CS156 Module 2 Week 3 Homework Assignment 1 (10 pts)

CHAPTER 2

1. Let us examine the rationality of various vacuum-cleaner agent functions.
   1. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational.

Answer:

In order to show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational, we need to understand the concepts of rationality and how the agent's function adheres to rational decision-making.

Rational Agent : **Rationality** in the context of artificial intelligence and agents refers to the ability of an agent to make decisions that maximize its expected performance measure, given its knowledge and goals. Rational agents make choices that are expected to lead to better outcomes.

Evaluation of the simple vacuum-agent:

1. Performance measure: defined as one point for each clean square at each time step, over 1000 time steps in a lifetime. Goal of agent : maximize the total number of clean squares over its lifetime.
2. Environment : “Geography” known beforehand. Clean square -> stay clean. Suck -> cleans current square. Left -> move left. Right -> move right. Except when agent is outside environment, where it remains as is.
3. Sensors: senses current square condition i.e., dirt presence and location.
4. Actuators: move left, move right, suck

In conclusion, based on the provided information and assumptions, this simple vacuum-cleaner agent is rational. It effectively maximizes its performance measure (total clean squares) by perceiving the environment, making decisions based on that perception, and taking actions that align with its goal. It may be a basic agent, but within the specified environment and task, it acts rationally to achieve its objective.

* 1. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?

Answer:

An example of a rational agent function for the case in which each movement costs one point is a grid world problem like a vacuum world.

**INITIAL STATE:** can be any state   
**ACTIONS: There can be 3 actions in two-cell world:** *Suck*, move *Left*, and move *Right*. Whereas in a two-dimensional multi-cell world we need more movement actions: *Upward* and *Downward*, giving us four **absolute** movement actions, or we could switch to **egocentric actions**: defined relative to the viewpoint of the agent-*Forward, Backward, TurnRight*, and *TurnLeft*.  
**TRANSITION MODEL:** In the transition model, "Suck" removes dirt from the agent's cell, and "Forward" advances the agent by one cell in its current direction, with no effect if it encounters a wall. *Backward* moves the agent in the opposite direction, while *TurnRight* and *TurnLeft* change the direction it is facing by 90 degrees.  
**GOAL STATES:** The states in which every cell is clean.  
**ACTION COST:** Each action costs 1.

(OR) we can represent agent function making some modifications to 1 (a) as below:

Rational Agent Function:

1. Performance Measure: The performance measure remains the same: one point for each clean square at each time step, over a "lifetime" of 1000 time steps. However, in this case, it's essential to subtract one point for each movement made by the agent since each movement costs one point.
2. Environment: “Geography” known beforehand. Clean square -> stay clean. Suck -> cleans current square. Left -> move left. Right -> move right. Except when agent is outside environment, where it remains as is.
3. Sensors: senses current square condition i.e., dirt presence and location.
4. Actuators: move left, move right, suck

Internal State:

In this case, the agent program does require some internal state. Specifically, the agent needs to keep track of its current location, the locations it has visited, and the locations that are still dirty. This internal state allows the agent to make informed decisions about where to move and when to clean, considering its goal of maximizing the performance measure while minimizing the cost of movements.

* 1. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?

Answer:

In cases where clean squares can become dirty, and the geography of the environment is unknown, designing effective agents becomes more challenging. In such dynamic and uncertain environments, learning from experience can be a valuable approach.

1. Learning Agents:
   * 1. Agents can employ machine learning techniques, such as reinforcement learning, to adapt to changing environments.
     2. The agent can learn a policy that maximizes its expected reward over time, considering the possibility of previously clean squares becoming dirty.
     3. It can also learn a value function to estimate the desirability of visiting different locations.
2. Exploration Strategies:
   * 1. In an unknown environment, the agent can employ exploration strategies to learn about the environment's layout and cleanliness dynamics.
     2. For example, the agent can initially explore randomly and update its internal map of the environment as it discovers new locations and dirt patterns.

Learning from experience is beneficial in these cases because the agent's knowledge of the environment is incomplete and dynamic. Learning allows the agent to adapt to changing conditions and make more informed decisions.

The agent should learn several key aspects:

1. **Environmental Dynamics**: It should learn how quickly clean squares become dirty, the likelihood of different locations becoming dirty, and how often the environment changes.
2. **Spatial Mapping**: The agent should build a map or model of the environment, marking the locations it has visited, their cleanliness status, and their proximity to each other.
3. **Cleaning Policies**: It should learn effective cleaning policies that prioritize squares based on their dirtiness and the likelihood of becoming dirty again.
4. **Exploration Strategies**: The agent should develop strategies for efficient exploration, focusing on areas of uncertainty and novelty.
5. For each of the following assertions, say whether it is true or false and support your answer with examples or counter examples where appropriate.
   1. An agent that senses only partial information about the state cannot be perfectly rational.

