Análise e Implementação do Algoritmo A* Busca Eficiente em Grafos

Klaus Pereira Becker
Engenharia da Computação
Universidade do Vale do Rio dos Sinos

***Resumo*—Este trabalho explora a aplicação do algoritmo A* na resolução de problemas de busca do caminho mais curto. Apresenta a fundamentação teórica do A*, sua evolução e características principais. Discute-se sua aplicação no planejamento de rotas urbanas, modelando-se o ambiente como um grafo. Complementarmente, detalha-se uma aplicação prática prévia do A* no controle de um carrinho seguidor de linha com obstáculos dinâmicos, demonstrando a versatilidade do algoritmo em diferentes contextos de automação e busca.**

I. Introdução

O planejamento de rotas otimizadas é um desafio fundamental em diversos campos da engenharia, da logística à robótica autônoma. A capacidade de encontrar o caminho mais eficiente em um ambiente, seja ele estático ou dinâmico, é crucial para a autonomia e o desempenho de sistemas computacionais. Dessa maneira, a seguir será apresentada uma análise do algoritmo A* (A-star), um poderoso algoritmo, amplamente reconhecido por sua eficiência na determinação do caminho de menor custo em grafos.

Este trabalho tem como objetivo analisar o algoritmo A*, detalhar sua implementação e demonstrar sua eficácia e adaptabilidade em contextos variados de busca de caminhos. Serão apresentadas a fundamentação histórica e teórica do A*, a descrição dos componentes principais (funções g, h e f), o passo a passo da implementação em C++ e a discussão dos resultados obtidos em aplicações reais, incluindo um carrinho seguidor de linha com obstáculos dinâmicos.

II. Fundamentação Teórica

A. Origem do Algoritmo

O algoritmo em questão foi inicialmente publicado em em 1968 por Peter Hart, Nils Nilsson e Bertram Raphael, pesquisadores do Stanford Research Institute (SRI), enquanto estavam trabalhando no projeto Shakey. Sua criação surgiu da necessidade de desenvolver métodos mais eficientes para planejamento de caminhos em robótica e inteligência artificial (IA), áreas então emergentes que exigiam algoritmos capazes de encontrar rotas ótimas em espaços de estado complexos.

Naquele período, os algoritmos de busca eram limitados em termos de eficiência e capacidade de lidar com ambientes dinâmicos. Algoritmos como o Dijkstra já existiam e garantiam a otimalidade ao explorar todos os caminhos possíveis até chegar no caminho com menor custo possível. No entanto, sua natureza tornava-o computacionalmente caro para problemas com grandes espaços de busca. O Best-First Search (BFS), por outro lado, aprimorou o processo de busca ao priorizar nós com base em uma função heurística, a qual trata-se de uma estimativa do custo restante para alcançar o objetivo. Contudo, o BFS não garantia a otimalidade, pois poderia escolher caminhos subótimos.

O A* surgiu como uma combinação inovadora entre a garantia de otimalidade de algoritmo e a eficiência da busca guiada por heurísticas. A função $f(n) = g(n) + h(n)$ foi o ponto chave, onde $g(n)$ trazia a informação do custo real (como em Dijkstra) e $h(n)$ trazia a estimativa heurística (como em BFS), permitindo um equilíbrio entre exploração e direcionamento ao objetivo. Isso o tornou o primeiro algoritmo a combinar a otimalidade com uma busca heurística eficiente, tornando-o um marco no campo da IA, que ainda estava engatinhando naquela época, e em problemas de busca de caminhos.

Sua eficácia reside na utilização de uma função de avaliação $f(n)$ para cada nó n, dada pela soma de dois componentes:

$$f(n) = g(n) + h(n)$$

Onde:

- $\boldsymbol{f(n)}$: Custo total estimado do nó inicial até o nó destino passando pelo nó atual n.
- $\boldsymbol{g(n)}$: Custo real do nó inicial até o nó atual n. Este valor é acumulado à medida que o algoritmo explora o grafo.
- $\boldsymbol{h(n)}$: Custo estimado heuristicamente do nó atual n até o nó destino.

