





**BACHELOR IN INFORMATION TECHNOLOGY**

First Assignment on

Artificial Intelligence

Submitted by: Submitted to:

Name: Manish Lama Name: Saroj Poudel

Year/ Semester: 3rd /5th Semester

LCID: 00017000524

**Date: 28-04-2021**

1. **What is an intelligent agent? Describe various type of task environments with examples.**

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals.

(GeeskfromGreeks, 2021, p. Javatpint)

Type of task environments with examples are:-

* Fully Observable vs. Partially Observable

A fully observable AI environment has access to all required information to complete target task.

Partially observable environments such as the ones encountered in self-driving vehicle scenarios deal with partial information in order to solve AI problems

Example of fully observable environment is Image recognition and partially observable environment is self-driving car. (jrodthoughts., 2021)

* Competitive vs. Collaborative

Competitive AI environments face AI agents against each other in order to optimize a specific outcome.

Example of competitive AI environments is Games such as GO or Chess.

Collaborative AI environments rely on the cooperation between multiple AI agents.

Examples of collaborative AI environments is Self-driving vehicles or cooperating to avoid collisions or smart home sensors interactions. (jrodthoughts., 2021)

* Deterministic vs. Non-deterministic

Deterministic AI environments are those on which the outcome can be determined base on a specific state. In other words, deterministic environments ignore uncertainty.

Example of deterministic is chess.

Most real world AI environments are not deterministic. Instead, they can be classified as stochastic.

Example of non-deterministic is self-driving car. (jrodthoughts., 2021)

* Static vs. Dynamic

static AI environments rely on data-knowledge sources that don’t change frequently over time.

Example of static is crossword puzzle.

Contrasting with that model, dynamic AI environments such as the vision AI systems in drones deal with data sources that change quite frequently.

Example of dynamic is self-driving car. (jrodthoughts., 2021)

* Discrete vs. Continuous

Discrete AI environments are those on which a finite [set of possibilities can drive the final outcome of the task.

Example of Discrete is Chess.

Continuous AI environments rely on unknown and rapidly changing data sources.

Example of Continuous is Vision systems in drones. (jrodthoughts., 2021)

* **Complete vs. Incomplete**

Complete AI environments are those on which, at any give time, we have enough information to complete a branch of the problem.

Example of Complete is Chess.

AI strategies can’t anticipate many moves in advance and, instead, they focus on finding a good ‘equilibrium” at any given time.

Example of Incomplete is Poker. (jrodthoughts., 2021)

1. **What is meant by admissible heuristic? What improvement is done in A\* search than greedy search? Prove that A\* search gives us optimal solution if the heuristic function is admissible.**

An admissible heuristic is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state. (GeeskfromGreeks, 2021)

Improvement done in A\* search than greedy search

* A\* is complete, optimal, and it has a time and space complexity of (𝑏𝑚)O(bm).
* A\* uses more memory than greedy BFS. (Exhange, 2021)

Prove:

It is a best-first search that uses the evaluation function:

f(n) = g(n) + h(n)

Here,

1. Since nodes are expanded in ascending order of 𝑓(𝑛)f(n) you know that no other node is more promising than the current one. Remember: ℎ(𝑛)h(n) is admissible so that having the lowest 𝑓(𝑛)f(n) means that it has an opportunity to reach the goal through a cheaper path that the other nodes in OPEN have not. And this is true unless you can prove the opposite, i.e., by expanding the current node.
2. Since 𝐴∗A∗ stops only when it proceeds to expand the goal node (as oppossed to stop when generating it) you are sure (from the first point above) that no other node leads through a cheaper path to it.

(GeeskfromGreeks, 2021)

1. **What is an intelligent agent? Briefly describe the type of agents with examples.**

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals.

(GeeskfromGreeks, 2021, p. Javatpint)

Type of agent environments

* Simple Reflex Agent

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived till date.

For example: Thermometer. It indicates the temperature of the current situation and does not rely on the past situation. (GeeskfromGreeks, 2021)

* Model-based Reflex Agent

It works by finding a rule whose condition matches the current situation. A model-based agent can handle **partially observable environments** by use of model about the world.

