CITS4211 Mid-semester test 2012

Fifty minutes, answer all three questions, total marks 60

Question 1. (20 marks) Briefly describe and contrast the principles, operation, and performance issues of *breadth-first* and *depth-first* search.

Illustrate your answer using the tree in Figure 1. Include enough detail to make it clear that you understand how the algorithms work, particularly which nodes are expanded, in what order, and why.

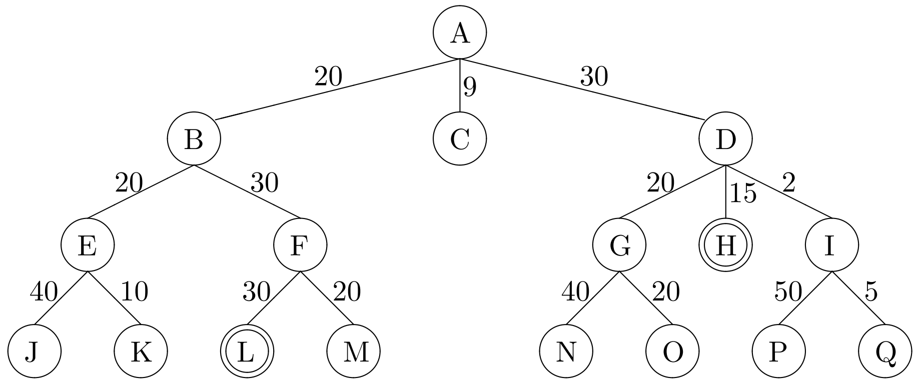

Figure 1: The search tree for Questions 1 and 2. Each arc is labeled with its cost. H and L are the only goal states.

Question 2. (20 marks) Briefly describe the principles, operation, and performance issues of *A**.

Illustrate your answer using the tree in Figure 1 and the heuristic function in Figure 2. Include enough detail to make it clear that you understand how the algorithm works, particularly which nodes are expanded, in what order, and why.

A	41	D	12	G	14	J	8	M	19	P	13
B	20	E	9	H	0	K	2	N	20	Q	13
C	34	F	30	I	12	L	0	O	30		

Figure 2: An admissible heuristic function for the tree in Figure 1.

Question 3. (20 marks) Briefly describe the principles, operation, and performance issues of *policy iteration.*

Illustrate your answer using the problem in Figure 3. Include enough detail to make it clear that you understand how the algorithm works, particularly how utilities are calculated, and how new policies are derived and justified. (Focus on doing one iteration correctly and thoroughly.)

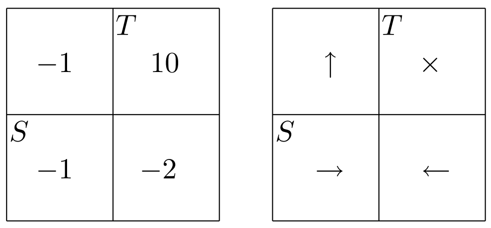

Figure 3: The path-planning problem for Question 3, and the initial policy. S is the start state, T is the only terminal state, and each state is labeled with its reward. Every attempted move has an 80% chance of working, a 10% chance of veering left 90°, and a 10% chance of veering right 90°. Walking into a wall means the agent stays put.

1. *Breadth-first* search (BFS) expands the shallowest node first: thus it orders the nodes of a tree such that all of the siblings of a node are processed before any of its children.

BFS is complete: it will always find a goal state. Also it is optimal: it will always find the shallowest goal state. It takes time proportional to b^d, which is typical of simple search, but the big problem is its space complexity: all unprocessed nodes must be held in memory, and this number also is proportional to b^d. BFS is unusable in practice for realistic problems.

In the example the nodes are expanded in this order:

A, B, C, D, E, F, G, H

and the shallowest goal state is found.

Depth-first search (DFS) expands a single sub-tree completely, then backtracks to expand alternatives: thus it orders the nodes of a tree such that all of the children of a node are processed before moving on to a different sibling.

DFS is neither complete nor optimal: it can get stuck following infinite branches of the tree, and it returns the first goal state that it happens to find. Its time complexity is bad for trees with very deep branches, but its space complexity is very good: it is proportional only to the length of the longest branch processed. DFS usually needs an occurs-check to avoid infinite loops. DFS is improved substantially if a depth can be identified before which a solution is known to occur.

