Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT-213
Aluno:

Relatório do Laboratório 2 - Busca Informada

1. Breve Explicação em Alto Nível da Implementação

1.1. Algoritmo Dijkstra

O algoritmo de dijkstra é um algoritmo de busca que consegue achar o menor caminho num grafo. Nele, primeiramente, cria-se um fila de prioridade implementada por heap, a qual armazena os vértices do grafo bem como seus custos, para que possamos passar de forma ordenada por menor custo de aresta por todos os vértices e atualizar sempre os custo. Os custos de cada vértice são inicializados como infinito, tendo apenas o nó inicial adicionado na fila custo nulo.Assim, enquanto a fila não estiver vazia retiram-se elementos da fila até que se ache um nó não visitado. Após esse elemento não visitado ser achado, o marcamos como closed. Caso esse nó já seja o nó alvo, já retorna-se o caminho até esse nó, bem como o custo desse menor caminho. Caso não, analisa-se os sucessores do nó e caso o sucessor tenha custo maior que o custo do nó atual+ custo da aresta, atualiza-se o custo para o custo do nó atual + custo da aresta, além de declarar o nó atual como parent ou pai do sucessor, para depois do processo se possa determinar o caminho mínimo percorrido pela função construct_path.

1.2. Algoritmo *Greedy Search*

A greedy search, também chamada de busca gulosa, é um algoritmo de busca baseado somente na heurística do problema, que nesse caso é a distância euclidiana, possuindo alta velocidade para achar um caminho, que não é ótimo, até o alvo. No nosso código, porém, foi necessário calcular o custo de seu caminho para mostrar que tal custo é maior. Primeiramente, cria-se uma priority queue com as operações feitas por heap, iniciando o custo inicial como a própria heurística do ponto até o objetivo e adicionando o nó inicial na fila de prioridade. Assim, enquanto a fila não estiver vazia retiram-se elementos da fila até que se ache um nó não visitado.Após esse elemento não visitado ser achado, o marcamos como closed, para que no próximo ciclo não seja visitado novamente. Assim, para cada sucessor do nó não visitado, verifica-se se o sucessor já é o vértice alvo, e, caso seja, retorna-se o caminho executado bem como seu custo, que deve ser calculado assim como no algoritmo de dijkstra. Também declara-se o nó atual como pai do sucessor. Caso o sucessor não seja o objetivo, atualiza-se a heurística do sucessor e adiciona-o na heap, a fim de que posteriormente sejam analisados também seus sucessores.

