Problema do Caminho Mínimo: Análise de Algoritmos

Marcos Henrique Santos Cunha[1]

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
Minas Gerais – MG – Brasil

marcos.hsc@aluno.ufop.edu.br

Abstract. *One of the most classical problems in Graph Theory is the Shortest Path Problem (SPP), which consists on finding the shortest path between two vertices in a graph. The efficient solution for this problem allows the operation of a huge variety of actual technologies, such as GPS and data traffic on computer networks. Many solutions to this problem was proposed, some of them are the Dijkstra, Bellman-Ford and Floyd Warshall algorithms, each one wtih its advantages and disadvantages. In this article, the structure, operation and differences between each algorithm will be presented. Besides that, a comparison between the execution time of the algorithms will be made, enabling a deeper understanding about its operation.*

Resumo. *Um dos problemas mais clássicos em grafos é o Problema do Caminho Mínimo (PCM), que consiste em encontrar o caminho mais rápido entre dois vértices de um grafo. Sua solução de forma eficiente possibilita o funcionamento eficaz de diversas tecnologias atuais, tais como o GPS e o tráfego de dados em redes. Diversas soluções para este problema foram propostas, dentre elas os algoritmos de Dijkstra, Bellman-Ford e Floyd Warshall, cada um com suas respectivas vantagens e desvantagens. Neste artigo, serão apresentados estrutura, funcionamento e diferenças entre cada um dos algoritmos. Além disso, será feita uma comparação entre seus tempos de execução, possibilitando um entendimento mais profundo acerca do funcionamento de cada um deles.*

1. Introdução

O estudo da Teoria dos Grafos permite que diversos problemas reais sejam modelados e resolvidos, utilizando diferentes abordagens. Ao modelar uma situação real como um grafo, abre-se a oportunidade para a utilização de uma grande variedade de algoritmos, possibilitando a resolução de diferentes problemas que venham a ser propostos nesta mesma situação.

Um dos problemas que podem ser resolvidos utilizando grafos é o Problema do Caminho Mínimo (PCM). Este problema consiste em, dado um grafo ponderado G e dois vértices $s, t \in G$, encontrar o caminho de menor custo que possui s como origem e t como destino. Existem diversas aplicações para o PCM: em sistemas de GPS, para encontrar o menor caminho de um local a outro; em redes de computadores, para encontrar qual a sequência de servidores pelos quais os dados devem passar, visando obter a menor latência possível. Além das aplicações listadas anteriormente, existem muitas outros problemas que podem ser modelados como um PCM, demonstrando a importância de seu estudo.

Neste artigo, serão apresentados três algoritmos que solucionam o PCM: Dijkstra, Bellman-Ford e Floyd-Warshall. Estes possuem diferenças bem claras entre si, que serão também explicadas no decorrer deste texto. Além disso, foram feitos diversos testes utilizando estes algoritmos, dos quais os resultados de desempenho serão apresentados posteriormente, para fins de comparação e discussão.

Todo o código referente ao assunto discorrido neste artigo pode ser visualizado através de um repositório do GitHub [CUNHA 2020].

Ao final, será possível descrever, com clareza, quais as conclusões obtidas a partir da análise dos resultados de cada um dos algoritmos, quais suas diferenças, pontos positivos e negativos. Dessa forma, será adquirido um melhor entendimento acerca de cada um deles.

2. Algoritmos

Nesta seção, serão detalhados os algoritmos implementados, seu pseudocódigo será apresentado, bem como a explicação de seu funcionamento.

2.1. Algoritmo de Dijkstra

O Algoritmo de Dijkstra, para um grafo ponderado G e um vértice origem $s \in G$, computa dois vetores, um de distâncias e um de predecessores, que representam os menores caminhos entre s e cada um dos vértices do grafo. Ele tem como sua ideia principal atualizar os menores caminhos a cada iteração do algoritmo, até que não hajam mais vértices a serem verificados.

Este algoritmo é um dos mais populares, especialmente devido á sua baixa complexidade computacional de pior caso, da ordem de $O(|V|^2)$, ou $O(|V|log|V|)$, caso seja implementado por uma fila de prioridades. Apesar de ser um algoritmo popular, ele não é capaz de garantir os caminhos mínimos em caso de arestas com peso negativo. No algoritmo 1, é apresentado o pseudocódigo do algoritmo de Djikstra [FONSECA 2020]:

