LAB2 - Busca Informada

CT-213 Aprendizado de Máquina para Robótica Móvel
Prof. Marcos Maximo

Rafael Mello Celente

March 29, 2022

1 Implementação

1.1 Algoritmo de Dijkstra

A lógica do algoritmo de Dijkstra é setar o valor de custo de um nó (o custo entre um nó inicial e o nó em questão) como infinito e tentar melhorar esse valor a cada etapa da busca.

A implementação do algoritmo foi feita através de um *Priority Queue (PQ)*, de forma que fosse dado prioridade pro nó de menor custo. O algoritmo começa por setando o custo do nó inicial como 0 e adicionando esse nó à PQ. Quando chegamos ao nosso objetivo com o menor custo possível significa que a PQ está vazia, uma vez que não existe nenhum nó a ser analisado que tenha um custo menor. Logo, o algoritmo roda enquanto a PQ ainda tiver pelo menos um elemento.

A cada iteração, o elemento de menor custo é retirado da PQ, marcado como visto e são analisados os seus sucessores. Se algum dos seus sucessores não tiver sido visitado ou o custo atual do sucessor for menor que o custo do nó sendo visitado agora mais o custo da aresta entre eles, o custo do sucessor é atualizado. Ademais, o nó em questão se torna o pai desse sucessor e o sucessor é adicionado à fila de prioridade PQ.

Quando a fila de prioridade esvaziar, podemos então reconstruir o caminho de menor custo a partir dos antecessores do nó objetivo.

1.2 *Algoritmo Greedy Search*

A lógica do algoritmo de *Greedy Search* é ignorar o custo dos caminhos até agora e apenas olhar para um custo simplificado. No caso de um mapa 2D como aqui implementado, o *Greedy Search* se importa apenas com a distância euclidiana do nó que o o algoritmo olha naquele momento até o nó objetivo.

A implementação do algoritmo, assim como no Dijkstra, foi feita através de um *Priority Queue (PQ)*, de forma que fosse dado prioridade pro nó de menor custo. O algoritmo começa setando o custo do nó inicial como a distância euclidiana do nó inicial até o nó objetivo e adicionando-o à fila de prioridade.

A cada iteração, o elemento de menor custo é retirado da PQ, marcado como visto e são analisados os seus sucessores. Se algum dos seus sucessores não tiver sido visitado, o nó pai daquele sucessor é então atualizado e ele é adicionado à fila de prioridade com um custo que é a distância do nó em questão até o nó objetivo. Assim, a PQ é responsável por selecionar sempre o nó com menor distância possível até o objetivo. Quando a fila de prioridade esvaziar, podemos então reconstruir o caminho a partir dos antecessores do nó objetivo.

É importante ressaltar que esse algoritmo não necessariamente irá encontrar o caminho de menor custo, uma vez que o algoritmo utiliza a distância euclidiana, ignorando obstáculos ou buracos. Entretanto, essa solução tende a ser a mais rápida, uma vez que visita menos nós.

1.3 Algoritmo A*

A lógica do algoritmo A* é ignorar bastante parecido com o algoritmo de Dijkstra. A diferença entre os dois está em uma heurística que é utilizada no A* que serve para "indicar" ao algoritmo uma direção mais suscetível a encontrar a solução ótima. O algoritmo então trabalhará com dois valores: g que é o custo real do nó até agora e; f que é a projeção de custo final do objetivo considerando a heurística h, de forma que $f = g + h$ para um nó intermediário. O objetivo do A*, portanto, é minimizar o valor de f, de forma "empurrar" o algoritmo para uma solução. Ambos algoritmos encontram a solução ótima, mas o A*, considerando uma heurística suficientemente boa, chega ao resultado mais rapidamente.

