Inteligência Artificial

Prof. Robson de Souza

Busca guiada por conhecimento

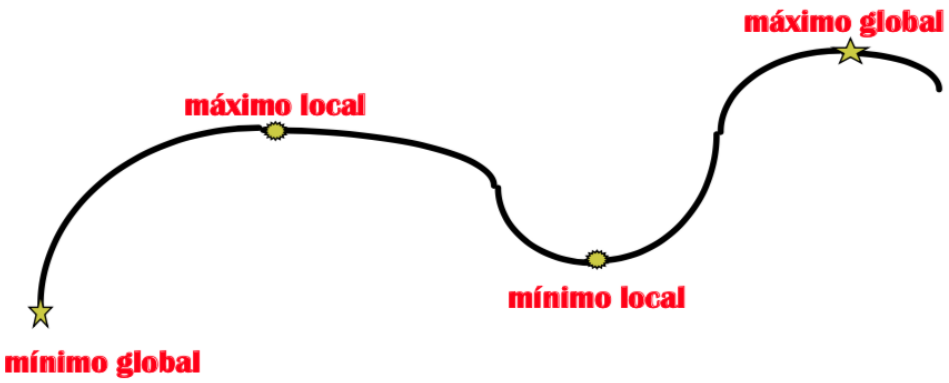
A Busca em Largura e Busca em Profundidade são métodos bastante gerais (pouco eficientes) de se efetuar uma busca, pois não dependem de qualquer conhecimento sobre como resolver o problema. A busca em largura e a busca em profundidade (com backtracking) são sempre capazes de fazer a busca progredir e sempre capazes de achar uma solução (caso ela exista) – são métodos completos.

Porém, esse tipo de busca não costuma ser eficiente, uma vez que não possui qualquer tipo de conhecimento para se guiar. Pensando nisso, às vezes é melhor se usar uma técnica de busca que pode não ser completa, mas que seja eficiente.

Busca Heurística

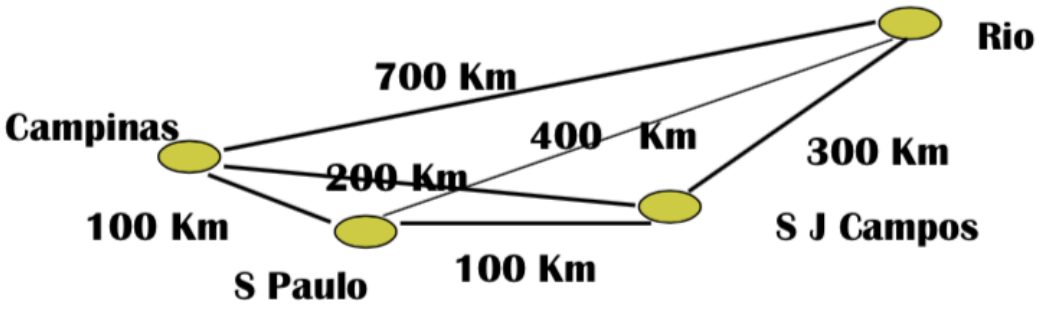
Um dos tipos de busca que podem ser utilizados para resolver problemas em um ambiente é a busca heurística. Uma heurística é um método para descobrir a solução de um problema. É um método prático, que nem sempre é garantido funcionar (mas na maioria das vezes funciona) mas que é um método eficiente de solucionar o problema (quando funciona) **embora muitas vezes produza uma solução sub-ótima**.

Na IA utilizamos os conceitos de **ótimos globais** e **ótimos locais**, um ótimo global é a melhor solução possível (solução ótima), enquanto um ótimo local é uma solução (solução satisfatória), porém não é a melhor possível. Observe a figura abaixo:



Seres humanos não costumam procurar soluções ótimas, mas sim soluções satisfatórias, porque o tempo para buscar a solução é sempre limitado e a eficiência é essencial. Como exemplo, observe o problema do caixeiro viajante:

Um caixeiro viajante tem que visitar todas as cidades em uma dada lista, mas visitando só uma vez cada cidade (a não ser a cidade de origem, que também é a cidade de destino). Existem estradas ligando todas as cidades da lista. As distâncias entre os vários pares de cidades são conhecidas. Ache o **caminho** que o caixeiro viajante deve percorrer de modo que a **distância** que ele percorra, saindo de e voltando para cada uma das cidades na rota, seja **mínima**.



Para resolver esse problema, temos a opção de utilizar uma solução via força bruta, que encontrará de forma garantida a melhor rota possível. Assim, ela deve calcular todos os caminhos possíveis e escolher o mais curto. O problema é que essa solução é muito ineficiente, pois é uma solução de complexidade exponencial.

