Network Flow: Task allocation using Bipartite Graph.

Francis N Tsigbey
Ciência da Computação
DCC-UFRR
Boa Vista, Brasil
tsigbeyfrancis@gmail.com

João L.S. Rodrigues
Ciência da Computação
DCC-UFRR
Boa Vista, Brasil
joaolucasidney@outlook.com

***Abstract*—This project of Analysis of Algorithms discusses about Bipartite Graphs modelled as Flow Networks to represent tasks allocation to calculate the maximum flow. It uses the standard Ford-Fulkerson method, with augmenting paths, minimum cuts and residual network, and its Edmond-Karp implementation, which uses Breadth-first search. It shows us the advantages of using bipartite graphs in everyday life application and the complexity of time in the computing world.**

I. Introdução

Os algoritmos fazem parte do dia a dia das pessoas. Seja uma receita culinária ou instruções para uso de medicamentos. Ou seja, é uma sequência de ações executáveis para chegar à solução de um determinado tipo de problema. Segundo *Edsger Dijkstra*, um algoritmo corresponde a uma descrição de um padrão de comportamento, expresso em termos de um conjunto finito de ações.

A análise de algoritmos (descrita e difundida por D.E. Knuth) tem como função determinar os recursos necessários para executar um dado algoritmo. Ela estuda a correção e o desempenho através da análise de correção e da análise de complexidade. Dados dois algoritmos para um mesmo problema, a análise permite decidir qual dos dois é mais eficiente.

Parte desses problemas de algoritimo são modelados em grafos, estruturas de dados muito importantes na computação e este trabalho ira focar no metodo Ford-Fulkerson e no algoritmo de Edmond-Karp que o implementa para solucionar problemas de alocação de tarefas e determinação de fluxo máximo em rede e grafos.

A seguir será apresentado a modelagem do problema, uma breve introdução aos algoritmos utilizados, a implementação do algoritmo de FordFulkerson e a conclusão.

II Modelagem do Problema

O problema do Fluxo Máximo foi modelado da seguinte forma: existe uma empresa de prestação de serviço querendo alocar suas filiais, contratadas de forma que possam solucionar uma determinada tarefa (vértices do grafo) o mais rápido possível do jeito mais eficiente possível, partindo do pressuposto de que cada tarefa tem uma carga(representada pela aresta do grafo) para ser resolvida e os funcionários(representados pelos vértices do grafo) tem uma capacidade adquirida ao longo do tempo para resolver algumas das tarefas. A empresa tem interesse em saber qual é a melhor forma de distribuir as tarefas para que os funcionários possam resolvê-las da forma mais otimizada. Por conta disso o problema foi modelado na forma de grafos bipartido, já que não há necessidade de haver relacionamentos entre os funcionários; Eles apenas efetuaram uma ou mais tarefas. Este grafo bipartido pode ser convertido em um fluxo de redes, adicionando um nodo fonte source(origem) e um nodo destino sink(destino), para podermos usar o algoritmo Edmond-Karp e encontrar o fluxo máximo de tarefas que podem ser realizadas pelos funcionários a qualquer momento. A fig 1 ilustra o grafo bipartido com a “A-origem” e o “B-destino”.

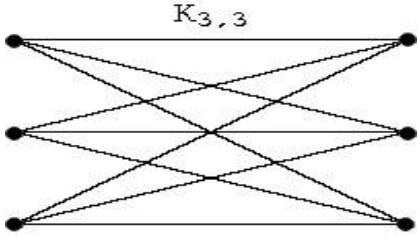

Fig.1.exemplo de um grafo bipartido.

A. Grafo bipartido

Um grafo bipartido pode ser definido da seguinte forma: Um grafo *G é* dito bipartido quando suas vértices *V(G)* podem ser divididos em dois conjuntos disjuntos *X* e *Y*, tais que toda aresta de *G* liga um vértice na parte *X* e outro na parte *Y*.

A notação *G= (X, Y, E) é a comumente usada* para nos referirmos ao grafo bipartido com partição *X, Y* e *E(G) = E*. [13] [9].

