A*: Refinamento de Buscas Heurísticas por meio do Algoritmo *A* Adaptativo*

Yan Victor dos Santos
yanvictor_ds@hotmail.com

Abstract

Funções heurísticas são definidas para otimizar processos de busca em grafos conexos que fornecem informações sobre o peso de suas arestas, permitindo encontrar o caminho menos custoso entre dois vértices. Em problemas reais, é possível descrever cenários por meio de um grafo, onde os vértices são estados de situações específicas de modo que a relação entre eles descreve o fluxo de continuidade de busca por uma ou mais soluções. O objetivo deste trabalho é mostrar como o algoritmo A* [1] consegue lidar com grafos bem definidos com informações de conhecimento, visando achar a melhor solução para um problema específico. Por fim, este artigo também descreve como a função heurística que define a forma de percorrimento pode ser refinada para otimizar a busca e diminuir a expansão de estados, sendo esta uma característica de *A* Adaptativo*. Atualmente, estes dois algoritmos possuem grande importância em buscas informadas, pois fornecem ótimas soluções.

Keywords: Algoritmo A*; Adaptative A*; Busca Gulosa; Busca em Largura; Busca de Custo Uniforme

1 Introdução

A Inteligência Artifical está sempre a pesquisar formas de tornar mais eficiente a busca heurística para problemas que envolvem cenários complexos. Para isto, algoritmos bastante conhecidos são usados como base para avançar no processo de resolução ótima e completa dos mais diversos problemas. Como forma de solução e divulgação, este trabalho apresenta os principais conceitos do algoritmo A*, que busca encontrar sempre a solução ótima usando o conhecimento fornecido pelo próprio problema. Além do mais, podemos apresentar formas de tornar este algoritmo ainda mais eficiente ao refinarmos a função heurística durante vários processos de busca. Além de ter o poder de diminuir a quantidade de estados expandidos, é possível aplicar a cenários que possuem incremento de custos nas arestas. Tal ideia é inserida dentro do algoritmo *A* Adaptativo* [2].

O resto do artigo está organizado da seguinte forma. A seção 2 fornece conceitos e informações necessárias para a compreensão do algoritmo A*. A seção 3 apresenta o próprio algoritmo, explicando seu funcionamento, vantagens e desvantagens. Um exemplo de busca usando A* também será apresentada nesta seção. A descrição do algoritmo *A* Adaptativo* será apresentado na seção 4, seguida da conclusão descrita na seção 5.

2 Base de Conhecimento

Para compreendermos o funcionamento do algoritmo A*, precisamos entender como funcionam também os algoritmos Guloso e Busca de Custo Uniforme, pois são essenciais para a formação do A* e de seu derivado A* Adaptativo.

2.1 Algoritmo Guloso

Um algoritmo guloso é "míope", isto é, toma decisões com base nas informações disponíveis na iteração corrente sem olhar as consequências que essas decisões terão no futuro. Um algoritmo guloso jamais se arrepende ou volta atrás: as escolhas que faz em cada iteração são definitivas. A função heurística *h(n)* é o que define o próximo passo durante o percorrimento de um grafo. Em geral, algoritmos gulosos são rápidos e eficientes, embora sua forma de execução não garante a resolução de problemas complexos e pode simplesmente não solucioná-los em todos os cenários

possíveis. Embora seja uma busca informada (heurística), é necessário compreender a existência de um outro algoritmo conhecido como Busca de Custo Uniforme, que complementará a análise do ambiente de atuação para solucionar problemas por meio do algoritmo A*.

2.2 Busca de Custo Uniforme

A estratégia de busca uniforme é parecida com a estratégia da busca em largura. Ao invés de pegar o primeiro nó da lista de processamento para ser expandido, o nó que possui o menor custo será o nó que deverá ser expandido. A função *g(n)* retorna o custo do nó inicial até o nó *n*. Desta forma o ramo da árvore com menor custo em relação ao nó inicial tende sempre a ser percorrido, fazendo com que a solução ótima seja encontrada em algum momento. O caso onde todos os nós do próximo nível possuem custo iguais acaba se tornando um caso especial da busca em largura.

Por ser um algoritmo de busca cega, sua função baseia-se sempre em "olhar para trás", enquanto o algoritmo guloso sempre tende a "olhar para frente". É por isso que o algoritmo A* sempre irá "olhar para frente e para trás ao mesmo tempo", uma vez que ele a abrange a noção dos dois algoritmos anteriores.

