Comparação entre os Algoritmos de Roteamento Dijkstra e Floyd-Warshall

Gustavo Macedo[1], **Heloisa Apª Alves**[1], **Luiz Fernando Becher de Araujo**[1]

[1]Colegiado de Ciência da Computação
Universidade Estadual do Oeste do Paraná (Unioeste)
Caixa Postal 711 – 85.819-110 – Cascavel – PR – Brasil

{gustavo.macedo3,heloisa.alves,luiz.araujo2}@unioeste.br

Resumo. *O estudo compara os algoritmos de Dijkstra e Floyd-Warshall para matrizes de adjacência de diferentes tamanhos. Os resultados mostram que o Floyd-Warshall apresentou melhor desempenho, com tempos de execução menores no cálculo das menores distâncias entre todos os pares de vértices. Isso se deve ao fato de que, enquanto o Floyd-Warshall processa todas as combinações diretamente, o Dijkstra precisa ser executado V vezes para obter o mesmo resultado, tornando-se menos eficiente em grafos maiores. A análise de desempenho, baseada no tempo de execução, confirmou a superioridade do Floyd-Warshall nesse cenário.*

1. Introdução

Os grafos são estruturas matemáticas amplamente utilizadas para modelar e solucionar problemas que envolvem relações entre diferentes elementos. No cotidiano, seu uso está presente em diversas aplicações, como redes sociais, sistemas de navegação, motores de busca e algoritmos de recomendação. Essas estruturas são formadas por vértices, que representam os objetos, e arestas, que indicam as conexões entre eles. Devido à sua versatilidade, os grafos são essenciais em áreas como ciência da computação, engenharia e logística, possibilitando a otimização de processos e a análise eficiente de grandes volumes de dados.

De forma geral "um grafo G = (V,E) consiste de V, um conjunto não vazio de vértices (ou nós) e de E, um conjunto de arestas. Cada aresta tem um ou dois vértices associados a ela, chamados de suas extremidades. Dizemos que cada aresta liga ou conecta suas extremidades."[1]

Para ilustrar melhor a representação de um grafo com custos, apresentada anteriormente, a Figura 1 exibe um exemplo em que se deseja encontrar o menor caminho entre os vértices A e D. Existem duas possíveis rotas: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D, com um custo total de 56, e A $\rightarrow$ C $\rightarrow$ D, com um custo total de 37. O custo de cada caminho é determinado pela soma dos pesos das arestas percorridas. Assim, o segundo caminho é o mais eficiente, pois apresenta o menor custo.

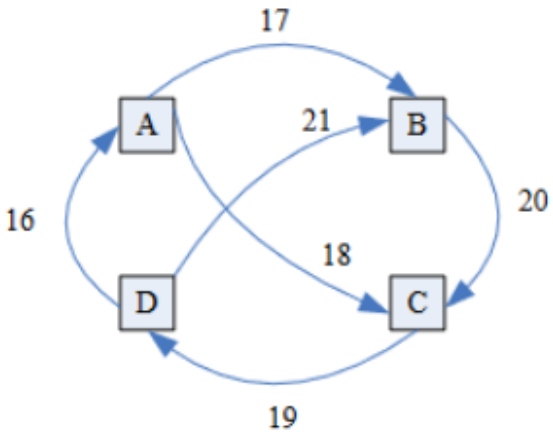

Figura 1. Grafo com custo. Fonte: Pedro S. Prieto et al [2]

Diante desse cenário, os algoritmos de Floyd-Warshall e Dijkstra são amplamente utilizados para encontrar caminhos mínimos em grafos, cada um com características específicas. O algoritmo de Dijkstra determina o caminho mais curto a partir de um único vértice de origem para todos os outros, sendo eficiente em grafos com pesos não negativos. Já o Floyd-Warshall calcula os caminhos mínimos entre todos os pares de vértices, tornando-se uma escolha adequada para problemas que exigem múltiplos cálculos simultâneos. Enquanto Dijkstra se destaca na busca de trajetos a partir de um único ponto, Floyd-Warshall é ideal para cenários que requerem uma visão global das distâncias.

Com base nesses algoritmos, este estudo buscou avaliar o desempenho dessas estratégias em diferentes conjuntos de dados. Para isso, foram realizados testes nos quais o principal critério de análise será o tempo de execução de cada método, permitindo uma compreensão mais aprofundada de seu comportamento em diversos tamanhos de cenários.

