# IA: Assignement 2

Benoit Daccache Christopher Castermane

October 17, 2012

## 1 Statements to be proven

### 1.1 Prove that depth-first is a special case of uniform-cost search

Depth-first is a special case of uniform-cost search when for each node, there is only one successor.

### 1.2 Prove that breadth-first is a special case of uniform-cost seach

Breadth-first search is a special case of uniform-cost search when for each node, the path cost is identical node.

let x be the path cost at node n.

let y be the depth of the node.

let p be the path cost from node n to node n+1

BFS will choose the shallowest node. UCS will choose the node with the smallest path cost. Therefore for each node the path cost will be incremented by 1 and the depth by one. Then x == y and BFS is equivalent to UCS

### 1.3 Prove that greedy-best-first search is a special case of A*

When h(n) is equal to 0 then greedy-BFS is a special case of A*.

For A* , $f(n) = g(n) + h(n)$

for BFS, $f(n) = h(n)$

if$h(n) = 0$ then for each nod

### 1.4 Prove that uniform-cost is a special case of A ?

Uniform cost is a special case of A* when $h(n) = 0$ .

For UCS, $f(n) = h(n)$

For A* $f(n) = g(n) + h(n)$

if $h(n) = 0$ then $f(n) = 0$ for UCS. Therefore it will choose the node using the lexicographic order.

UCS can be a special case of A* when $h(n) = 0$ and $g(n)$ orders the node in a lexicographic order.

### 1.5 Prove that if a heuristic is consistent, then it is admissible.

- Admissible means that the heuristic never overestimates the cost to reach the goal.
- Consistency means that the estimated cost of a node n is always lower than the children's estimated cost of n with the step cost from n to the children. ($h(n) \leq c(n, a, child(n)) + h(child(n))$)

Therefore :

$h(n) \leq c(n, a, child(n)) + h(child(n))$
$\Leftrightarrow h(n) - h(child(n)) \leq c(n, a, child(n))$
$\Leftrightarrow h(n) \leq c(n)$
$\Leftrightarrow h(n)\, is\, admissible$

## 2 $A*$versus uniform-cost search

### 2.1 Give a consistent heuristic for this problem. Prove that it is admissible

Manhattan distance is a consistent heuristic. Since we can move one block at a time in 4 possible directions, the best scenario is when we can reach the goal with following a straight line path. Therefore if this path is obstructed, the real cost will be higher than the one calculated using the manhattan distance heuristic. Thus, the manhatan distance is an admissible heuristic.

### 2.2 Show on the left maze the states (board positions) that are visited during an execution of A* graph search with a manhattan distance heuristic (ignoring walls). A state is visited when it is selected in the fringe and expanded. In the A* algorithm, we assume that when different states in the fringe have the smallest value, the algorithm chooses the state with the smallest coordinate $(i; j)((0; 0)$ being the bottom left position, i being the horizontal index and j the vertical one) using a lexicographical order

Table 1:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| 1 | 3 | 5 | 7 | 9 | 11 | € 13 |
| I | 2 | 4 | 6 | 8 | 10 | 12 |

### 2.3 Show on the right maze the board positions visited by a uniform-cost graph search. When several states have the smallest path cost, this uniform-cost search visits them in the same lexicographical order as A*

Table 2:

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 17 | 23 | 32 | 41 | 50 | |
| 7 | 16 | 22 | 31 | 40 | 49 | |
| 6 | 15 | 21 | 30 | 39 | 48 | |
| 5 | 14 | 20 | 29 | 38 | 47 | |
| 4 | 13 | 19 | 28 | 37 | 46 | |
| 3 | 12 | \| | 27 | 36 | 45 | |
| 2 | 11 | \| | 26 | 35 | 44 | |
| 1 | 10 | \| | 25 | 34 | 43 | € 52 |
| I | 9 | \| | 24 | 33 | 42 | 51 |

## 3 Your program

### 3.1 As illustrated on Figure 3 some situations cannot lead to a solution. Are there other similar situations? If yes, describe them.