3 Algoritmo A*

Uma vez tendo compreendido como funcionam os algoritmos de Custo Uniforme e Guloso, podemos entender facilmente o que é e como o algoritmo A* expande os nós na fronteira de forma a encontrar a solução ótima.

Obter informações do problema para buscar o próximo nó caracteriza um percorrimento conhecido como **busca heurística**. A heurística é a característica principal do A*, pois é essencial conhecer bem o o problema e seu ambiente de atuação para que seja possível "olhar para frente e para trás ao mesmo tempo".

O Algoritmo A* é um tipo de busca informada que engloba em si os conceitos da busca de custo uniforme e busca gulosa, ao considerar informações de custo do nó inicial até o um determinado nó *n*, e deste último até o nó final (objetivo) para encontrar a solução do problema. Ao utilizar o conhecimento adquirido no problema para tomar decisões de expansão de nós, a função de avaliação *f(n)* do A* retorna o custo da solução ótima do nó inicial até o nó objetivo. A função *f(n)* é definida da seguinte forma:

$$\boldsymbol{f(n) = g(n) + h(n)},$$

onde *g(n)* fornece o custo do caminho do nó inicial até o nó *n*, enquanto *h(n)* é a função heurística que calcula o custo estimado do caminho mais barato do nó *n* até o nó final (objetivo), que no caso do exemplo deste artigo é o "caminho reto" entre os dois nós. A ideia é expandir o nó que possui o menor valor de retorno de *f(n)* possível.

3.1 Vantagens e Desvantagens da utilização do algoritmo A*

Assim como a busca em largura, o algoritmo A* é **completo** e sempre buscará pela **solução ótima**, baseando-se na função heurística *h(n)* definida para calcular *f(n)*. Além do mais, este algoritmo pode ser modificado facilmente para encontrar uma solução ótima em cenários mais complexos, que o algoritmo original em si não poderia encontrar.

Uma das desvantagens do algoritmo, é o fato de que os valores das arestas precisam ser fixos e seu ambiente deve ser completamente conhecido. Isto reduz drasticamente a quantidade de problemas que o algoritmo pode resolver, uma vez que um ambiente com possíveis mudanças pode ferir completamente a integridade do resultado do algoritmo A*. Uma outra desvantagem está relacionada a problemas que atuam em tempo real, onde movimentação precisa ser realizada em paralelo ou agente pode executar movimento durante cada *slot* de tempo. O principal motivo é que A* precisa de um longo tempo para realizar a busca, e depois a movimentação.

3.2 Exemplo

A Figura 1 apresenta a solução da busca de *Arad* para *Bucharest* por meio do algoritmo A*. A tabela ao lado apresenta a distância em linha reta de cada nó para o nó final. Esta característica reflete o modo de busca heurística.

