

## Assignment #1 – Answer Sheet

**Search in discrete and continuous spaces**

Due: Start of Lab, Week 6 (1pm, 30 August 2006)

**1 Overview**

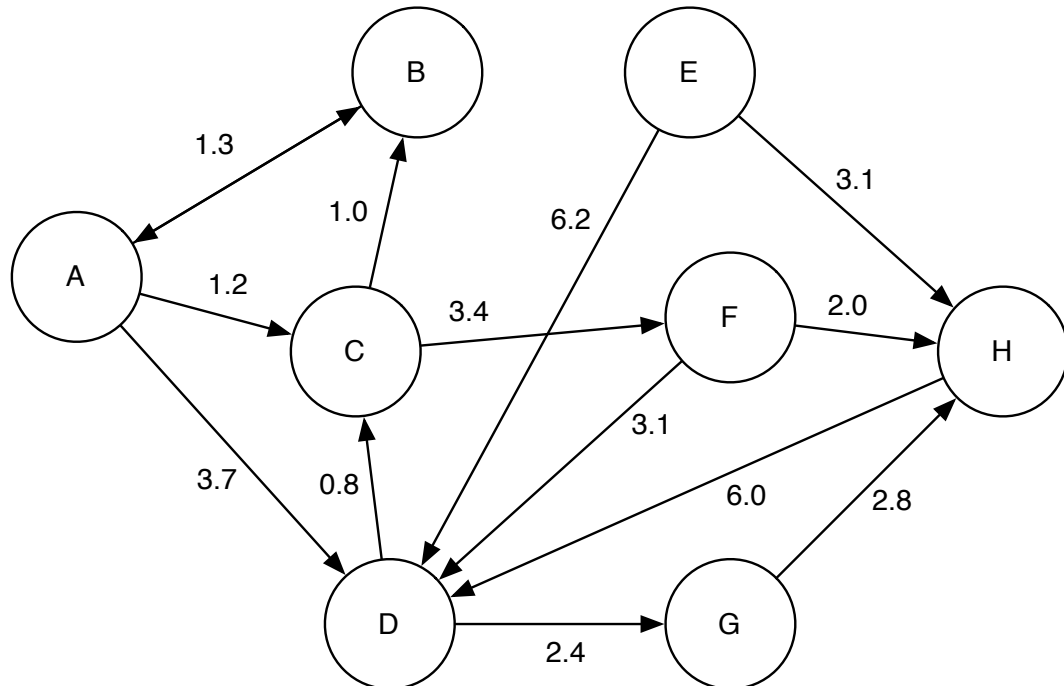
The goal of this assignment is to test your understanding of various search algorithms. While this assignment does not directly ask you to implement or trace all the algorithms discussed in class, it may be advantageous for you to do so. To answer the questions in this assignment, you will have to do some investigation of the algorithms outside of what was explicitly mentioned in class.

You should feel free to ask the Lecturer questions and discuss the assignment with others, but you should not copy someone else's work. (e.g. You can get someone to show you how to answer a question, but you should write your answer up yourself from what you remember at least 30 minutes after they've left.)

Total marks for this assignment: 90. It will be scaled to 15% of the comp3431 grade.

**2 Discrete Search**

In the following questions we'll be considering this graph:



With the following heuristic costs to node H:

Node	Distance
A	6.0
B	4.0
C	5.1
D	4.5
E	3.0
F	2.0
G	2.5
H	0.0

You should assume that for any node, if we need its children, we can iterate through them in alphabetical order. When asked to trace a discrete search algorithm, you should list the state of the fringe data structure just before a node is removed from it, you should also list what node is removed, and any state associated with that node. This should be repeated from when the start node is added to the fringe, until the algorithm is complete.

## 2.1 Question 2.1 (5 marks):

Write the pseudo-code for a generalised discrete search algorithm. Your algorithm should print out the final path between the start and goal nodes.

