Excercise 3
Implementing a deliberative Agent

Group №9: Baris Sevilmis, Doga Tekin

October 17, 2020

1 Model Description

1.1 Intermediate States

We use two concepts of state, we call the states that the agent moves around *intrinsic* States. We use another concept called *extrinsic* State in our algorithms to keep track of the costs and actions to reach *intrinsic* states, so we can find an optimal path to an *intrinsic* goal state. This representation is depicted in Figure 1. Our *extrinsic* state representation has three attributes:

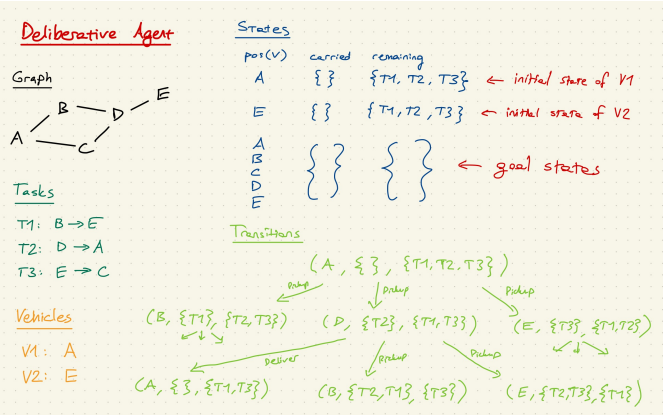


Figure 1: Intrinsic State Representation

- *Intrinsic* State:
  1. The agent's current city,
  2. Carried set of tasks by the agent
  3. Remaining set of tasks in topology(not carried or delivered by any agent yet)
- Cost of reaching the current *intrinsic* state starting from the initial *intrinsic* state
- List of actions to reach the current *intrinsic* state from the initial *intrinsic* state.

1.2 Goal State

The goal states are those where both the set of currently carried tasks and the set of remaining tasks are empty, as these states imply all tasks have been delivered, which is the agent's goal. Even though each agent plans to deliver all tasks, this does not indicate the an agent has to complete all the tasks on its own, rather an agent only has to complete all available tasks, whereas other agents may contribute to completing the remaining task set.

1.3 Actions

We argue that a deliberative agent cannot benefit from moving around randomly, so it will only take actions in order to either deliver a carried task or pick up a remaining task. In addition, agent is finished when the task sets are empty although some other agent may not be finished with its tasks yet, since moving randomly to another city is sub-optimal. Therefore, agents stay in their current cities when they don't have any more tasks to pickup or deliver.

2 Implementation

A key property of *intrinsic* state is demonstrated in our visited structure. The visited structure maps *intrinsic* states to the lowest cost observed to reach that state. This allows us to check whether a state has been visited already and with corresponding cost for arrival. Although BFS pseudo-code seems to be different, actually *intrinsic* states are not the sole identifiers of a state, the actions list and current cost have to examined as well. Therefore, observing an already visited *intrinsic* state does not mean the best path to that state has already been discovered, so the best cost and path to that *intrinsic* state (and its successors) might need to be updated.

2.1 BFS

We mostly try to follow the BFS pseudocode from the slides, we note our changes here:

• BFS queue holds *extrinsic* states. The initial *extrinsic* state consists of the initial *intrinsic* state, current cost of 0, and an empty action list.
• First set of goal states are not directly returned by algorithm, main goal is to find an optimal path to goal states. Therefore, we keep track of the goal state that has been reached with the lowest cost during the BFS, and only return the best path after the queue is empty and all paths have been explored.
• The data structure used to keep track of visited states remembers the cost taken to reach those states as well, so that in case they are reached through a more optimal path later they can be updated appropriately.

2.2 A*

A* follows a similar implementation to BFS with minor additions. A* utilizes a PriorityQueue instead of LinkedList and always puts at the head of the queue the state with the lowest cost + heuristic cost. Secondly, with regards to the PriorityQueue we know that the first goal state reached results in the optimal plan. Therefore, A* heuristic does end as soon as a goal state is reached.