O presente artigo está organizado da seguinte forma: a Seção 2 descreve a implementação, detalhando a linguagem de programação utilizada, o ambiente de teste, as entradas dos algoritmos, bem como as implementações propostas junto com os seus custos; a Seção 3 apresenta os resultados obtidos; e a Seção 4 contém as considerações finais.

2. Desenvolvimento

Para avaliar o desempenho dos algoritmos de roteamento, foram realizados testes com diferentes conjuntos de dados, analisando o tempo de execução e a alocação dinâmica de memória. Isso permitiu uma melhor compreensão do impacto computacional de cada método. Neste capítulo, são apresentados o ambiente computacional, incluindo a linguagem utilizada e as especificações da máquina na Seção 2.1. A implementação de cada estratégia é detalhada na Seção 2.2, com trechos de código ilustrativos. Já a Seção 2.3 discute o custo assintótico das abordagens, enquanto a Seção 2.4 analisa o custo empírico para cada conjunto de teste.

2.1. Ambiente Computacional

A modelagem e os testes dos algoritmos foram realizados na linguagem de programação C, utilizando o ambiente de desenvolvimento Visual Studio Code. As estratégias implementadas foram executadas em um computador equipado com um processador AMD

Athlon 3000G (2 núcleos, 4 *threads*, 3,50 GHz), com cache L1 de 32 KB para dados e 64 KB para instruções, cache L2 de 1 MB e cache L3 de 4 MB. O sistema contava com 8 GB de memória RAM DDR4 (2666 MHz) e um SSD NVMe PCIe 3.0 de 128 GB. O sistema operacional utilizado foi o Windows 11 Pro.

Os testes foram conduzidos com vetores de entrada lidos a partir de arquivos fornecidos pelo professor da disciplina via Google Drive. Esses arquivos foram organizados em tamanhos variados, sendo eles: 10, 25, 50, 75, 100, 150, 200, 250, 300, 400, 500, 650, 800, 1.000 e 1.500. Os dados de cada teste foram organizados em uma matriz de adjacência, uma estrutura bidimensional utilizada para representar grafos. Nessa matriz, cada célula indica a existência ou ausência de uma aresta entre dois vértices, possibilitando a modelagem das conexões do grafo de forma eficiente.

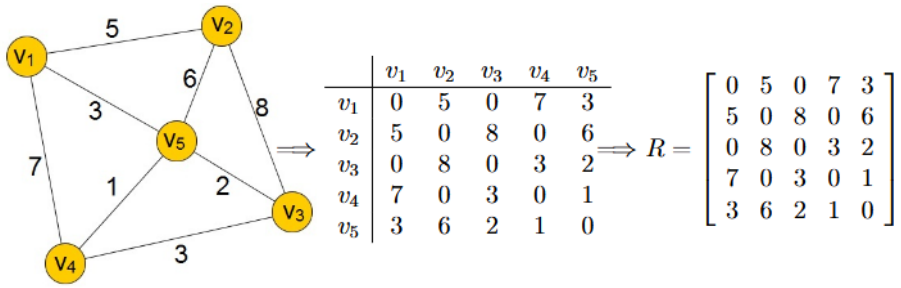

Figura 2. Grafo ponderado e sua matriz de adjacência. Fonte: Viana A.O. et al [3]

A leitura da matriz de teste utilizou alocação dinâmica via listas aninhadas. Após essa etapa, os arquivos foram lidos e seus valores armazenados dinamicamente em uma matriz de adjacência. Em seguida, cada abordagem de roteamento foi aplicada para calcular as menores distâncias entre os vértices, atualizando dinamicamente a matriz de distâncias ao longo da execução.

Para avaliar o impacto no tempo de processamento dos algoritmos de roteamento, os resultados dos testes foram apresentados em segundos (seg), utilizando a função *time()* para medir o tempo de execução. O tempo foi calculado a partir do valor inicial do relógio do sistema operacional, retornando um valor de precisão dupla que representa o número de segundos desde o início da medição. É importante destacar que a cronometragem inicia-se após a leitura dos arquivos de entrada, considerando apenas o tempo de execução da função dos grafos e sendo interrompida ao final do seu processamento.