$F \leftarrow \{s\}$  – the fringe

```

 $V \leftarrow \{s\}$  – the set of visited nodes
 $P \leftarrow \{s, \emptyset\}$  – the hash of parent pointers
while  $F \neq \emptyset$  do
   $n \leftarrow$  next state from  $F$ 
  if  $n = g$  then
    – print out the path to the goal
    – this prints it out reversed
    repeat
      print  $n$ 
       $n \leftarrow P(n)$ 
    until  $n = \emptyset$ 
    Exit
  end if
  for all children  $n'$  of  $n$  do
    if  $n' \in V$  then
      Choose which node you're keeping,  $n'$  or the current node
    end if
     $V \leftarrow V \cup \{n'\}$ 
     $P \leftarrow P \cup \{n', n\}$ 
     $F \leftarrow F \cup \{n'\}$ 
  end for
end while
Print “No path”

```

Something like the function Graph-Search on page 83 of Russell and Norvig would also work, if you remembered to store back-pointers.

## 2.2 Question 2.2 (5 marks):

Trace the behaviour of Depth First Search on the graph starting at node A, with a goal of node H.

I've used subscripts for parent pointers.

- Fringe:  $A_\emptyset$  Remove:  $A_\emptyset$
- Fringe:  $B_A, C_A, D_A$  Remove:  $D_A$
- Fringe:  $B_A, C_A, G_D$  Remove:  $G_D \leftarrow$  could be done here
- Fringe:  $B_A, C_A, H_G$  Remove:  $H_G$

## 2.3 Question 2.3 (5 marks):

Trace the behaviour of Breadth First Search on the graph starting at node A, with a goal of node H.

- Fringe:  $A_\emptyset$  Remove:  $A_\emptyset$
- Fringe:  $B_A, C_A, D_A$  Remove:  $B_A$
- Fringe:  $C_A, D_A$  Remove:  $C_A$
- Fringe:  $D_A, F_C$  Remove:  $D_A$
- Fringe:  $F_C, G_D$  Remove:  $F_C \leftarrow$  could be done here
- Fringe:  $G_D, H_F$  Remove:  $G_D$
- Fringe:  $H_F$  Remove:  $H_F$

## 2.4 Question 2.4 (5 marks):

Trace the behaviour of Best First Search on the graph starting at node A, with a goal of node H.

I've used superscripts to indicate heuristic values, and shown the fringe as a sorted list. Subscripts are still parent pointers.

- Fringe:  $A_\emptyset^{6.0}$  Remove:  $A_\emptyset^{6.0}$
- Fringe:  $B_A^{4.0}, D_A^{4.5}, C_A^{5.1}$  Remove:  $B_A^{4.0}$
- Fringe:  $D_A^{4.5}, C_A^{5.1}$  Remove:  $D_A^{4.5}$
- Fringe:  $G_D^{2.5}, C_A^{5.1}$  Remove:  $G_D^{2.5} \leftarrow$  could be done here
- Fringe:  $H_G^{0.0}, C_A^{5.1}$  Remove:  $H_G^{0.0}$

## 2.5 Question 2.5 (5 marks):

Trace the behaviour of Uniform Cost Search on the graph starting at node A, with a goal of node H.

Here superscripts are not heuristic values, but rather costs from the start. I've shown those costs as the old total plus the increment the first time. Again, I've shown the fringe as a sorted list. Subscripts are still parent pointers.