O algoritmo, por sua vez, opera mantendo uma fila de nós a serem explorados, sempre expandindo o nó com o menor valor de $f(n)$. Essa estratégia permite que a busca seja guiada de forma inteligente em direção ao objetivo, evitando a exploração desnecessária de caminhos menos promissores e mais custosos, bem diferente de outros algoritmos de busca que podem explorar caminhos irrelevantes ou subótimos.

B. Propriedades da Heurística

A qualidade e as propriedades da função heurística $h(n)$ são cruciais para a performance e a garantia de otimalidade do Algoritmo A*. A escolha de uma heurística adequada pode reduzir significativamente o tempo de execução do algoritmo, enquanto uma heurística inadequada pode levar a um desempenho ruim ou até mesmo à falha em encontrar o caminho ótimo. As duas propriedades mais importantes que uma heurística deve satisfazer são:

- **Admissibilidade:** Uma heurística $h(n)$ é considerada admissível se ela nunca superestimar o custo real do nó n até o nó objetivo. Formalmente, $h(n) \leq h^*(n)$, onde $h^*(n)$ é o custo real mínimo de n ao objetivo. Dessa forma, obtendo uma heurística admissível temos que o A* encontrará o caminho de menor custo.
- **Consistência:** Uma heurística é consistente se, para qualquer nó n e qualquer vizinho v de n, o custo estimado de n ao objetivo for menor ou igual ao custo real de n a v mais o custo estimado de v ao objetivo. Formalmente, $h(n) \leq custo(n, v) + h(v)$.

A fim de exemplo, temos a distância Euclidiana como uma heurística admissível e consistente em um espaço bidimensional, pois ela sempre subestima a distância real entre dois pontos. A qual é representada pela fórmula, onde (x_1, y_1) e (x_2, y_2) são as coordenadas dos pontos:

$$h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Em outra vertente, para sistemas baseados em grades, como jogos de tabuleiro ou carrinho seguidor de linha, a distância de Manhattan é uma heurística admissível e consistente. Ela é calculada como a soma das diferenças absolutas das coordenadas, representando o custo de movimento em um grid onde apenas movimentos ortogonais (horizontal ou vertical) são permitidos:

$$h(n) = |x_2 - x_1| + |y_2 - y_1|$$

Por fim, temos que a consistência implica na admissibilidade, mas o contrário não pode ser afirmado. Uma heurística consistente garante que o valor de $f(n)$ seja não decrescente ao longo do caminho, o que é essencial para a eficiência do A*.

C. Uso e Aplicações

Desde sua introdução, o algoritmo de busca A* tem sido amplamente utilizado, transcendendo as fronteiras da robótica e IA para aplicações em diversas áreas, tornando-se um padrão ouro para problemas de busca de caminhos, onde temos a eficiência e a otimalidade como requisitos essenciais. Dessa maneira, dentre as aplicações mais comuns do A* estão:

- **Planejamento de Rotas Urbanas:** Em sistemas de navegação e GPS, o algoritmo A* é amplamente empregado para planejar rotas otimizadas em ambientes urbanos complexos. Plataformas como Google Maps® e Waze® o utilizam para considerar o custo real do percurso ($g(n)$) e estimativas heurísticas ($h(n)$) de tempo de viagem. Isso permite incorporar fatores dinâmicos como tráfego e condições das vias na determinação do caminho mais eficiente.
- **Jogos Eletrônicos:** Em jogos, o A* é usado para controlar o movimento de personagens não jogáveis (NPCs), permitindo que eles encontrem caminhos eficientes em ambientes dinâmicos e complexos, desviando de obstáculos e interagindo com o ambiente de forma inteligente.
- **Robótica Autônoma:** Esse algoritmo é fundamental em robótica, onde robôs autônomos precisam navegar em ambientes desconhecidos ou dinâmicos. Sendo utilizado desde robôs aspiradores de pó até veículos autônomos, o A* permite que esses sistemas encontrem rotas seguras e eficientes, evitando obstáculos e adaptando-se a mudanças no ambiente.
- **Sistemas de Transporte Inteligente:** O A* é utilizado em sistemas de controle de tráfego para otimizar o fluxo de veículos e reduzir congestionamentos.