Indo para a área de análise de algoritmos, tem-se uma solução um pouco mais eficiente que é o branch-and-bound (ramifica-e-poda), basicamente, essa solução pede que o algoritmo fique gerando caminhos completos e guarde o caminho mais curto achado até o momento (correspondendo a uma distância total D); assim que o caminho que esteja correntemente sendo explorado corresponda a uma distância (parcial) maior que D, descarte este caminho. O branch-and-bound garantidamente acha uma solução e é bem mais eficiente que a solução via força-bruta, mas ainda é um algoritmo de complexidade exponencial.

Finalmente, indo para a área da IA, pode-se utilizar uma solução heurística, que é a heurística do vizinho mais próximo. Basicamente, essa heurística funciona do seguinte modo:

1 - Escolha uma cidade qualquer;
2 - para selecionar a próxima cidade, considere todas as cidades ainda não visitadas e escolha a cidade mais próxima da corrente;
3 - vá para ela;
4 - repita o passo 3 até que todas as cidades tenham sido visitadas.

A heurística do vizinho mais próximo é uma heurística bastante geral que executa em tempo proporcional a N^2 (onde N é o número de cidades a serem visitadas), ou seja, é bastante eficiente, mas não é garantido que ela ache a solução ótima – não é uma heurística completa. Isso significa que não é garantido que ela ache um caminho cuja distância total percorrida seja um mínimo global porque essa heurística opera de forma local: ela não investiga exaustivamente o espaço de possibilidades de forma global, ela só investiga a continuação do caminho na proximidade da cidade correntemente visitada, localmente.

Vale ressaltar que embora uma heurística nem sempre produza uma solução ótima, só nos piores casos (worst cases) a solução que ela produz é muito ruim – e os piores casos possíveis costumam ser raros.

Para melhorar a eficiência de uma busca heurística, é possível incorporar conhecimento sobre a tarefa em maior ou menor grau. **Conhecimento heurístico** é o conhecimento sobre a tarefa incorporado numa heurística. Conhecimento heurístico pode ser incorporado num procedimento de busca baseado em regras de produção, ou nas próprias regras ou até mesmo como uma função heurística a parte. Por exemplo:

No jogo de damas, se existe uma peça que pode virar dama com este movimento então mova essa peça de modo que ela vire dama; (regras de produção)

No caso de uma função heurística a parte, ela avalia cada estado do problema a ser considerado na busca e determina quão desejável é esse estado. Isso significa que uma **função heurística** é uma função que associa a cada estado do problema uma medida de quão desejável é esse estado. Geralmente essa medida é um número. Às vezes a função heurística indica quão bom é o estado; às vezes ela indica quão ruim é o estado. Isso é importante para saber se o objetivo será maximizar ou minimizar a função heurística.

Exemplos de funções heurísticas:

<u>Problema do Caixeiro Viajante:</u> A soma das distâncias até o momento.

<u>Jogo-da-velha:</u>

1 para cada linha, coluna ou diagonal em que podemos vencer e na qual já temos uma peça.
2 para cada linha, coluna ou diagonal em que podemos vencer e na qual já temos duas peças.

De acordo com o valor da função heurística calculada em um dado estado em um espaço de estados sabe-se se é provável ou não que esse estado esteja num caminho que leve à solução. Quanto mais precisa for a avaliação feita pela função heurística, mais acertada é a escolha do caminho a seguir na árvore de busca.

Entretanto, uma função heurística muito precisa pode ser muito cara em recursos. Isto vai contra a principal

vantagem de usar uma função heurística, que é a eficiência na busca. Portanto, existe um trade-off entre precisão e custo de avaliação da função heurística.

Existem algoritmos que são utilizados para calcular os valores das funções heurísticas. Um exemplo desses algoritmos é o **algoritmo minimax**, que permite lidar com situações como as encontradas em jogos em que dois adversários jogam alternadamente, ele se baseia no desenvolvimento da árvore de busca em níveis maiores de profundidade e no fato de que o objetivo de cada jogador é derrotar seu adversário.

