Algoritmos de busca

Wesley Ricardo Lamb
Instituto Federal Catarinense - Campus Videira
Videira - SC, Brasil
wesley.lamb@castorsoft.com.br

***Abstract*—Este artigo apresenta uma análise comparativa de quatro algoritmos de busca aplicados à navegação em um labirinto composto por múltiplas salas interconectadas. O estudo explora as características e o desempenho dos algoritmos Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra e A*, destacando suas diferenças em termos de eficiência, custo e caminho encontrado. Por meio da modelagem do labirinto como um grafo, o artigo demonstra como cada algoritmo explora o ambiente e seleciona rotas, evidenciando as situações em que cada método se mostra mais adequado para a resolução de problemas de busca em ambientes complexos.**

***Index Terms*—solução de labirintos, DFS, BFS, A*, Dijkstra, grafos**

I. Introdução

De acordo com [1], a busca por caminhos eficientes em ambientes complexos é um problema central na área de inteligência artificial (IA) e ciência da computação. Desde a navegação de robôs em espaços físicos até a resolução de jogos e planejamento de rotas em sistemas de transporte, a capacidade de explorar e encontrar soluções ótimas ou satisfatórias é essencial para o desenvolvimento de sistemas inteligentes. Um dos desafios clássicos é a navegação em labirintos, onde o agente deve identificar um caminho entre múltiplas salas interligadas, enfrentando incertezas e restrições de custo. Algoritmos de busca são ferramentas fundamentais para solucionar esses problemas, permitindo explorar grandes espaços de estados de forma sistemática e eficiente. Entre os mais conhecidos, destacam-se o Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra e A*, cada um com características específicas que influenciam seu desempenho e aplicabilidade. Compreender as diferenças entre esses métodos e suas implicações práticas é crucial para a escolha da abordagem mais adequada em diversos contextos de IA.

Neste artigo, investigamos a aplicação desses quatro algoritmos em um labirinto modelado como um grafo de salas interligadas, discutindo suas vantagens, limitações e o impacto de suas estratégias de busca no desempenho e na qualidade dos caminhos encontrados.

II. Fundamentação Teórica

A. Breadth-First Search

Para [1], o algoritmo BFS é uma técnica clássica de busca em grafos utilizada para explorar de maneira sistemática todos os vértices alcançáveis a partir de um vértice inicial. A ideia fundamental do BFS consiste em expandir os nós em camadas, explorando primeiro todos os vértices a uma certa distância do nó origem antes de avançar para vértices mais distantes. Essa abordagem garante que o caminho encontrado para qualquer vértice seja o de menor número de arestas (menor custo em termos de passos) em grafos não ponderados.

Para implementar essa estratégia, o BFS utiliza uma estrutura de dados do tipo fila (FIFO - First In, First Out) para armazenar os vértices que serão explorados. Inicialmente, o nó de partida é inserido na fila; em seguida, iterativamente, o algoritmo remove o vértice do início da fila, visita seus vizinhos não explorados, e os adiciona ao final da fila. Esse processo se repete até que todos os vértices acessíveis tenham sido visitados ou até que o objetivo seja encontrado.

B. Depth-First Search

Em [2] vemos que algoritmo DFS é um método fundamental para exploração de grafos que se caracteriza por sua abordagem de busca em profundidade. Diferentemente do BFS, que expande os vértices em camadas, o DFS procede explorando o caminho mais profundo possível a partir do nó inicial antes de retroceder (backtracking) para explorar outras ramificações. Essa estratégia é implementada geralmente por meio de uma pilha (LIFO - Last In, First Out) ou por recursão, onde o algoritmo visita um vértice e recursivamente explora um de seus vizinhos não visitados, aprofundando-se até que não existam mais vértices adjacentes a serem visitados.

O DFS é útil para diversas aplicações, como a detecção de ciclos, a ordenação topológica em grafos direcionados acíclicos, e a identificação de componentes conectados. Contudo, diferente do BFS, o DFS não garante a descoberta do caminho mais curto em grafos não ponderados, já que pode explorar caminhos mais longos antes de encontrar a solução desejada.

Em contextos de busca em inteligência artificial, o DFS pode ser eficiente em termos de memória, pois mantém apenas o caminho atual na memória, mas pode ser menos eficaz em garantir soluções ótimas ou completas dependendo da estrutura do espaço de busca.

