Instituto Tecnológico de Aeronáutica

CT-213: Inteligência Artificial Para Robótica Móvel

Lab 02 - Busca Informada

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

Os três algorítmos de busca testados nesses experimentos são Dijkstra, *Greedy Search*, e A^*. Eles usam a chamada função de custo (Equação 1) para tentar encontrar um caminho ótimo desde o início até o objetivo, onde n é um nó qualquer, $g(n)$ é o custo do caminho até esse nó, e $h(n)$ é uma função heurística que estima o custo desde esse nó até o objetivo. Cada nó armazena em si seus valores de f e g.

$$f(n) = g(n) + h(n) \tag{1}$$

Os algorítmos em estudo têm em comum uma estrutura geral com três componentes básicos. O primeiro deles é a configuração de uma situação inicial, na qual todos os nós têm seus valores de f e g igualados a infinito. Nesse estágio, também são definidos os valores de f e g para o nó inicial e uma fila de prioridade para armazenar os próximos nós a serem visitados.

Depois dessa situação inicial, os algorítmos iniciam um laço que é reiterado enquanto o nó objetivo não for visitado e enquanto ainda houverem novos nós a serem visitados, ou seja, enquando a fila de prioridade não estiver vazia. Dentro desse laço, o primeiro elemento da fila é retirado, e é usado como o nó que será visitado na presente iteração.

Ainda nesse primeiro laço, inicia-se um novo laço que analisa cada sucessor do nó atualmente sendo visitado. Nesse estágio, a função de custo é calculada e usada para selecionar os sucessores que são candidatos a fazerem parte do caminho final segundo a lógica de cada algorítmo. Esses candidatos são adicionados à fila de prioridade, e armazenam o nó sendo visitado como o parâmetro *parent*, que será posteriormente usado para reconstruir o caminho desde o nó objetivo até o nó de partida.

Apesar da semelhança na estrutura geral dos três algorítmos, eles seguem princípios distintos para efetuar buscas. Na seção seguinte, serão brevemente explicitadas as nuances de cada um.

2 Algorítmos de busca

2.1 Algorítmo de Dijkstra

O algorítmo de Dijkstra não utiliza nenhuma heurística para estimar o custo até o seu destino, buscando minimizar o custo para chegar até cada nó que é dado pela função denominada $g(n)$. Dessa forma, a sua função de custo é dada por $f(n) = g(n)$, então é necessário usar somente o parâmetro f no código.

Na implementação do algorítmo, antes de entrar nos laços, o valor de f do nó de partida é definido como sendo zero. Ademais, quando os sucessores de um nó estão sendo analisados, os sucessores selecionados para entrarem na fila de prioridade são aqueles que não foram visitados ainda, e cujos valores atuais de f são maiores do que a soma do custo para chegar no nó atual com o custo para ir do nó atual até o sucessor em questão. Quando essas condições forem satisfeitas, o valor de f desse sucessor receberá o valor dessa soma, pois significa que um caminho menor até ele foi encontrado.

Após verificar todos os sucessores do nó, ele será marcado para não ser visitado novamente, uma vez que o menor caminho até chegar nele já foi determinado. Um exemplo de caminho traçado pelo algorítmo de Dijkstra encontra-se na Figura 1.

