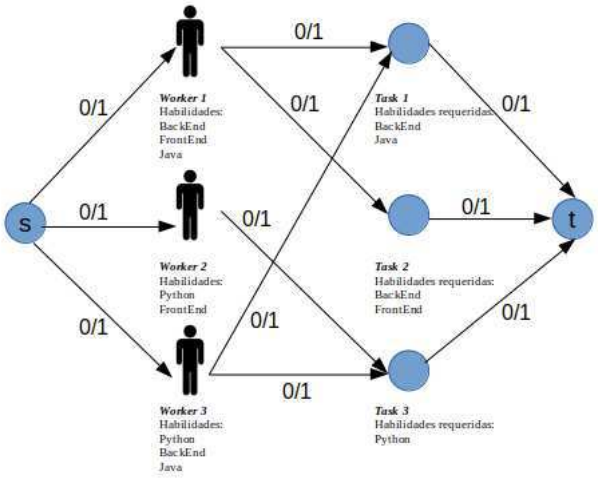

Figura 8: Rede representando a relação entre trabalhadores e tarefas.

Logo utilizando algum algoritmo de Fluxo Máximo teríamos o maior pareamento possível entre trabalhadores e tarefas. É possível observar que essa abordagem considera que um trabalhador poderá realizar no máximo uma tarefa, porém essa modelagem pode ser facilmente ajustada para considerar um limite maior de tarefas

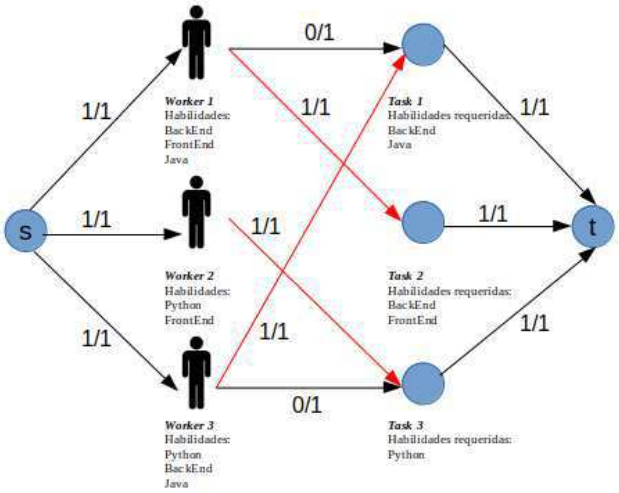

Figura 9: Rede representando o pareamento entre trabalhadores e tarefas após cálculo do Fluxo Máximo.

para cada trabalhador. Considerando que cada trabalhador v pode realizar no máximo $l(v)$ tarefas, basta criar $l(v)$ cópias do vértice que representa o trabalhador v no grafo e aplicar o emparelhamento máximo da mesma maneira.

3.2.2 Pareamento entre carros de aplicativo e passageiros. Existem situações em que o emparelhamento máximo por si só pode não ser tão eficiente, uma exemplificação é no contexto dos aplicativos de viagem. É muito comum nessas aplicações que haja situações em que várias pessoas solicitem viagens de forma simultânea. Nesse caso, deseja-se fazer associação entre carros e passageiros, considerando um exemplo ilustrativo em que três passageiros solicitam viagens de forma simultânea e existem três carros disponíveis, como apresentado na Figura 10.

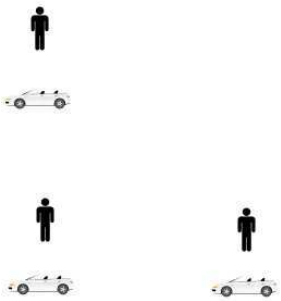

Figura 10: Representação ilustrativa de passageiros e carros.

Uma solução poderia ser criar um grafo bipartido em que uma partição representaria os clientes e outra partição representaria os motoristas e calcular o emparelhamento máximo. Porém esse emparelhamento pode gerar uma solução desbalanceada, como mostrado na Figura 11.

