Custo Uniforme

Eric Segala Jovelli e Marco Cezar da Silva

Representação (estrutura de dados) do estado

Custo uniforme

Para criar o algoritmo de custo uniforme foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:

- O estado do jogo
- O nodo anterior
- A distância da origem

A-star

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:

- O estado do jogo
- O nodo anterior
- A distância da origem
- O g (custo da origem até o nodo)
- O h (heurística)

A-star Manhattan

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:
O estado do jogo
O nodo anterior
A distância da origem
O g (custo da origem até o nodo)
O h (heurística)

Estrutura de dados para a fronteira e nodos fechados

Custo uniforme

A estrutura de dados para fronteira é a fila. Para armazenar os nodos fechados (os que já foram visitados) foi usado um array.

A-star

A estrutura de dados para fronteiras é uma fila. Para armazenar os nodos fechados (os que já foram visitados) foi usado um array.

A-star Manhattan

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:
O estado do jogo
O nodo anterior
A distância da origem
O g (custo da origem até o nodo)
O h (heurística)

Métodos ou funções principais e breve descrição do fluxo do algoritmo

Custo uniforme:

Classe nodo

Get_path: retorna o caminho até o nodo

Módulo

Check_game_over: verifica se o estado é o final
Print_stats: imprime as estatísticas da execução do algoritmo após encontrar a solução
Get_state_after_move_space_down: retorna um nodo após termos a troca do espaço com um nodo abaixo dele.
Get_state_after_move_space_up, get_state_after_move_space_right, get_state_after_move_space_left: semelhante a Get_state_after_move_space_down, mas relativo a troca do espaço com vizinhos de outras direções.
Get_derived_nodes: obtem os nodos derivados
Uniform_cost_search: algoritmo em si

Descrição do fluxo do algoritmo: é excutado o Uniform_cost_search, que itera sobre a fila de prioridade, removendo o primeiro da fila para que seja executado o Get_derived_nodes sobre ele. No Get_derived_nodes é executado Check_game_over e é verificada cada direção e é executado Get_state_after_move_space_down e/ou Get_state_after_move_space_up e/ou Get_state_after_move_space_right e/ou Get_state_after_move_space_left. Por fim é executado Print_stats quando o Check_game_over retorna true.

A-star

aStarSearch: o algoritmo em si.
isVisitedNode: checa se o nodo foi visitado
printResults: resultados da execução

getDerivedNodesFromAction: obter nodos derivados de um nodo
getActionsFromPuzzleNode: verifica possíveis jogadas
isFinished: checa se é game over
Na classe PuzzleNode:
getH: obtem a heurística
getF: obtém h + g
toString: retorna o estado
isEqual(nodo): verifica se o nodo é igual ao argumento
getPath: caminho ao estado inicial

A-star Manhattan

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:
O estado do jogo
O nodo anterior
A distância da origem
O g (custo da origem até o nodo)
O h (heurística)

Gerenciamento da fronteira, verificações, quais etapas foram feitas ao adicionar um estado na fronteira

Custo uniforme

A fronteira é gerenciada pela fila priority_queue. No início do game, o nodo inicial é adicionado a ela. Então, ocorre um loop while, ou seja, enquanto houver nodos na fronteira, esse loop é iterado. No loop. É removido o primeiro elemento da fila. É verificado se esse é o estado final. Se for, temos um game over. Caso contrário, tal nodo é adicionado à lista de nodos visitados. Então, obtemos todos os nodos derivados do nodo que foi removido da lista, a cada um desses nodos é atribuida a distância do nodo derivados acrescentada em um. Além disso, o nodo derivador é salvo em um atributo do nodo derivado, para reconstrução futura do caminho. Para cada nodo derivado, é checado se ele não está na lista de visitados. Se não estiver, ele é adicionado no fim da fronteira e a fronteira é então reordenada de acordo com a distância dos nodos do nodo inicial.

A-star

O funcionamento é semelhante ao custo uniforme, porém, há algumas diferenças. O ordenamento da fronteira passa a ser pelo valor de f de cada nodo. F é dado pelo atributo g (o custo até o nodo) + h (o valor da heurística)

A-star Manhattan

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:
O estado do jogo
O nodo anterior
A distância da origem
O g (custo da origem até o nodo)
O h (heurística)

Descrição das heurísticas e comparação da precisão delas

Custo uniforme

No caso do algoritmo de custo uniforme não houve uso de heurísticas.

A-star

O valor da heurística é dado pelo método getH. Ele funciona assim:

1. Inicia-se um valor h = 9
2. Para cada número na posição certa reduz 1
3. Se o espaço está na posição certa reduz 1
4. Retorna h

A-star Manhattan

Para criar o algoritmo a-star foi usado um grafo no qual cada possível estado de jogo é representado por um nodo.
Cada nodo tem como atributos:
O estado do jogo
O nodo anterior
A distância da origem
O g (custo da origem até o nodo)
O h (heurística)

Breve análise do desempenho com uma tabela comparativa

	Custo Uniforme	A-star	A-star Manhattan
Fácil	0.0	0.0	0.0
Médio	0.0012	0.00002	0.00001
Difícil	0.0022	0.00003	0.00001

Limitações da implementação

A execução seria mais eficaz caso cada inserção ocorresse diretamente na posição correta. No jeito implementado, ocorre a inserção no final da fronteira e então ele é reordenada com base na distância de cada elemento da origem