Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT213
Aluno: Tafnes Silva Barbosa

Relatório do Laboratório 2 - Busca Informada

1 Breve Explicação em Alto Nível da Implementação

Neste laboratório, foram implementados 3 algoritmos de planejamento de caminho: Dijkstra, *Greedy Search* e A*.

1.1 Algoritmo Dijkstra

O Dijkstra encontra o caminho ótimo, mas é muito lento por verificar mais nós. Sua implementação foi feita tomando como base o pseudocódigo dos *slides* da aula. O algoritmo segue os seguintes passos:

1. Reseta o *grid* e recebe o nó objetivo através do método `get_node(i, j)`;
2. Cria a fila de prioridades, recebe o nó inicial, seta o custo deste nó para 0 (`node.f = 0`) e o coloca na fila de prioridades (Usa-se o `node.f`, pois este algoritmo não usa heurística, logo f=g);
3. Enquanto a fila de prioridades não está vazia:
   (a) Extrai o nó de maior prioridade da fila;
   (b) Se o nó não está fechado: (faz-se isso para evitar voltar a um nó da fila que já foi retirado, dado que a estrutura em python não atualiza um valor dentro da fila de prioridades)
      i. Fecha o nó retirado da fila;
      ii. Se o nó é o nó objetivo, retorna o caminho até ele (através do método `construct_path(goal_node)`) e o custo para chegar ao nó objetivo; (Como o nó foi retirado da fila, seu custo é o menor possível para este algoritmo)
      iii. Para cada sucessor do nó retirado da fila, caso o custo do sucessor seja maior que o custo do nó retirado mais o custo de chegar a esse nó (através do método `get_edge_cost((i, j), successor_position)`, atualiza seu custo, atualiza seu pai e coloca o sucessor na fila de prioridades.

1.2 Algoritmo *Greedy Search*

O *Greddy Search* pode encontrar um caminho subótimo, mas é muito mais rápido. Sua implementação foi feita tomando como base o pseudocódigo dos *slides* da aula. O algoritmo segue os seguintes passos:

1. Reseta o *grid* e recebe o nó objetivo através do método `get_node(i, j)`;

2. Cria a fila de prioridades, recebe o nó inicial, seta o custo ótimo deste nó até o objetivo igual à distância euclidiana entre estes nós (`node.f = initial_node.distance_to(goal_position)`), seta o custo para chegar a este nó para 0 (`node.g = 0`) e o coloca na fila de prioridades (Usa-se o `node.f` igual à heurística, pois este algoritmo usa somente a heurística, logo f=h; contabiliza-se o `node.g` para evitar de voltar a um nó que já teve custo menor definido, dado que trabalha-se com grafos e não árvores);

3. Enquanto a fila de prioridades não está vazia:

   (a) Extrai o nó de maior prioridade da fila;

   (b) Se o nó não está fechado: (faz-se isso para evitar voltar a um nó da fila que já foi retirado, dado que a estrutura em python não atualiza um valor dentro da fila de prioridades)

      i. Fecha o nó retirado da fila;

      ii. Para cada sucessor do nó retirado da fila, caso o custo para chegar até o sucessor (`successor.g`) seja maior que o valor de atualização: (faz-se isso para evitar voltar a um nó que já teve custo `g` menor definido, dado que não se trabalha com árvores e sim com grafos)

         A. Atualiza seu pai e atualiza o custo para chegar até o nó sucessor (`node.g`);

         B. Se ele é o nó objetivo, retorna o caminho do nó inicial até o objetivo e o custo para chegar até este sucessor (`successor.g`);

         C. Atualiza o custo `successor.f` deste sucessor para o valor da heurística utilizada (distância euclidiana entre este nó sucessor e o nó objetivo, dado que este algoritmo só utiliza a heurística);

         D. Insere o sucessor na fila de prioridades.

1.3 Algoritmo A*

O A* encontra o caminho ótimo em um grafo dado que a heurística utilizada (distância euclidiana) é consistente, ou seja, respeita a "desigualdade triangular". Ele é mais rápido que o Dijkstra e mais lento que o *Greedy Search*. Sua implementação foi feita tomando como base o pseudocódigo dos *slides* da aula. O algoritmo segue os seguintes passos:

1. Reseta o *grid* e recebe o nó objetivo através do método `get_node(i, j)`;

2. Cria a fila de prioridades, recebe o nó inicial, seta o custo para chegar a este nó para 0 (`node.g = 0`), seta o custo (`node.f`) deste nó para a distância euclidiana entre este nó e o nó objetivo e o coloca na fila de prioridades (Este algoritmo usa tanto a eurística como o custo de chegar ao respectivo nó, $\texttt{f} = \texttt{h} + \texttt{g}$);