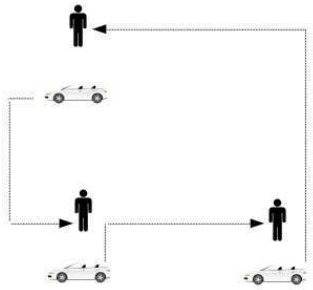

Figura 11: Emparelhamento ineficiente entre carros e passageiros.

Nesse caso, apesar de ter sido atribuído um carro para cada passageiro, percebe-se que os carros que estavam perto de certos passageiros foram atribuídos para passageiros mais distantes. O fato disso ocorrer se dá porque fatores como distância e tempo de viagem não estão sendo levados em consideração.

Para resolver isso, podemos utilizar a abordagem do Fluxo Máximo com Custo Mínimo, onde o custo de cada aresta seria a distância $d((v, w))$ entre o passageiro v e o carro w. Assim, aplicando o algoritmo de Fluxo Máximo com Custo Mínimo para calcular o emparelhamento máximo teríamos como solução o *matching* que minimiza a soma das distâncias entre os motoristas e os clientes, obtendo assim o seguinte emparelhamento mostrado na Figura 12.

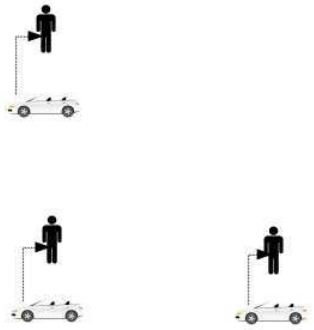

Figura 12: Emparelhamento eficiente entre carros e passageiros.

Outra solução que pode ser considerada é uma abordagem gulosa, na qual, a partir do momento em que uma solicitação é feita, o carro mais próximo é atribuído automaticamente ao usuário. Essa abordagem já foi utilizada pela Uber; no entanto, pode gerar soluções sub ótimas em relação ao tempo total de espera. Para ilustrar isso, conduzimos um experimento gerando disposições aleatórias de carros e passageiros em um plano cartesiano, considerando a distância entre eles como a distância de Manhattan. Exploramos casos em que desejamos alocar 5, 10, 15 e 20 carros para a mesma quantidade de passageiros, comparando a soma das distâncias de cada solução (gulosa e utilizando Fluxo Máximo com Custo Mínimo), obtendo assim o resultado exposto na Figura 13

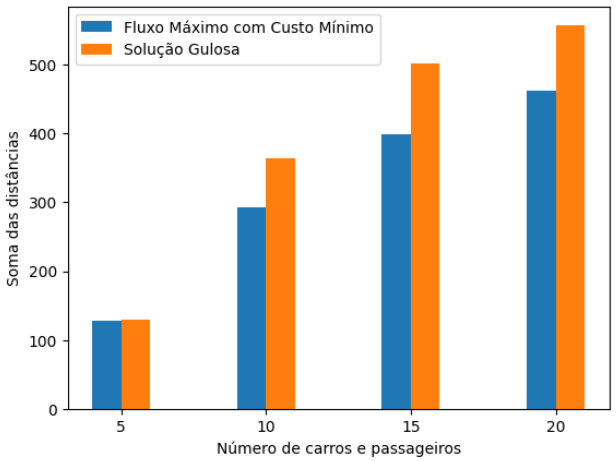

Figura 13: Comparação entre solução utilizando Fluxo e Guloso para distribuição de carros.

Portanto, percebe-se que uma solução utilizando fluxo pode gerar uma economia significativa em relação à soma das distâncias percorridas pelos carros. Em um dos casos testados, essa economia chegou a passar de 20%.

3.3 Possibilidade de vitória em torneio esportivo

Outra aplicação real dos algoritmos de fluxo é o cálculo da possibilidade de vitória em determinados torneios esportivos, essa abordagem foi documentada inicialmente por Schwartz [10] em um problema conhecido como *Baseball Elimination*. O problema original consiste em determinar quais times não possuem mais chance de título em um torneio de Basebol baseado na quantidade de vitórias de cada time e nos jogos restantes. No cenário em questão o que importa para a classificação de cada time é a quantidade de vitórias.