C. Algoritmo de Dijkstra

[3] diz que o algoritmo de Dijkstra é uma técnica clássica para a determinação do caminho mínimo em grafos ponderados, onde as arestas possuem custos ou pesos associados. A principal ideia do algoritmo é encontrar, a partir de um vértice origem, os caminhos de menor custo para todos os demais vértices do grafo, considerando os pesos das arestas.

Para isso, o algoritmo mantém um conjunto de vértices cujas distâncias mínimas da origem já foram determinadas

e iterativamente seleciona o vértice com a menor distância estimada que ainda não foi processado. Em cada iteração, ele atualiza as estimativas de distância dos vizinhos do vértice selecionado, relaxando as arestas e garantindo que as distâncias armazenadas sejam as menores possíveis até o momento.

A utilização de uma estrutura de dados adequada, como uma fila de prioridade, permite que o algoritmo opere de forma eficiente, reduzindo o custo computacional da seleção do próximo vértice a ser explorado. O algoritmo de Dijkstra é amplamente empregado em aplicações que envolvem roteamento, planejamento de trajetórias e problemas de otimização em redes, sendo garantido encontrar o caminho de custo mínimo em grafos com arestas de peso não negativo.

*D. Algoritmo A**

Por fim, em [4] vimos que algoritmo A* é um método heurístico de busca informada amplamente utilizado para encontrar caminhos mínimos em grafos ponderados. Ele combina a eficiência do algoritmo de Dijkstra com uma função heurística que estima o custo restante para alcançar o objetivo, permitindo uma exploração mais direcionada do espaço de estados.

A ideia central do A* é expandir nós com base em uma função de avaliação f(n)=g(n)+h(n), onde g(n) representa o custo acumulado do caminho desde o nó inicial até o nó n, e h(n) é uma heurística admissível que estima o custo mínimo do nó n até o objetivo. Essa combinação orienta a busca na direção do objetivo, priorizando os caminhos que aparentam ser mais promissores, o que pode reduzir significativamente o número de estados explorados em comparação com buscas não informadas.

III. Metodologia

A metodologia de representar um labirinto através de uma matriz constitui uma forma estruturada e eficiente de modelar ambientes discretos para aplicação de algoritmos de busca. Nessa abordagem, o labirinto é discretizado em células organizadas em uma matriz bidimensional, onde cada elemento indica o estado da célula — por exemplo, se é um espaço livre, uma parede, ou um ponto de interesse. Essa representação permite mapear o labirinto a um grafo implícito, onde cada célula corresponde a um nó, e as conexões entre células adjacentes representam arestas que podem ser exploradas pelos algoritmos.

Para realizar a exploração sistemática desse espaço, utilizam-se estruturas de dados fundamentais como listas, pilhas e filas, que definem a ordem em que os nós são visitados durante a busca. A fila (FIFO) é empregada no algoritmo Breadth-First Search (BFS), garantindo a expansão em níveis, isto é, a exploração de todos os nós a uma certa distância antes de avançar para os mais distantes. Já a pilha (LIFO) é a estrutura-base para o Depth-First Search (DFS), favorecendo a exploração profunda, onde se segue um caminho até o máximo antes de retroceder para outras ramificações.

No caso dos algoritmos de busca com custos, como Dijkstra e A*, utiliza-se uma fila de prioridade para ordenar os nós a serem explorados com base em funções que consideram o custo acumulado e, no caso do A*, também uma heurística estimada. Essas estruturas permitem que a busca seja direcionada para encontrar caminhos de custo mínimo de forma eficiente.

Assim, a combinação da representação matricial com o uso adequado dessas estruturas de dados possibilita a implementação efetiva dos algoritmos de busca, facilitando a navegação e a resolução de problemas de caminho em labirintos e outros ambientes discretos.

Todos os algoritmos foram implementados na linguagem C++ e utilizam as bibliotecas nativas para lidar com as estruturas de dados anteriormente listadas.

Vamos analisar os quatro algoritmos em uma situação partindo do canto superior direito até o canto inferior esquerdo. As células com letras são as salas, as células amarelas são passagens e as células pretas são paredes.

