Relatório de CT-213

Laboratório 1 – Máquina de Estados Finita e Behavior Tree

Autor:
Otávio Henrique Ribas Guimarães

Turma 23

Professor
Marcos Ricardo Omena de Albuquerque Maximo

Data de submissão do relatório: 30/03/2020

Instituto Tecnológico de Aeronáutica - ITA
Departamento de Computação

Sumário

1 Introdução

O planejamento de caminho configura a determinação de um algoritmo para fazer com que, saindo de um estado inicial, uma IA chegue a um estado objetivo. Esse algoritmo pode ser feito de varias formas diferentes, podendo privilegiar custo do caminho ou tempo computacional.
Tomamos uma malha de várias posições que podem ser tomadas por um robô, chamadas de nós. Nessa malha, são marcadas uma posição inicial e um objetivo, bem como são criados obstáculos diversos. Para determinar-se um caminho que ligue essas suas posições, implementou-se três algoritmos distintos: Dijkstra, Busca Gulosa (Greedy Best-First) e A*.
Para as implementações utilizadas, fez-se uso de uma fila de prioridade, de forma a melhor selecionar os nós a serem testados. A fila de prioridade visa reduzir a complexidade do algoritmo, fazendo com que ele opere mais rapidamente.
A Busca Gulosa e A*, ao contrário do algoritmo de Dijkstra, fazem ainda uso de uma função de avaliação heurística. Enquanto Dijkstra testa nós puramente baseado na distância entre o nó e a origem, os outros dois algoritmos, chamados por isso de Busca Informada, utilizam-se de uma função para estimar o quão provável é que um nó faça parte do caminho ótimo. No modelo programado, a função de avaliação heurística utilizada é a distância euclidiana, isso é, a distância entre o estado testado e o objetivo. A escolha decorre da malha ser considerada 8-conectada, isso é, do fato de o robô pode se movimentar na diagonal. A distancia de um nó a um adjacente na horizontal ou na vertical é convencinada como 1, enquanto na diagonal é $\sqrt{2}$.

$$h(n, g) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \qquad \text{(Eq. 1)}$$

Nota-se que, sendo a distância euclidiana, a heurística utilizada atende a desigualdade triangular. A isso, dizemos que ela é uma heurística consistente.

2 Implementação dos Algoritmos

Descreve-se, a seguir, a implementação de cada algoritmo. Nota-se que cada um deles utilizou uma implementação de fila com heap sort como fila de prioridades. Além disso, a Busca Gulosa e a A* fizeram uso também da função heurística (1).

2.1 Algoritmo de Dijkstra

Para a implementação do algoritmo de Dijkstra, foi criada uma matriz de mesmo tamanho da malha de nós estudada, que foi utilizada para salvar os custos de cada nó. Os custos são todos inicializados como sendo iguais a infinito, enquanto o custo da posição inicial é inicializado como zero. O nó inicial, ainda, é colocado na fila de prioridade.
A cada nó retirado da fila, fecha-se esse nó e analiza-se os demais nós que são adjacentes a ele. Um nó fechado não volta a ser colocado na fila de prioridades. Ao se analisar os vizinhos, compara-se se o valor do custo na matriz de custos para aquele vizinho é maior do que o custo para o nó recém retirado da fila somado à distancia entre o nó e esse

vizinho. Se for o caso, e esse vizinho não for fechado, substitui-se o custo desse vizinho na matriz de custos por esse novo valor, e atribui-se a esse vizinho o nó original como sendo seu pai. Enfim, esse vizinho é adicionado à fila de prioridades.
O procedimento ocorre até que seja retirado da fila de prioridades o nó objetivo. Nesse momento, esvazia-se a fila de prioridades. O custo do algoritmo é simplesmente o custo do nó objetivo na matriz de custos.

2.2 Busca Gulosa

A implementação do algoritmo da Busca Gulosa (Greedy Best-First) faz uso da função heurística (1) para determinar a posição de um nó na fila de prioridades. A prioridade de um nó é definida pela distância entre ele e o nó objetivo.
Inicialmente, se adiciona à lista de prioridade o nó inicial. Quando um nó é retirado da lista, fecha-se ele e avalia-se seus vizinhos. Para cada vizinho, atribui-se a ele o nó retirado da lista como pai. O nó vizinho é fechado e adicionado à lista de prioridades. Esse processo é repetido até que se retire da lista de prioridades o nó objetivo. Nesse momento, esvazia-se a lista de prioridades.
Determina-se o custo, enfim, a partir do caminho criado após a execução do algoritmo. Calculamos a distância entre cada dois nós adjacentes pertencentes ao caminho e somamos esses valores, determinando, assim, o custo total do caminho.