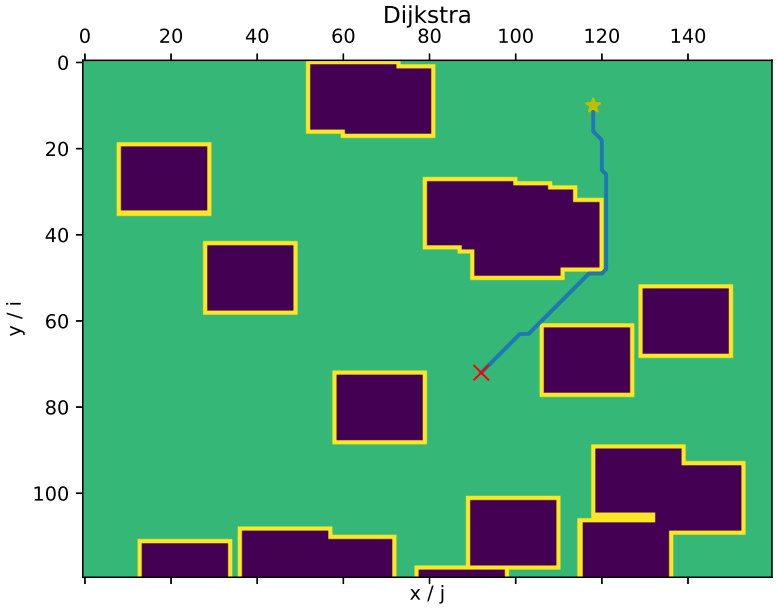

Figura 1: Caminho calculado pelo Algorítmo de Dijkstra.

2.2 Busca Gulosa (*Greedy Search*)

O algorítmo de Busca gulosa é essencialmente o oposto de Dijkstra: ignora-se o custo até chegar até um determinado nó, e leva em conta somente a estimativa do custo até chegar ao destino, ou seja, a função heurística $h(n)$, de tal forma que a função custo é dada por $f(n) = h(n)$. Nessa implementação, a heurística usada foi, simplesmente, a distância euclidiana do nó atual até o objetivo.

Nesse algorítmo, por mais que a função custo só leve em conta $h(n)$, os valores de g também foram armazenados para calcular o custo do caminho encontrado. Dessa forma, inicialmente define-se os valores de f e g do nó de partida como sendo, respectivamente, a distância dele até o nó objetivo, e zero. Dessa vez, todos os sucessores que ainda não foram descobertos são considerados como candidatos a constituirem o caminho final. Quando eles são analisados, define-se os seus valores de f e g como sendo, respectivamente, a distância deles até o nó objetivo, e a soma do valor de g do nó que está sendo visitado com o custo para ir desse nó até o sucessor em questão. Além disso, nesse caso os nós são marcados para não serem mais analisados assim que eles são descobertos como sucessores de outro nó, o que pode ser feito, uma vez que a função custo não muda para cada nó.

Ao término desse algorítmo, um caminho até o objetivo estará definido, como mostra um exemplo na Figura 2.

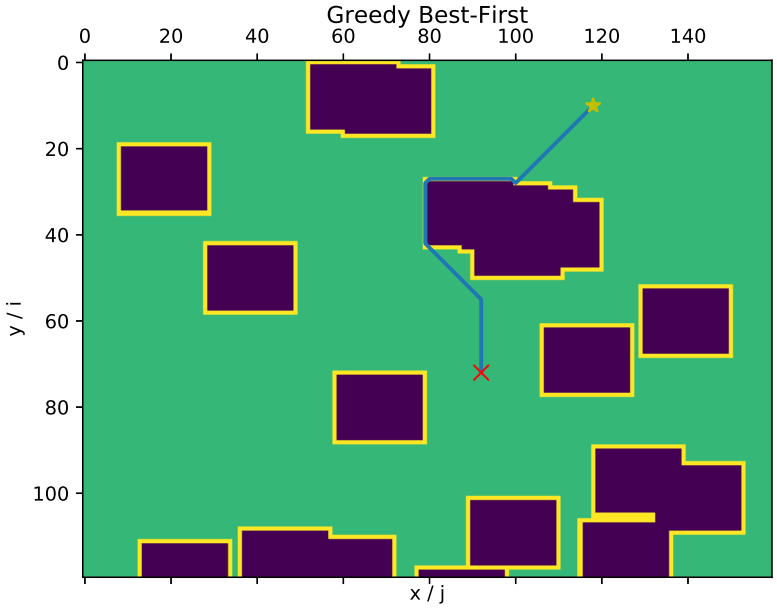

Figura 2: Caminho calculado pelo Algorítmo *Greedy Search*.

2.3 Algorítmo A*

Por fim, o algorítmo A^* utiliza o custo do caminho até cada nó ($g(n)$) para achar o caminho de menor custo, mas usa também a estimativa do custo de cada nó até alcançar o objetivo ($h(n)$) para otimizar o tempo de execução. Dessa forma, para esse algorítmo, a função custo é dada por $f(n) = g(n) + h(n)$. Novamente, a função heurística usada foi a distância euclidiana até o nó objetivo.