2.3 Heuristic Function

The aim of the heuristic function is to estimate the cost to reach a goal state from a state as well as possible without overestimating the true cost, i.e. while satisfying admissibility. Therefore, we use the maximum of the following two values:

• If there are carried tasks, the cost to travel (by the shortest path) to the furthest delivery city among the carried tasks.
• If there are remaining tasks, the cost to pick up and deliver the furthest (in terms of shortest pickup+deliver distance) remaining task.

It can be proved that this heuristic does not overestimate the true cost to reach a goal state and therefore leads to an optimal solution. We know that a goal state has no carried tasks and no remaining tasks. So if the agent is carrying tasks, it will at least have to deliver the furthest task. If there are remaining tasks, it will at least have to pick up and deliver the furthest task. The largest of these is our heuristic.

3 Results

Table 1 demonstrates that both BFS and A* find an optimal plan with the same low cost in comparison to naive plan costs. Although, found paths may differ between A* and BFS, they have the same cost.

Table 1: Cost Comparison for Single Agent

Costs	*4 Tasks*	*5 Tasks*	*6 Tasks*	*7 Tasks*	*8 Tasks*	*9 Tasks*	*10 Tasks*	*11 Tasks*	*12 Tasks*	*13 Tasks*
Naive Plan	11050	15450	19200	21100	22100	23050	25900	27750	29950	32800
BFS Plan	6100	6100	6900	8050	8550	8600	9100	OOM	OOM	OOM
A* Plan	6100	6100	6900	8050	8550	8600	9100	9100	9100	9100

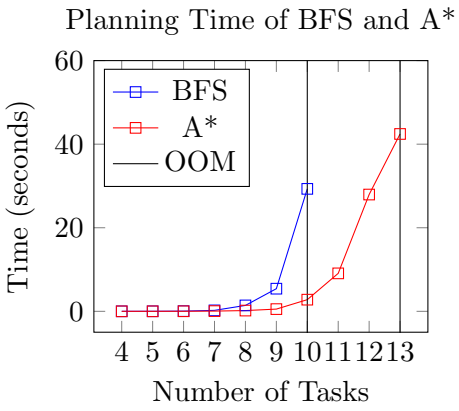
3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

In this experiment, we compare the planning phase of BFS and A* by their corresponding times, in which we increase the task amount for both BFS and A*. There is only a single agent and task amount is varied from 4 to 13 in general. We use the Switzerland topology.

3.1.2 Observations

It could be easily observed that A* is much more efficient than BFS as expected. This is especially clear given high amount of plannable tasks, where A* can plan up to 10 times as fast as BFS. Under favor of efficiency, A* can optimally plan for up to 13 tasks in 1 minute whereas BFS can only plan for up to 10 tasks in the same amount of time. There is an additional observation such that both algorithms find an optimal path with the same cost in all cases, although they sometimes find different optimal paths. Lastly, due to complete traversal of planning space of BFS unlike A*,it runs into memory problems at more than 10 tasks(Out of Memory). Even if we increase the memory as much as possible, the algorithm ends up with a timeout.



3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

In this experiment, multi-agent systems have been tested, where agent amount has been varied from 1 to 6 agents. Task amount has been set to 8 and Switzerland topology has been selected. All agents have been arranged such that their capacity and speed are equal.

3.2.2 Observations

Simulation steps required to complete all deliveries are decreasing as expected until a certain agent amount. This would correspond to decrease in task amounts per agent and therefore, total amount of tasks would be completed in less simulation steps. However, this decrease slows down/stops at a point. Planning steps are not included within simulation steps, though replanning amount increases due to conflicts for certain (planning times decreased due to less remaining task amount). Stabilization or decrease in efficiency could be interpreted as a result of non-communicative behaviour. Since agents do not communicate with each other but only observe the environmental changes, task distribution and planning are not efficient for increasing amount of agents after a certain point. Results for both BFS and A* are demonstrated within Figure 3.2.1.