Excercise 3
Implementing a deliberative Agent

Group №35: Aleksandar Hrusanov, Rastislav Kovac

October 20, 2020

1 Model Description

1.1 Intermediate States

Our state representation has a variable to store the *starting location* of the agent, which is used for creating the final plan. It also keeps track of the *current location* of the agent, as well as the task sets for currently *running tasks* (e.g. the tasks which the agent has picked up and is still yet to deliver) and *remaining tasks* (e.g. the tasks which the agent has still yet to pickup). The state updates current cost in the calculation of the BFS and ASTAR algorithm right after succesor states are generated. The cost is used for overall quality comparison of a state. We also have a variable to keep track of the agent's remaining capacity.

1.2 Goal State

A given state is considered to be a *goal state* if the agent has fulfilled all of the tasks, e.g. there are no more tasks on the map to be picked up and later delivered (the set of *remaining tasks* is empty) and there are no more tasks on the agent's vehicle to be delivered (the set of *running tasks* is empty).

1.3 Actions

For any given state, we consider two possible actions to generate the successor states. For each of the remaining tasks we generate a new state which reflects that the agent has moved to the task's pickup city and has picked up the task (e.g. agent's *current location* and *remaining capacity* is updated, as well as the sets of running and remaining tasks). Similarly, for each of the running tasks we generate a new state which reflects that the agent has moved to the task's delivery city and has delivered the task. Later, when we build the plan from initial state to the optimal goal state, between any two successive states we add the necessary *move* actions to the plan (e.g. moving from agent's location to task's pickup or delivery city along the shortest path).

In this way, we generate a successor state for each possible *pickup* and *deliver* action for the tasks in the corresponding task sets. Each successor state is a duplicate of the current state with the respective class fields changed as needed.

2 Implementation

2.1 BFS

Our BFS implementation uses a queue to store all states to be considered and keeps track of already visited states. It starts with the initial state, passed as an argument. For each considered state, we first check if it is a final state (see section 1.2). If not, then we generate its successor states. For each successor

state, we check if it is *redundant* - e.g. if we have already reached the same state through a different path but with a lower (or equal cost). For this purpose we keep a *hashset* of previously visited states and a *hashmap* mapping each visited state to the lowest cost we have reached it for. If the state is redundant, we simply throw away the state and do not follow its path any further. If not, then we add the state to the queue, update the data structures mentioned above, as well as a *hashmap* mapping a state to it's parent state (later used for building our plan). The *BFS* function finds the optimal final state, builds the plan, and returns it. We build the plan by backtracking through the hashmap between states and their parent states. For each state we add the respective action to the plan (*pickup* or *deliver*) and we connect two consecutive states by the corresponding *move* actions along the shortest path between the two states' locations.

We have implemented an *equals* function in order to make use of the *hashmap* and *hashset* classes. Lastly, since we only create a new state when the agent picks up or delivers a task, no two states in a given path will be considered the same (e.g. will have different running tasks and/or remaining tasks sets). This implies that no cycles are possible.

2.2 A*

The A* algorithm has a similar structure to the BFS algorithm with one important difference - we use a *priority queue* to order the states we want to explore next, rather than a regular FIFO queue. The ordering of the states in the priority queue is dictated by our chosen heuristic.

2.3 Heuristic Function

Our heuristic function gives the following cost estimation: $f(s) = g(s) + h(s)$. $g(s)$ is the cost of the plan until state s so far. $h(s)$ is the highest *single-task cost* which is defined to be the highest of the following: for every *running task* which the agent has picked up, the cost from the agent's current location to the task's delivery city; for every *remaining task* which the agent has yet to pick up, the cost from the agent's current location to the task's pickup city and from there to its delivery city. Now, we reason about the optimality of our heuristic. Assume that task $T = (T_p, T_d)$ with pickup city T_p and delivery city T_d has the highest *single-task cost*, so $h(s)$ is the cost of the shortest path from the agent's current location to T_p and then to T_d. We consider two cases. First, if T is *not* the last task the agent performs, then it incurs a cost higher than $h(s)$ since after picking up and delivering T, the agent has more tasks to handle which will increase the cost. Next, assume T is the last task the agent performs. There are two options: at least one of the other tasks is *not* on the shortest path to T_p and then to T_d. This implies that the agent will go through T_p and T_d via a more costly route than $h(g)$. And the second option is that all other tasks are on the shortest path to T_p and then T_d. This implies that the actual host is exactly $h(g)$. Thus, our heuristic never overestimates the cost, causing the A* algorithm to stay optimal.