Algoritmo minimax

O algoritmo minimax funciona do seguinte modo:

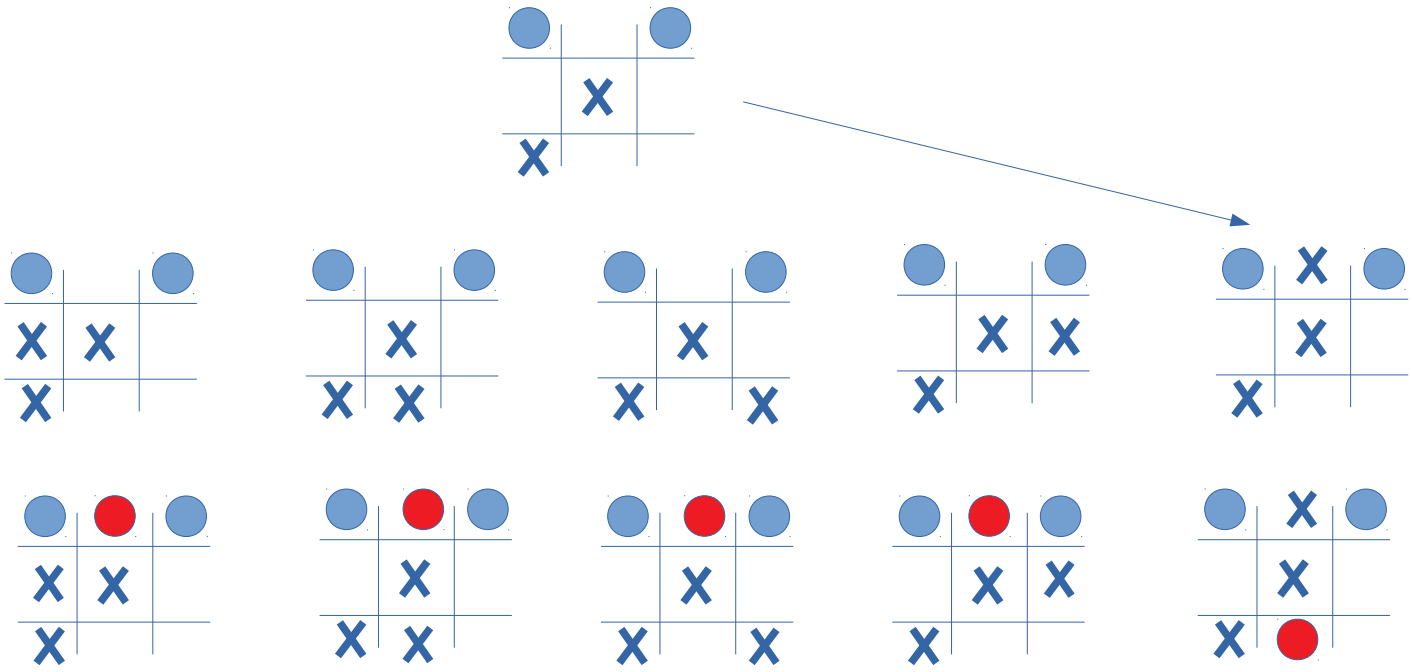
No momento em que um jogador (digamos, J1) planeja a sua jogada, ele gera e avalia, de acordo com a função heurística, dois níveis à frente:

1 - O da suas possíveis jogadas futuras e, para cada um desses estados no 1º nível futuro,
2 - O das possíveis respostas do adversário (J2) (já num 2º nível futuro).

Para avaliar a jogada (no 2º nível futuro) que o adversário irá escolher, J1 pressupõe que J2 irá escolher a pior opção para ele (J1), isto é, o estado associado ao menor valor da função heurística (que é sempre calculada do ponto de vista de J1).

O menor valor heurístico calculado no 2º nível é o escolhido como o valor heurístico associado ao respectivo estado que o gerou. Neste 2º nível o algoritmo minimiza a função heurística.
Comparando os valores da função heurística para os vários estados gerados no 1º nível futuro, o algoritmo escolhe, para J1, a jogada que maximiza a heurística.

Este processo de minimização e maximização da heurística, respectivamente para estados em que J2 e J1 jogam, pode continuar a ser desenvolvido para vários pares de níveis de profundidade, se assim desejado.



Abordagens primitivas em IA baseavam-se em busca, apoiada em poder computacional bruto, o que é o suficiente em casos simples. Porém, em situações mais complexas, as possibilidades crescem exponencialmente e enfrenta-se o problema da explosão combinatória das possibilidades a considerar na busca por uma solução. A saída então é fazer a busca ser guiada por conhecimento, para isso, é necessário heurísticas de busca e de bases de conhecimento.