Porém, como será mostrado nesta seção, essa estratégia pode ser expandida para situações onde existe empate e para cada resultado é atribuída uma pontuação, desde que a pontuação distribuída por uma partida seja sempre a mesma. O exemplo utilizado será o Torneio de Candidatos de 2020-2021. Este torneio de xadrez é um dos mais prestigiados do mundo, no qual o vencedor obtém o direito de desafiar o atual campeão mundial. O torneio adota um formato em que todos os participantes se enfrentam duas vezes, uma com as peças brancas e outra com as peças pretas. Em caso de vitória, o jogador ganha 1 ponto, enquanto em caso de empate, cada jogador recebe 0,5 pontos. A três rodadas do final do campeonato de 2020-2021, a classificação era a seguinte conforme apresentado na Tabela 1.

Tendo em vista esse contexto, surge a questão: quais jogadores ainda estão na disputa pelo título? Uma solução inicial seria computar todas as possibilidades de resultados e identificar quais jogadores sairiam vitoriosos em pelo menos uma dessas situações. No entanto, essa abordagem apresenta uma complexidade considerável, uma vez que cada partida pode resultar em três desfechos diferentes. Portanto, a complexidade de criar todos os cenários possíveis seria exponencial, com uma ordem de 3^g, em que g representa o número de partidas restantes.

Posição	Jogador(a)	Pontuação
1	Ian Nepomniachtchi	7
2	Anish Giri	6,5
3	Fabiano Caruana	6
4	Maxime Vachier-Lagrave	5,5
5	Alexander Grischuk	5,5
6	Wang Hao	5
7	Kirill Alekseenko	4,5
8	Ding Liren	4

Tabela 1: Classificação do Torneio de Candidatos 2020-2021 após 11 rodadas.

3.3.1 Modelando o problema como um grafo. Inicialmente, devido à natureza dos algoritmos utilizados, visando trabalhar com números inteiros, é viável multiplicar as pontuações atribuídas por 2, ou seja, considerar que uma vitória vale 2 pontos e um empate 1 ponto para cada jogador. Essa alteração não afeta o resultado final, uma vez que a proporção entre as pontuações é mantida.

Considerando que deseja-se verificar a possibilidade de vitória de um jogador qualquer k. Primeiramente deve-se considerar o cenário mais otimista para este, ou seja, ganhar todas suas partidas restantes, logo sendo $p(x)$ a pontuação atual do jogador x e $g(x)$ o número de partidas que faltam para o jogador x, podemos definir M como sendo sendo a pontuação máxima que k pode atingir:

$$M = 2 \times g(k) + p(k)$$

Caso M seja menor que a pontuação do líder do torneio já podemos afirmar que k não possui mais possibilidade de vitória, caso contrário precisamos checar se os pontos das partidas restantes entre os demais jogadores podem ser distribuídos de forma que nenhum jogador atinja uma pontuação maior que M. Dessa forma, podemos relacionar os demais jogadores e suas respectivas partidas como uma rede de fluxo.

A ideia da modelagem é representar a pontuação que será distribuída pelas partidas como unidades de fluxo, assim o algoritmo de fluxo vai fazer o papel de distribuir essa pontuação entre os jogadores. Primeiramente, para cada partida m entre dois jogadores x e y, tal que $x \neq k$ e $y \neq k$, e sendo s a fonte da rede, se constrói a seguinte relação no grafo conforme descrito na Figura 14.

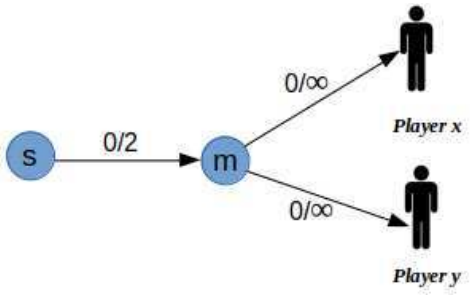

Figura 14: Relação entre partida de jogadores na rede de fluxo.

