University of Waterloo

Department of Electrical and Computer Engineering

ECE 457A Cooperative and Adaptive Algorithms
Final Examination
August 2nd, 2016, 4:00-6:30. Location: M3 1006

Answer All Questions

(a) What is the advantage of breadth-first search over depth-first search?

Breadth-first search is guaranteed to find the best (closest to the root) solution, whereas depth-first search may go unnecessarily deep into the tree and find a suboptimal solution.

(b) What is the advantage of depth-first search over breadth-first search?

Depth-first search has a space (memory) complexity of only O(n) for a maximum search depth of n, because only the current path needs to be stored at any time. Breadth-first search, on the other hand, actually builds the entire search tree up to the level of the solution, so its space complexity is $O(b^n)$ for branching factor b and solution level n.

c) Describe the strategy of iterative deepening and its advantages over breadth-first and depth-first search.

Iterative deepening is an iterated, depth-limited variant of the depth-first search algorithm. In each iteration, depth-first search is carried out with a maximum search depth of 0 (only the root is tested). If no solution is detected, the maximum search depth is increased to 1 and another search is started. With each iteration, the maximum search depth is increased by one, until a solution is detected.

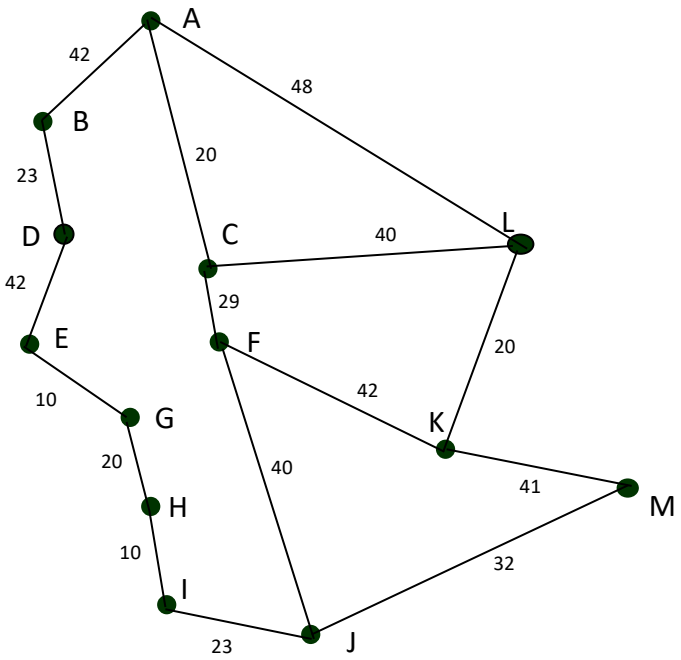
The advantage of iterative deepening over breadth-first search is its low space complexity O(n) for a solution located at level n. At the same time, A* is still guaranteed to find the shortest path to a solution. Its time complexity is slightly worse than that of breadth-first search, but for deep trees with a large branching factor, this difference is insignificant. The advantage of iterative deepening over depth-first search is that iterative deepening is guaranteed to find the best solution (i.e., the shortest path) without ever going unnecessarily deep into the tree.

Describe the differences between genetic programming and genetic algorithms.

(5 marks)

b) Consider the following map (not drawn to scale).

(16 marks)



Using the A* algorithm work out a route from town A to town M. Use the following cost functions.

- G(n) = The cost of each move as the distance between each town (shown on map).
- H(n) = The Straight Line Distance between any town and town M. These distances are given in the table below.

Provide the search tree for your solution, showing the order in which the nodes were expanded and cost at each node. You should not re-visit states previously visited. Finally, state the route you would take and the cost of that route.

Straight Line Distance to M

A	51
B	50
C	32
D	28

E	42
F	14
G	33
H	43

I	50
J	32
K	41
L	56

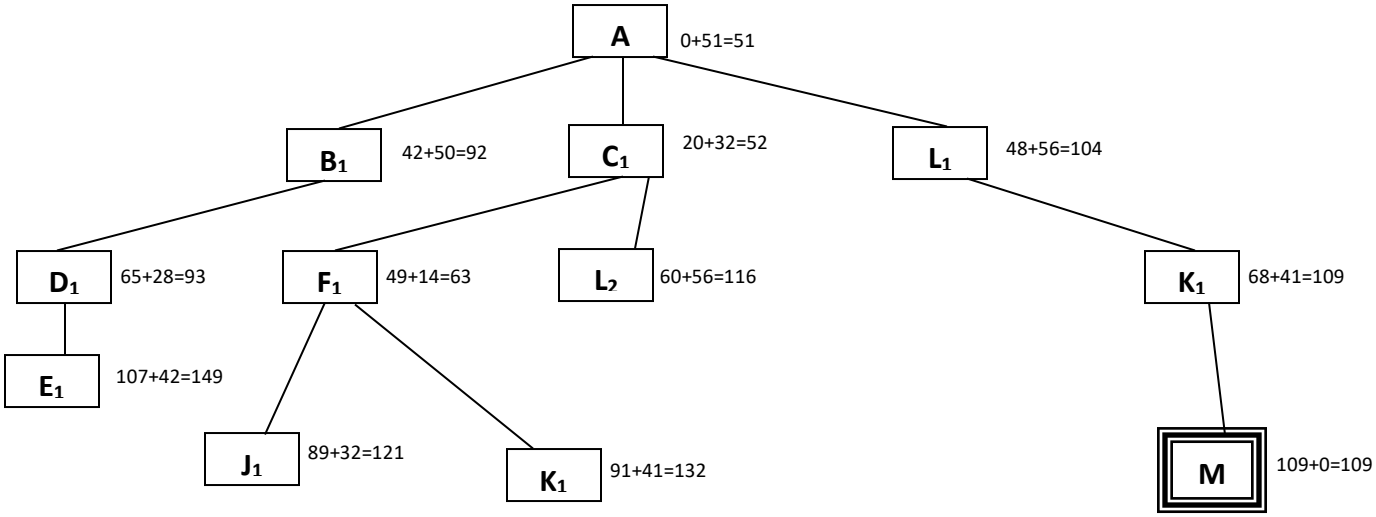
M	0

c) The straight line distance heuristic used above is known to be an admissible heuristic. What does this mean and why is it important?

b) Using A* Algorithm

The Search Tree

The figures next to each node represent the G(n) and H(n) functions, where
G(n) = The cost of the search so far (i.e. distance travelled)
H(n) = The heuristic value (i.e. the straight line distance to the target town)



The nodes would be expanded in the following order A, C_1, F_1, B_1, D_1, L_1, K_1 and then M (that is the lowest cost node is expanded next).

The route taken is A, L, K, M at a cost of 109.

c) The straight line distance heuristic used above is known to be an admissible heuristic. What does this mean and why is it important?

An admissible heuristic is one which never over estimates the cost to the goal. This is obviously the case with the straight line distance between two towns.

Having admissible heuristics is important as it allows the A algorithm to be proved to be optimal (i.e. always find the best solution).*

Question :

(a) Using Genetic Programming to learn a Boolean Function [13]

Assume you want to learn a Boolean function f(a,b,c,d) with 4-input variables a, b, c, d, such as[1] (a∧b) ∨ (c^~d) from training data, which have the following form:

[1] "~' represents the negation operator

a	b	c	d	output
1	1	0	0	1
1	1	1	0	1
0	0	1	0	1
0	1	0	1	0
1	1	0	0	1

For example, the Boolean function (a$\wedge$b) $\vee$ (c^~d) predicts the output of the first 4 examples correctly, but not the fifth training example.