Algoritmo A*

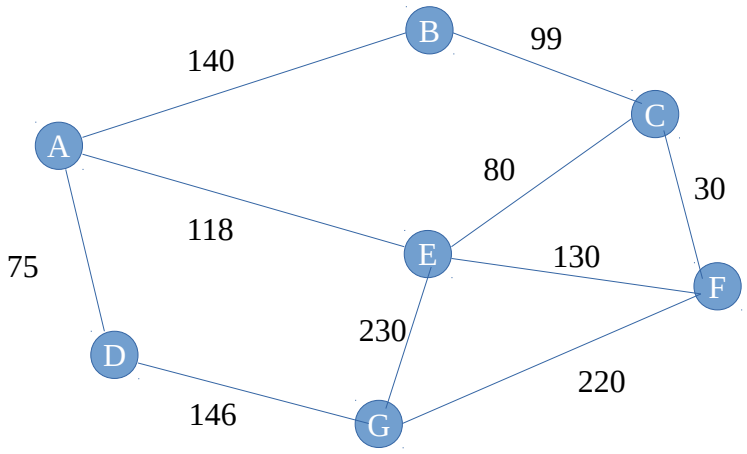
O algoritmo A* tem por objetivo melhorar o desempenho da busca por meio de algum conhecimento a ser utilizado juntamente da heurística. Nesse caso, o algoritmo utiliza mais de uma informação para decidir sobre qual ação tomar.

Por exemplo, a heurística do vizinho mais próximo utilizada no problema do caixeiro viajante pode ser melhorada utilizando o A*. Nesse caso, vamos considerar os valores calculados pela heurística do vizinho mais próximo como uma função g(n) e algum conhecimento a mais sobre o problema como uma função h(n). O algoritmo A* calcula uma função f(n) = g(n) + h(n). A busca então é guiada pela função de avaliação f.

Esse algoritmo é muito utilizado para encontrar rotas mais curtas em problemas que exigem isso, tais como movimentação em jogos digitais, GPS, etc. Geralmente a função g(n) calcula o valor da distância entre os vizinhos e a função h(n) calcula o valor total conhecido de cada vizinho até o destino, ao somar os dois valores, o vizinho que possui o menor valor é o escolhido e esse loop segue até que um estado objetivo seja encontrado. Vale ressaltar que esse algoritmo pode ser alterado para armazenar algum conhecimento específico sobre o problema e, com isso, tomar decisões mais satisfatórias.

Como exemplo, vamos fazer uma pequena alteração no algoritmo para calcular a rota mais curta entre duas cidades, nesse caso utilizaremos o conhecimento de qual a distância em linha reta de cada cidade até o destino. Com isso será possível entender o funcionamento do algoritmo, além de compreender o fato de que podemos alterar a função h para armazenar algum conhecimento específico e melhorar o algoritmo.

Observe o seguinte grafo com seus nós e arestas:



Supondo que o nó de origem seja A e o destino seja F, vamos adicionar as distância em linha reta conhecida entre todos os nós até o nó de destino F.

Distância em linha reta conhecida entre cada nó e o destino F	
A	250
B	110
C	20
D	330
E	120
F	0
G	200

Considerando a heurística do vizinho mais próximo, a rota escolhida para encontrar esse destino seria a seguinte: ADGF, com um custo total de 441.

Vamos utilizar o algoritmo A* adaptado com a função h(n) representando os valores das distâncias em linha reta até F. Nesse caso o algoritmo fará o cálculo da distância entre vizinhos somada à distância em linha reta de cada vizinho até F e escolherá o caminho de menor valor.

Na primeira iteração, começando da origem **A**, os vizinhos B, D e E terão seus custos em linha reta somados aos custos até A:

f(n) para B será 110 + 140 = 250
f(n) para D será 330 + 75 = 405
f(n) para E será 120 + 118 = 238

Desses 3, o **E** é o vizinho que possui menor valor, logo o algoritmo escolhe esse nó para prosseguir e o processo se repete a partir dele, sendo que seus vizinhos são C, F e G (Uma vez que o A já foi utilizado):

f(n) para C será 20 + 80 = 100
f(n) para F será 0 + 130 = 130
f(n) para G será 200 + 230 = 430

Nesse caso, o vizinho **C** é o escolhido por ter o menor valor e o processo se repete para C e seus vizinhos (que será nesse caso apenas o **F**, uma vez que os outros vizinhos já foram selecionados ou descartados):

f(n) para F será 0 + 30 = 30

Com isso a rota foi encontrada e é AECF. O custo total dessa rota (calculando os valores das arestas) é 118 + 80 + 30 = 228.

Fica claro que nesse exemplo o algoritmo A* (adaptado ao nosso contexto) teve um desempenho relativamente superior ao vizinho mais próximo básico, que traçou uma rota de valor total 441.

Obviamente, que esse algoritmo pode ter um desempenho ruim dependendo do contexto, mas o mais importante é que é possível adicionar conhecimento ao algoritmo de acordo com a necessidade visando sempre um melhor desempenho.

Referências bibliográficas:

RUSSELL, Stuart J.; NORVIG, Peter. Inteligência artificial. Elsevier, 2004.

RUSSELL, Stuart; NORVIG, Peter. Inteligência artificial. Rio de Janeiro: GEN LTC, 2013. ISBN 9788595156104.