ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Intelligent agents

CS-430

Lab3: Deliberative Agents

Submitted By Group 36:
Christopher Salotti & Marco Antognini

October 2015

Introduction

Continuing on the *Pick-up and Delivery Problem* with the *Logist* platform, we know focus on a *deliberative agent* that will build a strategy beforehand and apply it until the goal is reached or the strategy gets invalidated because the environment was altered by an event independent from the agent's will, in which case a new plan is computed based on the new environment.

As with the previous experiments, the goal is to deliver every available task to their respective destinations. The agent's strategy consists in planning a sequence of actions such as moving to another city, delivering a task or picking up a task. Here we will consider the following three algorithms to build such plan. The first algorithm we consider is a **Naive** one in order to have a baseline: the agent will consider only one task at a time, pick it up and deliver it to its destination. Then we consider **Breath-First Search**: We define a graph representation of the problem, where nodes are agent's state and a node successors is define by the set of valid action from that state, and we look for the shortest set of legal actions to any final state. Finally, we test the **A*** algorithm and three heuristic functions that predict the cost of a node in the search graph and focus on visiting nodes that have the lowest cost first and draw a few conclusions from the trade-off between a precise heuristic that can be used to find the best plan of action and other heuristics that converge more quickly but cannot be used to always find optimal plan.

State and Action Representation

In order to use graph-based search algorithms we have to define a proper representation for the state and the action of our deliberative agent. In the search graph we represent node with states and transition between nodes with actions.

In this *Pick-up and Delivery Problem*, the agent vehicles have a limited capacity and can move only to neighbouring cities. Once they have loaded a task they cannot throw it away; they must deliver it to its destination.

Based on the facts presented above, we endowed the agent's **State** with the following properties:

- the location of the agent in the topology; i.e. a **City**;
- the set of tasks loaded on the lorry ready for delivery; i.e. a **TaskSet**;
- the remaining capacity of the lorry; i.e. an integer;
- and the set of tasks available for pick-up; i.e. another **TaskSet**.

A State is considered *final* if the lorry is empty and no task remain to be picked-up in the environment.

We consider three kind of **Action**s:

- *Move(c)*: The agent moves from its current location to the city c.
- *PickUp(t)*: The agent picks up task t from its current location.
- *Deliver(t)*: The agent deliver task t to its current location.

If we consider **State** alone, we already have a considerably huge number of nodes in our search graph. We therefore limit the action set to the subset of legal actions. For example, an agent can only move to a neighbouring city c if 1) there is a task for pick-up in city c, or 2) the agent has a task on board to be delivered in city c, or 3) city c is the first step on the shortest path toward another city that has a task ready for pick-up. Similarly, we limit the *PickUp(t)* or *Deliver(t)* actions to agents that are in the origin or destination, respectively, of the given task t.

Implementation

Here we describe the overall structure of our implementation but leave out the *Naive* algorithm as it was not written by us.

Breadth-First Search We consider the algorithm presented for the that we modified according to the course[1]. We define a node as a set of actions that lead to a state and we compute the **optimal plan** when the first final node is encountered.

A* We considered the algorithm presented in class[2] with three different heuristics:

CONSTANT The simplest heuristic simply assigns a cost of 0 to any node in the search graph.

OPTIMISTIC This heuristic gives a cost of $-\sum_{t \in T} reward(t)$ where $T = AvailableTasks \bigcup LoadedTasks$.

DELIVERY The last heuristic ignore the remaining task to be picked-up and simply gives the sum of reward for the tasks already loaded (i.e. $T = LoadedTasks$).

Note that only the *OPTIMISITIC* heuristic is optimal: the only thing it doesn't take into account is the travelled distance, which will increase the cost. Hence it will always underestimate the real cost. The *CONSTANT* heuristic is consistent only when the overall reward is less than the cost of distributing the tasks, which is in fact usually not true, but can, as we will see, still produce interesting results when a considerable amount of tasks are available. Finally the *DELVIERY* heuristic suffer from a similar issue but in fewer situations and can achieve better result and furthermore still terminates with an important number of tasks.

We believe that some better heuristic, reducing the number of visited nodes in the search graph, could be built that actually takes the travelled distance into consideration by using a variant of TSP where we would build the shortest path linking all cities where we need to go but ignoring the cities where there is no task to be picked up or delivered. But by lack of time we could not implement it.

Results

			A*							
			Delivery		Constant		Optimistic		BFS	
	Naive Baseline	# Tasks	d	t	t	t	d	t	d	t
	1796 km	8	2.43	0s	1.47	0s	2.83	0s	2.53	26s
	2055 km	9	2.4	0s	1.4	0s	2.74	1.1s	†	†
NL	2481 km	11	2.6	0s	1.66	0s	3.4	32s	†	†
	20617 km	95	3.65	5.1s	2.71	18s	†	†	†	†
	32097 km	150	4.3	10.6s	†	†	†	†	†	†
	3250 km	8	1.8	0s	1.7	0s	2.5	0s	2.42	27s
	3656 km	9	2.35	0s	1.65	0s	2.64	1s	†	†
CH	4383 km	11	2.65	0s	1.6	0s	3	17.7s	†	†
	39176 km	95	4.4	0s	3.4	26s	†	†	†	†
	6289 km	150	4.7	1.6s	†	†	†	†	†	†

d: Distance improvement ratio compare to naive baseline.
t: Time to find a plan.
†: Algorithm computation go beyond 1 minute
Average results for both topology on 3 different RNGs.