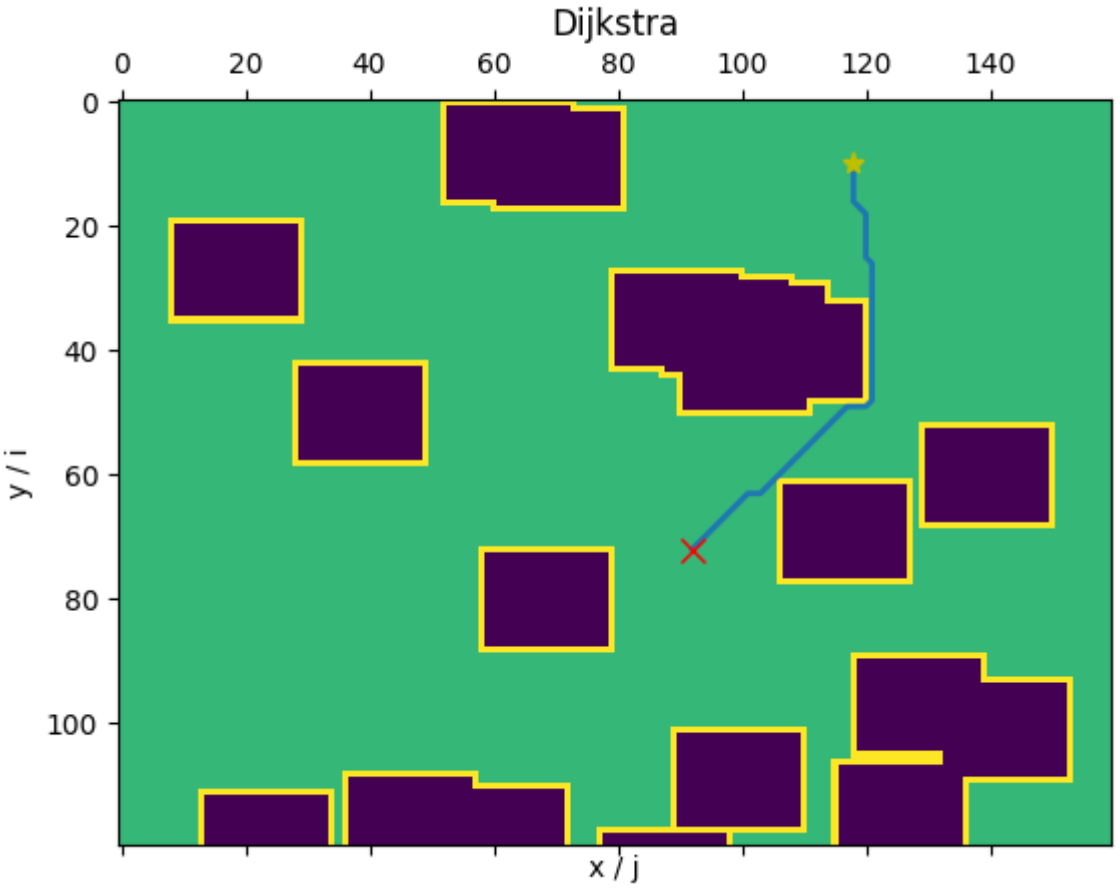
1.3. Algoritmo A*

O algoritmo A* é um algoritmo de Dijkstra melhorado por uma heurística, que é uma informação do domínio do problema que é capaz de estimar o melhor caminho e diminuir o tempo para achar esse caminho. Nesse caso, como o tabuleiro da busca era 8-conectado, a heurística utilizada foi a distância euclidiana do vértice atual até o vértice alvo. Assim, primeiramente cria-se uma fila de prioridade cujas operações são feitas por uma heap na qual os custos e os vértices do grafo serão armazenados, iniciando o g, isto é, o custo devido somente ao algoritmo de Dijkstra, que gera o caminho mínimo, como 0 e o f que é custo da soma entre g e h, sendo h a heurística, como o valor de h até do vértice de início ao vértice alvo. Insere-se nessa fila de prioridade o vértice inicial e seu custo f. Dessa

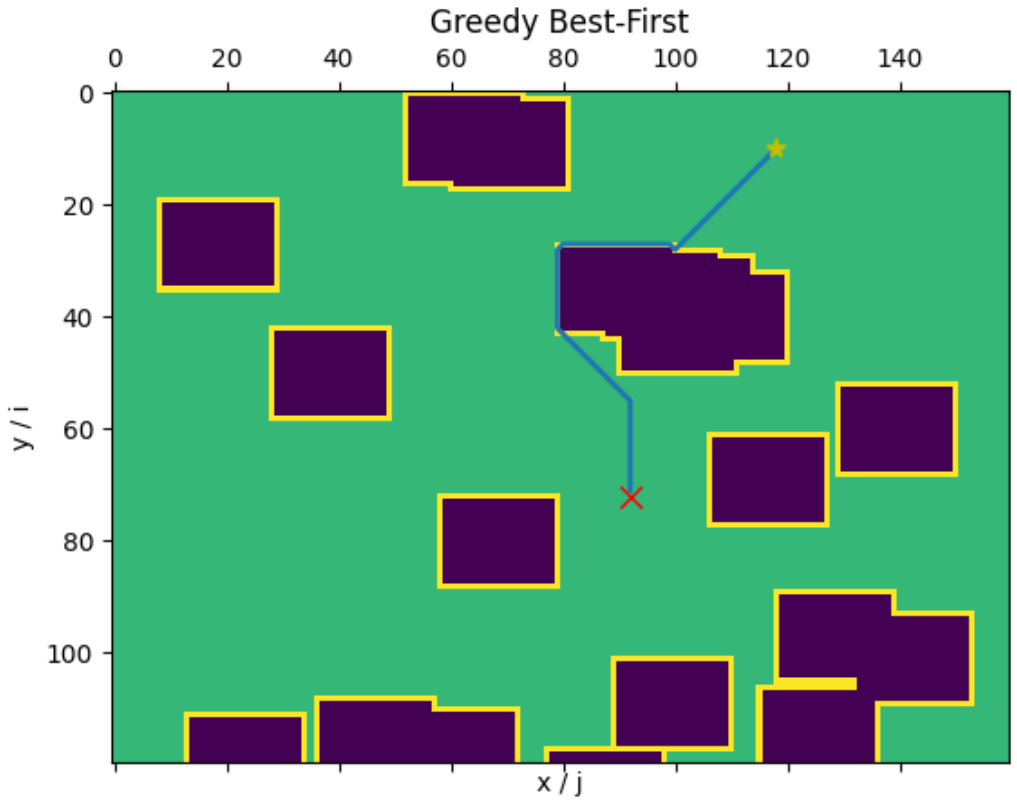
forma, enquanto a fila não estiver vazia, faz-se um longo procedimento. Primeiro retira-se um elemento da fila, assim como seu custo. Se esse nó já tiver sido visitado, ou seja, for closed, retiram-se mais elementos da fila até que apareça um nó não visitado. Quando este nó aparecer, já o marcamos como visitado para que no próximo loop do while já seja considerado visitado. Caso o nó já seja o nó alvo, retorna-se o caminho utilizado e o custo. Caso contrário, analisa-se os sucessores não visitados do nó, pois os sucessores visitados já têm seu custo determinado. Assim, se o custo total f do sucessor for maior que o custo do nó+custo da aresta entre o nó e o sucessor + custo da heurística do sucessor até o alvo, atualizam-se os custos do sucessor, sendo o custo g do sucessor o custo g do nó + o custo da aresta e os custo f do sucessor sendo o custo g + a heurística do sucessor. No fim coloca-se o parent do sucessor como o próprio nó para no fim gerar o caminho e adiciona-se o sucessor à fila de prioridade.