Answer: This statement is generally False as perfect rationality refers to the ability to make good decisions given the sensor information received.

Example:

An agent, even if lacking sensors, could be considered rational when its sole performance metric is to remain entirely stationary within an environment devoid of other agents, although such an agent may be deemed unremarkable due to its limited objectives.

* 1. There exist task environments in which no pure reflex agent can behave rationally.

Answer: This statement is generally True as pure reflex agent has lack of history, inability to plan and limited sensing.

Example: In a game of chess, a pure reflex agent that makes moves based solely on the current board configuration and without considering the opponent's past moves or any future consequences would not behave rationally. Chess requires strategic thinking and planning multiple moves ahead.

* 1. There exists a task environment in which every agent is rational.

Answer: This statement is generally True in relatively simple, well-defined, and cooperative tsk environments.

Example 1: In an environment where a light switch has only one position (on or off), any agent choosing the correct position based on whether they want the light on or off is behaving rationally. Here there is just a. Single path to reach desired optimal goal.

Example 2: In a known and deterministic environment like in a simple board game with no hidden information, the optimal moves are known in advance, and any agent following those moves is behaving rationally.

* 1. The input to an agent program is the same as the input to the agent function.

Answer: This statement is False as the agent function receives the complete sequence of percepts up to a specific point as input, while the agent program is provided with the sensory input perceived by the sensors.

Example: the agent program may preprocess or filter the raw sensory input before passing it to the agent function. For example, it might perform noise reduction or feature extraction to improve the quality of input data.

* 1. Every agent function is implementable by some program/machine combination.

Answer: This statement is False as feasibility and practicality of such implementations can vary widely depending on the specific function, the chosen technology stack, and the constraints of the real-world environment in which the agent operates.

Example: Extremely complex agent functions may require significant computational resources, and there may be practical limitations on the available hardware or execution time

* 1. Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.

Answer: This statement is True.

Example: Imagine a competitive, deterministic multi-agent game environment. Now, think about an agent whose success is determined by its capacity to remain unpredictable to its adversaries. In this context, an agent that uniformly selects actions from its available set of options would be highly rational, as it becomes challenging for other agents to forecast its actions.

* 1. It is possible for a given agent to be perfectly rational in two distinct task environments.

Answer: This statement is True.

Example: Setting aside the unattainability of perfect rationality, let's examine an agent whose performance measure hinges on two factors: evading pursuing enemies and sustaining prolonged periods underwater. Assume this agent is equipped with sensors and actuators tailored to excel in these areas without causing interference or harm. Now, let's explore two distinct task environments: one characterized by competitive interactions among multiple agents, where physical means are the sole method to outmaneuver opponents (excluding long-range weapons), and the other environment is a competition focused on endurance in underwater conditions. In both of these environments, the aforementioned agent would demonstrate perfect rationality.

* 1. Every agent is rational in an unobservable environment.

Answer: This statement is True.

Example: Let's contemplate an agent placed within an environment that cannot be directly observed. This agent's performance measure assigns secondary importance to avoiding jammed actuator-toes on bike cleats and primary importance to the ability to exit the current room. It's important to note that the priority lies in leaving the room rather than preventing toe injuries. Additionally, assume that the room contains bike cleats, and while the agent knows they exist, their exact locations remain unknown. In this scenario, the agent will strive to exit the room but will inevitably encounter and jam its actuator-toe on a bike cleat, demonstrating irrational behavior.

* 1. A perfectly rational poker-playing agent never loses.

Answer: This statement is False.

Example: Given the partially observable, multi-agent, stochastic, sequential, static, and discrete nature of poker, it's important to acknowledge that even a perfectly rational agent would face challenges in winning every game, unless it possesses complete omniscience. The complexity of the task environment involves numerous factors beyond the agent's control. However, it's worth noting that the agent could adopt a learning approach to enhance its playing ability, but it's likely that it may still experience losses.