- Fringe:  $A_\emptyset^{0.0}$  Remove:  $A_\emptyset^{0.0}$
- Fringe:  $C_A^{0.0+1.2=1.2}, B_A^{0.0+1.3=1.3}, D_A^{0.0+3.7=3.7}$  Remove:  $C_A^{1.2}$
- Fringe:  $B_A^{1.3}, D_A^{3.7}, F_C^{1.2+3.4=4.6}$  Remove:  $B_A^{1.3}$
- Fringe:  $D_A^{3.7}, F_C^{4.6}$  Remove:  $D_A^{3.7}$
- Fringe:  $F_C^{4.6}, G_D^{3.7+2.4=6.1}$  Remove:  $F_C^{4.6} \leftarrow$  could be done here

- Fringe:  $G_D^{6.1}, H_F^{4.6+2.0=6.6}$  Remove:  $G_D^{6.1}$
- Fringe:  $H_F^{6.6}, H_G^{6.1+2.8=8.9}$  Remove:  $H_F^{6.6}$  ← *strikeout indicates a second path considered but not taken because its cost from the start was higher than the one already in the fringe. That node is never actually added to the fringe.*

## 2.6 Question 2.6 (5 marks):

Trace the behaviour of A\* Search on the graph starting at node A, with a goal of node H.

In this example, superscripts show the cost from the start, the heuristic estimate to the goal, and the sum. Again, I've shown the fringe as a sorted list. Subscripts are still parent pointers.

- Fringe:  $A_\emptyset^{0.0+6.0=6.0}$  Remove:  $A_\emptyset^{0.0+6.0=6.0}$
- Fringe:  $B_A^{1.3+4.0=5.3}, C_A^{1.2+5.1=6.3}, D_A^{3.7+4.5=8.2}$  Remove:  $B_A^{1.3+4.0=5.3}$
- Fringe:  $C_A^{1.2+5.1=6.3}, D_A^{3.7+4.5=8.2}$  Remove:  $C_A^{1.2+5.1=6.3}$
- Fringe:  $F_C^{4.6+2.0=6.6}, D_A^{3.7+4.5=8.2}$  Remove:  $F_C^{4.6+2.0=6.6}$  ← *A\* cannot finish here and provably find the shortest path*
- Fringe:  $H_F^{6.6+0.0=6.6}, D_A^{3.7+4.5=8.2}$  Remove:  $H_F^{6.6+0.0=6.6}$

## 2.7 Question 2.7 (10 marks):

In normal A\* search, the priority of a node,  $n$ , is:  $f(n) = g(n) + h(n)$ . If we add a constant coefficient,  $\alpha$ , (greater than one) to the heuristic, we get  $f(n) = g(n) + \alpha h(n)$ . If the optimal path cost is  $C^*$ , is there a bound on the cost of the path found by our new search algorithm? If so, what is it?

Can you prove your bound?

One bound is that path found has its cost bounded by  $\alpha C^*$ . The proof uses the standard A\* proof setup - a proof by contradiction where we look at the last step of the search. Imagine we have a start node  $s$ , a goal node  $g$ , the node in the fringe furthest along the optimal path to the goal  $o$ . We will assume that the goal node is in the fringe, and is about to be pulled out of the fringe as it has the lowest  $f$  value.

$g(o) + h(o) < C^*$  because  $o$  is on the optimal path to the goal and our heuristic is admissible  $f(o) = g(o) + \alpha h(o) \leq \alpha g(o) + \alpha h(o) = \alpha(g(o) + h(o)) \leq \alpha C^*$   
 $f(g) = g(g) + \alpha h(g) = g(g) + 0$

If we are about to extract  $g$  from the fringe, then it must have the lowest  $f$  value. We see that  $f(o)$  no more than  $\alpha C^*$ , so the  $f$  value of  $g$  must be no more than  $\alpha C^*$ . If we assume that this path was going to violate our bound then, then we have a contradiction.

### 2.8 Question 2.8 (5 marks):

Give an admissible heuristic estimate for the minimum number of search steps from each node to H. Your heuristic should assign non-zero values to some nodes.

Node	Distance
A	3
B	4
C	2
D	2
E	1
F	1
G	1
H	0

The above heuristic is the highest possible. Anything underneath that would also be admissible.

### 2.9 Question 2.9 (5 marks):

Use your heuristic to perform Best First Search on the graph. How does use of this heuristic compare with the cost-heuristic above?

