BLG435E - Artificial Intelligence

Homework 1 Report

150130051 – İlay Köksal

Q1)

a) Domestic Service Robot

Performance Measure: percentage of clear floor, minimize energy consume, do not break objects while carrying (Can be more performance measurement criteria according to robot model)
Environment: rooms, objects, people who live in, house pets
Actuators: wheels, speaker (hands for holding objects maybe?)
Sensors: camera, microphone, (weight scale?)

Observability – Partially
Deterministicness – Stochastic
Episodicness – Episodic
Staticness – Dynamic
Discreteness – Continuous
Agents – Multi

Simple reflex agent
Agent decide the action by looking the situation of the room and considering condition-action rules.

b) E-mail sorting application based on preferences

Performance Measure: not showing unwanted mails, sorting performance, sorting speed, consuming less memory space
Environment: user, mails
Actuators: screen display
Sensors: keyboard

Observability – Fully
Deterministicness – Deterministic
Episodicness – Episodic
Staticness – Static
Discreteness – Discrete
Agents – Single

Simple reflex agent
By looking at the rules that given by user (condition-action rules), agent decides to keep email or delete it.

c) Autonomous car driver agent

Performance Measure: safe trip, legal, fast, minimize fuel consume
Environment: roads, traffic signs, traffic lights, pedestrians, passengers
Actuators: wheels, brake, gas pedal, horn, signal lights

Sensors: speedometer, gps, engine sensors, camera

Observability – Partially
Deterministicness – Stochastic
Episodicness – Sequential
Staticness – Dynamic
Discreteness – Continuous
Agents – Multi

Utility-based agent
There are multiple solutions to goal state. Some are better than others. A driver can choose multiple paths to goal location. But it should choose better one by looking utility function.

d) Activity recognition and anomaly detection software agent in an airport

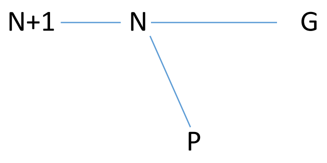
Performance Measure: fast recognition of anomaly, percentage of anomaly detection
Environment: passengers, security attendant
Actuators: alarm, screen display
Sensors: camera, keyboard

Observability – Partially
Deterministicness – Deterministic
Episodicness – Sequential
Staticness – Dynamic
Discreteness – Continuous
Agents – Single

Model based reflex agent
Considers the state with how state evolves. Because software should recognize activities and is partially observable, model based agent is suitable.

Q2)

I will prove every consistent heuristic is also admissible by induction.

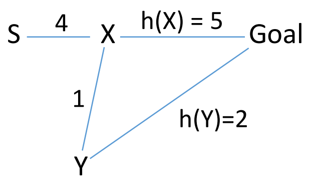


Let's assume real cost between (N+1) – N nodes is 1, N-G nodes is m and N-P nodes is n.
Consistency denotes,
$h(N) \leq h(P) + n$

Admissibility denotes,
$h(N) \leq m$

If h function is consistent we can say that $h(N+1) \leq h(N) + 1$

If we assume consistent heuristic is also admissible, we can write h(N) ≤ m so we can write
h(N+1) ≤ h(N) + 1 ≤ m + 1
We can found h(N+1) ≤ m +1 shows us every consistent heuristic is also admissible.



As seen in the graph, going to Goal from X node cost 4 + h(X) = 9
But going to Goal from Y node cost 4 + 1 + h(Y) = 7
But for consistency, it should be h(X) ≤ h(Y) + cost (X, Y)

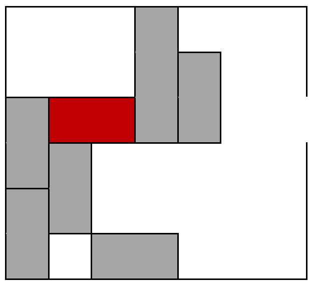
So this heuristic is not consistent but it is admissible.

Q3)

a)

Well defined form of a problem includes initial state, actions, goal test and path cost.

Initial State



Actions

- Move block up 1 square
- Move block down 1 square
- Move block left 1 square
- Move block right 1 square

Goal Test

- Is tail of the block reached (3,6) index in matrix?

Each block keeps its starting row and column number and also its size. So first we will calculate tail index by adding size to column index and then we will check if it is reached to 6.

Path Cost
Since we move blocks 1 square by 1 square, path cost is same and taken 1 for each move.

b)

For implementation of the problem, I created 3 classes. Block class, node class and puzzle class. Block class has 4 entities row, column, size and position. Node class has a block vector, an empty space matrix for keeping track of the empty spaces in the board, a pointer to successor and a vector for child nodes. Puzzle is a class for root node and search functions.

I started by writing canMoveLeft, canMoveRight, canMoveUp, canMoveDown and newChildren functions. canMove functions controls a block's movement possibilities. NewChildren creates all possible children for a given node by using canMove functions.

In my implementation, matrix index spans between 0-5 numbers and block coordinates kept according to upper left corner of the block. So my ending criteria was to get 5 when I add column index and size of a block. Also I choose the block which we try to take out as 1st block of a node block vector.

BFS is a complete search so we can say BFS guarantees a solution for puzzle on the contrary to DFS. But with this feature, it consumes memory largely. BFS's space complexity is really high according to DFS.

My implementation looks next nodes square by square. Like the one in the example. Because of this reason, it takes quite a time to find solution. Bu it finds it eventually for BFS. DFS takes more time than this, so I could not find any solution for it.

BFS
Generated Nodes: 705444
Expanded Nodes: 89610

c)