Algoritmo 1: Algoritmo de Dijkstra

Entrada: Um grafo $G = (V, E, w)$
Entrada: Origem s
para cada v *em* V **faça**
    $dist[v] \leftarrow \infty$;
    $pred[v] \leftarrow null$;
$dist[s] \leftarrow 0$;
$Q \leftarrow V$;
enquanto $Q \neq \emptyset$ **faça**
    $u \leftarrow i | min\{dist[i], \forall i \in Q\}$;
    $Q \leftarrow Q - \{u\}$;
    para cada v *adjacente a* u **faça**
        se $dist[v] > dist[u] + w(u, v)$ **então**
            $dist[v] \leftarrow dist[u] + w(u, v)$;
            $pred[v] \leftarrow u$;

As linhas (1-5) fazem a inicialização necessária para a execução do algoritmo. Nas linhas (7-8) é selectionado o elemento u, com a menor distância com relação a s, e este é removido do vetor Q. Em seguida, verifica-se nas linhas (9-12) por novos caminhos mínimos entre u e seus vértices adjacentes. O processo das linhas (6-12) é repetido até que o vetor Q esteja vazio. Ao final, obtêm-se os vetores de distâncias e predecessores, que podem ser utilizados para obter os caminhos mínimos de s a qualquer vértice do grafo.

A implementação do algoritmo de Dijkstra que será utilizada nos resultados deste artigo, foi feita utilizando a uma fila de prioridades, buscando a menor complexidade computacional possível para este algoritmo, da ordem de $O(|V|log|V|)$.

2.2. Algoritmo de Bellman-Ford

O Algoritmo de Bellman-Ford, tal como o de Dijkstra, gera um vetor de predecessores e distâncias, para um dado grafo ponderado G e um ponto de origem $s \in G$. Seu funcionamento consiste em verificar todas as arestas, de todos os vértices, atualizando os caminhos mínimos, caso melhores caminhos sejam encontrados.

Este algoritmo possui um custo computacional de pior caso bem mais alto que o de Dijkstra, sendo este da ordem de $O(|V| \times |E|)$. Diferentemente do algoritmo de Dijkstra, este algoritmo garante encontrar menores caminhos mesmo com arestas de peso negativo. Além disso, com uma pequena adaptação ao código, este algoritmo também possibilita encontrar ciclos de custo negativo. No algoritmo 2, segue o pseudocódigo do algoritmo de Bellman-Ford [FONSECA 2020].

Algoritmo 2: Algoritmo de Bellman-Ford

```
   Entrada: Um grafo G = (V, E, w)
   Entrada: Origem s
1  para cada v em V faça
2  |  dist[v] ← ∞;
3  |  pred[v] ← null;
4  dist[s] ← 0;
5  para i ← 0 até |V| − 1 faça
6  |  para cada (u, v) em E faça
7  |  |  se dist[v] > dist[u] + w(u, v) então
8  |  |  |  dist[v] ← dist[u] + w(u, v);
9  |  |  |  pred[v] ← u;
```

Nas linhas (1-4), é feita a inicialização dos objetos necessários para a execução do algoritmo. Então, repete-se $|V| - 1$ vezes o seguinte processo: verificar todas as arestas do grafo G verificando por novos melhores caminhos (linhas 6-9). Ao final, tal como no algoritmo de Dijkstra, existirá um vetor de distâncias e um de predecessores, que possibilitarão encontrar o menor caminho de s a qualquer vértice de G.

Existe, ainda, uma versão do algoritmo de Bellman-Ford onde é possível interromper a execução se nenhum caminho mínimo foi atualizado. O pseudocódigo adaptado é mostrado no algoritmo 3 [FONSECA 2020].

Algoritmo 3: Algoritmo de Bellman-Ford Eficiente

Entrada: Um grafo $G = (V, E, w)$
Entrada: Origem s
para cada v *em* V **faça**
    $dist[v] \leftarrow \infty$;
    $pred[v] \leftarrow null$;
$dist[s] \leftarrow 0$;
para $i \leftarrow 0$ *até* $|V| - 1$ **faça**
    $trocou \leftarrow False$;
    para cada (u, v) *em* E **faça**
        se $dist[v] > dist[u] + w(u, v)$ **então**
            $dist[v] \leftarrow dist[u] + w(u, v)$;
            $pred[v] \leftarrow u$;
            $trocou \leftarrow True$;
    se $trocou = False$ **então**
        $break$;

Como mostrado no pseudocódigo 3, a execução é similar ao primeiro algoritmo mostrado, porém é utilizada uma variável auxiliar para detectar quando algum caminho foi trocado. Se nenhum caminho foi trocado na presente iteração, a execução pode ser interrompida.

2.3. Algoritmo de Floyd Warshall

O algoritmo de Floyd Warshall é um algoritmo capaz de, para um dado grafo ponderado G, gerar uma matriz de distâncias e uma de predecessores, com as quais é possível descobrir qualquer caminho mínimo entre qualquer par de vértices do grafo. Esta é a principal vantagem da utilização deste algoritmo. Além disso, ele também garante caminhos mínimos para arestas com peso negativo.

