Análise de Desempenho dos Algoritmos de *Dijkstra* e *Bellman-Ford* na Determinação de Caminhos Mínimos

Gabriel Ranulfo Branquinho Cirillo[1], Letícia Siguinolfi de Lima[1], Thayanne Christine Ferreira Lima Martins[1], Vaniely Rodrigues da Cruz[1]

[1]Universidade Estadual do Oeste do Paraná (UNIOESTE)
Cascavel – PR – Brasil

{gabriel.branquinho, leticia.lima34, thayanne.martins, vaniely.cruz@unioeste.br}

Abstract. *This work compares the Dijkstra and Bellman-Ford algorithms in determining the shortest paths in weighted graphs. Simulations were conducted to evaluate the performance of both algorithms in different scenarios, executing seven repetitions and analyzing the average of the remaining executions. The Dijkstra algorithm proved to be faster in terms of execution time in graphs with non-negative weights. The analysis of the results revealed the advantages and limitations of each approach, offering insights into their practical applications in routing and network optimization systems.*

Resumo. *Este trabalho compara os algoritmos de Dijkstra e Bellman-Ford na determinação dos menores caminhos em grafos ponderados. Foram realizadas simulações para avaliar o desempenho de ambos em diferentes cenários, executando sete repetições e analisando a média das execuções restantes. O algoritmo de Dijkstra mostrou-se mais rápido em termos de tempo de execução em grafos com pesos não negativos. A análise dos resultados revelou as vantagens e limitações de cada abordagem, oferecendo insights sobre suas aplicações práticas em sistemas de roteamento e otimização de redes.*

1. Introdução

Algoritmos de roteamento desempenham um papel crucial em vários campos, como roteamento de rede, navegação geográfica e problemas de otimização. Entre esses algoritmos, *Dijkstra* e *Bellman-Ford* são amplamente utilizados para encontrar os caminhos mais curtos em grafos ponderados. O algoritmo de *Dijkstra* é conhecido por sua eficiência em grafos com pesos não negativos, enquanto *Bellman-Ford* pode lidar com grafos com pesos negativos e detectar ciclos de pesos negativos [AbuSalim et al. 2020].

Compreender os pontos fortes e as limitações desses algoritmos é essencial para escolher o método apropriado para aplicações específicas. Analisar um algoritmo significa prever os recursos necessários. Às vezes, recursos como memória, largura de banda de comunicação ou *hardware* de computador são a principal preocupação, mas com frequência é o tempo de computação que desejamos medir. Em geral, pela análise de vários algoritmos candidatos para um problema, pode-se identificar facilmente o mais eficiente [Cormen et al. 2012].

O principal objetivo deste estudo é implementar e comparar os algoritmos de *Dijkstra* e *Bellman-Ford* para determinar os caminhos mais curtos em grafos ponderados. Especificamente, o estudo visa: a) Avaliar o desempenho de ambos os algoritmos em termos de tempo de execução e precisão das distâncias calculadas; b) Analisar o comportamento dos algoritmos em diferentes cenários, incluindo grafos com diversos tamanhos e distribuições de peso; e c) Fornecer insights sobre as aplicações práticas de cada algoritmo e sua adequação para diferentes tipos de problemas de roteamento e otimização.

2. Fundamentação Teórica

2.1 A notação *Big O*

A notação *Big O* informa quão eficiente é um algoritmo em termos de escalabilidade. Por exemplo, se temos uma lista de tamanho n, e o tempo de execução do algoritmo é O(n), isso significa que o tempo de execução do algoritmo cresce linearmente com o tamanho da lista. No entanto, a notação *Big O* não fornece uma medida direta do tempo em segundos. Em vez disso, ela permite comparar o número de operações realizadas pelo algoritmo para diferentes tamanhos de entrada. Essa notação nos ajuda a entender como o desempenho do algoritmo se comporta à medida que o tamanho da entrada aumenta [Cormen et al. 2012].

