Relatório de Análise de Algoritmos de Caminho Mínimo

RA260382

November 28, 2025

1 Introdução

Este relatório apresenta uma análise comparativa de cinco algoritmos para encontrar o caminho mínimo em um grafo: o algoritmo de Dijkstra, a otimização de Dijkstra usando a estrutura de dados de Dial, a busca bidirecional de Dijkstra, a combinação da busca bidirecional com a otimização de Dial, e uma abordagem baseada em Programação Linear (PL). O objetivo é comparar o desempenho em termos de tempo de execução e a exatidão das soluções em um conjunto de grafos de teste.

2 Metodologia

Os algoritmos foram implementados em C++ (Dijkstra, Dial, Bidirecional Dijkstra, Bidirecional Dial) e Python (Programação Linear). Os testes foram executados em 10 grafos de entrada (‘arq01.in‘ a ‘arq10.in‘). O tempo de execução de cada algoritmo foi medido para cada caso de teste. A exatidão foi verificada comparando a saída dos algoritmos com um gabarito de referência.

3 Descrição dos Algoritmos

3.1 Dijkstra

O algoritmo de Dijkstra é um algoritmo guloso que encontra o caminho mais curto de um vértice de origem para todos os outros vértices em um grafo ponderado com arestas de peso não negativo. Ele mantém um conjunto de vértices visitados e, a cada passo, seleciona o vértice não visitado com a menor distância (custo) da origem. A implementação utiliza uma fila de prioridade para selecionar eficientemente o próximo vértice a ser visitado.

3.2 Dijkstra com Otimização de Dial

A otimização de Dial é aplicável quando os pesos das arestas são inteiros e pertencem a um intervalo pequeno. Em vez de uma fila de prioridade, ele usa um array de buckets, onde o bucket ‘i‘ armazena todos os vértices a uma distância ‘i‘ da origem. Isso permite que a operação de ”encontrar o mínimo” seja mais rápida em média, especialmente em grafos densos com pesos de aresta pequenos.

3.3 Dijkstra Bidirecional

A busca bidirecional de Dijkstra executa duas buscas simultaneamente: uma a partir do vértice de origem (busca direta) e outra a partir do vértice de destino (busca reversa), em um grafo reverso. A busca termina quando as duas fronteiras de busca se encontram. O caminho mais curto é então o mínimo entre o caminho encontrado no encontro e os caminhos já completados por cada busca. A principal vantagem é a redução do espaço de busca, o que geralmente leva a um desempenho significativamente melhor em grafos grandes.

3.4 Dijkstra Bidirecional com Otimização de Dial

Esta abordagem combina a busca bidirecional com a otimização de Dial. Duas estruturas de dados de Dial são mantidas, uma para a busca direta e outra para a busca reversa. Espera-se que essa combinação aproveite tanto a redução do espaço de busca da abordagem bidirecional quanto a eficiência da estrutura de dados de Dial para grafos com pesos de aresta pequenos e inteiros.

3.5 Programação Linear (PL)

O problema do caminho mínimo pode ser formulado como um problema de Programação Linear, especificamente como um problema de fluxo de custo mínimo. As variáveis de decisão, x_{ij}, representam o fluxo na aresta (i, j). A formulação é a seguinte:

$$\text{minimizar} \quad \sum_{(i,j)\in E} c_{ij} x_{ij} \tag{1}$$

sujeito a:

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{se } i = s \text{ (origem)} \\ -1, & \text{se } i = t \text{ (destino)} \\ 0, & \text{caso contrário} \end{cases} \quad \forall i \in V \tag{2}$$

$$x_{ij} \geq 0 \quad \forall (i, j) \in E \tag{3}$$

Onde c_{ij} é o custo da aresta (i, j), s é o nó de origem e t é o nó de destino. A primeira restrição é a de conservação de fluxo: para o nó de origem s, o fluxo líquido de saída é 1; para o nó de destino t, o fluxo líquido de entrada é 1; e para todos os outros nós, o fluxo que entra é igual ao fluxo que sai. A segunda restrição garante que o fluxo não seja negativo.

Embora seja uma abordagem muito geral e poderosa, a resolução de um modelo de PL é computacionalmente mais cara do que os algoritmos combinatoriais especializados como o de Dijkstra para o problema do caminho mínimo.

4 Resultados

Os tempos de execução para cada algoritmo nos 10 casos de teste estão apresentados na Tabela 1. Os testes de exatidão indicaram que todos os algoritmos produziram a resposta correta para a maioria dos casos. No entanto, foram observadas pequenas discrepâncias nos resultados de 'biDialDijkstra' para os arquivos 'arq04.in' e 'arq05.in', onde o valor obtido estava fora da margem de erro de 1%. Isso se deve ao arredondamento do custo das arestas para valores inteiros. Todavia, a solução final ainda sim se mostre bem próxima.