Apesar de extremamente útil, seu custo computacional é consideravelmente maior do que os algoritmos de Djikstra e Bellman-Ford, sendo da ordem de $O(|V|^3)$.

Seu funcionamento é baseado em verificar todos os caminhos possíveis no grafo de forma incremental, melhorando os caminhos mínimos encontrados a cada iteração. O pseudocódigo deste algoritmo é apresentado no algoritmo 4 [FONSECA 2020].

As linhas (1-10) fazem a inicialização da matriz de predecessores e da matriz de distâncias. Em seguida, nas linhas (11-16) é executado um laço triplamente aninhado, para fazer a verificação de todos os possíveis caminhos do grafo. Ao final da execução, será possível descobrir o caminho mínimo entre qualquer par de vértices do grafo, a partir das matrizes de distância e predecessores geradas.

3. Experimentação e Resultados

Nesta seção, será feita uma comparação entre os tempos de execução de cada um dos algoritmos, para diversos tipos de grafos. Além disso, será definida a metodologia utilizada para a execução dos testes.

Algoritmo 4: Algoritmo de Floyd Warshall

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

```
1  para i ← 0 até |V| − 1 faça
2      para j ← 0 até |V| − 1 faça
3          se i = j então
4              dist[i][j] ← 0;
5          senão se existe aresta (i, j) então
6              dist[i][j] ← w(i, j);
7              pred[i][j] ← i;
8          senão
9              dist[i][j] ← ∞;
10             pred[i][j] ← null;
11 para k ← 0 até |V| − 1 faça
12     para i ← 0 até |V| − 1 faça
13         para j ← 0 até |V| − 1 faça
14             se dist[i][j] > dist[i][k] + dist[k][j] então
15                 dist[i][j] ← dist[i][k] + dist[k][j];
16                 pred[i][j] ← pred[k][j];
```

3.1. Metodologia

Os grafos utilizados nos testes são grafos ponderados e não-orientados, gerados aleatoriamente, por um algoritmo que recebe como parâmetros o número de vértices, arestas, peso mínimo e peso máximo. O computador utilizado para os testes possui um processador Intel Core i5-4460, com quatro núcleos, operando a 3.20GHz cada, e 8GB de memória RAM.

Para facilitar o entendimento dos resultados, é apresentada na tabela 1, uma associação de uma sigla para cada uma das configurações de grafos, que serão utilizadas posteriormente nos resultados.

Tabela 1. Tabela explicativa para parâmetros utilizados nos testes.

Sigla	Vértices	Arestas	Peso mínimo	Peso máximo
P1	50	500	1	100
P2	50	1000	1	10
P3	100	1000	1	10
P4	100	5000	1	100
P5	500	5000	1	10
P6	500	10000	1	100
P7	1000	10000	1	100
P8	1000	50000	1	10

Os resultados de tempo de execução serão representados em segundos, com uma precisão de seis casas decimais. Além disso, utilizarão-se siglas para se referenciar aos algoritmos na tabela, sendo DJK (Dijkstra), BF (Bellman-Ford), BFE (Bellman-Ford Eficiente), FW (Floyd-Warshall).

3.2. Resultados

Na tabela 2 são mostrados os resultados de tempos de execução dos algoritmos, utilizando-se dos grafos mostrados na tabela 1.

Tabela 2. Resultados de tempo de execução.

Configuração	DJK	BF	BFE	FW
P1	0.000795	0.003194	0.000421	0.014647
P2	0.001005	0.004664	0.000630	0.014985
P3	0.002321	0.009569	0.001015	0.107393
P4	0.003841	0.044239	0.003632	0.112884
P5	0.053324	0.246592	0.014960	13.29160
P6	0.048637	0.480530	0.019797	13.80618
P7	0.307619	1.051188	0.060536	112.0867
P8	0.259119	7.308493	0.092778	120.7773

Com a observação da tabela de resultados, a primeira informação que chama a atenção é a nítida diferença de tempo de execução entre algoritmos de complexidades maiores. Ao observar os resultados de Dijkstra (DJK), Bellman-Ford (BF), e Floyd Warshall (FW), que possuem custos computacionais de pior caso, respectivamente $O(|V|log|V|)$, $O(|V|^2)$ e $O(|V|^3)$, verifica-se que a diferença de tempo de execução é extremamente alta, e aumenta cada vez mais, à medida que o tamanho do grafo aumenta. Perceba que, para a configuração P8, o algoritmo de Dijkstra levou uma fração de um segundo para executar, enquanto o de Floyd Warshall levou mais de dois minutos.