3. Enquanto a fila de prioridades não está vazia:

   (a) Extrai o nó de maior prioridade da fila;

(b) Se o nó não está fechado: (faz-se isso para evitar voltar a um nó da fila que já foi retirado, dado que a estrutura em python não atualiza um valor dentro da fila de prioridades)

i. Fecha o nó retirado da fila;

ii. Se o nó é o nó objetivo, retorna o caminho até ele (através do método `construct_path(goal_node)`) e o custo para chegar ao nó objetivo (`node.g`); (Como o nó foi retirado da fila, seu custo é o menor possível para este algoritmo)

iii. Para cada sucessor do nó retirado da fila, caso o custo do sucessor (`successor.f`) seja maior que o custo de chegar ao nó retirado (`node.g`) mais o custo de chegar a esse nó (através do método `get_edge_cost((i, j), successor_position)` mais a heurística para chegar ao objetivo a partir do sucessor: atualiza o custo para chegar a ele (`successor.g`), atualiza seu custo (`successor.f`), atualiza seu pai e coloca o sucessor na fila de prioridades.

2 Figuras Comprovando Funcionamento do Código

Nesta seção são mostradas figuras comprovando o funcionamento do código implementado. Em cada algoritmo há 3 figuras mostrando o funcionamento para mais de um par (início, objetivo).

2.1 Algoritmo Dijkstra

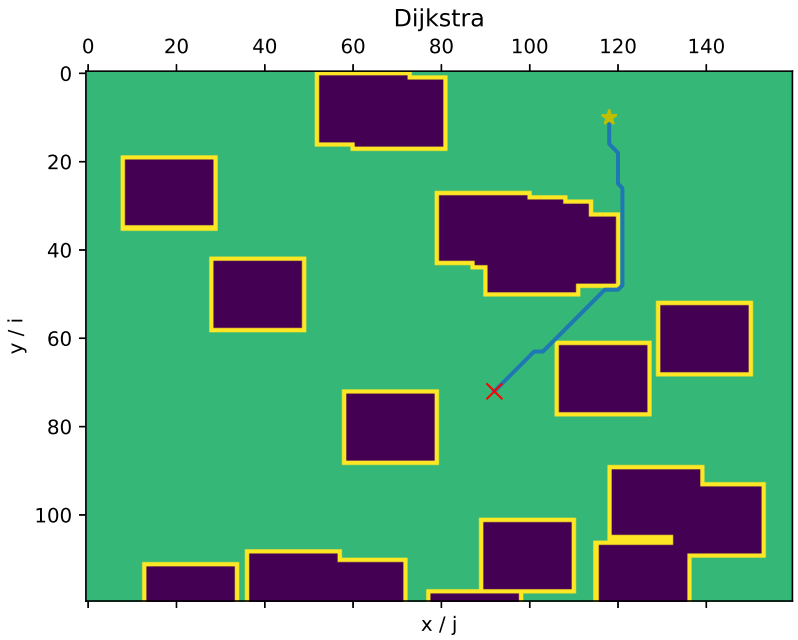

Figura 1: Primeiro caminho com Dijkstra, gerado com *seed*=15.

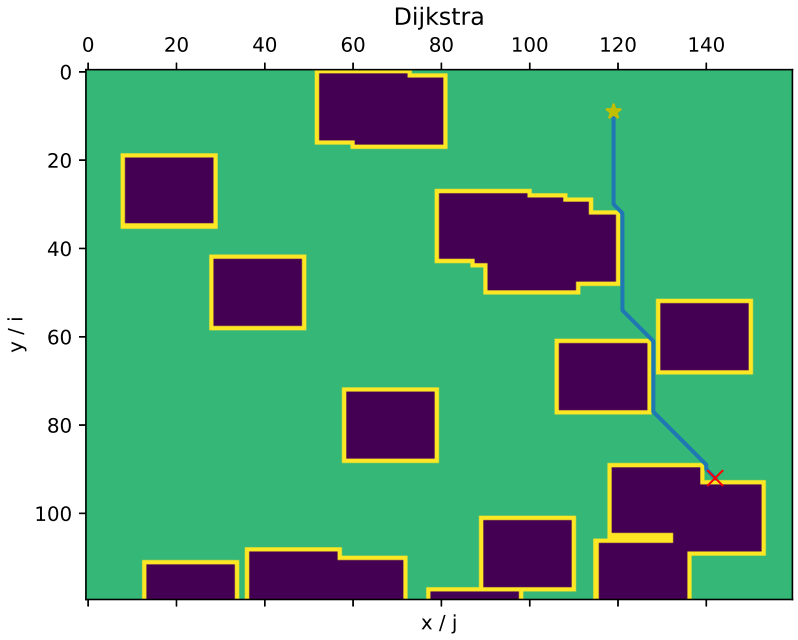

Figura 2: Quinto caminho com Dijkstra, gerado com *seed*=15.