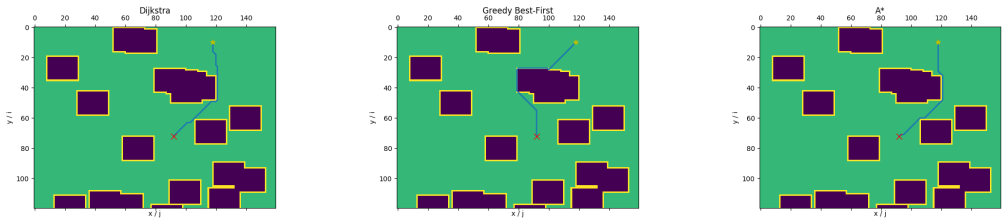
No exemplo aqui realizado de um mapa 2D com um movimento 8-conectado, a heurística mais propícia é a de distância euclidiana.

A implementação do algoritmo, assim como no de Dijkstra, foi feita através de um *Priority Queue (PQ)*, de forma que fosse dado prioridade pro nó de menor custo. O algoritmo começa por setando o custo g do nó inicial como 0 e o custo f como a distância euclidiana do nó inicial ao nó final. O nó então é adicionado à PQ. Quando chegamos ao nosso objetivo com o menor custo possível significa que a PQ está vazia, uma vez que não existe nenhum nó a ser analisado que tenha um custo menor. Logo, o algoritmo roda enquanto a PQ ainda tiver pelo menos um elemento.

A cada iteração, o elemento de menor custo é retirado da PQ, marcado como visto e são analisados os seus sucessores. Se um sucessor não tiver ainda sido visitado e seu custo f for maior que o custo g do nó visitado mais o custo entre eles e mais o custo h do sucessor ao nó objetivo, os valores de f e g do sucessor são atualizados e este é adicionado à fila de prioridade.

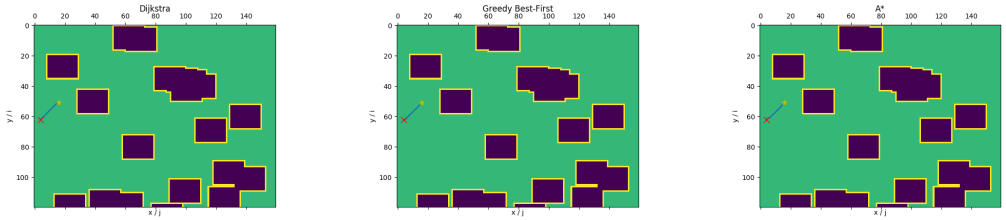
Quando a fila de prioridade esvaziar, podemos então reconstruir o caminho de menor custo a partir dos antecessores do nó objetivo.

2 Figuras de comparação entre algoritmos



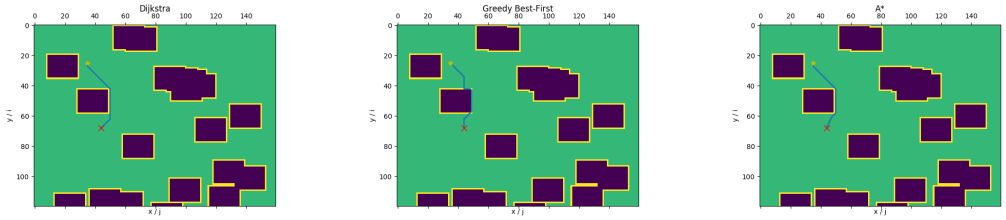
(a) (b) (c)

Figure 1: Simulação 1 entre os 3 algoritmos de busca. Em (a) Dijkstra, (b) *Greedy Search* e (c) A*.



(a) (b) (c)

Figure 2: Simulação 2 entre os 3 algoritmos de busca. Em (a) Dijkstra, (b) *Greedy Search* e (c) A*.



(a) (b) (c)

Figure 3: Simulação 3 entre os 3 algoritmos de busca. Em (a) Dijkstra, (b) *Greedy Search* e (c) A*.

3 Comparação entre algoritmos

Algoritmo	Tempo computacional		Custo do caminho	
	Média	Desvio padrão	Média	Desvio Padrão
Dijkstra	0,179541	0,096838	79,82920	38,57096
Greedy Search	0,010132	0,003953	105,0559	63,61328
A*	0,040073	0,030786	79,82920	38,57096

Table 1: Tabela de comparação entre os algoritmos de planejamento de caminho.