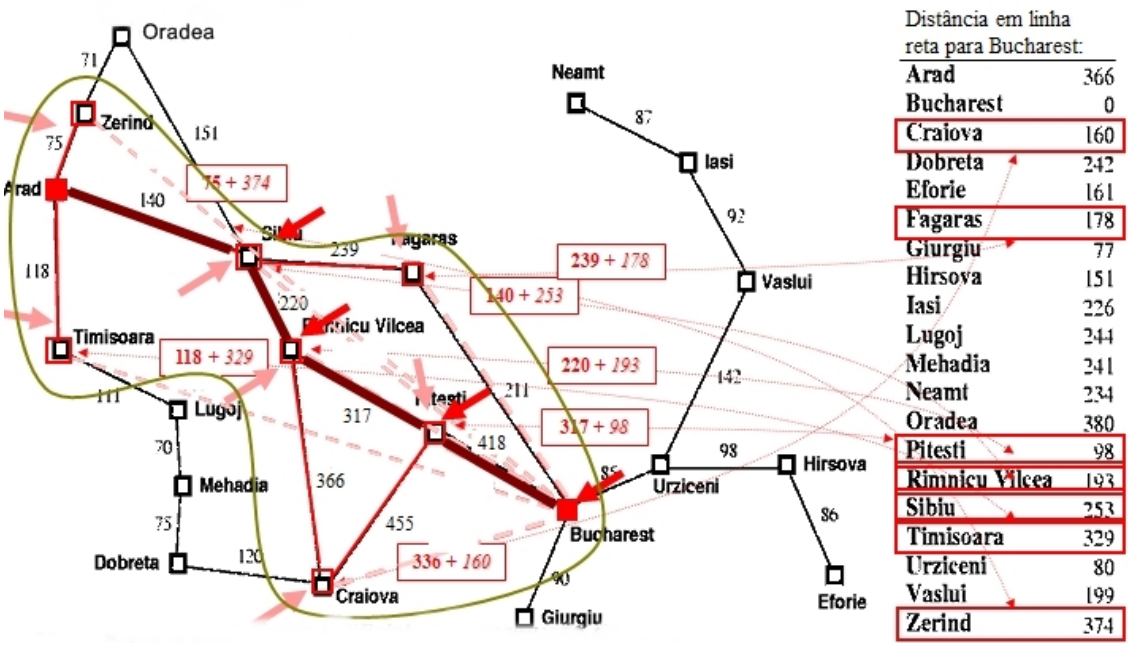

Figura 1. Exemplo Algoritmo A*

Para cada nó *n*, *f(n)* é calculada somando-se a distância de *Arad* até *n* com a distância reta de *n* até *Bucharest*. Para cada nível, o nó da fronteira que possuir o menor valor calculado é expandido até que o caminho mais barato seja definido. A solução ótima apresentada segue o seguinte caminho:

Arad-Sibiu-Rimnicu Vilcea-Pitesti-Bucharest.

Custo Calculado: 418.

4 A* Adaptativo

O *A* Adaptativo* encontra repetidamente caminhos de custo mínimo para problemas de pesquisa com o mesmo objetivo, onde o custo das arestas pode aumentar entre os percorrimentos. Ele usa buscas A* para encontrar os caminhos de custos mínimos e atualiza as heurísticas ao longo do tempo para torná-las mais informadas e, portanto, as buscas A* futuras mais focadas. Em outras palavras, para tornar o algoritmo A* [3] mais eficiente, podemos melhorar a função heurística *h(n)* simplesmente atualizando os valores usando informações de pesquisas anteriores.

Esta variação do A* é uma busca heurística incremental que resolve uma série de problemas semelhantes e mais rápido que A*. Este algoritmo é simples de entender e fácil de implementar, pois transforma os valores consistentes de *h* com relação ao nó final em valores mais consistentes e informados ainda.

4.1 Refinando Heurísticas

Para melhorar os valores da heurística durante o percorrimento do grafo, bastaria executar a seguinte atualização:

$$h(n) = f^* - g(n),$$

onde f^* é o custo mínimo encontrado após uma busca A*. É possível observar que, a cada iteração, o valor de f^* tende a diminuir para que o resultado desta função seja um caminho completamente ótimo.

Sendo este princípio usado em [4], mais tarde acabou tornando-se sólido e recorrente o uso criando assim o *A* Adaptativo*.

4.2 A versão ansiosa do A* Adaptativo

A versão ansiosa realiza uma busca A* e atualiza a heurística de todos os estados que foram expandidos durante o percorrimento de A* atribuindo *h[n] = f* - g[n].* As pesquisas A* futuras não podem expandir mais estados com a nova heurística do que a outra com a heurística antiga. Muitas vezes expandem poucos estados e, portanto, executam muito mais rápido, o que justifica a pequena quantidade de tempo de execução necessária para atualizar as heurísticas de todas as extensões após cada busca A*. Nesse sentido, *A* Adaptativo* consegue superar o algoritmo A*, uma vez que o mesmo só usa uma heurística para percorrer.

A versão ansiosa do Adaptive A* primeiro percorre e, em seguida, atualiza a heurística dos estados que foram expandidos durante a busca. Uma desvantagem de atualizar a heurística de todos os estados que foram expandidos é que provavelmente ele atualiza a heurística de estados que não são necessários em buscas futuras.

4.3 A versão preguiçosa do A* Adaptativo

A versão preguiçosa se lembra de algumas informações no momento em que um estado n é expandido durante a busca, juntamente com algumas informações do momento em que a pesquisa é encerrada (como o custo *f** do caminho encontrado) e, em seguida, usa essas informações para calcular a nova heurística *h[s]* do estado n **quando for necessário** durante buscas futuras. Esta versão evita calcular de forma desnecessária para estados que não serão utilizados futuramente, melhorando ainda mais a efetividade do algoritmo.