In the example the nodes are expanded in this order:

A, B, E, J, K, F, L

and the goal state found is not the shallowest, merely the "leftest".

2. *A** maintains a pool of nodes, which starts with just the root. The cost-estimate of each node X is the sum of the cost from the root to X and the heuristic estimate of the cost from X to the goal state. The next node expanded is one which has the lowest cost-estimate. It stops when a goal state is found.

*A** is provably optimal (it always finds the cheapest goal state) as long as its heuristic is admissible, i.e. it never over-estimates the cost from X to the goal state. Variants of *A** (SMA*, IDA*) have been described which also have good space behaviour.

In the example the pool of nodes evolves like this:

A	$0 + 41 = 41$

B	$20 + 20 = 40 \rightarrow 41$
D	$30 + 12 = 42$
C	$9 + 34 = 43$

D	$30 + 12 = 42$
C	$9 + 34 = 43$
E	$40 + 9 = 49$
F	$50 + 30 = 80$

C	$9 + 34 = 43$
I	$32 + 12 = 44$
H	$45 + 0 = 45$
E	$40 + 9 = 49$
G	$50 + 14 = 64$
F	$50 + 30 = 80$

I	$32 + 12 = 44$
H	$45 + 0 = 45$
E	$40 + 9 = 49$
G	$50 + 14 = 64$
F	$50 + 30 = 80$

H	$45 + 0 = 45$
E	$40 + 9 = 49$
Q	$37 + 13 = 50$
G	$50 + 14 = 64$
F	$50 + 30 = 80$
P	$82 + 13 = 95$

3. *Policy iteration* starts with an arbitrary policy; in each iteration it calculates the value of each non-terminal state under the current policy, then the optimal move in each non-terminal given those values and the transition function. If the policy needs to be updated, it iterates.

Policy iteration is generally faster than the alternative *Value iteration*, because it reaches equilibrium without calculating the precise value of each state.

In the example the first iteration of the algorithm goes like this (x is the utility of the top-left state, y is bottom-left, z is bottom-right):

$$\begin{aligned} x &= -1 + 0.9x + 1 && (1) \\ y &= -1 + 0.8z + 0.1y + 0.1x && (2) \\ z &= -2 + 1 + 0.8y + 0.1z && (3) \end{aligned}$$

(1) gives $x = 0$, so simplify and multiply by 10 throughout.

$$\begin{aligned} 9y &= -10 + 8z && (4) \\ 9z &= -10 + 8y && (5) \end{aligned}$$

Clearly $y = z$, so substitute z for y in (4) and solve.

$$x = 0, y = -10, z = -10$$

Top-left: clearly right is best.

$$\begin{aligned} up : 0.9(0) + 1 &= 1 \\ down : 0.1(0) + 1 + 0.8(-10) &= -7 \\ right : 0.1(0) + 8 + 0.1(-10) &= 7 \\ left : 0.9(0) + 0.1(-10) &= -1 \end{aligned}$$

Bottom-left: clearly up is best.

$$\begin{aligned} up : 0.8(0) + 0.1(-10) + 0.1(-10) &= -2 \\ down : 0.9(-10) + 0.1(-10) &= -10 \\ right : 0.1(-10) + 0.1(0) + 0.8(-10) &= -9 \\ left : 0.9(-10) + 0.1(0) &= -9 \end{aligned}$$

Bottom-right: clearly up is best.

$$\begin{aligned} up : 8 + 0.1(-10) + 0.1(-10) &= 6 \\ down : 0.9(-10) + 0.1(-10) &= -10 \\ right : 1 + 0.9(-10) &= -8 \\ left : 1 + 0.8(-10) + 0.1(-10) &= -8 \end{aligned}$$

Figure 4 shows the policy after the first iteration. One more iteration would establish that this policy is optimal.

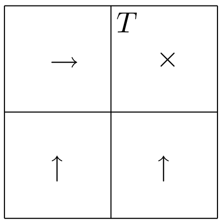

Figure 4: Policy after one iteration.