Diante disso, o presente capítulo apresenta os detalhes do desenvolvimento dos métodos de roteamento, organizado em três fases distintas: a análise dos algoritmos, que inclui os detalhes de sua implementação; a avaliação do custo assintótico de cada abordagem utilizada; e a análise do custo empírico associado à execução de cada uma delas.

2.2. Implementação das estratégias

Esta subseção apresenta estratégias para encontrar caminhos mínimos em grafos, abordando os algoritmos clássicos: Dijkstra, eficiente para pesos não negativos e busca a menor rota a partir de um vértice, e Floyd-Warshall, que calcula as menores distâncias entre todos os pares de vértices.

Serão apresentadas as funções principais implementadas para cada abordagem, detalhando suas estruturas, etapas de execução e otimizações utilizadas. Essas implementações demonstram como cada algoritmo processa os dados e encontram as soluções para o problema de caminhos mínimos, facilitando a compreensão de suas aplicações práticas.

A função apresentada no Código 1 implementa o algoritmo de Dijkstra. O algoritmo inicializa todas as distâncias como infinitas, exceto a do vértice inicial, que recebe valor zero. Além disso, um vetor de controle armazena quais vértices já foram visitados.

O algoritmo itera sobre os vértices, selecionando aquele com a menor distância ainda não visitado. Em seguida, ele verifica seus vizinhos e atualiza suas distâncias caso encontre um caminho mais curto. Esse processo continua até que todos os vértices tenham sido processados ou que não haja mais caminhos possíveis.

Ao final da execução, a função retorna a menor distância entre o vértice de origem e o destino especificado como parâmetro.

```
void dijkstra(int **graph, int src, int size, int *dist) {
    int visited[size];

    for (int i = 0; i < size; i++) {
        dist[i] = INF;
        visited[i] = 0;
    }

    dist[src] = 0;

    for (int count = 0; count < size - 1; count++) {
        int u = minDistance(dist, visited, size);
        visited[u] = 1;

        for (int v = 0; v < size; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INF &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
}
```

Código 1. Função principal do Dijkstra

A função apresentada no Código 2 implementa o algoritmo de Floyd-Warshall, utilizado para calcular as menores distâncias entre todos os pares de vértices em um grafo. Inicialmente, a matriz de distâncias é preenchida com os valores da matriz de adjacência, atribuindo um valor infinito para os pares de vértices que não possuem conexão direta. Além disso, os elementos da diagonal principal são inicializados com zero, pois a distância de um vértice para ele mesmo é sempre nula.

O algoritmo percorre a matriz de distâncias por meio de três laços aninhados. Para cada vértice intermediário k, verifica-se se ele oferece um caminho mais curto entre dois vértices i e j. Se for encontrada uma menor distância ao passar por k, a matriz de distâncias é atualizada com esse novo valor. Esse processo se repete até que todas as combinações

possíveis de caminhos tenham sido analisadas.

Ao final da execução, a função retorna a matriz contendo as menores distâncias entre todos os pares de vértices do grafo.

```
void floydWarshall(int **dist, int size) {
    for (int k = 0; k < size; k++) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
}
```

Código 2. Função principal do Floyd-Warshall

2.3. Custo Assintótico

Esta seção discute o custo assintótico de cada abordagem de roteamento, analisando seu impacto no desempenho à medida que o tamanho da entrada aumenta. O custo assintótico, expresso por meio da notação O-grande (O), descreve como o tempo de execução ou o uso de memória de um algoritmo cresce em relação ao tamanho da entrada. Essa análise permite comparar diferentes algoritmos e identificar quais são mais eficientes para grandes volumes de dados. Com isso, é possível prever limitações de escalabilidade e escolher a abordagem mais adequada para cada cenário.

O Dijkstra está sendo implementado usando uma abordagem simples e direta, sem o uso de estruturas de dados avançadas como Heap Binário ou Fibonacci Heap. A estratégia utilizada é baseada em uma busca linear para encontrar o próximo vértice com a menor distância ainda não visitado. O código percorre todos os vértices para encontrar aquele com a menor distância, o que resulta em uma complexidade de tempo de $\mathcal{O}(V^2)$, onde V é o número de vértices no grafo. Essa abordagem é menos eficiente para grafos grandes, especialmente em comparação com versões que utilizam um Heap Binário, com complexidade $\mathcal{O}(E \log V)$, ou Fibonacci Heap, com complexidade $\mathcal{O}([E + V] \log V)$, onde E é o número de aresta no grafo [4]. Vale lembrar que o Dijkstra foi executado V vezes, portanto, a complexidade desta implementação é $\mathcal{O}(V * V^2)$, ou seja, $\mathcal{O}(V^3)$.