2.2. Algoritmo de *Dijkstra*

O algoritmo de *Dijkstra*, desenvolvido por Edsger Dijkstra em 1956, é um método clássico utilizado para encontrar o caminho mais curto entre um nó de origem e todos os outros nós em um grafo ponderado, onde todos os pesos das arestas são não-negativos [Dijkstra, 1959; Tenenbaum et al. 1995]. Este algoritmo é amplamente utilizado em diversas aplicações, como roteamento de redes e sistemas de navegação.

Este algoritmo funciona através de uma abordagem gulosa, onde expande o menor caminho de forma incremental. O processo começa atribuindo uma distância inicial de zero ao nó de origem e infinitas para todos os outros nós. Em seguida, o nó com a menor distância não processada é selecionado e suas distâncias para os nós vizinhos são atualizadas. Esse processo continua até que todos os nós tenham sido processados.

O *Dijkstra* seria preferido se todas as arestas tivessem pesos não negativos, devido à sua eficiência computacional. No entanto, se existirem pesos negativos, o *Bellman-Ford* seria necessário para garantir a correção e para detectar possíveis ciclos negativos, que indicariam que não existe um caminho mais curto bem definido [Singh & Tripathi, 2018]

2.3. Algoritmo de *Bellman-Ford*

O algoritmo de *Bellman-Ford*, nomeado em homenagem a Richard Bellman e Lester Ford, é um método utilizado para encontrar o caminho mais curto de um único nó de origem para todos os outros nós em um grafo ponderado, sendo capaz de lidar com arestas

de peso negativo [Bellman 1958]. Além disso, o algoritmo pode detectar ciclos de peso negativo no grafo.

O algoritmo opera relaxando todas as arestas repetidamente. Inicialmente, as distâncias são definidas da mesma forma que no algoritmo de *Dijkstra*. Em seguida, para cada aresta, a distância é atualizada se a distância através dessa aresta for menor do que a distância atualmente conhecida. Este processo de relaxamento é repetido V-1 vezes, onde V é o número de vértices. Após as V-1 iterações, uma iteração adicional verifica a presença de ciclos negativos.

2.4. Diferenças entre os algoritmos

Segundo [Singh and Tripathi 2018] o algoritmo de *Bellman-Ford* lida de forma mais eficaz com uma pequena quantidade de nós, enquanto o algoritmo de *Dijkstra* é mais eficiente para uma grande quantidade de nós.

A complexidade de tempo do algoritmo de *Bellman-Ford* é O(V · E), onde V é o número de vértices e E é o número de arestas. O algoritmo de *Dijkstra* por sua vez tem uma complexidade O(E + V · log V). Com base na complexidade de tempo, podemos inferir que o algoritmo de *Bellman-Ford* leva mais tempo que o algoritmo de *Dijkstra*. O resumo das diferenças com a complexidade de tempo, espaço e quantidade de nós sugerida entre os dois algoritmos é mostrado na Tabela 1.

Tabela 1: Comparação dos dois algoritmos [AbuSalim et al. 2020]

Algoritmo	Complexidade de Tempo	Complexidade de Espaço	Qtde. de Nós Sugerida
Dijkstra	$O(E + V \cdot \log V)$	$O(V^2)$	Grande
Bellman-Ford	$O(V \cdot E)$	$O(V^2)$	Pequena

2.5. Aplicações e Limitações

O algoritmo de *Dijkstra* pode ser aplicado em roteamento de redes, sistemas de navegação *Global Positioning System* (GPS), planejamento de rotas em mapas. No entanto, não pode lidar com arestas de peso negativo e pode ser ineficiente em grafos densos sem estruturas de dados apropriadas [Corner 1998].

O algoritmo de *Bellman-Ford* pode ser aplicado em cenários onde pesos negativos são comuns, como modelos econômicos e detecção de oportunidades de arbitragem em finanças. Contudo, é mais lento em comparação com o algoritmo de *Dijkstra* para grafos grandes, devido à sua maior complexidade de tempo. Pode não ser prático para aplicações em tempo real sem otimizações adicionais [Ascencio and Araújo 2010].