1. For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Section 2.3.2.
   1. Playing soccer.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Playing soccer | Score goals, zero penalties, don’t allow the other team to score | Field for soccer, players, referee, coach, ball | Legs for kicking ball, running, hands, head | Eyes, ears |

* 1. Exploring the subsurface oceans of Titan.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Exploring the subsurface oceans of Titan | Accurately mapping oceans and identifying environment | Water, submersible | Propeller, body to swim | Camera, sonar, motor sensor |

* 1. Shopping for used AI books on the Internet.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Shopping for used AI books on the Internet | Low price cost in getting correct edition of the required book | Internet, shopping sites, customers | Computer to search the internet | Monitor displaying result |

* 1. Playing a tennis match.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Playing a tennis match | Win by getting highest score, don’t allow other side to score, zero penalties | Tennis ground, racket, referee, players, ball | person | Eyes, ears |

* 1. Practicing tennis against a wall.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Practicing tennis against a wall | trying to hit as many shots as possible to the wall | Racket, player, wall, ball | person | Eyes, ears |

* 1. Performing a high jump.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Performing a high jump | Jump to maximum height | Pole, referee, player, padding, jumper | person | NA (as there is no need of any, one can just follow the path and jump off. Single task) |

* 1. Knitting a sweater.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Knitting a sweater | Completely create a sweater | Needles, yarn, knitter | hands | eyes |

* 1. Bidding on an item at an auction.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Agent Type | Performance Measure | Environment | Actuators | Sensors |
| Bidding on an item at an auction | Get item at low cost | Bidders, items, auctioneer | voice | Eyes, ears |

1. This exercise explores the differences between agent functions and agent programs.
   1. Can there be more than one agent program that implements a given agent function? Give an example, or show why one is not possible.

Answer: Yes, there can be more than one agent program that implements a given agent function. For example, the navigation function of an agent can be realized through agent programs that employ stored tables as an approach, as opposed to using a map along with a general-purpose search.

* 1. Are there agent functions that cannot be implemented by any agent program?

Answer: Yes, there are agent functions that cannot be implemented by an agent program. Example: halting problem.

* 1. Given a fixed machine architecture, does each agent program implement exactly one agent function?

Answer: Yes, as long as it operates as a single thread and remains deterministic in nature.

* 1. Given an architecture with n bits of storage, how many different possible agent programs are there?

Answer: 2^n possible programs will be there for an architecture with n bits of storage.

* 1. Suppose we keep the agent program fixed but speed up the machine by a factor of two. Does that change the agent function?

Answer: No, speeding up the machine doesn’t affect agent function because the environment din’t change.

1. Write pseudocode agent programs for the goal-based and utility-based agents.

Answer:

Goal based agent:

function GOAL\_BASED\_AGENT(percept) **returns** an action

**persistent**:

*state*, the agent’s current conception of the world state

*goal*, a description of what the agent would like to achieve

*rules*, a set of condition-action rules

*action*, the most recent action, initially none

state <- UPDATE\_STATE (state, action, percept, goal)

rule <- RULE\_MATCH (state, rules, goal)

action <- rule.ACTION

**return** action

Utility based agent:

function UTILITY\_BASED\_AGENT(percept) **returns** an action

**persistent**:

*state*, the agent’s current conception of the world state

*possible states*, possible states that may maximize happiness

*rules*, a set of condition-action rules

*action*, the most recent action, initially none

state <- UPDATE\_STATE (state, action, percept, possible states)

rule <- RULE\_MATCH (state, rules, possible states)

action <- rule.ACTION

**return** action

CHAPTER 3:

**3.1** Explain why problem formulation must follow goal formulation.

Answer: Problem formulation should follow goal formulation because setting clear goals narrows down the problem's scope, making it solvable within a reasonable timeframe. Without a defined goal, an agent may face an overwhelming number of choices, making decision-making challenging.

For instance, imagine an agent without a specific goal, like sitting at a desk with a computer and a textbook. In this scenario, it could choose from a myriad of actions, such as browsing the web, playing games, reading random textbook pages, sleeping, or going outside. However, if the agent has a specific goal, like completing chapter 6, problems 1 to 10, from the textbook using the computer, its available choices become much more focused. Any action not contributing to the completion of these specific problems is immediately discarded.

Consequently, the agent is left with actions solely aimed at achieving its assignment.

In the absence of goal formulation preceding problem formulation, finding a solution becomes considerably more challenging, as there is no guiding objective to direct the agent's actions.

**3.2** Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall.

a. Formulate this problem. How large is the state space?

Answer: In this scenario, the problem begins with the robot positioned at the maze's center, initially facing north. Controlled by a user, the robot can orient itself in four directions: north, south, east, or west, and it can advance a specified distance. However, it's programmed to halt if it approaches a wall to prevent collisions. Given these capabilities, the robot can move a certain distance in any of these directions, either stopping when it reaches a wall or continuing through the maze. The robot's actions are limited to five possibilities: facing north, facing south, facing east, facing west, or moving. It can move in its current direction unless obstructed by a wall, in which case it stops. Thus, there are two potential outcomes for each action: either the robot advances in its current direction, or it stops due to a wall obstacle. With four possible movement directions, in a location of size 'n,' there exist '4n' possible world states. If n is infinite, there will be infinite possibilities.