Quanto ao Floyd-Warshall, sua complexidade assintótica é $\mathcal{O}(V^3)$. Isso se deve à estrutura do algoritmo, que utiliza três laços aninhados para iterar sobre todos os pares de vértices, verificando, para cada possível vértice intermediário k, se o caminho entre os vértices i e j pode ser encurtado passando por k. Apesar de não ser o algoritmo mais eficiente em termos de tempo, o Floyd-Warshall é bastante atrativo pela sua simplicidade de implementação e por fornecer uma solução completa para o problema de caminhos mínimos entre todos os pares de vértices, independentemente da conectividade do grafo [5].

2.4. Custo Empírico

Nesta seção, é apresentado o custo empírico dos algoritmos implementados, medido em nanossegundos, ou seja, o tempo necessário para a execução das abordagens escolhidas:

Dijkstra e Floyd-Warshall. O objetivo dessa análise é comparar o desempenho das estratégias utilizadas na busca pelos caminhos mínimos em grafos de diferentes tamanhos, avaliando a eficiência de cada método à medida que a complexidade da entrada aumenta.

Para garantir uma avaliação mais precisa e confiável, os testes foram realizados utilizando os arquivos de entrada descritos na Seção 2. Cada algoritmo foi executado cinco vezes para cada arquivo, e a média dos tempos de execução foi calculada para minimizar possíveis variações causadas por fatores externos, como oscilações no desempenho do hardware ou do sistema operacional. Essa abordagem permite observar com maior precisão o comportamento de cada algoritmo em diferentes cenários, identificando suas limitações e vantagens em termos de tempo de processamento.

Os resultados obtidos estão organizados na Tabela 1, que apresenta o tempo médio de execução dos algoritmos Dijkstra e Floyd-Warshall para cada conjunto de entrada. Essa comparação possibilita uma análise quantitativa do desempenho das duas abordagens, auxiliando na compreensão do impacto do crescimento do número de vértices no custo computacional de cada estratégia.

Tabela 1. Comparação do Tempo Médio de Execução entre Dijkstra e Floyd-Warshall para cada Entrada em Segundos

Tamanho da Entrada	Dijkstra (seg)	Floyd-Warshall (seg)
10	0.000008	0.000003
25	0.000091	0.000036
50	0.000589	0.000253
75	0.001689	0.000823
100	0.003705	0.001854
150	0.011140	0.006089
200	0.024645	0.013794
250	0.046144	0.026472
300	0.077899	0.044539
400	0.176548	0.103155
500	0.335229	0.199526
650	0.723406	0.430607
800	1.325136	0.797951
1000	2.555671	1.537959
1500	8.462967	5.114127

Os resultados obtidos nos testes demonstraram que o algoritmo de Floyd-Warshall apresentou um desempenho superior em termos de custo de execução em todos os cenários testados, quando comparado ao algoritmo de Dijkstra. Em cada um dos arquivos de entrada, o tempo de processamento do Floyd-Warshall foi consistentemente mais baixo, evidenciando sua eficiência na obtenção das menores distâncias entre todos os pares de vértices. Isso pode ser atribuído à sua abordagem de programação dinâmica, que permite calcular todas as combinações possíveis de caminhos de maneira sistemática, enquanto o algoritmo de Dijkstra, apesar de eficiente para buscas a partir de um único vértice, requer execuções múltiplas para alcançar o mesmo resultado, tornando-se menos vantajoso conforme o número de vértices do grafo aumenta.

Além da análise de desempenho, também foi realizada uma comparação dos resultados obtidos por cada algoritmo. Na Tabela 2, são apresentados os melhores caminhos encontrados para cada arquivo de entrada, tanto pelo algoritmo de Dijkstra quanto pelo de Floyd-Warshall. Essa comparação permite verificar a consistência das soluções geradas pelas duas abordagens, garantindo que ambos os métodos identifiquem corretamente os menores caminhos nos grafos testados.