5 Conclusão

Algoritmos de busca heurística são ótimas formas de encontrar soluções para diversos tipos de problemas. Uma vez que a busca gulosa não garante sucesso em uma parte dos casos por sempre "engolir" a melhor opção futura, podemos usar A* para melhorar sua precisão. "Olhar para frente e para trás" durante o percorrimento de um grafo pode ser uma maneira completamente ótima de solucionar problemas assim, embora não seja tão eficiente quanto o desejado.

Em problemas maiores e mais complexos, percorrer baseando-se na característica gulosa ao mesmo tempo que se analisa a distância que se tem do início pode demorar mais do que o desejado. Por isso o algoritmo precisa se adaptar e renovar a heurística antiga para diminuir ainda mais o tempo de busca, de forma a tornar ainda mais eficiente o uso do algoritmo A* em cenários que demandam uma rápida execução de código. Este algoritmo pode se tornar ótimo e eficiente, uma vez que se adaptar durante a execução de buscas é um grande método para diminuir tempo, como é definido no *A* Adaptativo*.

Embora se adaptar ao percorrimento em um grafo o torne mais eficiente, ainda assim é inviável aplicar a problemas em tempo real que demandam resposta rápida. Para problemas deste nível temos uma série de variações, tomando como exemplo o algoritmo Time-Bounded A*. É requerível aplicar mudanças em A* para tornar isto possível. Estas mudanças podem ser aplicadas facilmente, sendo este um de seus principais pontos positivos.

6 O Mundo de Wumpus

Para resolvermos o problema do Mundo de Wumpus, vamos provar que o Wumpus está na caverna (1,3).

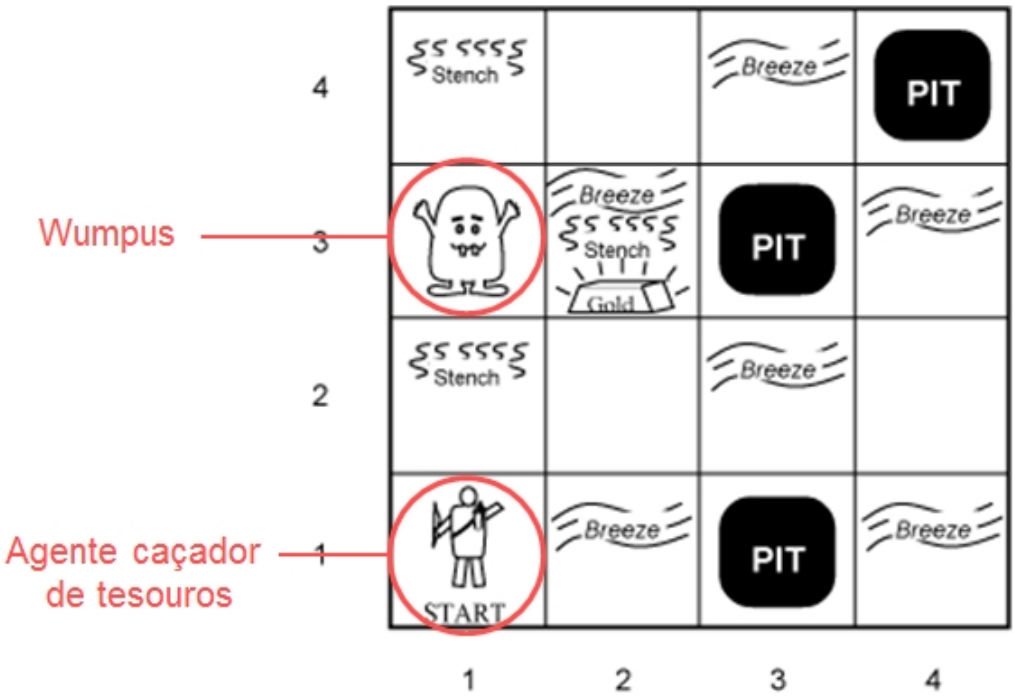

Figura 2. O Mundo de Wumpus

Passo 1: Estado Inicial A(1,1)