A dead state can occcur when a box is surrounded with 2 walls. In that case, the sokoban cannot move the box anymore and thus, the game cannot be resolved. Another case that can lead to a dead state is one a block is stuck on a column or line when the goal is not on this column/line.

### 3.2 Why is it important to identify dead states in your successor function? How are you going to implement it?

Dealing with dead states is important because it avoid expanding a node that cannot lead to a solution. Therefore we can save some computation time. To implement it, we mark on a grid all positions that lead to a dead state with a '*'. This grid is computed only when launching the algorithm (to avoid recalculating the grid at every iteration) and is based on the goal grid. For instance:

Table 3:

| | | | | | | |
|---|---|---|---|---|---|---|
| # | # | # | # | # | # | # |
| # | * | # | . | # | * | # |
| # | * | # | | # | * | # |
| # | * | | | | * | # |
| # | * | | | | * | # |
| # | . | | # | * | # | # |
| # | # | # | # | # | # | # |

- The * represents the dead state positions.
- The . represents a goal
- The # reprensents a wall

### 3.3 Describe possible (non trivial) heuristic(s) to reach a goal state (with reference if any). Is(are) your heuristic(s) admissible and/or consistent?

- A consistent heuristic is to sum the distance between one block and the closer goal for each block using a straight line distance (a improvement can be to consider wall positions)
- A second heuristic is to sum the distance between block one and goal one, block 2 and goal 2 and so on.
- Another heuristic is to count the number of blocks that are not on a goal.

### 3.4 Experiment, compare and analyze informed (astar_graph_search) and uninformed (breadth_first_graph_search) graph search of aima-python3 on the 15 instances of sokoban provided. Report in a table the time, the number of explored nodes and the number of steps to reach the solution. Are the number of explored nodes always smaller with astar_graph_search , why? When no solution can be found by a strategy in a reasonable time (say 5 min), explain the reason (time-out and/or swap of the memory)

Table 4:

| | A* | | | BFS | | |
|---|---|---|---|---|---|---|
| Grid # | Time | # Nodes visited | # steps | Time | # Nodes visited | # steps |
| 1 | 0.14s | 899 | 15 | 0.28s | 2108 | 15 |
| 2 | 0.95s | 5133 | 65 | 0.94s | 5183 | 65 |
| 3 | 77.7s | 237505 | 135 | 77s | 239126 | 135 |
| 4 | 3.94s | 21413 | 28 | 5.1s | 28605 | 26 |
| 5 | 3.88s | 19729 | 108 | 3.88s | 19743 | 108 |
| 6 | 5.72s | 29197 | 27 | 10.8s | 54610 | 27 |
| 7 | 1.0s | 4091 | 98 | 1.06s | 4218 | 98 |
| 8 | 1.4s | 6307 | 90 | 1.4s | 6325 | 90 |
| 9 | 23.3s | 99591 | 101 | 23.3s | 100629 | 101 |
| 10 | 0.4s | 2463 | 43 | 0.4s | 2662 | 42 |
| 11 | 57.4s | 187811 | 180 | 58s | 190754 | 179 |
| 12 | 16.6s | 50704 | 70 | 21.8s | 65366 | 70 |
| 13 | 37.1s | 126631 | 182 | 37.8s | 127184 | 182 |
| 14 | 33s | 149386 | 66 | 35s | 158755 | 65 |
| 15 | 6.7s | 28420 | 56 | 8.2s | 34383 | 56 |

### 3.5 What are the performances of your program when you don't perform dead state detection?

When not using dead state detection the performance of the program drops significantly. For example, it takes 5 times longer to solve the example 2 when not using dead state detection. Unfortunatly, our heuristic is not good enough to decrease significantly the execution time by using the A* search. We can see that the difference of nodes explored between the two tables is really low, which show the poor efficiency of the heuristic function. The number of nodes explored by using A* is still always lower that when using breadth-first, but we would expect a much lower value than what we see on the table.

A* finishes all the benchmark in 272s, breadth-first finishes all the benchmark in 285s, which is only a 4% gain. When using A*, the total number of nodes explored is only 7% lower than what we got with breadth-first.