Teorema:

- Um grafo *G* é bipartido se é somente se todo ciclo de G possuir comprimento par. [8]
- Um grafo é bipartido se, e somente se, seu número cromático é menor que ou igual a 22. Isso significa que podemos colorir todos os vértices usando apenas duas cores de maneira que uma aresta nunca ligue dois vértices da mesma cor. [13]

Pontes de konigsberg: Problema muito conhecido na área da computação ao, já que foi resolvido com a de teoria´ dos grafos. O problema consiste em encontrar o caminho que atravesse todas as pontes uma vez só e que retorne ao ponto de partida, considerando que há duas ilhas e 7 pontes no rio. [8]

Demonstração do Problema da Ponte de Konigsberg resolvido com o uso de grafos bipartidos:

Ida: Sejam ***X*** e ***Y*** as duas partições de **G**, todo caminho em **G** alterna um vértice de ***X*** com um vértice de ***Y*** *(*consequência da definição de grafo bipartido). Supondo que um ciclo contém um vértice v_i em uma das duas partições. Para voltar a esse vértice, é preciso ir na outra partição e voltar um número par de vezes. [8]

Volta: Seja ***G*** um grafo onde todo ciclo é de comprimento par. Seja um vértice v_i de *G*.

Colocamos num conjunto ***X*** o vértice v_i e todos os outros que são a uma distância par de v_i . Os outros vértices formam o conjunto ***Y***. Caso não houvesse nenhuma aresta ligando dois vértices de ***X*** ou de ***Y***, respeitaremos as condições para que o grafo seja bipartido. Suponhamos agora que existe uma outra aresta entre dois vértices *a* e *b* de ***X*** (ou ***Y***). Já temos um caminho par entre *a* e *b*. Acrescentando a nova aresta, obteríamos um ciclo de comprimento ímpar, o que contradiz a hipótese. Portanto, não pode existir outra aresta entre qualquer par de vértice que já está em ***X*** (igualmente par ***Y***) e o grafo é bipartido. [8] Note que essa prova indica de maneira direta qual seria o algoritmo par determinar se um grafo e bipartido ou não. [8]˜

A noção de grafo completo pode ser estendido aos grafos bipartidos. Um grafo bipartido completo e um grafo onde todos os vértices da partição *X* e ligado por uma aresta a todos os vértices da partição *Y*. Seja *m* e *n* o número de vértices em *X* e Y, respectivamente, o grafo completo bipartido será denotado $K_{m,n}$. [8]

B. Fluxo máximo

A seguir mostraremos alguns conceitos de fluxo máximo que serão utilizados na aplicação do trabalho desenvolvido na disciplina de análise de algoritmo.´

Definição: O fluxo máximo consiste em mostrar o número máximo de unidades de fluxo que pode ser enviar através da rede desde o nó origem (vértice origem *S*) ate o nó destino (vértice destino *T*) sem violar nenhuma restrição de capacidade. [11] [13]

Teorema: Seja *X* o conjunto de nos que pode ser atingido a partir de *S através* de um caminho não cheio, e Seja *Y* o conjunto dos nós restantes. $T \in Y\ T \in X$ e não se encontra numa situação de fluxo máximo.
[11]

Considere-se o corte composto pelos ramos com uma extremidade em ***X*** e outra em ***Y***. Todos os arcos dirigidos de um no ***V*** em ***X*** para um no ***W*** em ***Y*** estão cheios pois caso contrário *W* pertenceria a ***X*** e não a ***Y***. Qualquer arco dirigido de ***Y*** para ***X*** tera fluxo nulo pois caso contrário isso seria um retorno de fluxo de ***Y*** para ***X*** que, se fosse anulado, aumentaria o fluxo que efetivamente chega a *T*, o que contraria a hipótese inicial de a rede estar na situação de fluxo máximo. Então, a capacidade do corte, que é igual ao fluxo de *X* para *Y*, e igual ao fluxo na rede, que e máximo. [11]