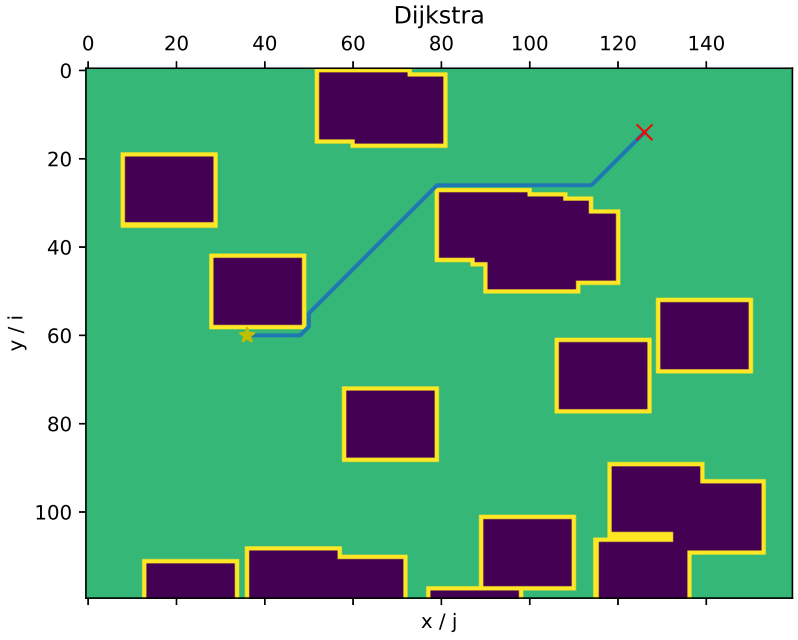

Figura 3: Décimo caminho com Dijkstra, gerado com *seed*=15.

2.2 Algoritmo *Greedy Search*

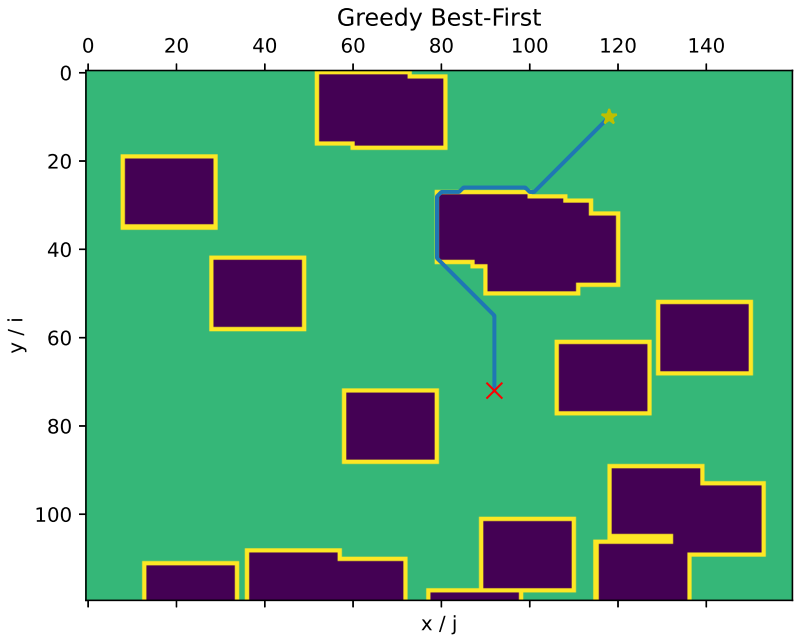

Figura 4: Primeiro caminho com *Greedy Search*, gerado com *seed*=15.

