Assignment 2

CHAPTER 3

Ex. 3.8

(a) However long or circuitous a path might be, it having a (large) negative cost would actually lower the cost function, and thus make it seem to the algorithm that the longer path is a good choice mathematically. Eventually, the algorithm will end up exploring the entire space to chalk out the most (mathematical) optimal path.

(b) It doesn't help. In case of a tree, a negative constant *c* might undermine the optimality of the selected path. As for a graph, the algorithm might keep looping along the negative cost path, lowering the cost function steadily at a rate of *c* per traversal.

(c) The agent being mathematically optimal, in that the objective is to reach the goal state by keeping its cost function **as low as possible**, it would keep going around the loop forever, until it chances upon an even better path.

(d) Humans usually set aside time to account for time of travel. We don't drive along scenic loops indefinitely because every traversal increases the time one would need to arrive at one's destination. The same goes with fuel as well, which isn't a resource available freely. If an artificial agent is penalized for the amount of time it takes to reach its destination, it's unlikely to spend too much time looping indefinitely along a particular path, no matter how much it helps the cost function. If such a time-penalty cannot be levied, the other way would be to have a function in place that diminishes the reward(s) of going through the same path more than once.

(e) Being a drug addict would be a real domain example.

Ex. 3.9

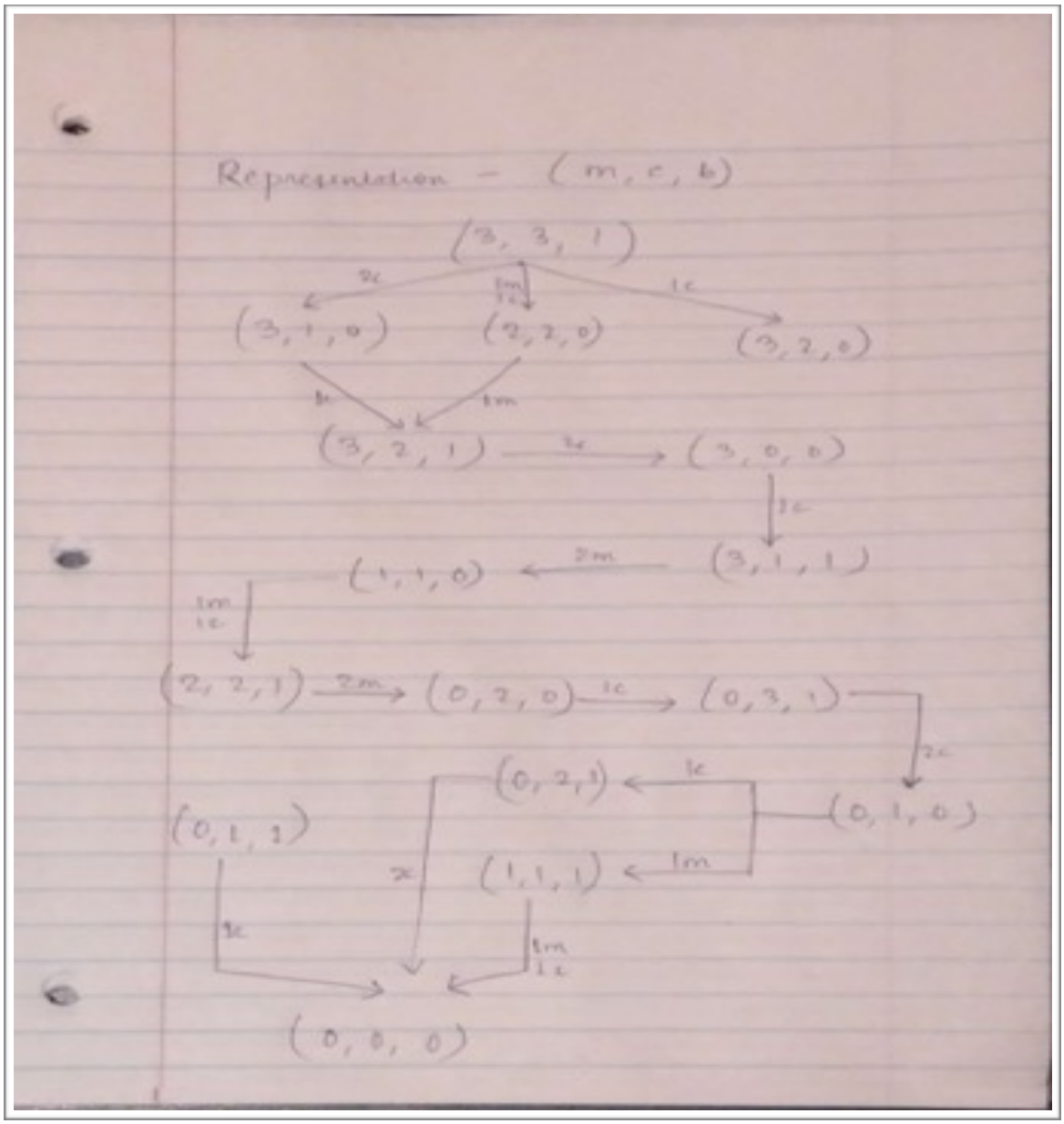
(a) The number of missionaries should always be greater than or equal to number of cannibals on any side of the bank. At most, only 2 persons can use to boat the row over to the other side. A list [m, c, b] could represent the number of missionaries, cannibals and boat on the wrong side: the starting state would be [3, 3, 1] and the goal state would be [0, 0, 0]. The entire state space according to this representation would be: `[3, 3, 1], [3, 1, 0], [2, 2, 0], [3, 2, 0], [3, 2, 1], [3, 0, 0], [3, 1, 1], [1, 1, 0], [2, 2, 1], [0, 2, 0], [0, 3, 1], [0, 1, 0], [0, 2, 1], [1, 1, 1], [0, 1, 1], [0, 0 ,0].`

The diagram is on the next page.

(b) Please refer to the files `missionaries and cannibals BFS.py` and `missionaries and cannibals BFS output.doc`

It's a good idea to check for repeated states, because not doing so might undo the action of transporting missionaries and/or cannibals to the other side.

(c) As easy as it to understand that the number of missionaries on one bank should not exceed the number of cannibals, it's not as simple to make the distinction between legal and illegal moves. Also, the branching factor appears to be huge.



Ex. 3.21

(a) When the step costs are equal for all parent-child node pairs, uniform-cost search produces the same result as breadth-first search. That is to say that BFS cannot be applied if the step-costs are unequal, unlike uniform-cost search.
(b) Best-first search expands nodes according to a particular criterion. If the criterion happens to something along the lines of “expand deepest node first”, it’s essentially depth-first search.

(c) If the heuristic function in A* search is set to *zero*, it gets reduced to uniform-cost search. This implies uniform-cost search is a special case of A* search.

Ex. 3.22

Please refer to the files `8 puzzle (A star).py` and `8 puzzle (A star).doc`