A capacidade de 2 do vértice fonte para a partida garante que a soma dos pontos recebidos por cada jogador não ultrapasse 2. Percebe-se que caso *f((s, m)) = c((s, m))* = 2, o fluxo de *(m, x)* e *(m, y)* irá representar cadas um dos 3 cenários da partida:

- $f((m, x)) = 1$ e $f((m, y)) = 1$: Significa que houve empate entre o jogador x e o jogador y.
- $f((m, x)) = 2$ e $f((m, y)) = 0$: Significa que o jogador x venceu a partida.
- $f((m, x)) = 0$ e $f((m, y)) = 2$: Significa que o jogador y venceu a partida.

Para completar a construção da rede, é necessário estabelecer a relação entre os jogadores e o sumidouro. Essa relação é representada por arestas, onde a capacidade de cada aresta indica o máximo de pontos que o respectivo jogador pode fazer, de modo que não ultrapasse o competidor k que está sendo testado. Assim, sendo t o sumidouro, para cada jogador $x, x \neq k$, é criada a seguinte relação no grafo (ver Figura 15).

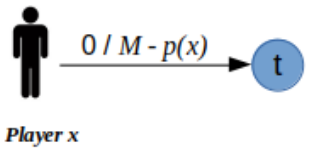

Figura 15: Relação entre jogador e sumidouro na rede de fluxo.

Essa configuração da rede faz com que ao ser aplicado algum algoritmo de Fluxo Máximo, este vai fazer o papel de distribuir as unidades de fluxo, que nesse caso representam os pontos disputados, entre os jogadores, de forma que nenhum jogador atinja uma pontuação maior que o jogador que está sendo testado.

3.3.2 Aplicando o algoritmo de Fluxo Máximo. Após a modelagem da rede, pode-se executar algum dos algoritmos para calcular o Fluxo Máximo entre a fonte e o sumidouro. Para checar se o jogador k ainda possui possibilidade de vitória basta conferir se todas as arestas que saem da fonte estão saturadas, caso isso seja verdade significa que é possível que todos os pontos restantes do campeonato sejam distribuídos de forma que nenhum jogador supere a pontuação máxima que k pode atingir. De maneira mais formal, sendo *(V, E)* a rede, o jogador que está sendo testado k ainda possui chance de título caso:

$$\forall v \in V / \exists (s, v) \in E : f((s, v)) = c((s, v))$$

Caso essa condição não seja satisfeita, significa que o algoritmo não conseguiu distribuir todos os pontos das partidas restantes, implicando que não há nenhuma combinação de resultados que culmine na vitória do jogador k.

3.3.3 Análise prática. Tomando como ponto de partida a tabela previamente exposta do torneio de candidatos, pode-se analisar se cada jogador ainda possui chance de título da seguinte forma; primeiramente, como está se considerando as pontuações dobradas, pode-se observar na Tabela 2 como ficaria a classificação faltando 3 rodadas para o fim do torneio.

Posição	Jogador(a)	Pontuação
1	Ian Nepomniachtchi	14
2	Anish Giri	13
3	Fabiano Caruana	12
4	Maxime Vachier-Lagrave	11
5	Alexander Grischuk	11
6	Wang Hao	10
7	Kirill Alekseenko	9
8	Ding Liren	8

Tabela 2: Classificação do Torneio de Candidatos 2020-2021 após 11 rodadas com as pontuações dobradas.

Jogador(a)	Possibilidade de vitória
Ian Nepomniachtchi	✓
Anish Giri	✓
Fabiano Caruana	✓
Maxime Vachier-Lagrave	✓
Alexander Grischuk	✓
Wang Hao	✓
Kirill Alekseenko	✓
Ding Liren	×

Tabela 3: Indicação dos jogadores com possibilidade de título faltando 3 rodadas para o fim do torneio.

