Intelligent agents

CS-430

Lab4: Centralized Agents

Submitted By Group 36:
Christopher Salotti & Marco Antognini

November 2015

Introduction

Continuing the *Pickup and Delivery* problem, we now focus on a viable solution for multiple agents. As we observed with the *deliberative* solution, multiple agents are *toxic* to each others if not coordinated. Unlike the deliberative algorithm, the centralized solution organizes the plan of each agent so that they can cooperate and therefore improve the overall cost of the generated plans. This is done by having a central entity that computes the best plan for a multi-agents system given a set of tasks.

There exist multiple ways of finding the best actions for each agent. We will consider two strategies: a **NAIVE** planner, which only purpose is to serve as a baseline, and a **Stochastic Local Search** (**SLS**) planner. On one hand, the dummy planner assigns all tasks to the same agent which will process them sequentially. On the other hand, the SLS algorithm will search for more optimal plans. This search is based on two main features: a) it is stochastic, meaning that it might explore sub-optimal solutions, and b) it is local, meaning that the generated solutions are *similar* to the current solution. We modified the provided SLS algorithm to what follows:

$\mathcal{I}, \mathcal{S} \leftarrow 0$ `/* iteration and stall iteration counts, respectively */`
$\mathcal{A}, \mathcal{A}^* \leftarrow$ CreateInitialPlan()
while $\mathcal{I} \leq$ *maximum number of iterations* $\mathcal{I}^\dagger$ **and** *hasn't run out of computation time* **do**
  $\mathcal{I} \leftarrow \mathcal{I} + 1$
  $\mathcal{N} \leftarrow$ GenerateNeighbors($\mathcal{A}$)
  switch *the result of an unfair coin tossed* **do**
    case *head:* $\mathcal{A} \leftarrow$ SelectBestOf($\mathcal{A}$, $\mathcal{N}$)
    case *tail:* $\mathcal{A} \leftarrow$ SelectAnyOf($\mathcal{N}$)
  if $\mathcal{A}^*$ *is better than or as good as* $\mathcal{A}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} + 1$
  else
    $\mathcal{S} \leftarrow 0$
    $\mathcal{A}^* \leftarrow \mathcal{A}$
  if $\mathcal{S} \geq$ *maximum number of stalled iterations* $\mathcal{S}^\dagger$ **then**
    $\mathcal{S} \leftarrow 0$
    $\mathcal{A} \leftarrow$ CreateInitialPlan()
    $\mathcal{A}^* \leftarrow$ SelectBestOf($\mathcal{A}$, $\mathcal{A}^*$)
return $\mathcal{A}^*$

In other words, given a maximum number of iterations, we stochastically explore the neighborhood of the current plan $\mathcal{A}$, which gets reset to an initial plan when we are stuck in a local minimum for too long, while keeping track of the best plan $\mathcal{A}^*$. The fairness of the tossed coin defines how randomly we explore the search space. We denote by **p** the probability of getting *tail*.

Model Description

In order to implement the SLS, we followed the instruction given in the annex of this lab. Since we have to consider agents that can carry multiple tasks simultaneously we split the pickup and delivery of a task into two distinct events. Furthermore, we defined the variables for the **Constraints Satisfaction Problem** (**CSP**) as follows:

1. Each agent should have a plan, possibly empty;
2. Each task should be picked up exactly once, and therefore by only one agent;
3. Each task should be delivered exactly once, and therefore by only one agent;

4. Each task should be delivered by the same agent that picked it up;
5. Each task should be delivered after being picked up;
6. No agent should be overloaded at any point in time.

Each of these constraints have to be respected when we generate the *Initial plan*, which we define later, and the *Neighboring plans*, which are computed using the following five kind of *mutations*:

1. The first task of an agent can be transferred to another agent **if** this other agent has enough capacity;
2. The pickup time for a given task can be advanced **if**, at no point in time, the agent is overloaded;
3. The pickup time for a given task can be postponed **if** the delivery time is still after the pickup time;
4. The delivery time for a given task can be advanced **if** the delivery time is still after the pickup time;
5. The delivery time for a given task can be postponed **if**, at no point in time, the agent is overloaded.

Note that we do not generate the full set of neighbors as it would be too big and slow. Instead, the neighboring plans are stochastically selected: we randomly select an agent with at least one assigned task and apply the 5 rules on each of its tasks.

Implementation

The code is split into three main classes:

`VehicleAction` This class represents the action that an agent can perform; it is a task-event pair where the event is either picking up the task or delivering it.

`GeneralPlan` This class represents the overall plan $\mathcal{A}$ made of a `Map` of vehicles to `List` of `VehicleActions`. It also implements both *CreateInitialPlan* and *GenerateNeighbors* functions.

`CentralizedTemplate` This class implements the `CentralizedBehavior` interface as well as the *NAIVE* and *SLS* planner previously mentioned using `GeneralPlan`.