For example:  the model-based agent uses GPS to understand its location and predict upcoming drivers. (GeeskfromGreeks, 2021)

* Goal-based Agent

These kind of agents take decision based on how far they are currently from their **goal**(description of desirable situations). Their every action is intended to reduce its distance from the goal.

For example: A goal based drone drops the product when it reaches the customer's location. (GeeskfromGreeks, 2021)

* Learning Agent

A learning agent in AI is the type of agent which can learn from its past experiences or it has learning capabilities.  
It starts to act with basic knowledge and then able to act and adapt automatically through learning.

For example: home thermostat, when temperature of house reaches to certain point, it starts heating or cooling. (GeeskfromGreeks, 2021)

1. **Write a python program to solve 8 puzzle problem with the algorithm of your choice and explain why did you choose the algorithm?**

|  |
| --- |
| from copy import deepcopy |
|  | import numpy as np |
|  | import time |
|  |  |
|  | # takes the input of current states and evaluvates the best path to goal state |
|  | def bestsolution(state): |
|  | bestsol = np.array([], int).reshape(-1, 9) |
|  | count = len(state) - 1 |
|  | while count != -1: |
|  | bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0) |
|  | count = (state[count]['parent']) |
|  | return bestsol.reshape(-1, 3, 3) |
|  |  |
|  |  |
|  | # this function checks for the uniqueness of the iteration(it) state, weather it has been previously traversed or not. |
|  | def all(checkarray): |
|  | set=[] |
|  | for it in set: |
|  | for checkarray in it: |
|  | return 1 |
|  | else: |
|  | return 0 |
|  |  |
|  |  |
|  | # calculate Manhattan distance cost between each digit of puzzle(start state) and the goal state |
|  | def manhattan(puzzle, goal): |
|  | a = abs(puzzle // 3 - goal // 3) |
|  | b = abs(puzzle % 3 - goal % 3) |
|  | mhcost = a + b |
|  | return sum(mhcost[1:]) |
|  |  |
|  |  |
|  |  |
|  |  |
|  | # will calcuates the number of misplaced tiles in the current state as compared to the goal state |
|  | def misplaced\_tiles(puzzle,goal): |
|  | mscost = np.sum(puzzle != goal) - 1 |
|  | return mscost if mscost > 0 else 0 |
|  |  |
|  |  |
|  |  |
|  | #3[on\_true] if [expression] else [on\_false] |
|  |  |
|  |  |
|  | # will indentify the coordinates of each of goal or initial state values |
|  | def coordinates(puzzle): |
|  | pos = np.array(range(9)) |
|  | for p, q in enumerate(puzzle): |
|  | pos[q] = p |
|  | return pos |
|  |  |
|  |  |
|  |  |
|  | # start of 8 puzzle evaluvation, using Manhattan heuristics |
|  | def evaluvate(puzzle, goal): |
|  | steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8], 3),('left', [0, 3, 6], -1),('right', [2, 5, 8], 1)], |
|  | dtype = [('move', str, 1),('position', list),('head', int)]) |
|  |  |
|  | dtstate = [('puzzle', list),('parent', int),('gn', int),('hn', int)] |
|  |  |
|  | # initializing the parent, gn and hn, where hn is manhattan distance function call |
|  | costg = coordinates(goal) |
|  | parent = -1 |
|  | gn = 0 |
|  | hn = manhattan(coordinates(puzzle), costg) |
|  | state = np.array([(puzzle, parent, gn, hn)], dtstate) |
|  |  |
|  | # We make use of priority queues with position as keys and fn as value. |
|  | dtpriority = [('position', int),('fn', int)] |
|  | priority = np.array( [(0, hn)], dtpriority) |
|  |  |
|  |  |
|  |  |
|  | while 1: |
|  | priority = np.sort(priority, kind='mergesort', order=['fn', 'position']) |
|  | position, fn = priority[0] |
|  | priority = np.delete(priority, 0, 0) |
|  | # sort priority queue using merge sort,the first element is picked for exploring remove from queue what we are exploring |
|  | puzzle, parent, gn, hn = state[position] |
|  | puzzle = np.array(puzzle) |
|  | # Identify the blank square in input |
|  | blank = int(np.where(puzzle == 0)[0]) |
|  | gn = gn + 1 |
|  | c = 1 |
|  | start\_time = time.time() |
|  | for s in steps: |
|  | c = c + 1 |
|  | if blank not in s['position']: |
|  | # generate new state as copy of current |
|  | openstates = deepcopy(puzzle) |
|  | openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank] |
|  | # The all function is called, if the node has been previously explored or not |
|  | if ~(np.all(list(state['puzzle']) == openstates, 1)).any(): |
|  | end\_time = time.time() |
|  | if (( end\_time - start\_time ) > 2): |
|  | print(" The 8 puzzle is unsolvable ! \n") |
|  | exit |
|  | # calls the manhattan function to calcuate the cost |
|  | hn = manhattan(coordinates(openstates), costg) |
|  | # generate and add new state in the list |