Após a normalização dos dados são repetidos os passos expostos anteriormente para cada jogador, construindo a rede, calculando o Fluxo Máximo e fazendo a checagem necessária como pode-se observar no Algoritmo 3. Dessa forma temos a Tabela 3 indicando quais jogadores ainda possuem chance de terminar o torneio na primeira colocação.

Assim, chegamos à conclusão que apenas o jogador Ding Liren não possui mais possibilidade de título nesse cenário. Apesar dele ter a possibilidade de atingir a mesma pontuação do jogador Ian Nepomniachtchi, o chaveamento do torneio faz com que não haja uma configuração que resulte na sua vitória.

Algoritmo 3 Checa Possibilidade de Vitória

```
1:  function CheckPossibleWinner(graph, source, sink, tournament, tested_player)
2:      target_score ← current_score(tested_player)
3:      remaining_points ← 0
4:      for all (x, y) ∈ get_remaining_games(tournament) do
5:          if tested_player = x or tested_player = y then
6:              target_score ← target_score + 2
7:          else
8:              remaining_points ← remaining_points + 2
9:              create_edge(source, (x,y), 2)
10:             create_edge((x,y), x, ∞)
11:             create_edge((x,y), y, ∞)
12:     for all player ∈ get_players(tournament) do
13:         if target_score < current_score(player) then
14:             return false
15:         if tested_player ≠ player then
16:             max_score ← target_score - current_score(player)
17:             create_edge(player, sink, max_score)
18:     max_flow ← FordFulkerson(graph, source, sink)
19:     if max_flow = remaining_points then
20:         return true
21:     else
22:         return false
```

4 CONCLUSÕES

Neste artigo, exploramos como problemas reais de diferentes áreas podem ser modelados como redes de fluxo, permitindo sua resolução eficiente através de algoritmos de Fluxo Máximo. Ao expor três problemas distintos do mundo real e suas respectivas soluções utilizando técnicas de Fluxo Máximo, como emparelhamento máximo, Fluxo Máximo com Custo Mínimo e uma adaptação do problema *Baseball Elimination*, evidenciamos a versatilidade e a aplicabilidade desses algoritmos em contextos diversos.

Entre os potenciais benefícios de se utilizar tais estratégias, destaca-se o poder que os algoritmos de Fluxo Máximo possuem de alocar recursos de forma otimizada, como pôde ser observado na distribuição de carros em redes de transporte e na atribuição de tarefas entre trabalhadores.

Além de demonstrar a eficácia dessas abordagens específicas, é importante ressaltar o poder da modelagem de problemas como grafos. Ao transformar questões complexas em estruturas de rede compreensíveis, abrimos portas para o desenvolvimento de novas soluções e ideias. Essa capacidade de representação não apenas facilita a resolução de problemas existentes, mas também estimula a inovação, incentivando a criação de novas abordagens e algoritmos.

À medida que continuamos a enfrentar desafios complexos em diversas áreas, desde logística e transporte até planejamento e tomada de decisões, a modelagem de problemas como redes de fluxo permanecerá uma ferramenta valiosa. No entanto, é fundamental reconhecer que novos problemas surgirão, e com eles, a necessidade de adaptação e desenvolvimento contínuo de técnicas de resolução.

4.1 Trabalhos Relacionados

Em 1966, Schwartz [10] apresentou como podemos utilizar o Fluxo Máximo para determinar os times que não possuem mais chance de título em um Torneio de Basebol, a partir dos jogos restantes e da pontuação de cada time. O que fizemos neste artigo foi uma pequena mudança na construção da rede de fluxo objetivando adaptar o problema para o contexto do xadrez, visto que diferentemente do basebol, no xadrez existe a possibilidade de empate entre os jogadores.

