COMP3702 Artificial Intelligence (Semester 2, 2024)
Assignment 1: Search in BEEBOT – **Report Template**

Name: Harsha Joshi

Student ID: 45561623

Student email: s4556162@uq.edu.au

Note: Please edit the name, student ID number and student email to reflect your identity and **do not modify the design or the layout in the assignment template**, including changing the paging.

Question 1 (Complete your full answer to Question 1 on the remainder page 1)

Dimension	Value	Justification
Planning horizon	Indefinite Stage	From P & M's 1.5: "Planner is an agent that looks ahead some finite, but not predetermined, number of stages." BeeBot utilises the constants.py and state.py to look ahead for it's next moves and does this continuously for the duration the user plays the game.
Representation	States	From P & M's 1.5: "A state of the world specifies the agent's internal state (its belief state) and the environment state." BeeBot has a state.py and constants.py that determine the current and environment state.
Computational limits	Perfect Rationality	From P & M's 1.5: "Perfect rationality is where an agent reasons about the best action without taking into account its limited computational resources."BeeBot is assumed to always take the best action, given the local perimeters.
Learning	Knowledge is Given	From P & M's 1.5: "Learning means finding the best model that fits the data." BeeBot has a completely local environment that does not account for previous knowledge/data/experience. As such, each iteration is assumed to start fresh from the same local environment (environment.py).
Sensing uncertainty	Fully observable	From P & M's 1.5: "Fully observable means the agent knows the state of the world from the stimuli." BeeBot has given fixed inputs (state.py/constants.py) and performs differently in accordance with the search algorithm and heuristic implemented.
Effect uncertainty	Deterministic	From P & M's 1.5: "Deterministic when the state resulting from an action is determined by an action and the prior state." BeeBot utilises it's old position to determine the new one – most evident by the compute_heuristic function under solution.py.
Number of agents	Single agent	From P & M's 1.5: "Single agent reasoning means the agent assumes that there are no other agents in the environment or that all other agents are part of nature, and so are non-purposive." BeeBot only contains the bee that is moving and acting in an agent specific way, there are no other attributes reliant on this in the game.
Interactivity	Offline	From P & M's 1.5: "Offline reasoning is the computation done by the agent before it has to act". BeeBot does not require any online agents to determine its actions. The environment, states and constants given are all that is required for the bee (agent) to determine its movement.

Question 2 (Complete your full answer to Question 2 on page 2)

BeeBot is comprised of various components. It gains its functionality mostly from the testcases/ex1, 2, 3, 4, 5.txt, environment.py and state.py.

These are the following components of BeeBot (from Module 1, pg. 8):

- Action Space: The set of all possible actions the agent can perform. This is predominately found in the constants.py file, as it contains a list of all the actions that that the bee can do in the game.
- Percept Space: The set of all possible things the agent can perceive. This is predominately found in the environment.py file, as it contains all possible functionality that the bee can do – specifically perform_action(), is_solved() being notable answers.
- State Space: The set of all possible configurations of the world the agent is operating in. This can be found from state.py and the testcases/ex1, 2, 3, 4, 5.txt. In tandem, these files can help to define all the states the bee can operate in.
- Transition Function: A function that specifies how the configuration of the world changes when the agent performs actions in it. The Solver class under solution.py computes how the bee interacts with the world and describes various methods of that interaction (as in solve_ucs versus solve_a_star).
- Utility/Cost Function: A function that maps a state (or a sequence of states) to a real number, indicating how desirable it is for the agent to occupy that state/sequence of states. In this case, it is evident that solve_a_star's compute_heuristic would be the more optimal way of the bee (agent) interacting with this world and would be the ideal utility function. In part it calculates all possible paths and then optimises the appropriate one given the inputs provided. It aims to return the ideal cost using an admissible heuristic.

Question 3 (Complete your full answer to Question 3 on page 3)

Method	**Nodes Visited**	**Nodes Expanded**	**Nodes on Frontier**	**Run Time**
UCS – Test Case 1	1599	1759	34	0.038
A* – Test Case 1	1771	1633	20	0.038
UCS – Test Case 2	25142	26706	471	0.643
A* – Test Case 2	25216	27253	487	0.757
UCS – Test Case 3	1049688	1050690	15760	44.803
A* – Test Case 3	1076360	1080713	16211	54.861
UCS – Test Case 4	1936732	1938828	29082	83.906
A* – Test Case 4	1891568	1896400	28466	100.048
UCS – Test Case 5	6818932	6827467	102 412	323.114
A* – Test Case 5	7169625	7179319	107 689	408.245

The key difference between UCS and A* search algorithms, is that A* is function of the UCS in addition with a heuristic. Because of the heuristic, A* theoretically is expected to expand fewer nodes and provide a more optimal solution as compared to UCS. Though multiple heuristics can and should be added for A* for the best optimisation, in this implementation only the distance was accounted for between the bee and its target. As such, the table above is to be reflective of these conditions.

