Disciplina de Inteligencia Artificial

Pesquisa no espaço de estados

Alunos:
Tiago Ávila - Nº 31565
Tiago Baião - Nº 18860
Engenharia Informática

Trabalho:
Resolução de problemas como problemas de pesquisa no espaço de estados

Docente:
Irene Pimenta Rodrigues
Engenharia Informática

19 de março de 2016

Introdução

Neste primeiro trabalho de inteligência artificial iremos colocar em prática os conhecimentos adquiridos nas aulas para a resolução de problemas como problemas de pesquisa no espaço de estados. O problema apresentado é o seguinte: Considere que um agente esta numa sala de uma caverna que tem nXn salas, como na figura abaixo, e que cada sala tem 4 portas, uma para cada sala vizinha. Suponha que o Agente esta na sala (2,2) e pretende ir para a sala (8,8) mas que as portas entre as salas (1,2) e (1,3),(2,3) e (2,2),(3,4) e (4,4),e (4,5) e (3,5) estão bloqueadas. Com este problema temos que resolver três exercícios. Como primeiro exercício é nos pedido para representar em Prolog o espaço de estados e os operadores de transição de estados para este problema, o código em Prolog do algoritmo de pesquisa não informada mais eficiente para resolver este problema e para justificar a escolha do algoritmo se por exemplo a caverna tivesse 30X30 salas e o problema fosse de ir da sala (18,18) para a sala (26,26). Depois de resolver este problema é pedido que digamos o numero total e exato de estados visitados e o numero máximo exato de estados que tem que estar simultaneamente em memoria. Como segundo exercício vamos propor duas heuristicas admissíveis para estimar o custo de um estado até à solução do problema. Iremos apresentar o código em Prolog do algoritmo de pesquisa informada mais eficiente para resolver este problema com as heuristicas definidas e vamos indicar também para cada função heuristica o numero total e exato de estados visitados e o máximo numero exato de estados que têm que estar simultaneamente em memoria. Como terceiro exercício é nos pedido para resolver o problema usando um algoritmo de local search. Fazer também uma representação para os estados deste problema, definir o operador vizinho que dado um estado gera todos os estados vizinhos, definir uma função de avaliação que permite ordenar os estados por grau de adequação e implementar o Hill Climbing e resolver o problema indicando o numero de estados expandido até encontrar a solução.

Primeiro Exercicio

Na primeira parte deste exercício é nos pedido para representar em Prolog o espaço de estados e os operadores de transição de estados para o problema, o código em Prolog do algoritmo de pesquisa não informada mais eficiente para resolver este problema e justificar a escolha do algoritmo se a caverna tivesse 30X30 salas e o problema fosse ir da sala (18,18) para a sala (26,26).
Testamos três algoritmos de pesquisa para a resolução deste exercício, um algoritmo de pesquisa em profundidade, um algoritmo de pesquisa em largura e um algoritmo de pesquisa iterativo. Depois de variados testes verificamos que o algoritmo com menor custo é o de largura, o que visita menos nos é o de profundidade e o que guarda o menor numero de nos em memoria é o iterativo. Comparamos de seguida os algoritmos usados. Em relação à comparação entre o algoritmo de largura e o de profundidade verificamos que o de profundidade visita menos nos mas em relação ao custo e à ocupação de nos em memoria o de largura é menos dispendioso, fazendo o de largura mais eficiente que o de profundidade. Em relação ao de profundidade e ao iterativo verificamos que que o iterativo custa menos que o de profundidade, ocupa menos nos em memoria, fazendo dele mais o mais eficiente entre os dois. Na comparação entre o algoritmo de largura e o iterativo verificamos que o de largura custa menos e visita menos nos apesar de ocupar mais nos em memoria, mas a diferença dos nos em memoria não é significativa logo entre o algoritmo de largura e o iterativo, o de largura é mais eficiente. Depois destas comparações chegamos à conclusão que o algoritmo de largura é o mais eficiente para o nosso problema. Em seguida segue o código em Prolog pedido na primeira parte do trabalho e segue também em anexo nos ficheiros ”operacoes.pl”, ”pni.pl”e ”pi.pl”.

```
:- dynamic(path/1).

% restrições do tabuleiro

restri((1,2),(1,3)).
restri((1,3),(1,2)).
restri((2,3),(2,2)).
restri((2,2),(2,3)).
restri((3,4),(4,4)).
restri((4,4),(3,4)).
restri((4,5),(3,5)).
restri((3,5),(4,5)).

estado_inicial((18,18)).
estado_final((26,26)).

% operações para mudança de local

% cima
op((X,Y),cima,(Z,Y),1):- X > 1, Z is X-1, (path((Z,Y))
-> fail
;assertz(path((Z,Y)))).
% baixo
op((X,Y),baixo,(Z,Y),1):- X < 30, Z is X+1, (path((Z,Y))
-> fail
;assertz(path((Z,Y)))).
% direita
op((X,Y),direita,(X,Z),1):- Y < 30, Z is Y+1, (path((X,Z))
-> fail
;assertz(path((X,Z)))).
% esquerda
op((X,Y),esquerda,(X,Z),1):- Y > 1, Z is Y-1, (path((X,Z))
-> fail
;assertz(path((X,Z)))).
```