Exemplo de aplicação: fluxo de um líquido através de rede de tubos, fluxo de mercadorias do produtor ao consumidor, casamento de desempregados com empregos: [8]

- Grafo com múltiplas origens e/ou múltiplos destinos. [12] • Encontrar o corte mínimo que separa o grafo em duas partes. [12]
- Encontrar número de caminhos que não usem as mesmas arestas. [12]
- Encontrar o maior emparelhamento num grafo bipartido. [12]

Prova: Seja *f* um fluxo que respeite as capacidades. Seja *C* um corte e *S* a margem superior do corte. Pela propriedade dos saldos, a intensidade de *f* e igual ao saldo de *f* em *S*. Esse saldo e *out(S)– in(S)* e assim não

passa de *out(S)*. Como *f* respeita as capacidades, *out(S) não* passa da capacidade do corte *C*. [10]

III. Os Algoritmos

A. Ford-Fulkerson

O algoritmo de Ford-Fulkerson ou “algoritmo de caminhos aumentados”, funciona assim: enquanto existir caminho aumentante de origem até o destino, envia fluxo por esse caminho e sendo utilizado para encontrar o fluxo de valor máximo que faça o melhor uso possível da capacidade disponível deste.

B. Edmond-Karp

O algoritmo de Edmonds-Karpe é apenas uma implementação do método FordFulkerson que usa busca em largura para encontrar os caminhos aumentados.

IV. Implementaçâo

O arquivo main.c começa lendo uma linha de entrada contendo 4 números inteiros: o número de vértices (interseções), o número de arestas (cargas), o número de tarefas (origens) e o número de funcionários (destinos). Após essa leitura, é criado um Grafo com dois nós extras, implementando a ideia de origem A e destino B, conhecidos como source e sink, respectivamente. Em seguida, é lido um número de linhas correspondente ao número de arestas, onde cada linha contém 3 inteiros: um vértice de origem, um vértice de destino e o peso (fluxo máximo) dessa aresta. Também é criada uma variável resposta, que receberá o valor retornado pela função "Ford-Fulkerson" e seus caminhos.

No módulo "FordFulkerson", os parâmetros de entrada são o número de vértices e o grafo. Durante a execução, é criado um grafo de fluxo residual e o vetor caminho é alocado. Em seguida, entra-se em um laço while, que é mantido enquanto a função de Busca em Largura (buscaLargura) retorna 1, indicando que um caminho da origem A ao destino B foi encontrado. Dentro desse laço, é determinado o incremento máximo, a partir do valor máximo anterior, percorrendo os vértices do último até o primeiro, verificando os fluxos e determinando o incremento máximo com base no fluxo do caminho. Em seguida, as arestas existentes no grafo residual são atualizadas, substituindo o valor do fluxo pelo valor anterior menos o incremento, de acordo com a direção em que estamos percorrendo. Ao final, obtemos o fluxo máximo.

No módulo "buscaLargura", os parâmetros de entrada são o número de vértices, o vetor caminho, o grafo e o grafo de fluxo (grafo residual). Nesse módulo, a origem A é determinada como o número de vértices + 1 e o destino B como o número de vértices + 1. É criada uma fila dinamicamente usando o módulo "fila" e um vetor de visitados (com tamanho vértice + 2). Esse vetor é inicializado com 0, indicando que o vértice correspondente ainda não foi visitado. Para iniciar o processo, a origem é enfileirada e, na função enfileirar, o valor de visitado desse vértice é alterado para 1, indicando que ele foi visitado, mas ainda há vizinhos a serem visitados. O valor do caminho é definido como -1, pois ele é o ponto de partida. Em seguida, entra-se em um laço while que continua enquanto a fila não estiver vazia. A cada iteração, o primeiro elemento da fila é desenfileirado, e o valor de visitado desse vértice é alterado para 2, indicando que ele e todos os seus vizinhos foram visitados. Em seguida, percorremos todos os vizinhos desse vértice em um loop for, desenfileirando e enfileirando os vizinhos válidos. Os vizinhos válidos são os vértices com valor 0.

no vetor visitado e que possuem um valor de fluxo que ainda não excedeu a capacidade. Por fim, verificamos o valor de visitado do último vértice para confirmar se o algoritmo conseguiu chegar ao destino [16].