A versatilidade do algoritmo permite sua aplicação em diversos contextos e tipos de grafos. Isso garante flexibilidade no projeto de heurísticas específicas para cada problema, equilibrando precisão e desempenho computacional.

III. Implementação do Algoritmo

A implementação do A* requer o gerenciamento de dois conjuntos de nós: aqueles que ainda precisam ser avaliados e aqueles que já foram. Para isso, são utilizadas duas estruturas de dados centrais: a **Lista Aberta** e a **Lista Fechada**. Sendo a eficiência dessas estruturas fundamental para o desempenho do algoritmo.

A. Estruturas de Dados Essenciais

- **Lista Aberta (Open List):** Armazena os nós que foram descobertos, mas cujos vizinhos ainda não foram totalmente explorados. É, essencialmente, uma fila de prioridade (*priority queue*). A cada passo, o nó com o menor custo $f(n)$ é removido da lista para ser processado. A escolha de uma fila de prioridade é crucial para garantir que o algoritmo sempre expanda o nó mais promissor.
- **Lista Fechada (Closed List):** Armazena os nós que já foram visitados e avaliados. Sua função é evitar que o algoritmo reavalie os mesmos nós, prevenindo a ocorrência de laços infinitos e garantindo que cada nó seja processado apenas uma vez. Uma tabela de hash (*hash set*) é uma escolha eficiente para esta lista, pois oferece tempo de busca de $O(1)$.

B. Passo a Passo do Algoritmo

O fluxo de execução do A* pode ser resumido nos seguintes passos:

1) **Definição do Grafo:** O grafo deve ser representado de forma que cada nó tenha um custo associado para alcançar seus vizinhos. Isso pode ser feito utilizando uma matriz de adjacência ou uma lista de adjacência, dependendo da densidade do grafo.
2) **Heurística:** Escolha uma função heurística $h(n)$ adequada ao problema. A heurística deve ser admissível e, preferencialmente, consistente para garantir a eficiência do algoritmo.
3) **Inicialização:**
   - Adicione o nó inicial à `lista aberta`.
   - O custo $g(inicial)$ é 0. O custo $f(inicial)$ é calculado usando a heurística $h(inicial)$.
   - A `lista fechada` inicia vazia.

4) **Loop Principal:** Enquanto a `lista aberta` não estiver vazia:
   - Selecione o nó `atual` da `lista aberta` que possui o menor valor de $f(n)$.
   - Se `atual` for o nó objetivo, a busca terminou. Inicie a reconstrução do caminho (detalhada a seguir).
   - Mova `atual` da `lista aberta` para a `lista fechada`.
5) **Expansão de Vizinhos:** Para cada vizinho `v` do nó `atual`:
   - Se `v` já estiver na `lista fechada`, ignore-o e continue.
   - Calcule o custo provisório passando por `v`, o qual denota-se $g(\mathtt{v})$, que é dado por: $g(\mathtt{atual}) + \mathtt{custo}(\mathtt{atual}, \mathtt{v})$.
   - Se `v` não estiver na `lista aberta`, adicione-o. Calcule seu $f(n)$, armazene seu $g(n)$ e defina `atual` como seu nó "pai", para que assim seja possível reconstruir o caminho posteriormente.
   - Se `v` já estiver na `lista aberta` e o novo caminho através de `atual` for mais curto (ou seja, $g(\mathtt{v})$ tentativo $< g(\mathtt{v})$ existente), atualize o $g(n)$ de `v` e seu nó pai para `atual`. Recalcule seu $f(n)$.
6) **Falha:** Se a `lista aberta` ficar vazia e o nó objetivo não for encontrado, significa que não existe um caminho válido.