3. Metodologia

3.1. Configurações do Ambiente de Testes

Para a realização deste projeto, foram utilizados os seguintes materiais:

- *Notebook* Dell Inspiron 5406 2-in-1, executando *Linux* Ubuntu 24.04, equipado com um processador Intel(R) Core (TM) i3-1115G4 de 11ª geração @ 3.00GHz e 16 GBytes de memória RAM DDR4.

- Ambiente de desenvolvimento integrado (IDE): Visual Studio Code (versão 1.91.1).
- Linguagem de programação: Python 3.12.3.

3.2. Cenário de Realização dos Experimentos

O código fonte e os conjuntos de dados utilizados neste projeto estão disponíveis através do endereço: *https://github.com/gabrielranulfo/avaliacaoedaa-2bimestre*. Este repositório contém os arquivos necessários para replicar os experimentos e análises.

Os conjuntos de dados são formados por arquivos de texto contendo matrizes de adjacência $n \times n$ de números inteiros, onde n representa o número de nós no grafo e os valores representam os pesos. Essas matrizes são amplamente utilizadas para representar grafos, pois indicam tanto a presença de arestas entre os nós quanto os pesos associados a essas arestas.

Os tamanhos das matrizes variam entre {10, 25, 50, 75, 100, 150, 200, 250, 300, 400, 500, 650, 800, 1000 e 1500}, representando a quantidade de nós em cada grafo. Os pesos das arestas são indicados por valores inteiros positivos, enquanto aqueles contendo o valor 0 denotam a ausência de conexão entre os nós correspondentes. A Figura 1 ilustra como essas representações são dispostas na matriz, facilitando a visualização e compreensão das relações e distâncias entre os diferentes nós. Essa organização é crucial para o estudo e aplicação de algoritmos de grafos em problemas de otimização.

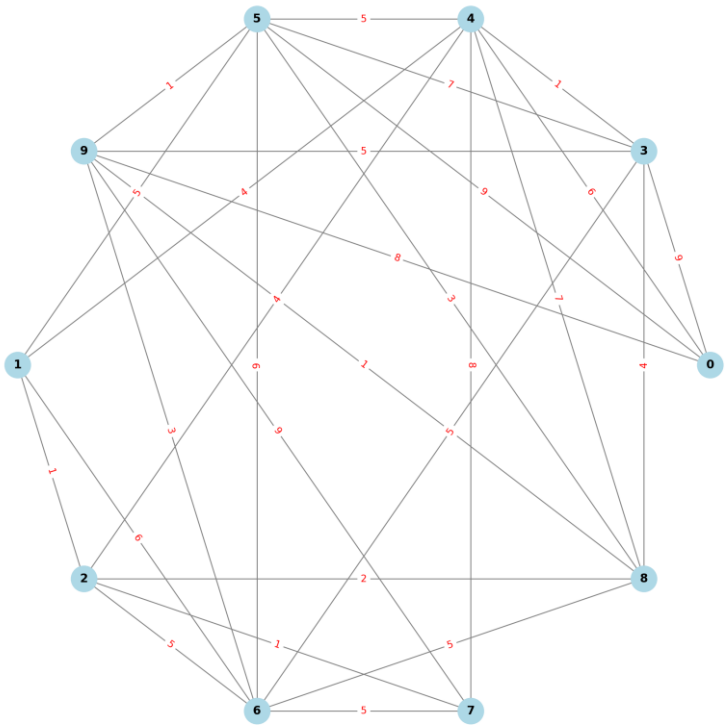

Figura 1: Representação de Grafo para Entrada de 10 nós

A Figura 1 representa um grafo não direcionado onde os nós representam vértices e as arestas têm pesos associados, indicando a distância ou custo para se mover de um nó a outro. O objetivo de um algoritmo de caminho mais curto seria identificar a rota mais eficiente de um nó de origem (por exemplo, nó 0) para todos os outros nós no grafo. Esse tipo de grafo é frequentemente utilizado para ilustrar a busca de caminhos mais curtos entre os vértices, uma tarefa comum em problemas de otimização e que pode ser abordada por algoritmos como Dijkstra e Bellman-Ford [Madkour et al., 2017].