V. Complexidade

Aqui será apresentada a análise do custo teórico de tempo e de espaço dos principais algoritmos e estruturas utilizadas. foram analisadas as funções:

Edmond-Karp: A complexidade de tempo e O(fluxo máximo*número de arestas), já que executamos um loop enquanto nos for dado um caminho existente. No pior caso, podemos adicionar 1 unidade de fluxo em cada iteração a complexidade do tempo torna-se O(fluxo-˜ máximo*número de arestas). Neste caso, como usamos também o Busca em Largura, temos que será O(vértices * número de arestas 2 * fluxo-maximo).

buscaLargura : A complexidade do tempo do algoritmo da Busca em Largura pode ser expressa como O(número de vértices + número de arestas), uma vez que cada vértice e cada aresta ser aos explorados no pior caso. A complexidade de espaço e O (número de vértices), ou seja, O(n).

criaFila: Função inicializada apenas a fila com ponteiros˜ nulos, o que traz uma complexidade de O(1).
enfileira: Função que insere o elemento no fim da fila. Uma operação nada custosa nesta situação. Como ele somente muda o endereço de onde seus ponteiros apontam, esta função e vista como O(1).
Desenfileira: Função que retira o primeiro elemento da fila. Uma operação nada custosa nesta situação. Esta função é vista como O(1).´

A complexidade total do programa e a maior complexidade´ dentre as citadas acima. Temos, então, ela como O (vértices * número de arestas²* fluxoM*aximo*).

VI. RESULTADOS

Avaliação Experimental: A implementação do algoritmo de Ford-Fulkerson e Edmonds-Karp não funcionou corretamente devido a um provável erro na lógica do código nos arquivos ford_fulkerson.c ou busca_em_largura.c. Infelizmente, esse erro não foi identificado e corrigido até o prazo final de entrega deste trabalho. Esses erros podem ter afetado o correto funcionamento dos algoritmos, resultando em resultados imprecisos ou inesperado.

Conclusão: Nessa abordagem, foi adotada a estratégia de distribuir o peso arbitrário de uma tarefa entre o maior número possível de funcionários para sua resolução, e devido os problemas explanados no tópico anterior, não é possível tirar maiores conclusões

REFERENCES

[1] Thomas H. CORMEN; Charles E. LEISERSON; Ronald L. RIVEST, “Algoritmos: teoria e prática”. Rio de Janeiro: Campus, 2002. 916.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rded., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/tutorial/

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand.

[6] https://en.wikipedia.org/wiki/Ford%E2%80%93Fulkerson_algorithm#Complexity

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[8] https://www.geeksforgeeks.org/ford-fulkerson- algorithm-for-maximum-flow-problem/

[9] Feofiloff, P. (14 de Julho de 2017). Algoritmos para Grafos. Fonte: Instituto de Matematica e Estat´ ´ıstica da Universidade de Sao Paulo:˜ https://www.ime.usp.br

[10] Oliveira, J. F. (s.d.). Problemas de Fluxo Maximo. Fonte: Faculdade de Engenharia da Universidade do Porto: https://web.fe.up.pt

[11] Ribeiro, P. (2015). Fluxo Máximo. Fonte: http://www.dcc.fc.up.pt

[12] Siaudzionis, L. (s.d.). Grafos Bipartidos. Fonte: http://www.codcad.com/lesson/65

[13] Valeriano A. de Oliveira, S. R. (s.d.). Teoria dos Grafos. Fonte: UNESP: https://www.ibilce.unesp.br

[14] Maximum flow - Ford-Fulkerson and Edmonds-Karp. (s.d.). Fonte: CPAlgorithms: https://cp-algorithms.com/graph