- Base de Conhecimento: ¬f(1,1), ¬b(1,1), ¬W(1,1).
  Pela BC, se não há fedor na caverna (1,1), então quer dizer que não há Wumpus em nenhuma caverna adjacente a ela. Portanto, o Agente terá uma regra de inferência para cada caverna no seu ambiente.
- Regras de Inferência:
  1. R1: ¬f(1,1) => ¬W(1,1) **e** ¬W(1,2) **e** ¬W(2,1)
  2. R2: ¬f(2,1) => ¬W(1,1) **e** ¬W(2,1) **e** ¬W(2,2) **e** ¬W(3,1)
  3. R3: ¬f(1,2) => ¬W(1,1) **e** ¬W(1,2) **e** ¬W(2,2) **e** ¬W(1,3)

Passo 2: Agente anda para a caverna (1,2), pois na BC contemos ¬b(1,1) e ¬W(1,1) indicando que não há Wumpus ou Buraco na caverna (1,2).

- Nova Base de Conhecimento: **f(1,2)**, ¬f(1,1), ¬b(1,1), ¬W(1,1).
- Regras de Inferência: *Nova regra R4.*
  1. R1: ¬f(1,1) => ¬W(1,1) **e** ¬W(1,2) **e** ¬W(2,1)
  2. R2: ¬f(2,1) => ¬W(1,1) **e** ¬W(2,1) **e** ¬W(2,2) **e** ¬W(3,1)
  3. R3: ¬f(1,2) => ¬W(1,1) **e** ¬W(1,2) **e** ¬W(2,2) **e** ¬W(1,3)
  4. **R4: f(1,2) => W(1,3) ou W(1,2) ou W(2,2) ou W(1,1)**

Para provar que o Wumpus está na caverna (1,3), vou mostrar que ele não está em nenhuma outra caverna, e então concluir, por eliminação, que ele está em (1,3).
Usando as Regras de infêrencia R1, R2 e R4:

- 1. Aplicando *Modus Ponens* em ¬f(1,1) e R1, temos: ¬W(1,1) **e** ¬W(1,2) **e** ¬W(2,1).
  2. Por E-eliminação em R1, concluímos: ¬W(1,1), ¬W(1,2), ¬W(2,1).
  3. Aplicando *Modus Ponens* em ¬f(2,1) e R2, temos: ¬W(1,1) **e** ¬W(2,1) **e** ¬W(2,2) **e** ¬W(3,1).

4. Por E-eliminação em R2, concluímos: ¬W(1,1), ¬W(2,1), ¬W(2,2), ¬W(3,1).
5. Aplicando *Modus Ponens* em ¬f(1,2) e R4, temos: W(1,3) **ou** W(1,2) **ou** W(2,2) **ou** W(1,1).
6. Por Resolução de Unidade, temos: Seja A=W(1,3) **ou** W(1,2) **ou** W(2,2) e B=W(1,1). Pelo passo 2, temos que B e ¬W(1,1) gera um *Absurdo*, fazendo com que B seja *falso*.
7. Por Resolução de Unidade, temos: Seja A=W(1,3) **ou** W(1,2) e B=W(2,2). Pelo passo 4, temos que B e ¬W(2,2) gera um *Absurdo*, fazendo com que B seja *falso*.
8. Por Resolução de Unidade, temos: Seja A=W(1,3) e B=W(1,2). Pelo passo 2, temos que B e ¬W(1,2) gera um *Absurdo*, fazendo com que B seja *falso*.
9. Como W(1,3) não leva a nenhum *Absurdo* pela regra de inferência de R1 ou R2, apenas bastou aplicar *Modus Ponens* em f(1,2) e R4, f(1,2) => A, onde A=W(1,3). Desta forma, obtemos W(1,3) como *verdadeiro*, **provando que o Wumpus está na caverna (1,3)**.

References

[1] N. N. P. Hart and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths.," *IEEE Transactions on Systems Science and Cybernetics*, pp. 2:100–107, 1968.

[2] S. K. Xiaoxun Sun and W. Yeoh, "Generalized adaptive a*.," *7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pp. 02–05, 2008.

[3] S. Koenig and M. Likhachev., "A new principle for incremental heuristic search: Theoretical results.," *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 402–405, 2006.

[4] R. Z. R. Holte, T. Mkadmi and A. MacDonald., "Speeding up problem solving by abstraction: A graph oriented approach. artificial intelligence," *Artificial Intelligence*, p. 85(1–2):321–361, 1996.