3.3. Métricas de Avaliação

Para realizar a comparação dos métodos a análise foi realizada sob duas óticas. Em um primeiro momento foram comparados o tempo (cronológico) gasto para que os algoritmos fossem executados sobre todos os vértices dos grafos. Em seguida foram comparadas as distâncias obtidas pelos métodos implementados.

4. Execução dos Experimentos

4.1. Procedimentos de Teste

Primeiro, cada arquivo com as matrizes de adjacência é aberto e lido para a memória do computador. Em seguida, uma função em *Python* executa os algoritmos para identificar as menores distâncias para cada nó. Por fim, essas distâncias são salvas em listas do *Python* e uma comparação entre as listas é realizada a fim de identificar se os algoritmos apresentaram os mesmos resultados. Os resultados são salvos em um arquivo CSV com os dados do experimento.

4.2. Repetição dos Experimentos

Cada experimento foi repetido 7 vezes para garantir a consistência dos resultados. Os tempos de execução variam devido a fatores como a carga de trabalho do sistema e variações de desempenho intrínsecas.

Para obter uma medida representativa do desempenho dos algoritmos, os melhores e piores tempos registrados em cada método foram descartados. Em seguida, a análise foi baseada na média dos cinco tempos restantes, e representadas em escala logarítmica em segundos para melhor leitura pois os resultados são exponenciais.

5. Resultados

5.1. Análise cronológica

A análise dos resultados apresenta uma significativa diferença entre os algoritmos com destaque positivo para o *Dijkstra*, que demonstrou excelente comportamento quando comparado diretamente com o *Bellman-Ford* em tempos de execução. Os experimentos, conforme apresentados na Figura 2, demonstram que o algoritmo *Bellman-Ford*, ao executar uma quantidade de 10 nós, teve uma média de tempo de 0,00008264 segundos,

o que equivale a cerca de 0,08264 milissegundos, enquanto o algoritmo de Dijkstra apresentou uma média de 0,00001893 segundos, ou cerca de 0,01893 milissegundos para a execução do algoritmo para esse tamanho de entrada, uma diferença de aproximadamente 4,37 vezes, o comportamento dos resultados para o restante das entradas também pode ser observado na Figura 2.