- Fringe:  $A_\emptyset^3$  Remove:  $A_\emptyset^3$
- Fringe:  $C_A^2, D_A^2, B_A^4$  Remove:  $C_A^2$
- Fringe:  $F_C^1, D_A^2, B_A^4$  Remove:  $F_C^1$  ← could be done here
- Fringe:  $H_F^0, D_A^2, B_A^4$  Remove:  $H_F^0$

It is faster, although if a poor heuristic is chosen, it could be slower.

## 3 Continuous Search

It is possible to perform a line-search analytically in some situations. Consider a function on a two dimensional space,  $z = f(x, y)$ . If we restrict ourselves to the line  $y = mx + c$  then we can substitute that into the original function to get  $z = f(x, mx + c)$ . This is now a function of one variable,  $x$ . We can differentiate and set the result to 0 to get a minimum or maximum.

For example; we start with  $z = x^2 + y^2 + 2xy$ , and want to search along the line  $y = 3x + 7$ . We substitute to get  $z = x^2 + (3x + 7)^2 + 2(3x + 7)x = 16x^2 + 56x + 49$ . Which we can differentiate to get  $z' = 32x + 56$ . The differential is 0 at  $x = -56/32 = -1.75$ . We can back-substitute to get  $y = 1.75$  for a minimum point along the line of  $(-1.75, 1.75)$ .

### 3.1 Question 3.1 (10 marks):

Show the steps involved in Powell's Direction Set method of minimisation for the function  $z = 2x^2 + y^2 + 2xy$ . Assume our initial direction set consists of the  $(1, 0)$  and  $(0, 1)$  unit vectors, and we start minimising from the point  $(2, 3)$ . Use the analytic method above to perform the each line minimisation.

- initially, at the point  $(2, 3)$ ,  $z = 8 + 9 + 12 = 29$
- start by minimising in one direction, say  $(1, 0)$ . This is the line  $y = 3$ . Substituting gives  $z = 2x^2 + 6x + 9$ , which has a derivative of  $z' = 4x + 6$  which is zero at  $x = -6/4$ . First min:  $(-1.5, 3)$ . At this point  $z = 4.5$ .
- now we iterate through all directions and minimise. I'll start with the second vector to make sure I don't try an minimise in exactly the same direction twice in a row.
  - minimize along direction  $(0, 1)$  from the point  $(-1.5, 3)$ . This is the line  $x = -1.5$ . Substituting gives  $z = 4.5 + y^2 - 3y$ , which has a derivative of  $z' = 2y - 3$  which is zero at  $y = 1.5$ . Second min:  $(-1.5, 1.5)$ . At this point  $z = 2.25$  for a drop of 2.25.
  - minimize along direction  $(1, 0)$  from the point  $(-1.5, 1.5)$ . This is the line  $y = 1.5$ . Substituting gives  $z = 2x^2 + 3x + 2.25$ , which has a derivative of  $z' = 4x + 3$  which is zero at  $x = -3/4$ . Third min:  $(-0.75, 1.5)$ . At this point  $z = 1.125$  for a drop of 1.125.
- The total change in location over that set of minimizations is  $(-0.75, 1.5) - (-1.5, 3) = (0.75, -1.5)$ . This will be our new minimization direction. We'll replace the direction that had the largest drop in the last set of minimizations which was  $(0, 1)$ . Our new directions are  $(0.75, -1.5)$  and  $(1, 0)$ .
- We now minimise along each of these directions in turn
  - minimize along direction  $(0.75, -1.5)$  from the point  $(-0.75, 1.5)$ . This is the line  $y = -2x$ . Substituting gives  $z = 2x^2 + 4x^2 - 4x^2$ , which has a derivative of  $z' = 4x$  which is zero at  $x = 0$ . Back-substituting we find that  $y = 0$  at this point. Fourth min:  $(0, 0)$ . At this point  $z = 0$  for a drop of 1.125.
  - minimize along direction  $(1, 0)$  from the point  $(0, 0)$ . This is the line  $y = 0$ . Substituting gives  $z = 2x^2$ , which has a derivative of  $z' = 4x$  which is zero at  $x = 0$ . Fifth min:  $(0, 0)$ . We haven't moved!
- Noting that we haven't moved, and that the vector set is not linearly dependent, we decide we must be at the minimum at  $(0, 0)$ .