2.3 A*

O algoritmo A* faz, assim como a Busca Gulosa, uso da Heurística (1) para abstrair o quão promissor é um nó. Todavia, a posição de um nó é determinada mutuamente pelo valor da função heurística e pela distância entre o nó e o nó inicial. Denotamos de g a distância entre um nó e a origem, semelhante ao que é calculado no algorítimo de Dijkstra, e de f a função dada por:

$$f(n) = g(n) + h(n) \qquad \text{(Eq. 2)}$$

São criadas duas matrizes auxiliares de custos, uma matriz para os valores de g e uma para os valores de f. Ambas são inicializadas com o valor de infinito em todas as células, exceto para o nó inicial, inicializado com zero para g e com o valor da heurística para f. À fila de prioridade, é adicionado o nó inicial.
A fila de prioridades tem sua ordem definida pelo valor de f dos nós. Sempre que um nó é retirado da fila de prioridade, fecha-se esse nó e analiza-se seus vizinhos. Para cada vizinho aberto, checa-se se o valor de seu f é superior ao valor de g do nó retirado da fila, somado ao custo da distância entre esses dois nós e ao valor da heurística para o vizinho. Se o f salvo for superior, então substituímos seu valor por esse novo valor calculado e substituímos o g do vizinho pelo g do nó retirado da fila somado à distância entre esses dois nós. Enfim, atribuimos à esse vizinho o nó retirado da fila como pai, e adicionamos esse viznho à fila de prioridades.
O processo é repetido até que seja retirado da fila de prioridades o nó objetivo. Determina-se o custo, enfim, a partir do caminho criado após a execução do algoritmo. Calculamos a distância entre cada dois nós adjacentes pertencentes ao caminho e somamos esses valores,

determinando, assim, o custo total do caminho. A distância também pode ser obtida pelo valor de g para o nó objetivo, uma vez que sabe-se que o caminho obtido por A* também é ótimo, pelo fato de a heurística (1) ser consistente.

3 Resultados e Discussão

Para cada algoritmo, executou-se o programa com diferentes números de iterações. Em cada caso, registrou-se os custos médios e os tempos computacionais. Vale ressaltar que o programa foi executado em um computador com um processador *Intel Core i5-5200U*.

3.1 Algoritmo de Dijkstra

Pela implementação do algoritmo de Dijkstra e executando o programa para diferentes números de iterações, obteve-se os seguintes resultados de custo médio e de tempo computacional.

Número de Iterações	Custo médio	Tempo Computacional (s)
1	78.1837	0.2111
10	80.5881	0.2630
100	79.8291	0.2851

Tabela 1: Resultados obtidos na execução do algoritmo de Dijkstra

Tem-se, graficamente, a representação dos caminhos criados pelo algoritmo de Dijkstra.

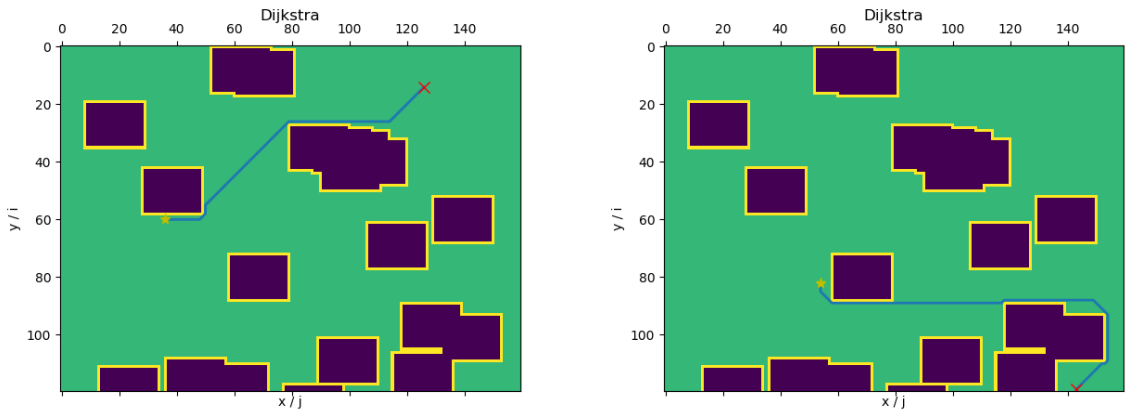

Figura 1: Caminhos criados pelo algoritmo de Dijkstra nos casos 9 e 55

3.2 Busca Gulosa

Pela implementação da Busca Gulosa (Greedy Best-First) e executando o programa para diferentes números de iterações, obteve-se os seguintes resultados de custo médio e de tempo computacional.

Número de Iterações	Custo médio	Tempo Computacional (s)
1	135.3259	0.0105
10	106.9219	0.0105
100	103.3419	0.0103

Tabela 2: Resultados obtidos na execução da Busca Gulosa

Tem-se, graficamente, a representação dos caminhos criados pela Busca Gulosa.