C. Reconstrução do Caminho

Após o algoritmo encontrar o nó objetivo, o caminho ótimo não é retornado diretamente. Em vez disso, ele é reconstruído de forma retroativa. Como cada nó armazena uma referência ao seu "pai" (o nó a partir do qual foi alcançado com o menor custo), o caminho pode ser traçado partindo do nó objetivo e seguindo os ponteiros de pai em pai até chegar ao nó inicial. O resultado é o caminho inverso, que pode ser simplesmente revertido para obter a sequência correta de nós do início ao fim.

IV. Aplicação Prática

Para demonstrar a eficácia do A*, esta seção detalha sua aplicação em um projeto de um Veículo Autoguiado (AGV). O projeto consiste em um robô autônomo que navega em um ambiente estruturado no formato de uma grade, onde as linhas formam um grafo. O robô é equipado com sensores infravermelhos para detectar obstáculos e seguir as linhas.

A. Modelagem do Ambiente e Estratégia de Navegação

O ambiente é modelado como um grafo, onde as interseções das linhas formam os nós e os segmentos são as arestas. A cada nó, o robô precisa tomar uma decisão de qual caminho seguir, com suas ações limitadas aos movimentos cardinais: Norte, Sul, Leste ou Oeste. O desafio do sistema de controle é, portanto, gerar uma sequência ótima de movimentos direcionais que leve o robô da sua posição atual até o destino, contornando quaisquer bloqueios.

B. Execução do A* e Adaptação a Obstáculos

Para otimizar a rota, foi implementado o A* em um Arduino® utilizando os seguintes componentes: o custo do caminho ()$g(n)$) como o número de movimentos já realizados e a heurística ($h(n)$) como a Distância de Manhattan. O algoritmo então calcula a rota mais eficiente usando a função $f(n) = g(n) + h(n)$, resultando em uma sequência de comandos direcionais (ex: "Norte, Oeste..."). O principal motivo da escolha do A* foi sua capacidade de adaptação. Se um sensor detecta um obstáculo, o robô interrompe sua sequência, o mapa interno é atualizado com o novo nó bloqueado e o A* é reexecutado a partir da posição atual para recalcular o caminho de forma rápida e eficiente.

V. Conclusão

Este trabalho apresentou uma análise detalhada do algoritmo A*, conectando sua fundamentação teórica e contexto histórico à sua implementação e aplicação prática em um sistema robótico. A exploração demonstrou que a eficiência do A* emerge de um balanço preciso entre o custo real acumulado, $g(n)$, e a estimativa heurística inteligente, $h(n)$. Essa combinação, que se mostrou robusta no Veículo Autoguiado (AGV) ao lidar com obstáculos dinâmicos em tempo real, confirma o A* como uma solução de alta performance para problemas de busca de caminho, mesmo em hardware com recursos limitados.

A análise aprofundada do algoritmo revelou-se particularmente interessante, não apenas por sua mecânica, mas por sua história. Compreender seus objetivos iniciais, no contexto do projeto Shakey, permitiu uma apreciação maior de sua engenhosidade e do pensamento visionário de seus criadores. Fica evidente que o A* transcende sua função como um mero algoritmo; ele representa uma filosofia de design fundamental para a criação de sistemas inteligentes, onde a tomada de decisão informada supera a força bruta.

Conclui-se, portanto, que o domínio do A* é indispensável na engenharia e na ciência da computação, não apenas como uma ferramenta clássica, mas como um exemplo duradouro de como uma solução elegante e bem-fundamentada resiste ao teste do tempo, mantendo-se relevante e poderosa em um campo em constante evolução.

Referências

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968.

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.

[4] K. Becker and C. Souza, *Automated Guided Vehicle (AGV)*. GitHub repository, 2024. [Online]. Available: https://github.com/klsbecker/automated-guided-vehicle