Formulating the problem:

Initial Condition: The robot begins at the center of the maze, oriented to the north.

Successor Function: In the current state, the robot can adjust its orientation to face a new direction,: ’n’ (north), 's' (south), 'e' (east), or 'w' (west).

State Space: Considering there are 'n' potential locations within the maze and the robot can move in four possible directions, there exist a total of '4n' conceivable world states.

Available Actions: The robot possesses five available actions: facing north, facing south, facing east, facing west, and moving forward.

Objective: The goal is to reach any point outside of the maze

b. In navigating a maze, the only place we need to turn is at the intersection of two or more corridors. Reformulate this problem using this observation. How large is the state space now?

Answer: If the intersections within the corridors constitute a portion of the state's 'n' locations (specifically, 'n – i'), and there remain only two potential outcomes for an action taken at any state ('2(n – i)'), while the four directional actions of the robot are confined solely to intersections (meaning the prior state space of '4n' now becomes '4i'), then the revised state space equates to '4i + 2(n – i)'.

To elaborate:

Given that the robot exclusively alters its direction at corridor intersections (represented as 'i') rather than at any point within the state (as denoted by 'n'), the state space, previously '4n' in Part A, is now redefined as '4i'.

Furthermore, as the intersections evidently form a subset of the state's locations, 'i' is

inherently encompassed within 'n'. Additionally, since the robot's directional changes occur solely at 'i,' it follows that the robot is in motion at all other locations within 'n' except for 'i,' hence the term 'n – i.' Lastly, the only two possible movement outcomes of the robot (moving forward or stopping upon encountering a wall) are incorporated, resulting in the total state space being '4i + 2(n – i).'

**c.** From each point in the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot’s orientation now?

Answer: If the robot gains autonomous movement capability in any direction, and the only remaining action to be considered is turning, then the elements '4' and '2(n-i)' previously included in the state space can be omitted. This is because '4' represented the four directions in which the robot could move, and '2(n-i)' represented the locations within the state where the robot moved and the potential outcomes of those movements—all of which are no longer relevant in this part of the problem. Consequently, the sole representation required for the state space is 'i.' Additionally, as a result of excluding these actions, there is no longer a need to track the robot's orientation, as it autonomously performs all actions without user intervention.

**d.**  In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

Answer**:**

* One initial observation I made pertains to the absence of information regarding the robot's design. It appears to have the capability to make turns but lacks the ability to move diagonally, restricting its movement to straight lines in four distinct directions.
* Additionally, there is a notable absence of details concerning the maze's layout. Should the maze contain dead ends, the robot could become immobilized since it can only execute turns at intersections.
* Furthermore, it's worth noting that this problem assumes flawless sensor functionality, implying that the robot will never encounter issues or collisions with walls

**3.3** Suppose two friends live in different cities on a map, such as the Romania map shown in Figure 3.1. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city i to neighbor j is equal to the road distance d(i, j) between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin.

We want the two friends to meet as quickly as possible.

**a**. Write a detailed formulation for this search problem. (You will find it helpful to define some formal notation here.)

Answer:

This search problem comprises the following elements:

**Goal**: To place both individuals, denoted as p1 and p2, within the same city, denoted as i, while minimizing the total time T.

**Definitions**: Distances between cities are represented by dist(i, j), defined as the straight-line distance between two cities i and j.

Problem:

**States**: The pair of cities, i and j, representing the current locations of p1 and p2, respectively, with the possibility that i and j may refer to the same city.

**Actions**: Relocating p1 to city i' (i-prime) and p2 to city j' (j-prime), resulting in an increase in T determined by the maximum of dist(i, i') and dist(j, j').

Solution:

The solution consists of two equal-length sequences of cities, one for p1 and one for p2, in the format i -> j -> ... -> k -> l -> m and a -> b -> ... -> c -> d -> m. The final city in each sequence must be the same, and the sequences must not share any other cities at the same index. Furthermore, the solution aims to minimize the following function:

Distance Calculation:

Calculate the distances traveled by p1 and p2 in sequence 1 (ds1) and sequence 2 (ds2), both of length n.

* 1. • Determine the largest distance for each position using the map and max functions applied to the zipped ds1 and ds2.

This approach ensures that both individuals are in the same location while minimizing the overall distance traveled, resulting in an optimal solution.