3 Results

3.1 Experiment 1: BFS and A* Comparison

Algorithm	Tasks	7	8	9	10	11	12	13	14
BFS	Time (ms)	158	348	1200	4316	12011	44925	Timeout	Timeout
	Nodes	8425	28108	91612	298405	939193	2709589	Timeout	Timeout
	Distance (km)	1610	1710	1720	1820	1820	1820	Timeout	Timeout
A*	Time (ms)	103	237	510	1844	5189	13327	20475	90012
	Nodes	2242	8210	21088	85433	231269	573227	650744	1688244
	Distance (km)	1610	1710	1720	1820	1820	1820	1820	1820

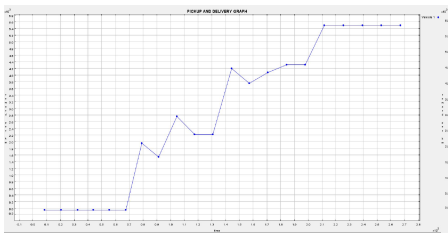

Figure 1: One Agent

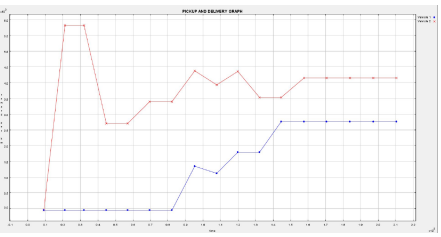

Figure 2: Two Agents

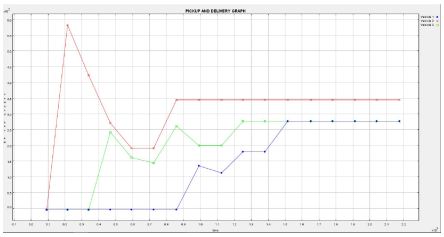

Figure 3: Three Agents

3.1.1 Setting

This experiment was performed on the Switzerland topology with 8 tasks and a task seed 23458. We experimented with various combinations for task weights, agent speed, and agent capacity. Each task has weight of 3kg and the agent has speed of 5km/h and same capacity 30kg. This was meant to test if the agent can indeed carry multiple tasks at the same time. Since time depends on the machine, we have provided the number of nodes an algorithm has traversed through.

3.1.2 Observations

The table shows us that both BFS and ASTAR algorithms were capable of finishing 12 tasks under a minute. Moreover, ASTAR finished 13 tasks under a minute and 14 tasks in one minute and half. BFS timed out after 12 tasks. Moreover, ASTAR was able to find a solution of the same cost as BFS. Since BFS returns an optimal solution, we can conclude that our heuristics does not overestimates.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

This experiment was performed on the Switzerland topology with 8 tasks and a task seed 23458. We experimented with various combinations for task weights, agent speed, and agent capacity. For easier analysis we kept those equal - each task has weight of 3kg and all three agents have the same speed (5km/h) and same capacity (30kg). The one-agent simulation from Figure 1 took about 2100 ticks and the agent started in Lausanne. The two-agent simulation from Figure 2 took about 1700 ticks and the additional agent started in Zurich. The three-agent simulation from Figure 3 took about 1500 ticks and the additional agent started in Bern. Some tasks relevant for the analysis of the experiment: (Task 0, Aarau - Sion), (Task 2, Zurich - Bern), (Task 4, Zurich - Luzern), (Task 6, Aarau - Basel)

3.2.2 Observations

In the one-agent simulation we see the agent executing it's initial optimal plan for the given task environment. In the beginning of the two-agent simulation, the red agent gains some reward at the expense of the blue agent. This is because in the blue agent's initial plan tasks 2 and 4 were among the first three tasks to be delivered, but in the red agent's initial plan those tasks were the first two to be picked up. Since the red agent started in Zurich, the pickup location of both tasks, it picked them up right away. In the three-agent simulation we see that the reward/km of the blue agent does not change compared to the two-agent simulation, but the red agent's one does. This implies that the green agent only interfered with the red agent's behavior. This is due to tasks 0 and 6 (the first two tasks to be picked up in the green agent's initial plan) since their pickup city, Aarau, is closer to the green agent's starting location than the red agent's.