Como no algorítmo *Greedy Search*, o nó de partida recebe os valores de f e g como sendo, respectivamente, a distância até o nó objetivo, e zero. Quando itera-se os sucessores de um dado nó, os que são escolhidos como candidatos a fazerem parte do caminho final são aqueles cujo valor atual de f é maior do que a soma do caminho percorrido até alcançar o nó atual, do custo para ir desse nó até esse sucessor, e da distância desse sucessor até o nó objetivo. Esses candidatos têm seu valor de f igualado a essa soma, e seu valor de g atualizado para o novo possível menor caminho até ele que foi encontrado. Após iterar todos os sucessores, o nó é marcado para não ser visitado novamente.

Ao término desse algorítmo, o caminho encontrado poderá ser reconstruído através dos parâmetros *parent* configurados. Um exemplo de caminho encontrado pelo A^* encontra-se na Figura 3.

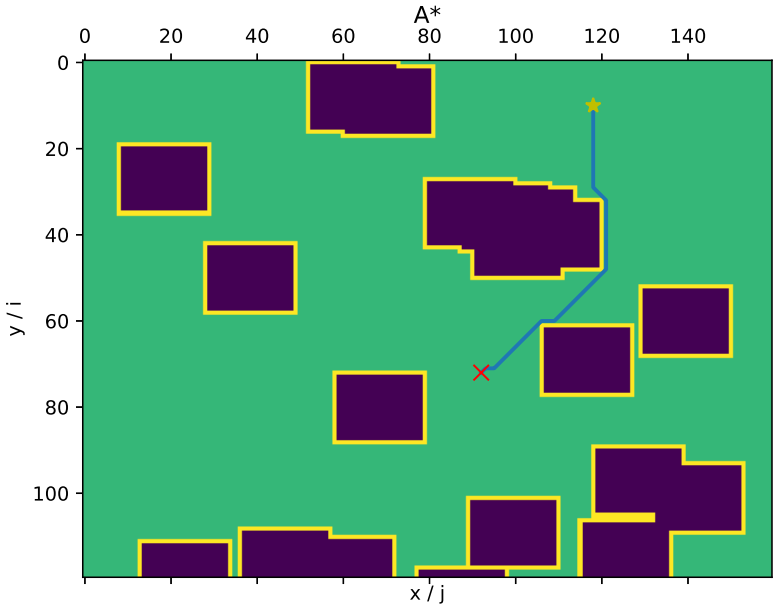

Figura 3: Caminho calculado pelo Algorítmo A*.

3 Método de *Monte Carlo*

Para comparar o desempenho de cada algorítmo nos quesitos custo computacional e eficácia na minimização do custo de caminho, foi feita uma análise de cada um deles usando o método de *Monte Carlo*. Eles foram usados para calcular uma série de 100 caminhos distintos. Dos resultados desses cálculos foram obtidos a média e o desvio padrão do tempo de execução e do custo de caminho, os quais se encontram na Tabela 1.

Algorítmo	Tempo Computacional (s)		Custo do Caminho	
	Média	Desvio padrão	Média	Desvio padrão
Dijkstra	0,15422	0,08451	79,829	38,571
Greedy Search	0,00640	0,00101	103,34	59,41
A*	0,04804	0,04588	79,829	38,571

Tabela 1: Resultados do Método de *Monte Carlo* para os algorítmos estudados.

Com os resultados obtidos, é possível perceber o que já era esperado pela descrição dos algorítmos usados. O algorítmo de Dijkstra é capaz de encontrar caminhos com custo médio baixo, porém, para fazer isso ele visita muitos nós, então acaba tendo um custo computacional alto. Por outro lado, o *Greedy Search* é muito mais rápido do que Dijkstra, porém ele encontra caminhos piores. O A^*, finalmente, combina as vantagens dos outros dois algorítmos para encontrar caminhos que, no caso desse experimento, são tão bons quanto os encontrados pelo Dijkstra, mas com um custo computacional significativamente menor, uma vez que ele se vale da heurística para reduzir a quantidade de nós que precisam ser visitados.