Para a segunda parte deste exercício foi nos pedido o numero total de estados visitados e o numero máximo de estados que tem que estar simultaneamente em memoria.Para mostrar esses resultados incluímos uma tabela com todos os valores dos algoritmos usados. Aqui segue essa tabela.

Algoritmo	Custo	Nos Visitados	Nos em Memoria
Largura	16	500	52
Profundidade	116	116	141
Iterativo	20	5935	43

Em seguida seguem os comandos para executar as funções no terminal GNU Prolog.
?-[pni].
?-pesquisa(operacoes,largura).
?-pesquisa(operacoes,profundidade).
?-pesquisa(operacoes,it).

Segundo Exercicio

No segundo exercício vamos propor duas heuristicas admissíveis para estimar o custo de um estado até à solução do problema.Em seguida vamos apresentar o código em Prolog do algoritmo de pesquisa informada mais eficiente para resolver este problema com as heuristicas definidas e vamos indicar também para cada função heuristica o numero total e exato de estados visitados e o máximo numero exato de estados que têm que estar simultaneamente em memoria. // As duas heuristicas escolhidas são a distancia euclidiana e o declive da reta. Em matemática, distância euclidiana (ou distância métrica) é a distância entre dois pontos, que pode ser provada pela aplicação repetida do teorema de Pitágoras. Aplicando essa fórmula como distância, o espaço euclidiano torna-se um espaço métrico. O declive é utilizado para medir a inclinação da reta em relação ao eixo das abcissas (eixo do xx). Por exemplo, uma reta com declive 2, em linguagem corrente pode ser explicado da seguinte forma: por cada duas unidades que a reta ”sobe”na vertical, ela ”avança”uma unidade na horizontal. Em seguida apresentamos o codigo em Prolog com as heuristicas definidas e em anexo no ficheiro ”operacoes.pl”, ”pni.pl”e ”pi.pl”.

```
:- dynamic(path/1).
% restrições do tabuleiro

restri((1,2),(1,3)).
restri((1,3),(1,2)).
restri((2,3),(2,2)).
restri((2,2),(2,3)).
restri((3,4),(4,4)).
restri((4,4),(3,4)).
restri((4,5),(3,5)).
restri((3,5),(4,5)).

estado_inicial((18,18)).
estado_final((26,26)).

% operações para mudança de local
%% pni %%

% cima
op((X,Y),cima,(Z,Y),1):- X > 1, Z is X-1, (path((Z,Y))
-> fail
;assertz(path((Z,Y)))).
% baixo
op((X,Y),baixo,(Z,Y),1):- X < 30, Z is X+1, (path((Z,Y))
-> fail
;assertz(path((Z,Y)))).
% direita
op((X,Y),direita,(X,Z),1):- Y < 30, Z is Y+1, (path((X,Z))
-> fail
;assertz(path((X,Z)))).
% esquerda
op((X,Y),esquerda,(X,Z),1):- Y > 1, Z is Y-1, (path((X,Z))
-> fail
;assertz(path((X,Z)))).

% 1 ª heuristica

h((Xi,Yi),V):- estado_final((Xf,Yf)), V is (Xf - Xi) + (Yf - Yi).

% 2 º heuristica

h((Xi,Yi),V):- estado_final((Xf,Yf)), Xf \= Xi, V is (Yf - Yi)/ (Xf - Xi).
h((Xi,Yi),V):- estado_final((Xf,Yf)), Xf = Xi,V is 0.
```

Nesta tabela seguem o numero total e exato de estados visitados e o máximo numero exato de estados que têm que estar simultaneamente em memoria para cada função heuristica.

Heuristica	Custo	Expansão de Nos	Nos Visitados	Nos em Memoria
Euclidiana	16	128	129	39
Declive De Reta	16	478	479	94

Em seguida seguem os comandos para executar as funções no terminal GNU Prolog.

?-[pni].

?-pesquisa(operacoes,a).

Nota: Comentar no código a heuristica que não pretende utilizar.