Table 1: Tempo de Execução Médio e Pior Caso (ms)

Algoritmo	**Tempo Médio (ms)**	**Pior Tempo (ms)**
Dijkstra	2.612	6.461
Dial Dijkstra	1.588	4.051
Bi-Dijkstra	2.053	2.265
Bi-Dial Dijkstra	1.275	1.935
Prog. Linear	5547.632	16818.504

5 Resultados Detalhados

Table 2: Resultados Comparativos dos Algoritmos

Arquivo	**Saída**	**Dijkstra**		**Dial**		**Bi-Dijkstra**		**Bi-Dial**		**PL**	
	Esperada	**Saída**	**Tempo(ms)**	**Saída**	**Tempo(ms)**	**Saída**	**Tempo(ms)**	**Saída**	**Tempo(ms)**	**Saída**	**Tempo(ms)**
arq01.in	44.0	44.0	0.915	44.0	0.488	44.0	1.830	44.0	1.078	44.0	9.331
arq02.in	43.0	43.0	0.967	43.0	0.480	43.0	1.991	43.0	1.042	43.0	4.668
arq03.in	17.0	17.0	0.962	17.0	0.587	17.0	2.265	17.0	1.038	17.0	6.455
arq04.in	21.883	21.883	0.947	21.0 (WA)	0.688	21.883	2.230	21.0 (WA)	1.001	21.883	8.270
arq05.in	31.388	31.388	0.985	32.0 (WA)	0.526	31.388	2.120	32.0 (WA)	0.998	31.388	8.224
arq06.in	48.787	48.787	1.075	49.0	0.510	48.787	1.933	49.0	0.991	48.787	10.154
arq07.in	839.0	839.0	4.050	839.0	2.624	839.0	2.059	839.0	1.536	839.0	11976.792
arq08.in	853.0	853.0	6.461	853.0	4.051	853.0	2.047	853.0	1.935	853.0	16818.504
arq09.in	794.0	794.0	3.609	794.0	2.366	794.0	2.013	794.0	1.412	794.0	12410.165
arq10.in	943.510	943.510	6.154	944.0	3.556	943.510	2.046	944.0	1.722	943.510	13223.963

6 Discussão

Os resultados confirmam que as implementações em C++ foram ordens de magnitude mais rápidas que a abordagem com Programação Linear em Python, como era esperado. A análise detalhada dos tempos de execução revela nuances importantes sobre o desempenho dos algoritmos combinatoriais.

Entre os algoritmos que garantem a solução ótima, o **Dijkstra Bidirecional (Bi-Dijkstra)** se mostrou o mais eficiente e consistente, especialmente nos casos de teste maiores (‘arq07‘ a ‘arq10‘), superando o Dijkstra padrão. Isso valida a eficácia da busca bidirecional na redução do espaço de busca em grafos maiores.

A otimização de **Dial** (tanto na versão unidirecional quanto na bidirecional) demonstrou ser eficaz na redução do tempo de execução, resultando nos algoritmos mais rápidos no geral. O **Bi-Dial Dijkstra** foi consistentemente o mais veloz nos grafos maiores. No entanto, essa velocidade vem ao custo da precisão, pois a estrutura de Dial, em sua implementação atual, opera com pesos inteiros. Isso levou a respostas incorretas (marcadas como WA) para grafos com custos de aresta fracionários, como ‘arq04.in‘ e ‘arq05.in‘.

A abordagem de **Programação Linear**, embora tenha produzido resultados corretos, foi a mais lenta em todas as instâncias, inviabilizando seu uso para aplicações que exigem respostas rápidas. A sobrecarga computacional para construir e resolver o modelo de PL é significativamente maior do que a dos algoritmos especializados. Contudo, sua flexibilidade para adicionar restrições complexas a torna uma ferramenta poderosa para variantes mais elaboradas do problema de caminho mínimo.

7 Conclusão

A análise comparativa dos algoritmos permite extrair conclusões claras sobre a melhor abordagem para o problema do caminho mínimo, considerando o compromisso entre velocidade e precisão.

Para cenários que exigem a solução exata, o **Dijkstra Bidirecional (Bi-Dijkstra)** provou ser a escolha mais robusta e eficiente, superando consistentemente o Dijkstra unidirecional, especialmente em grafos de maior escala.

Quando a velocidade é o critério principal e os pesos das arestas são inteiros (ou uma pequena margem de erro é aceitável), o **Bi-Dial Dijkstra** se destaca como o algoritmo mais rápido. No entanto, sua implementação baseada em pesos inteiros o torna inadequado para grafos com custos fracionários, onde a precisão é fundamental.

A Programação Linear, embora conceitualmente aplicável, é computacionalmente inviável para este problema em comparação com os algoritmos combinatoriais, devendo ser reservada para problemas de otimização com restrições mais complexas que não podem ser modeladas de outra forma.

Portanto, o **Dijkstra Bidirecional** representa o melhor equilíbrio entre desempenho e exatidão para o problema geral do caminho mínimo em grafos com pesos não negativos.