**b**. Let D(i, j) be the straight-line distance between cities i and j. Which of the following heuristic functions are admissible? (i) D(i, j); (ii) 2 ・ D(i, j); (iii) D(i, j)/2.

Answer: The heuristic D(i, j)/2 and D(i,j) are considered admissible. In an ideal scenario, both individuals advance towards each other at an equal pace, effectively halving the distance reduction compared to a situation where one of them exclusively travels to the other. Consequently, as the solution aligns more closely with this heuristic, it tends to become increasingly optimal. Also, straight line distance is always an admissible heuristic as it never overestimates actual cost of traversal.

**c**. Are there completely connected maps for which no solution exists?

**Answer:** In the case of completely connected maps, some situations may have no viable solution. The simplest example is a two-node graph, depicted as:

(P1)-------(P2)

As both individuals must move simultaneously, they end up merely swapping places and can never converge to the same city:

(P1)-------(P2)

|

V

(P2)-------(P1)

|

V

(P1)-------(P2)

In this scenario, they continuously exchange positions without ever reaching the same city.

**d**. Are there maps in which all solutions require one friend to visit the same city twice?

Answer: Indeed, in certain specific scenarios, all feasible solutions necessitate one person revisiting the same node, as exemplified below:

(A)---( )---( )---( )---( )---( )---( )---(C)---(B)---( )---( )

\ /

\ /

( ) ( )

\ /

( )

In this case, A and B can meet at C, but this entails B circumventing the loop and returning to B's starting point. Any other potential solutions, even if they involve more steps, would either involve retracing one's path or require A to follow a similar course and return to its initial entry point within the loop.

**3.4** Show that the 8-puzzle states are divided into two disjoint sets, such that any state is reachable from any other state in the same set, while no state is reachable from any state in the other set. (*Hint:* See Berlekamp *et al.* (1982).) Devise a procedure to decide which set a given state is in, and explain why this is useful for generating random states.

Answer: The 8-puzzle problem is a classic puzzle involving a 3x3 grid of numbered tiles, with one tile missing. The objective is to rearrange the tiles from an initial state to a goal state by sliding them one at a time into the empty space. In this puzzle, it is indeed possible to divide the states into two disjoint sets with specific properties.

To do this, we can classify states based on the parity of their inversions. An inversion occurs when a tile precedes another tile with a lower number, but they are in the reverse order of their goal positions. Specifically, let's denote the two sets as "even permutation states" and "odd permutation states."

Even Permutation States:

In these states, the number of inversions is even. This means that it is possible to reach any state within this set from any other state within the same set through a series of legal moves. The reason for this is that the number of inversions directly affects the parity of the permutation, and states with the same parity can be transformed into each other. This property can be proved mathematically.

Odd Permutation States:

Conversely, in these states, the number of inversions is odd. Just like in the even permutation states, it is possible to reach any state within this set from any other state within the same set. However, states in the odd permutation set cannot be transformed into states in the even permutation set (and vice versa) because the parity of the permutations is different.

To decide which set a given state is in, you can count the number of inversions in that state. If the number of inversions is even, the state belongs to the even permutation set; if it's odd, the state belongs to the odd permutation set.

Why is this useful for generating random states?

This property is valuable for generating random states in the 8-puzzle because it ensures that you can manipulate any state within one set to reach any other state in the same set without crossing over to the other set. It makes the random state generation process more predictable and manageable. For example, if you want to generate a random solvable state, you can start with a solved state (which belongs to the even permutation set) and perform a series of random legal moves, ensuring that the resulting state remains in the same set. This way, you can guarantee that the generated state has a solution and is not trapped in a different permutation set where a solution is impossible.

The conventional representation of the 8-puzzle can be outlined as follows:

**States**: A state description precisely defines the position of each tile within the puzzle.

**Initial State**: Any state can be chosen as the initial state. It's noteworthy that a parity property segregates the state space—each possible goal can be attained from precisely half of the potential initial states.

**Actions**: In the abstract sense, an action is the movement of the empty space Left, Right, Up, or Down, as if it were the blank tile that moves. Keep in mind that not all actions are applicable if the blank space is situated at an edge or corner.

**Transition Model**: This model maps a state and an action to the resulting state; for example, applying the "Left" action to the initial state causes the positions of the "5" tile and the blank space to switch, as illustrated in Figure 3.3.

**Goal State**: Although any state could theoretically serve as the goal, it's customary to specify a state in which the numbers are arranged in ascending order.

**Action Cost**: Each action incurs a cost of 1.

REF

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). Winning Ways, For Your Mathematical Plays. Academic Press.