Além disso, surge outra observação interessante: ao verificar os resultados de Dijkstra (DJK) e de Bellman-Ford Eficiente (BFE), percebe-se que o algoritmo de Dijkstra executou num tempo muito maior que o de Bellman-Ford Eficiente, mesmo tendo um custo computacional de pior caso menor. O que acontece é que, devido à melhora feita no algoritmo BFE, que permite que a execução seja interrompida caso nenhum caminho seja atualizado, este raramente chegará em seu custo computacional de pior caso, terminando sua execução bem mais rápido. Enquanto isso, observa-se que o algoritmo de Dijkstra possui uma tendência a estar sempre próximo de seu custo computacional de pior caso, independente da execução ou do conjunto de dados. Assim, verifica-se que o algoritmo de Bellman-Ford Eficiente, mesmo possuindo uma complexidade de pior caso elevada, costumeiramente executa mais rápido do que o algoritmo de Dijkstra, visto que esta complexidade raramente é atingida.

Vale também fazer uma comparação importante, entre o algoritmo de Bellman-Ford (BF), e sua versão melhorada, o Bellman-Ford Eficiente (BFE). A diferença absurda de tempos de execução mostra que pequenos ajustes num algoritmo podem melhorar drasticamente a sua performance, por mais que seu custo computacional de pior caso permaneça igual. Dessa forma, estudar maneiras de desenvolver algoritmos cada vez mais inteligentes se mostra interessante.

O algoritmo de Dijkstra (DJK) manteve um bom tempo de execução ao longo dos testes, precisando de uma fração de segundo para achar os menores caminhos, até mesmo para grafos relativamente grandes (P7, P8). Já o algoritmo de Bellman-Ford (BF), manteve bons tempos de execução, porém estes começaram a aumentar bastante com

o crescimento do grafo, o que pode tornar seu uso inviável para grafos extremamente grandes. Bellman-Ford Eficiente (BFE) gerou ótimos resultados, foi o melhor algoritmo dentre os testes, e pode ser utilizado para grafos bem maiores, mantendo um tempo de execução viável. Por fim, o algoritmo de Floyd Warshall (FW) não teve bons resultados de tempo de execução, principalmente devido á sua complexidade computacional alta. Seu tempo de execução aumentou absurdamente, à medida que o número de vértices do grafo aumentou também. Este algoritmo se mostra inviável para grafos com uma grande quantidade de vértices.

4. Conclusão

Ao final deste trabalho, conclui-se, primeiramente, que os algoritmos de menor caminho são extremamente importantes para o progresso da civilização, tendo diversas aplicações práticas que, sem elas, o mundo globalizado poderia não ser possível.

Além disso, é evidente que o estudo de algoritmos de menor caminho é importantíssimo, uma vez que a redução da complexidade de tais problemas pode acarretar em um imenso benefício, em todas as áreas que utilizam desta abordagem. Ademais, verifica-se que pequenas mudanças em algoritmos podem trazer melhoras significativas de performance, por mais que seu custo computacional de pior caso continue o mesmo.

Nota-se que algoritmos com complexidade alta se tornam inviáveis para problemas que envolvem grafos com um grande conjunto de vértices. Logo, para grafos com grande número de vértices, recomenda-se a utilização do algoritmo de Dijkstra, e também do algoritmo de Bellman-Ford Eficiente, observando sempre seu custo de pior caso, pois por mais que seus tempos de execução sejam baixos, um tempo de execução em pior caso é possível e deve ser considerado no momento da escolha do algoritmo. O algoritmo de Floyd Warshall pode ser efetivamente utilizado em grafos com poucos vértices, com o benefício da obtenção de todos os caminhos mínimos possíveis.

A evolução da tecnologia acontece de forma cada vez mais acelerada, e isto se dá principalmente devido à um grande investimento em pesquisa científica. Mesmo que técnicas tenham sido fortemente testadas e aprimoradas ao longo dos anos, é extremamente necessário que o setor de pesquisa continue buscando por melhorias. Trazendo para o contexto deste trabalho, é extremamente importante a continuidade da pesquisa no que se refere à problemas de caminho mínimo, pois sempre existe a possibilidade de uma melhora ser encontrada, por mais que os algoritmos atuais já estejam fortemente estabelecidos. Quando uma melhora é descoberta, toda a sociedade é beneficiada.

Referências

CUNHA, M. H. S. (2020). Algoritmos de menor caminho. `https://github.com/Marcoshsc/ShortestPathAlgorithms`.

FONSECA, G. H. G. (2020). A04: Percursos e conectividade.