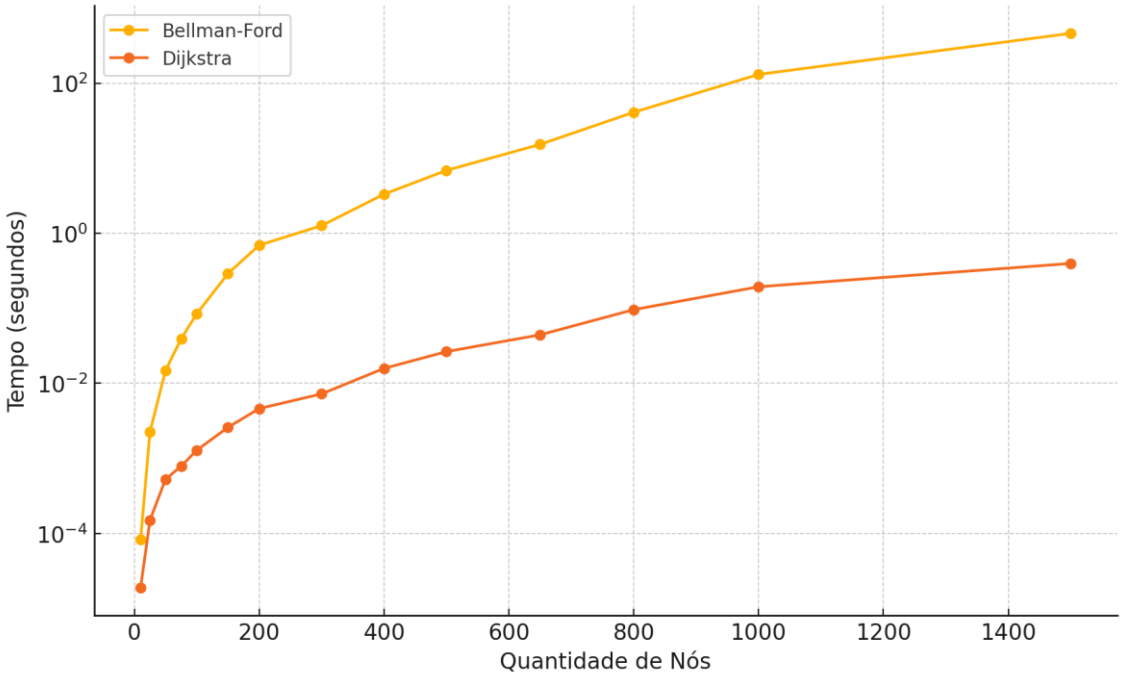

Figura 2: Tempo médio de execução dos algoritmos em escala logarítmica por quantidade de nós

Conforme ilustrado na Figura 2, ao analisarmos os valores intermediários e finais do gráfico, observamos que existe um crescimento exponencial no tempo de execução. Para 500 nós, o algoritmo *Bellman-Ford* apresentou um tempo médio de 15,25457225 segundos, enquanto o *Dijkstra* teve um tempo médio de 0,04433465 segundos, ou cerca de 44,33465 milissegundos. Isso representa uma diferença de aproximadamente 344,08 vezes entre os algoritmos para a mesma quantidade de nós como entrada.

Para 1.500 nós, o algoritmo *Bellman-Ford* registrou uma média de 459,73213105 segundos, enquanto o *Dijkstra* teve uma média de 0,39570866 segundos. Isso representa uma diferença de tempo de aproximadamente 1.161,79 vezes para execução dos algoritmos. Esses resultados reforçam a eficiência superior do *Dijkstra* em grandes grafos com diferenças exponenciais de tempo de um algoritmo para o outro. Enquanto o *Bellman-Ford* não foi útil para o experimento uma vez que é sugerido para grafos em que as arestas têm pesos negativos, e o *Dijkstra* é preferível em cenários onde se busca eficiência e os pesos são não negativos.
Esses resultados reforçam a importância de escolher o algoritmo correto com base nas características específicas do problema e dos dados, levando em consideração fatores como a presença de arestas de peso negativo e o tamanho do grafo.
A Tabela 2 abaixo apresenta o resultado de todas as comparações realizadas por tamanho de entrada no experimento e o valor entre a diferença de um para o outro.

Tabela 2: Comparação de desempenho entre os algoritmos *Bellman-Ford* e *Dijkstra*

Quantidade de Nós	Bellman-Ford (segundos)	Dijkstra (segundos)	Diferença (vezes)
10	0,00008264	0,00001893	4,37
25	0,00225501	0,00015068	14,97
50	0,01493449	0,00052862	28,25
75	0,03929386	0,00078902	49,80
100	0,08477979	0,00128803	65,82
150	0,29056015	0,00259600	111,93
200	0,69293895	0,00461612	150,11
250	1,26375871	0,00724463	174,44
300	3,31472335	0,01584091	209,25
400	6,87526078	0,02648888	259,55
500	15,25457225	0,04433465	344,08
650	40,91127825	0,09600482	426,14
800	68,12366874	0,11853614	574,71
1.000	130,51417370	0,19374270	673,65
1.500	459,73213105	0,39570866	1.161,79

5.2. Análise das distâncias

Ao analisar as distâncias, os resultados mostraram um comportamento já esperado com as matrizes, indicando que ambos os algoritmos calculam as mesmas distâncias para todos os pares de nós. As distâncias mínimas entre os nós foram consistentemente iguais em todos os casos, evidenciando que tanto o *Bellman-Ford* quanto o *Dijkstra* podem ser utilizados para cálculo de menor distância nos cenários propostos. A Figura 3 abaixo apresenta o resultado para a entrada com 10 nós, com as distâncias apresentadas entre os eixos de origem e destino iguais para ambos os algoritmos, comportamento observado para todas as entradas do experimento.