|  | q = np.array([(openstates, position, gn, hn)], dtstate) |
|  | state = np.append(state, q, 0) |
|  | # f(n) is the sum of cost to reach node and the cost to rech fromt he node to the goal state |
|  | fn = gn + hn |
|  |  |
|  | q = np.array([(len(state) - 1, fn)], dtpriority) |
|  | priority = np.append(priority, q, 0) |
|  | # Checking if the node in openstates are matching the goal state. |
|  | if np.array\_equal(openstates, goal): |
|  | print(' The 8 puzzle is solvable ! \n') |
|  | return state, len(priority) |
|  |  |
|  |  |
|  | return state, len(priority) |
|  |  |
|  |  |
|  | # start of 8 puzzle evaluvation, using Misplaced tiles heuristics |
|  | def evaluvate\_misplaced(puzzle, goal): |
|  | steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8], 3),('left', [0, 3, 6], -1),('right', [2, 5, 8], 1)], |
|  | dtype = [('move', str, 1),('position', list),('head', int)]) |
|  |  |
|  | dtstate = [('puzzle', list),('parent', int),('gn', int),('hn', int)] |
|  |  |
|  | costg = coordinates(goal) |
|  | # initializing the parent, gn and hn, where hn is misplaced\_tiles function call |
|  | parent = -1 |
|  | gn = 0 |
|  | hn = misplaced\_tiles(coordinates(puzzle), costg) |
|  | state = np.array([(puzzle, parent, gn, hn)], dtstate) |
|  |  |
|  | # We make use of priority queues with position as keys and fn as value. |
|  | dtpriority = [('position', int),('fn', int)] |
|  |  |
|  | priority = np.array([(0, hn)], dtpriority) |
|  |  |
|  | while 1: |
|  | priority = np.sort(priority, kind='mergesort', order=['fn', 'position']) |
|  | position, fn = priority[0] |
|  | # sort priority queue using merge sort,the first element is picked for exploring. |
|  | priority = np.delete(priority, 0, 0) |
|  | puzzle, parent, gn, hn = state[position] |
|  | puzzle = np.array(puzzle) |
|  | # Identify the blank square in input |
|  | blank = int(np.where(puzzle == 0)[0]) |
|  | # Increase cost g(n) by 1 |
|  | gn = gn + 1 |
|  | c = 1 |
|  | start\_time = time.time() |
|  | for s in steps: |
|  | c = c + 1 |
|  | if blank not in s['position']: |
|  | # generate new state as copy of current |
|  | openstates = deepcopy(puzzle) |
|  | openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank] |
|  | # The check function is called, if the node has been previously explored or not. |
|  | if ~(np.all(list(state['puzzle']) == openstates, 1)).any(): |
|  | end\_time = time.time() |
|  | if (( end\_time - start\_time ) > 2): |
|  | print(" The 8 puzzle is unsolvable \n") |
|  | break |
|  | # calls the Misplaced\_tiles function to calcuate the cost |
|  | hn = misplaced\_tiles(coordinates(openstates), costg) |
|  | # generate and add new state in the list |
|  | q = np.array([(openstates, position, gn, hn)], dtstate) |
|  | state = np.append(state, q, 0) |
|  | # f(n) is the sum of cost to reach node and the cost to rech fromt he node to the goal state |
|  | fn = gn + hn |
|  |  |
|  | q = np.array([(len(state) - 1, fn)], dtpriority) |
|  | priority = np.append(priority, q, 0) |
|  | # Checking if the node in openstates are matching the goal state. |
|  | if np.array\_equal(openstates, goal): |
|  | print(' The 8 puzzle is solvable \n') |
|  | return state, len(priority) |
|  |  |
|  | return state, len(priority) |
|  |  |
|  |  |
|  |  |
|  | # ---------- Program start ----------------- |
|  |  |
|  |  |
|  | # User input for initial state |
|  | puzzle = [] |
|  | print(" Input vals from 0-8 for start state ") |
|  | for i in range(0,9): |
|  | x = int(input("enter vals :")) |
|  | puzzle.append(x) |
|  |  |
|  | # User input of goal state |
|  | goal = [] |
|  | print(" Input vals from 0-8 for goal state ") |
|  | for i in range(0,9): |
|  | x = int(input("Enter vals :")) |
|  | goal.append(x) |
|  |  |
|  |  |
|  |  |
|  | n = int(input("1. Manhattan distance \n2. Misplaced tiles")) |
|  |  |
|  | if(n ==1 ): |
|  | state, visited = evaluvate(puzzle, goal) |
|  | bestpath = bestsolution(state) |
|  | print(str(bestpath).replace('[', ' ').replace(']', '')) |
|  | totalmoves = len(bestpath) - 1 |
|  | print('Steps to reach goal:',totalmoves) |
|  | visit = len(state) - visited |
|  | print('Total nodes visited: ',visit, "\n") |
|  | print('Total generated:', len(state)) |
|  |  |
|  | if(n == 2): |
|  | state, visited = evaluvate\_misplaced(puzzle, goal) |
|  | bestpath = bestsolution(state) |
|  | print(str(bestpath).replace('[', ' ').replace(']', '')) |
|  | totalmoves = len(bestpath) - 1 |
|  | print('Steps to reach goal:',totalmoves) |
|  | visit = len(state) - visited |
|  | print('Total nodes visited: ',visit, "\n") |
|  | print('Total generated:', len(state)) |