2. Figuras Comprovando Funcionamento do Código

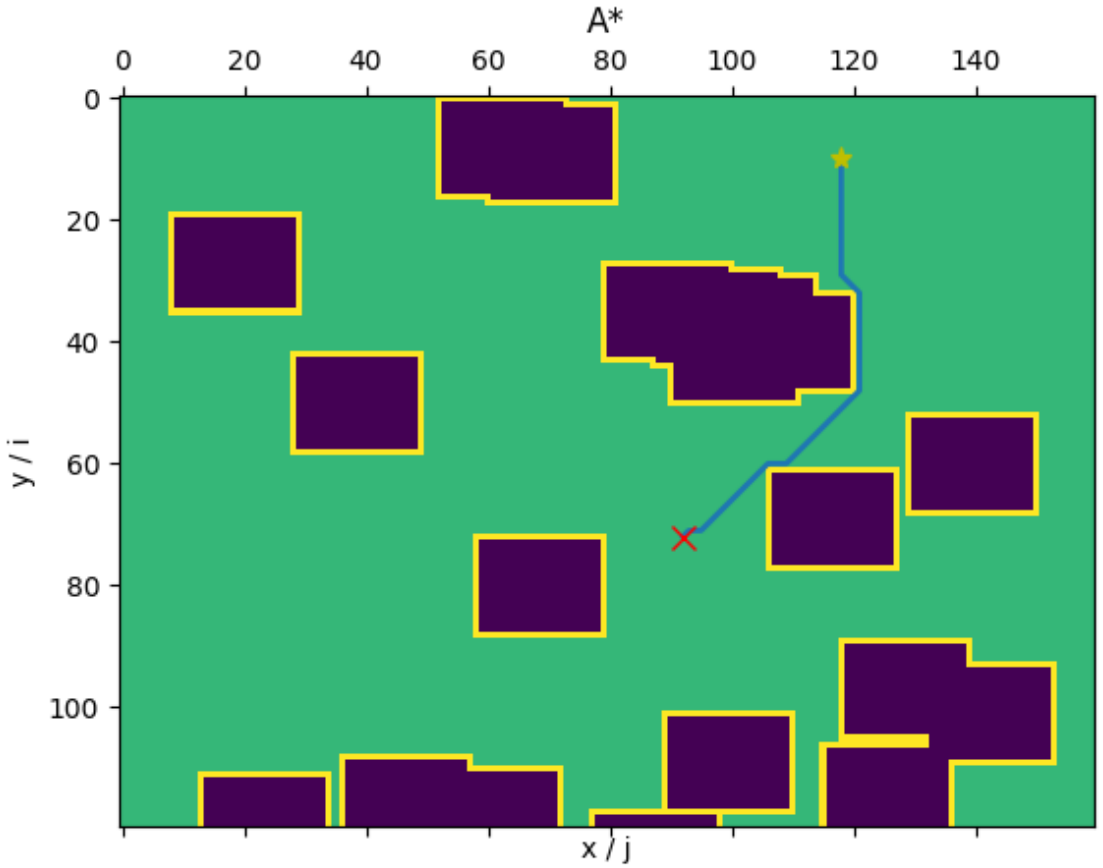
2.1. Algoritmo Dijkstra



2.2. Algoritmo *Greedy Search*



2.3. Algoritmo A*



3. Comparação entre os Algoritmos

Basta preencher a tabela.

Tabela 1 com a comparação do tempo computacional, em segundos, e do custo do caminho entre os algoritmos usando um Monte Carlo com 100 iterações.

Algoritmo	**Tempo computacional (s)**		**Custo do caminho**	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Dijkstra	0.438	0.246	79.82	38.57
Greedy Search	0.016	0.006	105.05	63.61
A*	0.084	0.069	79.82	38.57

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