The initial plan can be created in two ways: either we randomly distribute tasks to agents, taking into account their respective weights and capacities, or we assign all tasks to the agent with the biggest capacity. The first alternative is used by our `SLS_RANDOM_INITIAL` kind of agent, denoted **SLS RI** below, while the second one is used by regular **SLS** agents.[1]

Results

Table 1 reports the score of the different algorithms described above for multiple situations.

The First observation we can make is the following: the classic **SLS** performs better than **SLS RI**. This observation is quite well correlated with the following one: as shown on Figure 1(c), the **SLS** algorithm has a tendency to prefer allocating a lot of tasks to a single agent. An agent initially close to a *loop* or highly connected city will have an advantage over the other agents. We observed this situation when agent are quite far from the center of the map and from each other. Fortunately, the improvement compare the the **NAIVE** implementation is quite significant.

Another key point of **SLS** is the parameters for the computation of the optimal plan. Since the local search isn't exhaustive, it is sensitive to local minima or maxima. The introduction of a maximum number of iterations $\mathcal{I}^\dagger$ and of stalled iterations $\mathcal{S}^\dagger$ as well as keeping track of the best plan found so far was a good way to avoid this problem. We observed that the optimal $\mathcal{I}^\dagger$ was $50'000$, as bigger values do not

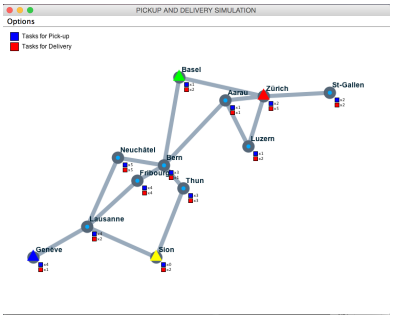
[1] See agents.xml

		4 agents			3 agents			2 agents		
		NAIVE	SLS	SLS RI	NAIVE	SLS	SLS RI	NAIVE	SLS	SLS RI
ENG	p = 0.3	68731	12059	16017	68731	11887	16918	68731	11958	14716
	p = 0.5		12120	13729		12267	13233		12501	13844
	p = 0.7		16960	17367		16898	17343		17050	17434
	p = 1.0		47717	49325		49123	49442		49877	48992
CH	p = 0.3	56400	10870	14260	56400	10530	14560	56400	10640	14280
	p = 0.5		10870	12350		10980	13750		10710	12730
	p = 0.7		14590	15450		13820	15380		13920	16140
	p = 1.0		40740	41390		40510	40880		40770	40590

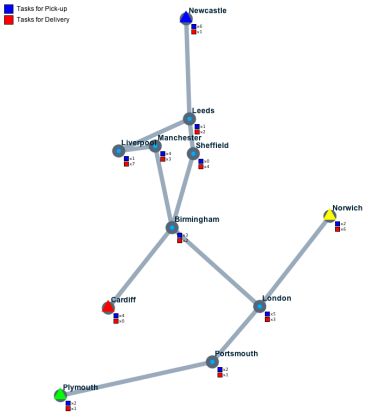
Table 1: Mean of the overall cost for multiple agents, map and probabilities

significantly improve improve the results, and $\mathcal{S}^{\dagger}$ set at $5'000$ gives good performances while smaller $\mathcal{S}^{\dagger}$ values will actually decrease the quality of the generated plan.

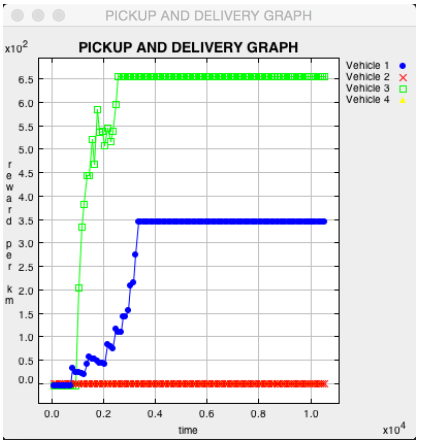
Furthermore, as shown in Table 1, the best solutions are achieved using a probability $p = 0.3$ for **SLS**. The improvements of plan cost, for 2 agents, compared to the **NAIVE** baseline are 83% and 81% for England and Switzerland, respectively. Similar values are obtained when more agents are involved because, as we discussed above, most agents are at rest.



(a) Agents on Switzerland map



(b) Agents on England map



(c) Typical centralized agents behavior

Figure 1: maps & agent behavior

An important note to add is the topological differences between *Switzerland* and *England*. As we can observe, agents are situated to the extremes in *England*, compared to *Switzerland* where they are situated in a cycle. Furthermore, our assumptions on agents positions can be verified. Observations showed that, even if the vehicles positions is quite different, **SLS** still assign most of the tasks to a single agent.

Conclusion

The centralized agent was meant to bring cohabitation and collaboration between agents that were toxic for each other with the *deliberative* solution. But as we have seen the **SLS** algorithm is *unfair* as, most of the time, assigning every tasks to a single agent will perform better than distributing the tasks among several agents. The experimental results let us conclude that **SLS** isn't a good way for a diversity of user agents to improve performance and optimal collaboration between agents.