|  |
| --- |
| (harshkv, 2021) |
|  |
| 1. **Design an Artificial Intelligent System with**    1. The problem you want to solve with its environment   Creating a Automated self-flying drone for product delivery.     * 1. PEAS description of your agent   **Performance: safety, trip, maximum distance**  **Environment: birds, airplane, poles, trees.**  **Actuator: break, propellers**  **Sensor: camera**   * 1. Description of search space   INITIAL STATES: From the distribution center.  ACTIONS: flying left, right, up, down, and rotate. Drop and Carry Product.  TRANSITION MODEL: If the location is assigned it flies towards its destination.  GOAL STATE: Reach to the customer.  ACTION COST: Each propeller rotate costs 1.   * 1. Description of the knowledge required and to use.   It should have the knowledge of the location, distance cover, time requires to reach the destination and distraction and obstacles to avoid during the trip. Drop the product when the destination is reached. (Poudel, how do we implement the system?, 2021)   * 1. Description on how you would plan to implement the system.   To implement this system, I will buy the hardware required to create a drone. Then I will assemble the drone and I will write the program to operate the drone. After that, I will do some testing on my own by placing a product on the drone and assign the location to my relative's home and see the self-flying drone delivering the product. (Poudel, how do we implement the system?, 2021) **References** Javapoint. Artificial Intelligence. kathmandu, 3, Nepal.  Javatpoint. *agent-environment-in-ai*. Retrieved from javatpoint.com: https://www.javatpoint.com/agent-environment-in-ai  GeeskfromGreeks (2021, 4 23). *A\*\_search\_algorithm*.  Retrieved from GitHubGist: https://gist.github.com/thiagopnts/8015876  Poudel, S. (2021, 4 18). how do we implement the system? kathmandu, 3, Nepal. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |