Algoritmo A*: encontrando o menor caminho entre dois pontos

Ana Vitória V. Cordeiro

Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brasil
avvc@icomp.ufam.edu.br

1. Introdução

Achar o caminho mais curto entre dois pontos é algo que se parece difícil à primeira vista, considerando que em um jogo, por exemplo, existem diversos caminhos que um personagem pode trilhar para atingir um objetivo, além de obstáculos que podem, eventualmente, impedir seu movimento. Dependendo do ambiente de atuação, pode tornar-se complicado encontrar uma boa solução para este problema em um tempo aceitável, de fato, mas graças a elegantes algoritmos propostos por estudiosos como Dijkstra, Peter Hart, Nils Nilsson e Bertram Raphael, é possível ter uma ideia clara e geral de como isso pode ser feito. Estes algoritmos atuam por meio da modelagem do ambiente em *grafos*, que são estruturas matemáticas onde pontos podem ser representados por vértices e uma relação entre eles (no caso, vizinhança), arestas que conectam estes vértices.

O algoritmo proposto por Dijkstra trabalha visitando os vértices do grafo a partir do ponto inicial, expandindo sua busca pelos vértices mais próximos do ponto inicial que ainda não foram visitados, até que o objetivo seja atingido. Este algoritmo dá a garantia de que o caminho mais curto será encontrado, porém não é tão rápido como, por exemplo, o algoritmo conhecido como Greedy BFS (Best-First-Search) ou Busca Gulosa em Profundidade. Este funciona de forma parecida, mas, ao invés de usar como critério para escolher um novo vértice sua distância do ponto inicial, utiliza uma *heurística* que fornece a estimativa de distância entre um ponto e o objetivo. Este algoritmo pode funcionar de maneira muito mais rápida que o de Dijkstra, mas não há garantias de que achará o caminho mais curto se existirem obstáculos, por exemplo [Patel]. O algoritmo tema deste trabalho, conhecido como A*, consiste numa fusão destas duas formas de busca de caminhos em um grafo. Foi descrito pela primeira vez em 1968 por Peter Hart, Nils Nilsson, e Bertram Raphael e é utilizado, principalmente, na produção de *jogos* e em robótica.

2. Algoritmo A*

Tomando $g(x)$ como o custo total para se alcançar determinado nó x e $h(x)$ o custo estimado para atingir o objetivo da busca (heurística), tem-se que $f(x) = g(x) + h(x)$ é o custo estimado total do caminho que x intermedia. Dessa forma, se $g(x)$ é o caminho mais curto até x e $h(x)$ o custo estimado do menor caminho de origem em x, então $f(x)$ é o custo da solução mais barata que contém x [Russel e Norvig 2010]. Esta é a base do funcionamento do A*.

O algoritmo terá um resultado ótimo se algumas condições forem atendidas. Se o espaço de busca é uma árvore e a função heurística adotada é admissível, ou seja, nunca calcula exageradamente o custo para alcançar o nó de destino, haverá um resultado ótimo.

Isto também vale se o espaço de busca for um grafo e $h(n)$, consistente, isto é, para cada nó n e seu sucessor n', vale que $h(n) \leq custo(n, n') + h(n')$ [Muñoz-Avila 2005]. Uma função heurística consistente sempre será admissível. Exemplos de heurísticas consistentes utilizadas em alguns contextos de aplicação do A* são: distância Euclidiana, distância de Manhattan, entre outras.

O algoritmo possui um funcionamento simples. Existem dois conjuntos, o de *nós abertos* e o de *nós fechados*. O primeiro contém os nós que são candidatos a serem examinados. Inicialmente, este conjunto possui apenas um elemento: o nó inicial. Já o conjunto de nós fechados contém os elementos que já foram examinados, assim, inicialmente será vazio. Para cada nó, é armazenada a informação referente ao vizinho pelo qual foi alcançado, ou seja, seu nó *pai*. A partir disto, é construído um laço de repetição onde o elemento da lista de abertos com menor valor de $f(n)$ é examinado. Se esse nó corresponder ao objetivo, então o caminho já foi encontrado. Senão, o nó é removido da lista de abertos e posto na de fechados. A partir daí, cada vizinho n' desse nó é examinado e um *custo de tentativa* é calculado, pela soma de $g(n)$ com o custo de n a n'. Se n' não está na lista de abertos e este custo de tentativa é menor que $g(n')$, então n será armazenado como pai de n' e seu $g(n')$ receberá o valor do custo de tentativa. Se n' não estiver, de fato, no conjunto de nós abertos, será inserido nele.

Algoritmo 1 A*

função A*($origem, destino$)
    $nos\_fechados \leftarrow \{\}$; $nos\_abertos \leftarrow \{destino\}$;
    $mapa\_navegados \leftarrow \{\}$;
    $g\_custo[origem] \leftarrow 0$;
    $f\_custo[origem] \leftarrow g\_custo[origem] + h(origem, destino)$;
    enquanto $nos\_abertos$ não está vazio **faça**
        $atual \leftarrow no$ em $nos\_abertos$ com menor valor de $f\_custo[]$;
        se $atual = destino$ **então**
            retorne $reconstruir\_caminho(mapa\_navegados, destino)$;
        fim se
        $nos\_abertos \leftarrow nos\_abertos \backslash \{atual\}$;
        $nos\_fechados \leftarrow nos\_fechados \cup \{atual\}$;
        para $vizinho \leftarrow$ primeiro dos $vizinhos(atual)$ **até** último dos $vizinhos(atual)$ **faça**
            se $vizinho$ está em $nos\_fechados$ **então**
                continue;
            fim se
            $g\_custo\_tentativa \leftarrow g\_custo[atual] + custo\_entre(atual, vizinho)$;
            se $vizinho$ não está em $nos\_abertos$ ou $g\_custo\_tentativa < g\_custo[vizinho]$ **então**
                $mapa\_navegados[vizinho] \leftarrow g\_custo\_tentativa$;
                $f\_custo[vizinho] \leftarrow g\_custo[vizinho] + h(vizinho, destino)$;
                $mapa\_navegados[vizinho] \leftarrow atual$;

```
22:                 se vizinho não está em nos_abertos então
23:                     nos_abertos ← nos_abertos ∪ {atual};
24:                 fim se
25:             fim se
26:         fim para
27:     fim enquanto
28: fim função
29:
30: função RECONSTRUIR_CAMINHO(mapa_navegados, destino)
31:     caminho ← {atual};
32:     enquanto atual está em mapa_navegados faça
33:         atual ← mapa_navegados[atual];
34:         caminho ← caminho ∪ {atual};
35:     fim enquanto
36: fim função
```

3. Implementação

A implementação do algoritmo A* foi realizada na linguagem C++, com compilador de suporte à versão C++11. Os testes foram realizados no terminal de comandos do Linux.

Esta implementação busca solucionar, especificamente, o problema de locomoção dentro do país de Romênia, proposto por [Russel e Norvig 2010]. O objetivo, originalmente, é realizar a viagem entre a cidade de "Arad" e "Bucharest" passando pelo menor caminho possível. Para o funcionamento do algoritmo neste cenário, o autor propõe uma função heurística baseada na distância Euclidiana entre dois pontos. Assim, fornece uma tabela indicando a distância de cada cidade do mapa a "Bucharest". Esta tabela foi adotada na implementação da heurística deste cenário, e isto faz com alguns testes não retornem o caminho mais curto, como será explicado posteriormente.

3.1. Testes

#1

```
Origem: Arad
Destino: Bucharest

Menor caminho: Arad-> Sibiu-> Fagaras-> Bucharest
```

#2

```
Origem: Fagaras
Destino: Craiova

Menor caminho: Fagaras-> Sibiu-> Rimnicu Vilcea-> Craiova
```

#3

```
Origem: Rimnicu Vilcea
Destino: Urziceni

Menor caminho: Rimnicu Vilcea-> Pitesti-> Bucharest-> Urziceni
```

#4

```
Origem: Timisoara
Destino: Craiova

Menor caminho: Timisoara-> Lugoj-> Mehadia-> Dobreta-> Craiova
```

#5

```
Origem: Urziceni
Destino: Dobreta

Menor caminho: Urziceni-> Bucharest-> Fagaras-> Sibiu-> Arad->
 Timisoara-> Lugoj-> Mehadia-> Dobreta
```

Neste último teste, o caminho retornado não foi o mínimo. A explicação para isto é que, como a função heurística utilizada no problema foi baseada na tabela do livro de [Russel e Norvig 2010], que fornece as distâncias aproximadas das cidades a Bucharest, o resultado perde admissibilidade quando a origem está próxima de Bucharest e o destino, longe. Isto acontece porque o algoritmo passa a adotar um comportamento oposto ao apropriado: escolhe os pontos mais distantes do destino para compor o caminho. Este comportamento não se aplica aos outros testes pois o destino está localizado sempre perto de Bucharest e a origem, distante, logo, retornam soluções ótimas para o problema.

4. Conclusões

O algoritmo A* é muito utilizado no problema de achar o menor caminho em um ambiente repleto de obstáculos, como nos jogos. Sem dúvidas, é uma opção de algoritmo eficiente, rápido e com garantia de bons resultados (se a heurística utilizada for consistente). Além de jogos, possui aplicação em diversos problemas que podem ser modelados em grafos e envolvem a busca de determinado caminho.

Referências

Muñoz-Avila, H. (2005). A* algorithm. *Topics on AI and Computer Game Programming*.

Patel, A. Introduction to a*. *http://theory.stanford.edu/ amitp/GameProgramming/AStar Comparison.html. Acesso em 05/06/2015*.

Russel, S. e Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. 3ª edição.