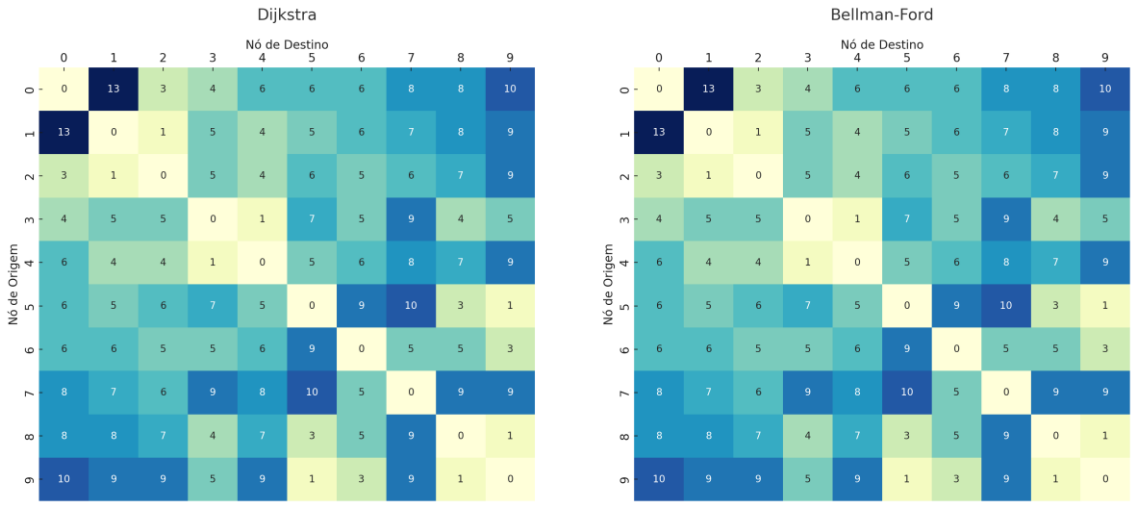

Figura 3: Matriz de distâncias para entrada de 10 nós

6. Discussão

6.1. Interpretação dos Resultados

Além da análise inicial, é importante considerar a complexidade teórica dos algoritmos, o *Bellman-Ford*, pode se tornar impraticável para grafos muito grandes devido ao tempo de execução crescente de modo exponencial, conforme observado nos experimentos. Em contrapartida, o algoritmo de *Dijkstra*, ao utilizar uma fila de prioridade, mostrou um desempenho significativamente melhor, particularmente em grafos com maior quantidade de nós.

Esses resultados destacam a necessidade de escolher o algoritmo adequado baseado nas características específicas do problema, como a presença de pesos negativos nas arestas, onde o *Bellman-Ford* seria mais apropriado, ou a ausência deles, favorecendo o uso de *Dijkstra* conforme demosntrado a partir deste experimento.

6.2. Razões para os Comportamentos Observados

Os comportamentos observados nos tempos de execução dos algoritmos *Bellman-Ford* e *Dijkstra* podem ser atribuídos a várias razões fundamentadas na complexidade e na estrutura dos algoritmos, como, por exemplo: a complexidade dos algoritmos e o número de iterações necessárias, pois *Bellman-Ford* realiza $V - 1$ iterações, atualizando todas as arestas em cada iteração para garantir a correção mesmo com pesos negativos.

Por outro lado, *Dijkstra* atualiza os caminhos de forma mais eficiente, explorando o nó com a menor distância conhecida em cada passo, o que reduz o número de operações necessárias. Outra razão para o comportamento é a natureza dos pesos utilizados. O algoritmo *Dijkstra* assume que todos os pesos das arestas são não-negativos, permitindo otimizações que não são possíveis com *Bellman-Ford*, que deve lidar com pesos negativos. Isso faz com que *Dijkstra* seja mais rápido em grafos onde essa condição é satisfeita.