Tabela 2. Melhor Caminho Encontrado entre Dijkstra e Floyd-Warshall para cada Entrada

Tamanho da Entrada	Dijkstra Melhor Caminho	Floyd-Warshall Melhor Caminho
10	8	8
25	5	5
50	8	8
75	8	8
100	7	7
150	4	4
200	6	6
250	16	16
300	13	13
400	17	17
500	13	13
650	11	11
800	13	13
1000	10	10
1500	15	15

A análise dos resultados exibidos na Tabela 2 possibilita ainda a observação de possíveis diferenças entre as soluções, especialmente em casos onde múltiplos caminhos de custo mínimo podem existir. Dessa forma, além de avaliar a eficiência computacional, também é possível validar a precisão e a aplicabilidade prática de cada estratégia na resolução do problema.

Apesar das diferenças no custo de execução, ambos os algoritmos, Dijkstra e Floyd-Warshall, encontraram os mesmos caminhos mínimos para todos os arquivos de entrada testados. Isso evidencia que, apesar das abordagens distintas adotadas por cada método ambos são capazes de identificar corretamente as soluções ótimas. Essa consistência nos resultados reforça a precisão de ambos os algoritmos na resolução do problema, independentemente das diferenças em sua implementação e desempenho computacional.

3. Resultados

Nesta seção, serão apresentados e analisados os resultados obtidos a partir da execução dos algoritmos de roteamento. Os dados coletados incluem o tempo de execução medido em milissegundos para vetores com características variadas e com tamanhos progressivamente maiores, descritos na Seção 2. Para facilitar a compreensão e destacar as diferenças entre as abordagens, os resultados serão ilustrados por meio de gráficos. Esses gráficos

permitirão uma análise visual clara do desempenho de cada estratégia, evidenciando os cenários de maior eficiência e os casos em que o custo computacional foi mais elevado.

A Figura 3 apresenta o desempenho do Dijkstra, enquanto a Figura 2 apresenta o desempenho do Floyd-Warshall. Onde, o eixo horizontal corresponde ao tamanho dos arquivos de entrada, e o eixo vertical o tempo gasto para as execuções em segundos. Ao analisá-las, fica claro que tanto o Floyd-Warshall quanto o Dijkstra executado V vezes apresentam crescimento cúbico do tempo de execução. Em grafos pequenos, o tempo total de ambos é baixo e não há grandes diferenças na prática. À medida que o número de vértices cresce, no entanto, o custo aumenta rapidamente nas duas abordagens.