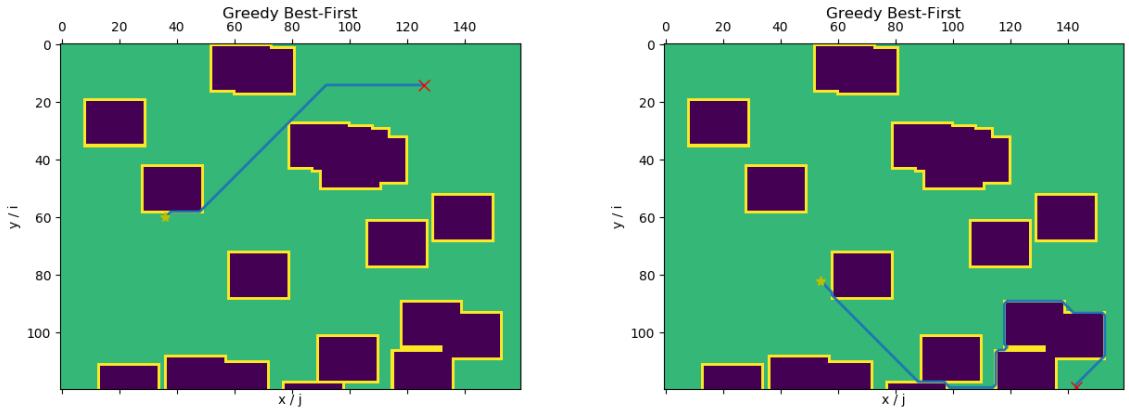

Figura 2: Caminhos criados pela Busca Gulosa nos casos 9 e 55

3.3 A*

Pela implementação do algoritmo de A* e executando o programa para diferentes números de iterações, obteve-se os seguintes resultados de custo médio e de tempo computacional.

Número de Iterações	Custo médio	Tempo Computacional (s)
1	78.1837	0.0700
10	80.5881	0.0623
100	79.8291	0.0604

Tabela 3: Resultados obtidos na execução da A*

Tem-se, graficamente, a representação dos caminhos criados pelo algoritmo de A*.

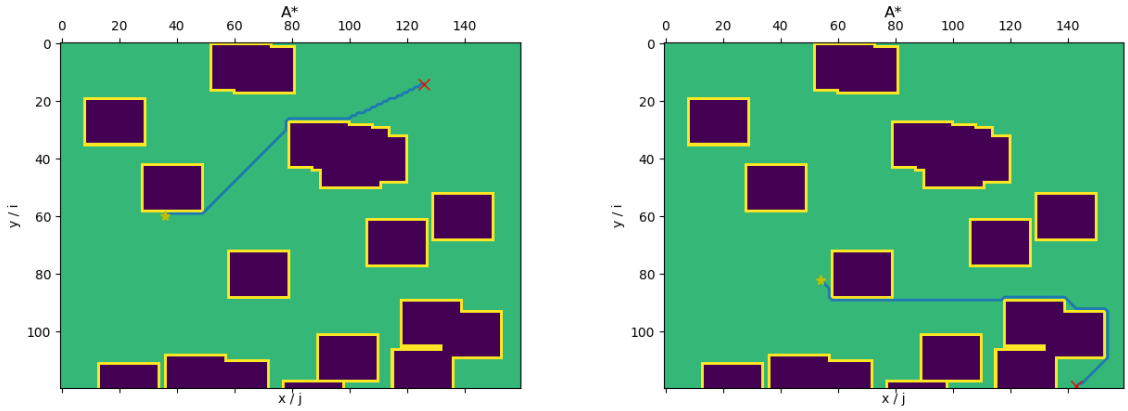

Figura 3: Caminhos criados pelo algoritmo de A* nos casos 9 e 55

3.4 Comparação dos resultados

Tomamos para cada algoritmo o valor do caminho médio e do tempo computacional obtidos pelo Método de Monte Carlo (100 iterações). A partir desses valores, podemos comparar os resultados obtidos para a implementação de cada um dos algoritmos.

Algoritmo	Custo médio	Tempo Computacional (s)
Dijkstra	79.8291	0.2851
Busca Gulosa	103.3419	0.0103
A*	79.8291	0.0604

Tabela 4: Resultados obtidos para cada algorítmo no Método de Monte Carlo

Dos resultados obtidos, percebemos que o algoritmo de Dijkstra, apesar de fornecer o caminho mínimo, dentre os algoritmos utilizados, apresenta tempo computacional com ordem de grandeza 10 vezes superior aos outros dois algoritmos. De forma oposta, a busca gulosa (Greed Best-First) apresenta o menor tempo computacional, mas obtém um caminho subótimo, apresentando um custo médio consideravelmente maior. O algotimo de A*, enfim, obtem o mesmo custo médio do algoritmo de Dijkstra, uma vez que a heurística utilizada (1) é consistente. Apesar disso, apresenta tempo computacional de ordem próxima à Busca Gulosa, uma vez que configura uma busca informada.