Em resumo, os comportamentos observados são resultado direto das diferenças na complexidade algorítmica, nas estratégias de atualização de caminhos e nas estruturas de dados utilizadas pelos algoritmos *Bellman-Ford* e *Dijkstra*. Essas características explicam por que *Dijkstra* exibe um desempenho superior, especialmente à medida que o número de nós no grafo aumenta, conforme demonstrado na Figura 2.

7. Conclusão

7.1. Resumo dos Resultados

Neste estudo, analisamos o desempenho dos algoritmos *Bellman-Ford* e *Dijkstra* em termos de tempo de execução, conforme a quantidade de nós no grafo aumentava. Observamos que o algoritmo *Dijkstra* apresentou consistentemente tempos de execução significativamente menores do que o *Bellman-Ford.*

Para grafos com 10 nós, *Dijkstra* demonstrou um desempenho 4,37 vezes superior ao *Bellman-Ford*. Essa tendência se manteve ao longo dos experimentos, com o *Dijkstra* mantendo uma eficiência maior, especialmente evidente em grafos maiores, como

mostrado para 500 e 1.500 nós onde essa diferença salta para 344,08 e 1.161,79 vezes respectivamente.

7.2. Implicações Práticas

As diferenças observadas nos tempos de execução têm implicações práticas importantes para a escolha de algoritmos de caminho mínimo em aplicações reais. O algoritmo *Dijkstra* é claramente mais eficiente para grafos sem arestas de peso negativo, tornando-se a escolha ideal para sistemas de navegação GPS, redes de roteamento de pacotes, e outras aplicações onde a velocidade de cálculo é crucial.

Por outro lado, o *Bellman-Ford* conforme revisão bibliográfica continua sendo relevante em cenários onde pesos negativos estão presentes o que não pode ser avaliado neste experimento.

7.3. Sugestões para Trabalhos Futuros

Para trabalhos futuros, sugerimos explorar a performance desses algoritmos em diferentes tipos de grafos para fornecer uma visão mais abrangente de seus comportamentos. Além disso, a análise de algoritmos alternativos, como por exemplo *Floyd-Warshall* e A-estrela, poderiam complementar este estudo.

Outro campo promissor seria a utilização de algoritmos híbridos que combinem o melhor de cada técnica. Por fim, a utilização de pesos negativos e diferentes valores não utilizados neste experimento para ampliar os resultados aqui apresentados.

Referências

AbuSalim, S. W., Ibrahim, R., Saringat, M. Z., Jamel, S., and Wahab, J. A. (2020). Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization. In IOP Conference Series: Materials Science and Engineering, volume 917,page 012077. IOP Publishing.

AbuRyash, H., & Tamimi, A. (2015). Comparison Studies for Different Shortest Path Algorithms. International Journal of Computers and Applications, Vol. 14.

Ascencio, Ana Fernanda Gomes, & Araújo, Graziela Santos de. (2010). Estruturas de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 24 jul. 2024.

Bellman, R. (1958). On a Routing Problem. Quarterly of Applied Mathematics, 16(1), 87-90.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2012). Algoritmos – Teoria e Prática. Elsevier, Rio de Janeiro, 3 edition.

Corner, Douglas E. (1998). Interligação em rede com TCP/IP (Vol. 1). Rio de Janeiro: Elsevier.

Dinitz, Y., & Itzhak, R. (2017). Hybrid Bellman–Ford–Dijkstra algorithm. Journal of Discrete Algorithms, 42, 35–44.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1(1), 269-271.

MADKOUR, A.; AREF, W. G.; REHMAN, F. U.; RAHMAN, M. A.; BASALAMAH, S. M. A survey of shortest-path algorithms. Disponível em: http://arxiv.org/abs/1705.02044. Acesso em: 26 de julho de 2024.

Singh, J. and Tripathi, R. (2018). Investigation of bellman–ford algorithm, dijkstra's algorithm for suitability of spp. International Journal of Engineering Development and Research (IJEDR), 6(1).

Tenenbaum, Aaron M., Langsam, Yedidyah, & Augenstein, Moshe J. (1995). Estrutura de dados usando C. São Paulo: Makron Books.