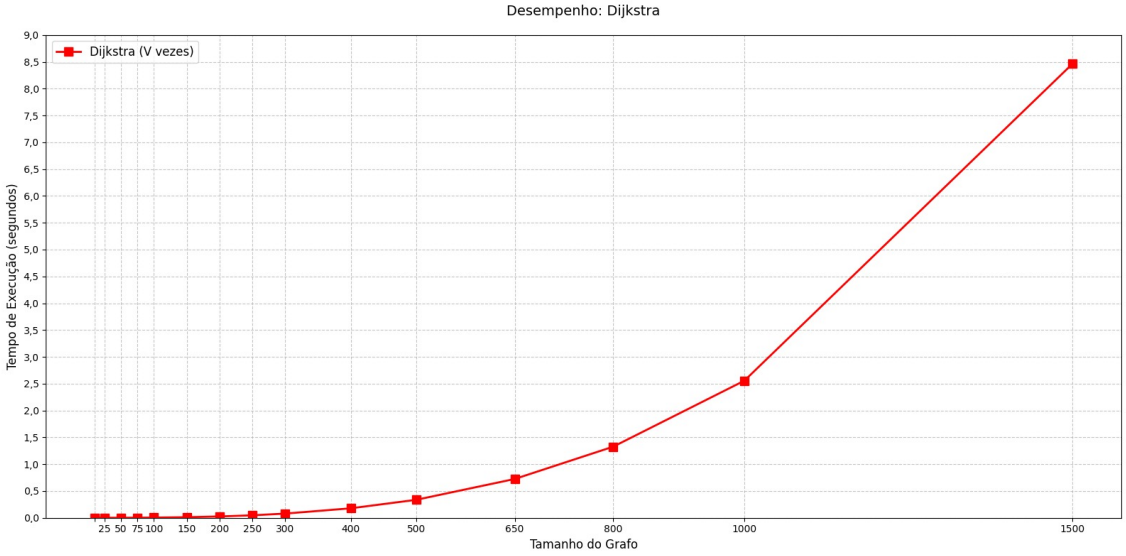

Figura 3. Desempenho do Dijkstra

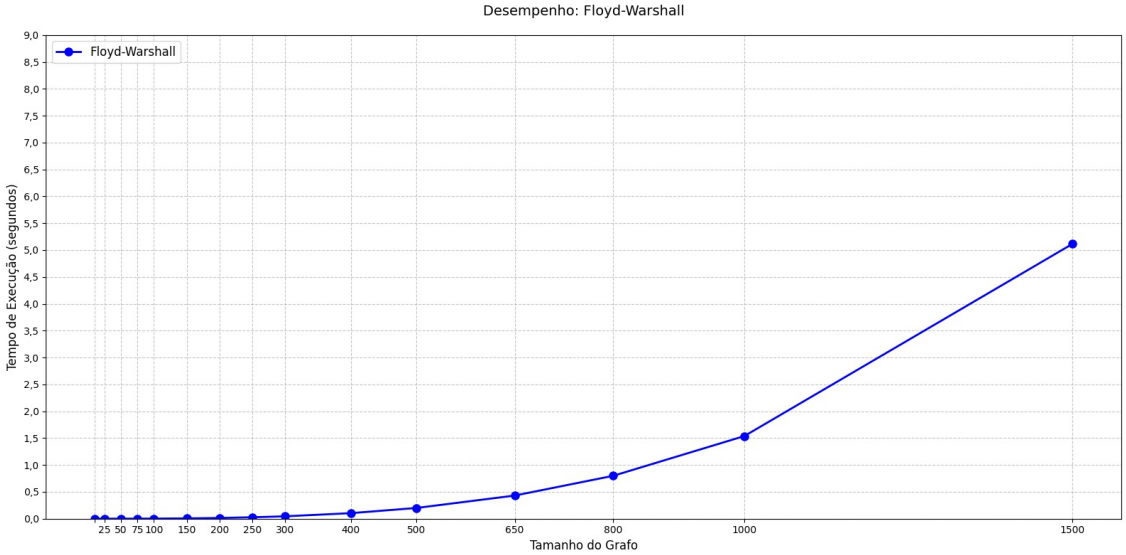

Figura 4. Desempenho do Floyd-Warshall

A Figura 5 junta ambos os algoritmos no mesmo gráfico. No caso do Floyd-

Warshall, percebe-se um crescimento suave no início, mas que se torna bem acentuado a partir de certo tamanho de grafo. Já o Dijkstra, quando executado V vezes, também sobe de forma semelhante conforme o número de vértices aumenta. Embora em alguns pontos um possa parecer um pouco mais rápido que o outro, no regime assintótico ambos mantêm a mesma ordem de complexidade e acabam consumindo tempos bastante próximos para grafos maiores.