Table 1: Performance comparison of algorithms for Switzerland & The Netherlander

[1] Slide 17 of deliberative-agent.pdf.
[2] Slide 6 of IA0607_Exercises_3_-_Deliberative_Agent.pdf.

From Table 1, we can observer the following results for a single agent. Only our *OPTIMISTIC* heuristic is optimal although we cannot run it on more than 12 tasks. *DELIVERY* and *CONSTANT* are suboptimal but they can perform well and are more scalable than BFS, which has to visit too many nodes when the number of tasks grows. We can clearly see that with more than 8 tasks our agent won't be able to perform well by using BFS since it grows exponentially.

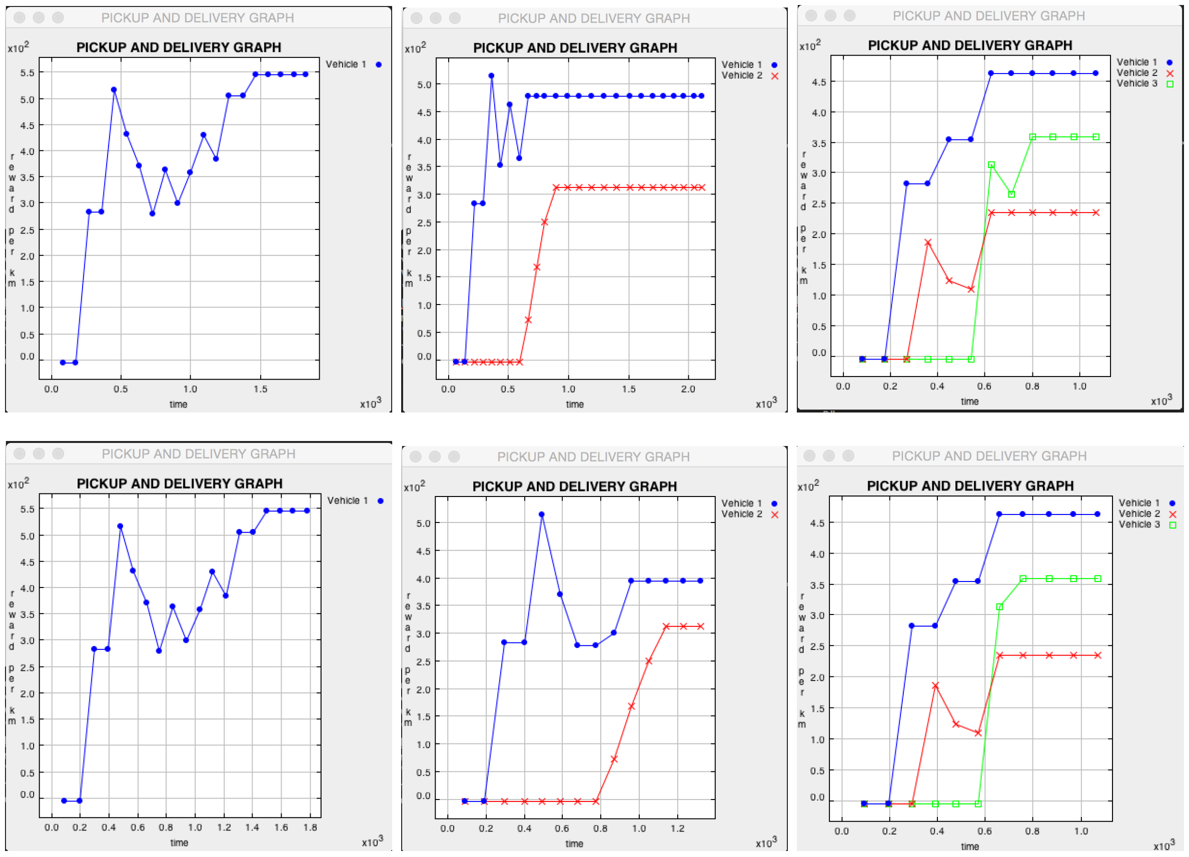

Figure 1: Running 1, 2 or 3 agents on 8 tasks: 1^{st} row is BFS agents (comp. time: 30 sec). 2^{nd} row is A* *OPTIMISTIC* agents (comp. time: less than 1 sec.)

When running multiple agents at the same time (cf. Fig. 1) we can observe that for few tasks (8) we have relatively close results with A* and BFS. However, the computational time is hugely reduced by A*, hence the reaction time of BFS agents is much bigger. This is a critical point that needs to be taken into consideration when using such algorithms when the computed plans that might get invalidated and therefore recomputed.

However, with both algorithms, it appears that having only 1 agent of a kind outperforms having more agent simultaneously. This fact can be simply explained: both A* and BFS planning algorithms don't take into consideration the actions of the other agents during the computation of the best plan.

Conclusion

We have found two algorithms that resolve our agent decision problem in an optimal fashion. The first one, **BFS**, takes ages to compute a situation above 10 tasks. The second one, **A***, uses a more *local* search to have a better efficiency in term of computation time and, even when the heuristic is sub-optimal, can find a better results.