### 3.2 Question 3.2 (5 marks):

For the function  $z = 2x^2 + y^2 + 2xy$ , what direction is conjugate to the vector  $(1, 0)$ ?

This was calculated by Powell's Method. It is the direction  $(0.75, -1.5)$ . It was because these directions were conjugate that Powell's method took us straight to the minimum when it started minimising in those directions.

## 4 Representation

Consider a small  $5 \times 5$  gridworld. (i.e. a small world with 25 states laid out on a  $5 \times 5$  rectangular grid.) We have four actions, each of which moves our agent to a neighbouring state, North, South, East or West. We will assume our world is deterministic. If the agent collides with a wall, then it does not move. We will assume that there are walls around the edge of the world, and between some, but not all, of the states.

There is one defined start state, and one defined goal state.

### 4.1 Question 4.1 (5 marks):

What is the standard transformation of this problem into a state space search?

Each state in the world corresponds to a node in the graph we search. Actions correspond to edges that move us between states/nodes. A path through the world corresponds to a path through the graph.

Consider the following plan space representations of the problem:

- Partial policy: Each state in our search space is a 25-vector of discrete values. Each element of this vector is drawn from the set {North, South, East, West, Undecided}. The elements of the vector correspond with the actions to be taken in the states of the original problem. The vectors in this space are restricted so that vectors are only in the search space if taking the assigned actions in turn will lead us to the goal in all states for which an action is assigned.
- Full policy: Each state in our search space is a 25-vector. Each element of this vector is drawn from the set {North, South, East, West}.
- Stochastic policy: Each state in our search space is a  $25 \times 4$  matrix of numbers. Each column in the matrix corresponds with a state in the original space. Each row corresponds with an action in the original space. The columns are normalised to find the probability that your agent takes that action in that state.
- Plan: Each state in the search space is a list of actions to be followed consecutively from the start state.

## 4.2 Question 4.2 (20 marks):

Describe a search technique that could be used with each representation. Include a description of any heuristics or evaluation functions you think are necessary.

- **Partial policy:** This could be searched with a discrete best-first search. A simple 'childrenOf()' operator you might think of is: The children of a node are the partial policies that have one more action assigned. Unfortunately, this doesn't fit with the constraint given. Instead you must choose the subset of those children that meet the specification given. A new node will meet the spec given if the action you just added takes you to a state with an already assigned action. A good heuristic for the best-first search would be to choose the distance from the start state to the closest state with an assigned action.
- **Full Policy:** I'd probably choose a local search for this plan space search. You get neighbours of a plan by simply changing one action at random. The interesting part is the evaluation function.
  - Note that it is possible to check nodes back from the goal state, searching backwards only where the assigned action matches (is the inverse of) where you're trying to search. This search will peter out before reaching the start state if the current policy can't get you to the goal. Look to have large search trees of this form.
  - Start at the start state and follow the policy. At some point you'll either reach the goal or start to cycle. The heuristic is the closest you ever get to the goal state.
- **Stochastic Policy:** here you have a continuous space, not a discrete one. That means you want to use a continuous search method. Conjugate gradient would be a good first choice. To evaluate each plan, you could use the expected number of steps to reach the goal. This could be found by sampling – run 100 trials and measure the average number of steps to reach the goal. If you never reach the goal, then record the closest you get. If no policies are reaching the goal, then you could use the 'closest' option, until most of them are reaching the goal, then switch to improving the speed.
- **Plan:** This is almost identical to the Partial Policy setup, except that all actions are assigned in a row. You could use best first search, with a heuristic of the distance to goal when you have finished executing the plan from the start state.