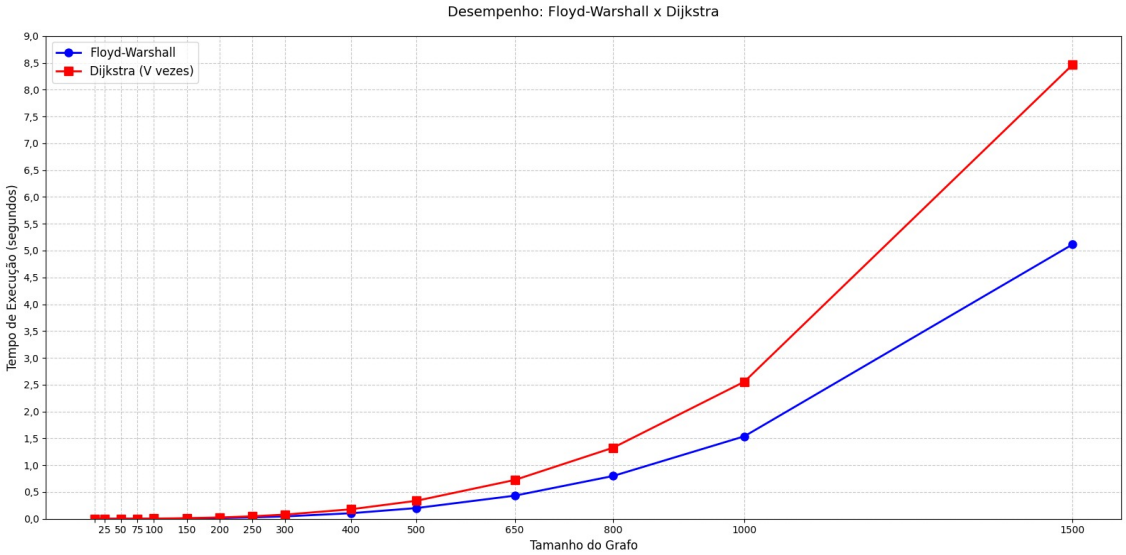

Figura 5. Desempenho do Dijkstra e do Floyd-Warshall

4. Conclusões

Os resultados obtidos nos testes demonstraram que o algoritmo de Floyd-Warshall apresentou um desempenho superior ao do algoritmo de Dijkstra em todos os cenários analisados. Enquanto o Floyd-Warshall calcula diretamente as menores distâncias entre todos os pares de vértices em uma única execução, o Dijkstra precisa ser repetido V vezes para alcançar o mesmo resultado, tornando-se menos eficiente conforme o número de vértices do grafo aumenta. Esse comportamento é particularmente evidente em grafos de tamanho intermediário a grande, onde a repetição sucessiva do Dijkstra impõe um custo computacional mais elevado.

Isso ocorre porque o algoritmo de Dijkstra foi projetado para encontrar o menor caminho a partir de um único vértice até os demais, sendo mais adequado para problemas onde apenas uma origem precisa ser considerada. No entanto, quando a necessidade envolve a obtenção das menores distâncias entre todos os pares de vértices, a execução repetida do Dijkstra adiciona uma sobrecarga significativa ao processamento. Em um grafo com 100 vértices, por exemplo, seria necessário executar o Dijkstra 100 vezes, acumulando um custo muito maior do que a execução única do Floyd-Warshall.

Embora em alguns cenários específicos um dos algoritmos possa parecer um pouco mais rápido do que o outro, no regime assintótico ambos mantêm a mesma ordem de complexidade, $O(V^3)$, e acabam consumindo tempos bastante próximos para grafos pequenos. No entanto, até certo ponto, o Floyd-Warshall ainda se mostra significativamente mais vantajoso, pois evita a repetição do Dijkstra e consegue calcular todas

as distâncias de forma sistemática e direta. Para grafos muito grandes, a diferença de desempenho entre os dois algoritmos tende a se estabilizar, e fatores como otimizações computacionais e a estrutura específica do grafo passam a ter um papel mais relevante na escolha do algoritmo ideal.

Dessa forma, os testes realizados indicam que, para a obtenção das menores distâncias entre todos os pares de vértices, o Floyd-Warshall compensa muito mais para grafos de tamanho considerável, reduzindo significativamente o tempo total de processamento quando comparado à abordagem repetitiva do Dijkstra. Entretanto, à medida que o tamanho do grafo continua crescendo indefinidamente, a vantagem inicial do Floyd-Warshall se equilibra com o custo acumulado do Dijkstra, tornando as diferenças menos perceptíveis em escalas extremamente grandes.

Referências

[1] A. L. de Souza, "Teoria dos grafos e aplicações," Master's thesis, Universidade Federal do Amazonas, 2013.

[2] F. L. Pedro S. Prieto, "Uma aplicação de teoria dos grafos para otimização de custos de rotas aeronáuticas internacionais," Master's thesis, Universidade Presbiteriana Mackenzie, 2022.

[3] A. C. L. A. Viana, Anderson Oliveira, "Algoritmos em grafos e o problema do caixeiro viajante: uma abordagem no ensino médio utilizando planilhas eletrônicas," Master's thesis, Universidade Federal de São João Del-Rei, 2016.

[4] T. B. da Silveira, E. M. Duque, S. J. F. Guimarães, H. T. Marques-Neto, and H. C. de Freitas, "Proposal of fibonacci heap in the dijkstra algorithm for low-power ad-hoc mobile transmissions," *IEEE Latin America Transactions*, vol. 18, no. 03, pp. 623–630, 2020.

[5] S. Kumar, S. Karthik, S. Srilakshmi, and P. D. Viginesh, "Performance analysis of floyd-warshall algorithm: Sequential and parallel execution using intel oneapi," in *2024 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 205–211, IEEE, 2024.