Theoretically, A* should visit lower number of explored states (nodes visited and nodes expanded), because the heuristic is to enable A* to avoid unnecessary paths that UCS may explore. Similarly, the runtime should also be reduced for A* in comparison to UCS, because of the lower number of states visited. A similar concept is applied to the nodes on the frontier for A* (Panchawate, 2023).

Looking at the table above, it is evident that there are certain instances where UCS performs slightly better or the same as A*. This is best summarised by the overall run time. Test Case 1 shows that both algorithms take about 0.038 seconds to search and compute the next state. Further Test Case 3's runtimes shows that A* performs worse than it's UCS counterpart with approximately 10 second difference. Additionally, this is evident as the number of nodes visited keeps increasing with each following test case.

Similarly, the nodes on frontier, nodes expanded, and nodes visited demonstrate that A* and UCS are quite efficient in computing the next position in terms of memory required to access the nodes. Ideally, A* should be accessing less nodes compared with UCS due to the heuristic however it is evident that there are instances where this does not happen such as Test Case 5. Particularly the most evident gap is between the nodes expanded, whereby A* is accessing more than 30 000 nodes as compared to UCS.

This demonstrates that A* though theoretically more computationally efficient, is quite similar to UCS for BeeBot. It is believed that this is primarily because of the choice of heuristic which will be elaborated more on in Question 4.

Question 4 (Complete your full answer to Question 4 on pages 4 and 5, and keep page 5 blank if you do not need it)

There was one heuristic applied to optimise the A* search algorithm: total_cost = abs(tgt[0] - int(state.BEE_posit[0])) + abs(tgt[1] - int(state.BEE_posit[1])) (Patel, 1970). The purpose of this function is to estimate the minimum cost from the bee's current state to any target state by calculating the absolute distance between projected target state and the bee's current position in its state. A technique called casting is implemented to ensure that the cost is appropriately calculated, without neglecting any contributing factors, such that it can be determined whether the total cost of this heuristic is admissible or not.

An admissible heuristic must underestimate the true cost that the A* algorithm would need to reach its target. If a heuristic does not underestimate the true cost it takes for the bee to reach its target practically, that heuristic is deemed as inadmissible. This is the reason why in the results tabulated in Question 3, there are certain instances where seemingly the A* algorithm performs worse than it's UCS counterpart. It is evident that a lower estimate is always preferable to an overly generous higher estimate.

In this case, the chosen heuristic is conservative in its estimation for this reason. Particularly in the node and runtime values from Question 3 (Test Case 1 and 2) demonstrate this A*'s efficiency over UCS, given they are similar or less than. This heuristic also accounts for distance values extremely well utilising casting in its implementation to compute the overall cost function effectively and without missing any values. This ensures all possible paths are accounted for by the bee.

However, both Test Case 3 and 5 demonstrated a more computationally efficient UCS when compared to A*. It is expected that if further variation in the heuristic was to be implemented, perhaps accounting for time or widgets, A* could have been better optimised. Similarly, understanding which heuristics A* algorithm responds best could also help in the optimisation process of this algorithm.

Appendix/References

Panchawate, P. (2023). *Uniform cost search (UCS) is a special case of A* algorithm*. Medium. https://medium.com/@pranjalpanchawate/uniform-cost-search-ucs-is-special-case-of-a-algorithm-ca7f828c62fe

Patel, A. J. (1970). *Hexagonal grids*. Red Blob Games. https://www.redblobgames.com/grids/hexagons/

Poole, D., & Mackworth, A. (2023). *Artificial intelligence foundations of computational agents: State space search*. 3rd Edition. Cambridge University Press. https://artint.info/3e/html/ArtInt3e.Ch1.S5.html

Code Sourced From: Tutorial 3 2024 Solutions (COMP3702)

Question 2: Module 1, pg. 8 Lecture Notes