Em relação à aplicação da distribuição de automóveis entre passageiros, um estudo feito em 2016 [5] analisa 4 diferentes estratégias para fazer o pareamento entre veículos autônomos e clientes, entre elas uma abordagem utilizando o emparelhamento máximo com custo mínimo. O artigo cita como uma das limitações da solução o fato de que apesar do emparelhamento máximo com custo mínimo minimizar a espera total, alguns passageiros específicos podem ter um tempo de espera individual elevado. Entretanto, ainda vale a pena considerar uma abordagem utilizando fluxo, uma vez que é

possível adaptar o algoritmo para descartar arestas muito custosas, impedindo assim que um usuário específico seja tão penalizado.

4.2 Trabalhos Futuros

Como trabalhos futuros, iremos aprofundar mais as aplicações abordadas neste trabalho, como por exemplo a estratégia de pareamento entre carros e passageiros, onde foi dado apenas um exemplo ilustrativo, um próximo passo seria coletar dados reais de determinadas aplicações para analisar a eficácia dessa abordagem na prática, podendo inclusive fazer uma comparação com as estratégias utilizadas atualmente e ver a viabilidade de aplicar algum algoritmo de fluxo.

Outro passo importante seria explorar as aplicações do Fluxo Máximo no contexto de Inteligência Artificial (IA). Algoritmos de fluxo já são utilizados em alguns campos da IA, como na segmentação de imagens, porém há muitas outras possibilidades onde podemos explorar o uso de algoritmos de fluxo, principalmente no que se diz respeito à alocação de recursos, visto o grande volume de dados que as aplicações de IA lidam, poderíamos testar a possibilidade de aplicar algoritmos de fluxo para alocar esses dados de forma eficiente.

AGRADECIMENTOS

Agradeço ao Professor Rohit Gheyi pela orientação neste trabalho e por todo o suporte recebido durante a graduação. Agradeço à minha família, especialmente aos meus pais, Ana Patrícia e José Aluísio, e ao meu irmão, Lucas de Matos, pelo apoio e carinho incondicionais. Por fim, gostaria de expressar minha gratidão aos meus amigos por tornarem esta jornada mais leve e à minha namorada, Thainá Andrade, por ser meu porto seguro nos momentos em que mais preciso.

REFERÊNCIAS

[1] Richard Bellman. 1958. On a routing problem. *Quart. Appl. Math.* 16 (1958), 87–90.
[2] Richard M. Edmonds, Jack; Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19, 2 (1972), 248–264.
[3] L. R. Ford and D. R. Fulkerson. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 19 (1956), 399–404. https://doi.org/10.4153/CJM-1956-045-5
[4] A.V. Goldberg, É. Tardos, and R.E. Tarjan. 1989. Network flow algorithms. *Stanford University CS Dept.* (1989).
[5] Josiah P. Hanna, Michael Albert, Donna Chen, and Peter Stone. 2016. Minimum Cost Matching for Autonomous Carsharing. *IFAC-PapersOnLine* 49, 15 (2016), 254–259.
[6] T. E. Harris and F. S. Ross. 1955. Fundamentals of a Method for Evaluating Rail Net Capacities. (1955).
[7] Éva Kleinberg, Jon; Tardos. 2006. The Maximum-Flow Problem and the Ford-Fulkerson Algorithm. *Algorithm Design First ed* (2006), Section 7.1.
[8] João Marcos Lima Medeiros and Rohit Gheyi. 2022. Compressão de arestas de grafos utilizando árvore de segmentos. *Universidade Federal de Campina Grande* (2022). http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/29259
[9] A. E. Roth, T. Sönmez, and M. U. Ünver. 2005. Pairwise kidney exchange. *Journal of Economic Theory* 125, 2 (2005), 151–188.
[10] B. L. Schwartz. 1966. Possible Winners in Partially Completed Tournaments. (1966). https://doi.org/10.1137/1008062
[11] J.L Szwarcfiter. 1988. Grafos e Algoritmos Computacionais. *Ed.Campos* (1988), Cap 6.