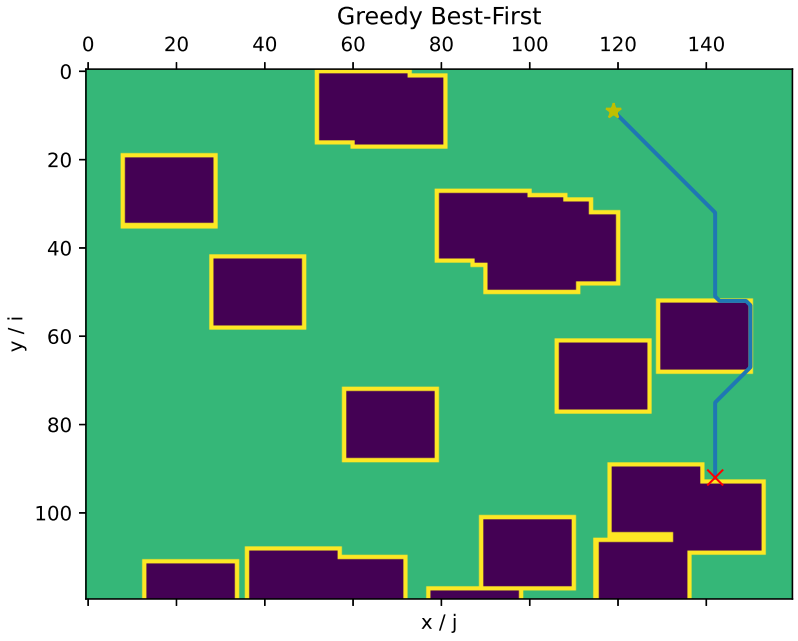

Figura 5: Quinto caminho com *Greedy Search*, gerado com *seed*=15.

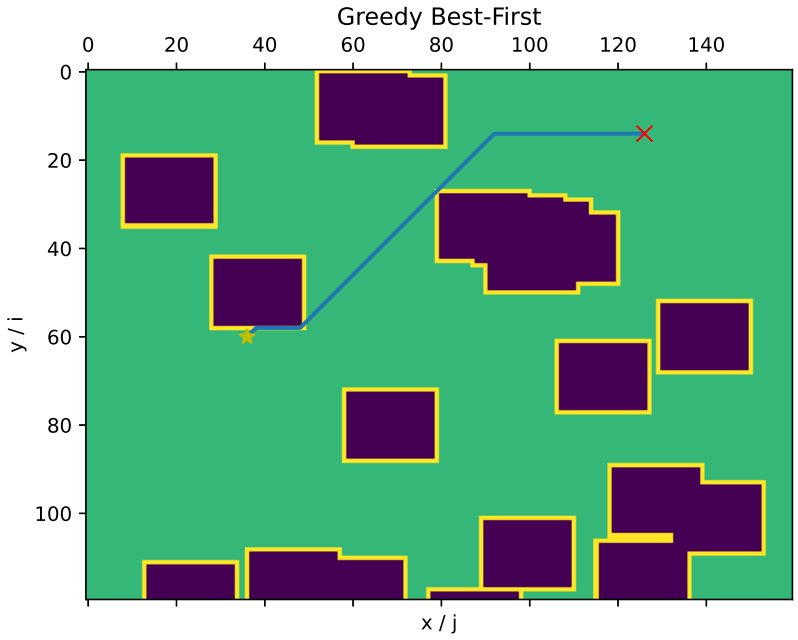

Figura 6: Décimo caminho com *Greedy Search*, gerado com *seed*=15.

2.3 Algoritmo A*

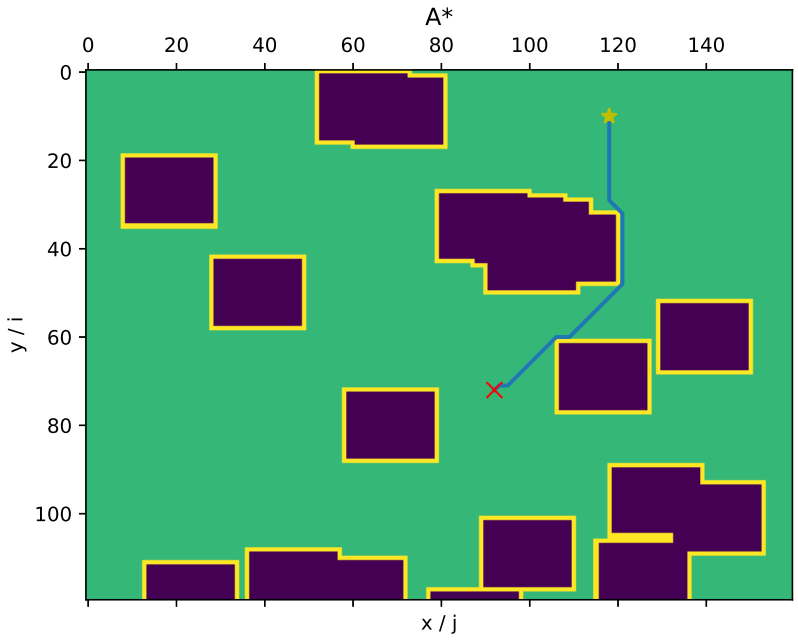

Figura 7: Primeiro caminho com A*, gerado com *seed*=15.