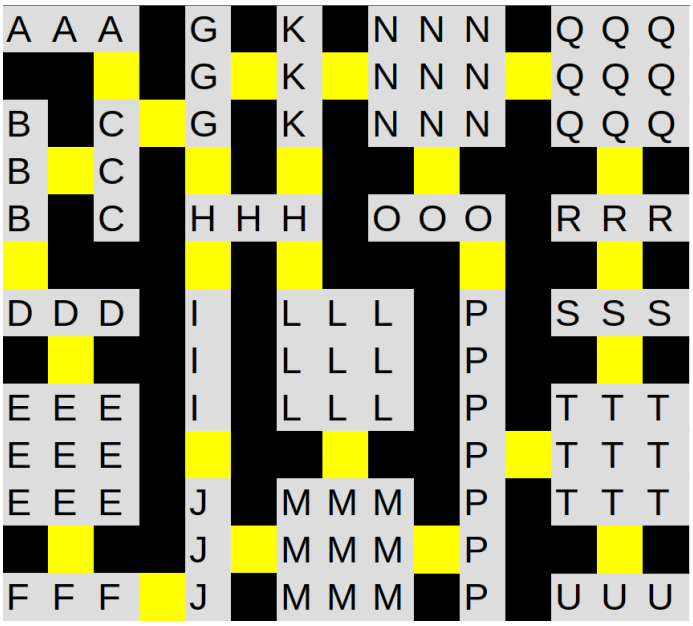

Fig. 1. Labirinto a ser analisado

IV. Resultados e Discussão

A. BFS

O algoritmo BFS acessou 286 posições e conseguiu chegar a solução perfeita de 28 passos seguindo o caminho a - c - g - k - n - q - r - s - t - u.

B. DFS

O algoritmo DFS acessou 62 posições porém a solução foi a pior dos testes, 38 passos seguindo o caminho a - c - b - d - e - f - j - m - p - t - u.

C. Dijsktra

O algoritmo de Dijsktra acessou 59 posições e encontrou uma solução boa de 32 passos seguindo o caminho a - c - g - h - l - m - p - t - u.

D. A*

O algoritmo A* acessou 259 posições e chegou a solução perfeita de 28 passos seguindo o caminho a - c - g - k - n - q - r - s - t - u.

V. Conclusão

Este estudo apresentou uma análise dos quatro principais algoritmos de busca em Inteligência Artificial — Busca em Largura (BFS), Busca em Profundidade (DFS), Dijkstra e A* — com o objetivo de compreender seu comportamento, desempenho e aplicabilidade na resolução de problemas de navegação em ambientes estruturados, como um labirinto.

Através da implementação prática dos algoritmos em um cenário de labirinto, foi possível observar diferenças significativas em relação ao tempo de execução, consumo de memória, e qualidade da solução (isto é, o custo ou comprimento do caminho encontrado).

O BFS se mostrou eficaz em encontrar soluções ótimas em grafos não ponderados, porém apresentou alto consumo de memória, especialmente em grafos grandes.

O DFS, embora mais econômico em termos de memória, frequentemente encontrou caminhos subótimos e, em certos casos, falhou ao explorar completamente o espaço de estados, dependendo da profundidade e da estrutura do labirinto.

O algoritmo de Dijkstra demonstrou robustez e garantiu sempre o caminho de menor custo, sendo especialmente adequado em cenários com pesos variados. No entanto, seu desempenho pode ser inferior ao de A* em termos de eficiência, por não utilizar heurísticas.

Por fim, o A* destacou-se como o algoritmo mais eficiente em termos de balanceamento entre custo e velocidade, aproveitando heurísticas admissíveis para reduzir significativamente o número de nós explorados, mantendo a optimalidade da solução.

Concluímos que a escolha do algoritmo de busca mais adequado depende diretamente das características do problema, como a presença de pesos, a necessidade de encontrar caminhos ótimos e os recursos computacionais disponíveis. Em aplicações práticas, o A* tende a oferecer o melhor compromisso entre desempenho e qualidade da solução, desde que uma heurística apropriada esteja disponível.

References

[1] M. Kurant, A. Markopoulou, and P. Thiran, ”On the bias of BFS (Breadth First Search)”, 2010, pp. 1-8.

[2] S.A.M. Makki and G. Havas, ”Distributed algorithms for depth-first search”, vol. 60. IPL, 1996, pp. 7-12.

[3] C. Wilt, J. Thayer, and W. Ruml, “A Comparison of Greedy Search Algorithms”, SOCS, vol. 1, no. 1, pp. 129-136, Aug. 2010.

[4] X. Liu and D. Gong, ”A comparative study of A-star algorithms for search and rescue in perfect maze”, 2010, pp. 24-27.