Excercise 3
Implementing a deliberative Agent

Group №34: Anh Nghia Khau, Stephane Cayssials

October 25, 2016

1 Model Description

1.1 Intermediate States

In this project, we define a state with these following characteristics:

- **mVehicle**: the vehicle of the agent
- **mAvailableTasks**: the set of all avalaible tasks on the map
- **mCarriedTasks**: the set of all present tasks in this vehicle
- **mPlan**: the plan for this state
- **mCurrentCity**: the current city
- **mInitialCity**: the city where we create the plan
- **mFreeWeight**: the free weight for this vehicle
- **isFinal**: this state is the final state or not
- **mCost**: the cost of this vehicle

1.2 Goal State

The final state is that we will try to pick up all available tasks and deliver all of them until we have no more task on the map.

1.3 Actions

- In the system of one-agent, the planning is computed at the begining. At a given state, agent will choose the next state on the planning depend on the value of the current state: mAvailableTask, mCarriedTask, mCost, mFreeWeight... or depend on the strategy of each agent (NAIVE, BFS, ASTAR) until there is no task on the map.
- In the multi agent system, the plainning is also computed at the beginning. But at a given moment, when the agent realised that there was an another agent who steal his task, so his planning is broken and he have to re-compute his planning again.

For each current state, we observe the list of available tasks and the list of carried tasks to create all possible next states. So all tasks possible at a current state is : AllStateOne + AllStateTwo

- AllStateOne: for each task on the mAvailableTasks list, we check if the vehicle can take this task (constraint on the capacity of the vehicle), if yes we say this is a new possible state, if not we discard this state and continue checking through all the list.
- AllStateTwo: for each task on the mCarriedTask list, we will create a new state for delivering this task.

2 Implementation

2.1 BFS

```
LinkedList<State> Q = new LinkedList<State>();
...
State state = Q.poll(); // always take the head of the queue to analyse (property of BFS)
if (! C.contains(state)){
    C.add(state);
    for (State s : state.succ())
      if (! Q.contains(s)){ // check if this state is already in the queue
         Q.add(s);
       }
}
```

2.2 A*

```
//ASTAR - Method
PriorityQueue<State> Q = new PriorityQueue<State>(new Comparator<State>() {
        @Override
        public int compare(State o1, State o2) {
           return o1.compareTo(o2); //compareTo base on function f(n) = g(n) + h(n)
        }
     });
   // Q.poll() always return a state in the queue with the smallest cost
   State analysedState = Q.poll();
  // Merge
  for (State s : analysedState.succ()){
      if (! Q.contains(s))
          Q.add(s);
  }
```

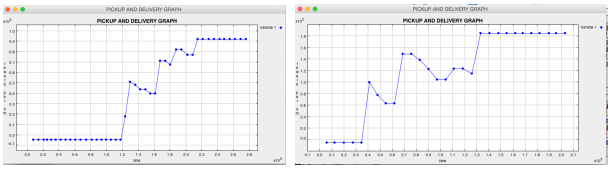
2.3 Heuristic Function

For the heuristic function, idea comes from the problem of Minimal Spanning Tree. We want to discover all nodes with a minimal weight, for this, at each step we will move to the next node that has a smallest weight (Prim's Algorithm)
For the AvailableTask list, we will find the task that has the smallest cost (in this case, cost = (distance(pickup) + distance(delivery))*costPerKm) (suppose that task "a" has the smallest cost in AvailableTask) AND and also for the CarriedTask list, we find the task that has the smallest cost (in this case, cost = distance(delivery)) (suppose that task "b" has the smallest cost in CarriedTask). Then agent will take the task that will give a minimun cost between "a" and "b" and this smallest cost is called "Heuristic cost". So g(n) is the total cost of this agent so far and h(n) is the "Heuristic Cost".

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting



		Number of task					
		5	6	7	8	9	10
BFS	minCost	11550	13500	16300	17550	18650	N/A
	# iters	730	2458	8425	28108	91612	N/A
	Time(ms)	27	145	800	8671	126516	N/A
A-STAR	minCost	6900	6900	8050	8550	8600	N/A
	# iters	213	439	2573	10141	23296	N/A
	Time(ms)	14	23	165	2828	25013	N/A

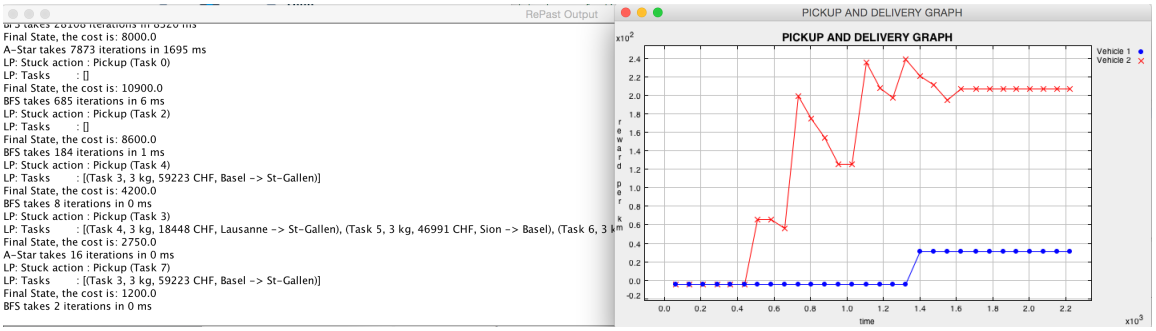
An Agent and his 6 tasks with BFS (left) and ASTAR (right) Algorithm

3.1.2 Observations

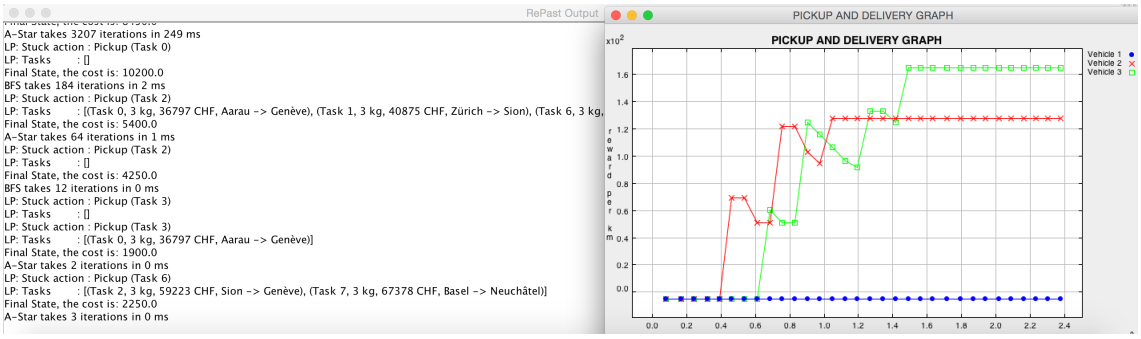
ASTAR is much more efficient than BFS. In BFS algorithm, we lost a lot of cost for moving (discover each layer without analyse the cost) while in ASTAR algorithm, we choose the task that has smallest cost each time.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting



Two agents and 8 tasks: one BFS in blue and one ASTAR in red



Three agents and 8 tasks: one BFS in blue and two ASTAR in red and green

3.2.2 Observations

ASTAR is always more efficient that BFS, but in the system multi agents, there is a lot of conflit between these agents, so we have to re-compute the planning each time when there was a conflit. In addition, agent lost some cost to come for picking a task but the task is stolen by another agent. This is a reason why here the award per km are not so much in comparison to the one of sigle agent.