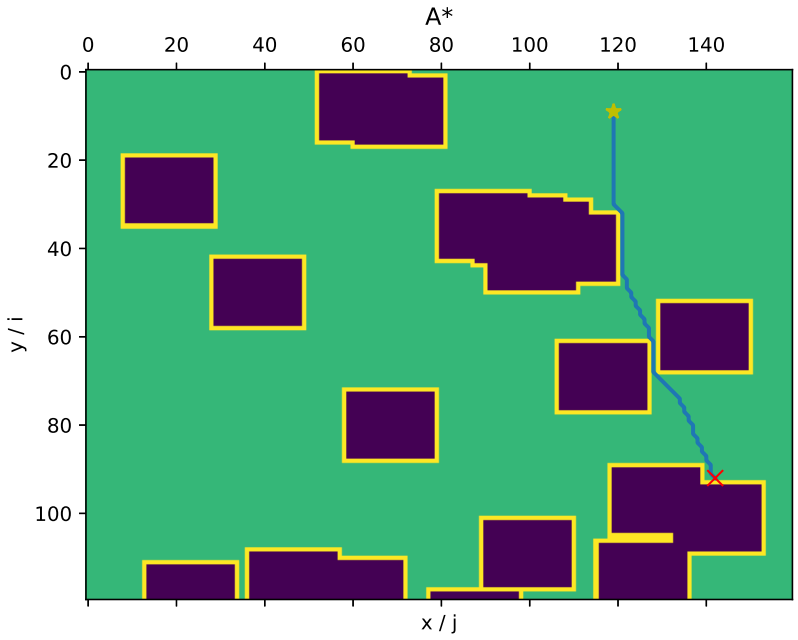

Figura 8: Quinto caminho com A*, gerado com *seed*=15.

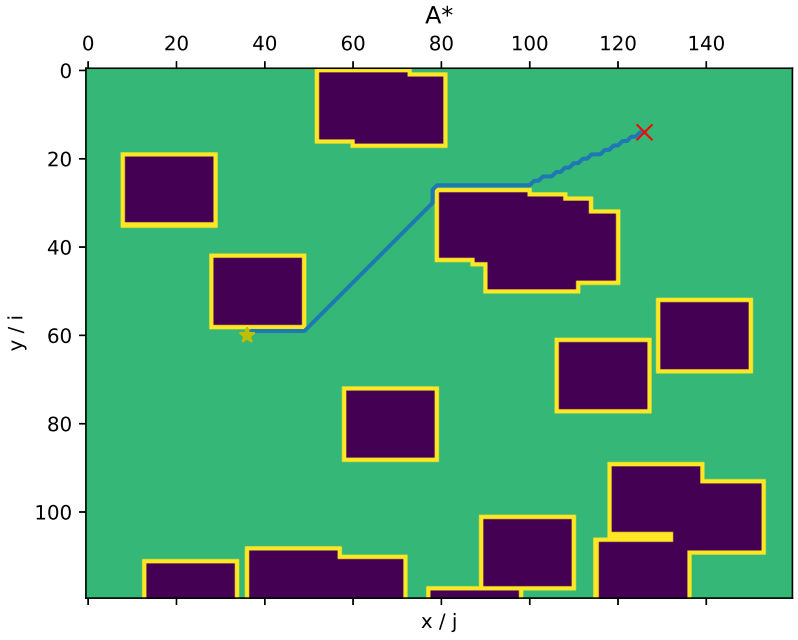

Figura 9: Décimo caminho com A*, gerado com *seed*=15.

3 Comparação entre os Algoritmos

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

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

Algoritmo	**Tempo computacional (s)**		**Custo do caminho**	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Dijkstra	0.1428	0.0807	79.8292	38.5710
Greedy Search	0.0063	0.0064	103.1175	58.7940
A*	0.0372	0.0337	79.8292	38.5710