Tec de Monterrey Campus Querétaro

Arturo Rivera Flores - A01202457
Cuauhtemoc Suarez - A01206503
Francisco Villanueva - A01202413

Artificial Intelligence
04/10/16

Lab 3: Implementing (Un)Informed Search Algorithms

Test your program with a couple of different problems. Increase the size of the problem to test the limits of your program.

Make a table comparing how **many nodes are searched** to find the answer for each problem. For this table, you should compare a number of different problems (at least 3) to avoid a statistical bias.

Your report needs to explain at least the following:

1. Which heuristics did you use for the A* algorithm?
2. Which of the four algorithms searches the least nodes and which one take the most?
3. Why does this happen?
4. Which algorithms are optimal? Why?
5. In your opinion what are the benefits of simpler algorithms versus more complex ones?

1.- The used heuristics were:

- Inconsistent.- For a heuristic to be consistent it needs to be admissible (not overestimating the cost) and to comply with the triangle inequality for the costs and the heuristic between two nodes… Thus, by making it inadmissible it also won't be consistent so we chose a constant value. Furthermore, this constant value must be higher than the minimum cost that a transition from one node to another has, this is 2 as the cost from taking the box and leaving it is 1 and the cost from one tile to another is 1 so the constant value must be higher than 2.
- Consistent.- Analyzing different ways for defining a heuristic for this one, we tried to find the best one (the one that has the minimum error between the actual cost and the heuristic). Thus, we "erased" the rule referring to the order of the Boxes in the stack so you can move one of the bottom and put it in the bottom of the other stack. Therefore, our heuristic sums the cost of moving the boxes to their corresponding tile ignoring the previous rule, as an example:

[B,A],[C],[] (initial state)

[B],[C],[A] (final state)

h = 3

As it can be seen, the heuristic is taking into account only the cost of moving A from tile 0 to tile 2 so the cost is 3 mins and is ignoring the fact that B is over A, the rest of the boxes are already in the correct position.

2.- For this part we tested the algorithm with 4 different problems listed on the tables below. The algorithm that expands less nodes is A* consistent algorithm, and the algorithm that expands more nodes is Depth First (in some cases is the Breadth First)..

[A],[B],[C] =>[A,B][C][]

Algorithm	**Nodes expanded**
Depth First	74
Breadth First	53
a* Inconsistent	45
a* Consistent	43

Cost

Algorithm	**Cost**
Depd111ee111th First	60
a* Inconsistent	22
Breadth First	10
a* Consistent	10

[A,C],[B],[] =>[C,B][A][]

Algorithm	**Nodes expanded**
Breadth first	100
A* Inconsistent	93
Depth First	76

A* consistent	48

Cost

Algorithm	**Cost**
Depth First	58
a* Inconsistent	18
Breadth First	13
a* Consistent	11

[B,A],[C],[D] =>[D][A][C,B]

Algorithm	**Nodes expanded**
Breadth first	954
Depth First	444
A* Inconsistent	374
A* consistent	306

Cost

Algorithm	**Cost**
Depth First	177
a* Inconsistent	39
Breadth First	21
a* Consistent	21

[B,A],[C,E],[D] =>[D,E][A][C,B]

Algorithm	**Nodes expanded**
Breadth first	7960

Depth First	1169
A* Inconsistent	4981
A* consistent	977

Cost

Algorithm	**Cost**
Depth First	904
a* Inconsistent	71
Breadth First	20
a* Consistent	20

Tables 1.1, 1.2, 1.3, 1.4 Different Tested Problems

3.- A* consistent expands less nodes as it is aware of the combined heuristic (cost + heuristic) at every time so it is always expanding the node with the lesser value assuring always finding the optimal path with the minimum required expansions (in general), it has to be noted that if there were many movements with equal combined heuristic value, all should be expanded (rare case). For the depth first, a tree search is used it can be forever expanding the nodes trying to reach the bottom, for a graph search it will expand always the "left-most" until finding the bottom or the answer and then moving to another one, if the answer is in the "right-most" node of the first expansion, the answer will be found until all of the other nodes are expanded; for the breadth-first the number of expansions is directly related to the level where the answer is found as it is expanding all the nodes per level.

Note: For the code the algorithms were developed using a graph search for assuring an answer, also the functions receive a value as a maximum depth for not saturating memory.

4.- Furthermore, A* always find the optimal solution as it knows the lesser cost and also the extra information that provides the heuristic. The priority queue helps to organize the nodes in the frontier for expanding them according to the combined cost (as it was mentioned earlier) so it is assuring that the answer that it finds requires the minimum cost. The other algorithms can also find an optimal solution but this depends on the situation (problem).

5.- Benefits of simpler algorithms:

1. It is easier to program than a complex algorithm. The Breadth First was developed quicker than A *.
2. With a correct implementation it assures finding an answer, if you don’t require the optimal one then it works.
3. Tools for implementing them are easier. For example, the priority queue is not a “normal” data structure so you need to implement it.

The biggest disadvantage is the memory and time needed for them to find a solution and also they don’t always find the optimal answer.