# Intelligent Agents - Centralized

Valentin Rutz, Ana Manasovska

November 2015

## 1 Introduction

In this assignment, again we try to solve a version of the Pickup and Delivery problem. In this version, we were supposed to distribute set of tasks among multiple vehicles, in order to minimize the cost.

## 2 Model

We model the problem as a constraint optimization problem i.e. we try to find a plan that minimize the total cost, while the constraints hold. The given constraints are:

1. all the tasks have to be delivered
2. every task has to be picked up before delivering it
3. the vehicles have a limited number of tasks that they can carry (the sum of the weights of the tasks that a given vehicle carries cannot exceed its capacity).

### 2.1 Variables

We have given a set of tasks $T = \{t_1, ..., t_n\}$ and set of vehicles $V = \{v_1, ..., v_m\}$. We can define the variables of our optimization problem.

*nextAction* - an array of $2N_T + N_V$ variables. The array contains two variables for every task, one for the pickup and one for the delivery, and one variable for every existing vehicle.
The domain of values that nextAction can have is either pickup of a task or delivery of a task

$$nextAction(x) \in \{p(t_1)...p(t_N), d(t_1)...d(t_N)\}$$

where with $p(t)$ we denote the action of picking up task t, and with $d(t)$ we denote the action of delivering the task t. Similar to the referenced paper, $nextAction(v) = a$ means that the first action of vehicle v is a, and $nextAction(a_i) = a_j$ means that the action $a_j$ is going to be taken after the action $a_i$.

We also use the arrays *time* and *vehicle* in a same way as presented in the given paper Finding the Optimal Delivery Plan: Model as a Constraint Satisfaction Problem, in order easier to check if the constraints hold.

### 2.2 Objective function

The functon that we want to optimize (minimize) is the total cost, calculated as following

$$C = \sum_i dist(v_i, nextAction(v_i) * cost(v_i) + \sum_i dist(a_i, nextAction(a_i) * cost(vehicle(a_i))$$

where $dist(v_i, nextAction(v_i)$ is the distance between the start city of the vehicle $v_i$ and city of the its first action, and $dist(a_i, nextAction(a_i)$ is the distance between the cities of the actions.

### 2.3 Constraints

1. PickUp before delivery - The vehicle that has the pickup action of a given task in its plan, has to have the delivery of same task too. Moreover, in the list of actions the pickup action has to be before the delivery.
2. Capacity - An action can be added to a plan of a vehicle, only if it doesn't exceed its capacity. The constraints for the arrays time and vehicle are similar to the ones in the referenced paper.

## 3 SLS

We apply SLS algorithm in a similar manner as in the reference paper. The details of our implementation are the following.

### 3.1 Initial solution

The SLS algorithm is very likely to end up in a local minimum. Therefore, with different initial solution we can end up in a different final result. We have tried two possibilities for an initial solution:

1. give all of the tasks to the biggest vehicle
2. randomly give tasks to all vehicles

For each task, we create two actions, Pickup and Delivery, and when we give a task to a vehicle, that means we give the both actions in order to satisfy the Pickup-before-delivery constraint.

### 3.2 Choose Neighbours

For generating neighbouring solutions, we choose one vehicle at random, we perform local operations on its current plan. The local operations are:

1. swap the order of two actions (as long as they don't break the constraints)
2. take an action and its counter part (the corresponding pickup or deliver action) and give it to another vehicle.

For the chosen vehicle, we apply the swap operation for every pair of actions in its plan. Also, we apply the transferring of actions, for every action, and try to give it to another vehicle. Exploring larger neighbourhood may help to avoid local minimum.

### 3.3 Local Choice

In local choice, with probability p we move to a neighbour, and with probability 1-p we keep the previous solution. The neighbou to which we are going to move is chosen randomly from an array of neighbours which is consisted of all neighbours that are very close to the best neighbour. We have defined being "very close" as differing from the best cost with less than 3% of the best cost. We have explored different values for this parameter and confirmed that 3% give fairly good results in most of the cases.

### 3.4 Stopping criterion

The stopping criterion in our case was the number of iterations. In general, the algorithm converges to some local minimum. With different number of iterations we could end up in a different minimum, especially if we use larger neighbourhood so that we avoid getting stuck in a minimum. After 10000 iterations, in most of the cases, we have observed convergence with a minimum with a satisfying cost, and execution time less than a minute.

## 4 Performance

In many cases, the best plan included only one vehicle, or if there are multiple vehicles included in the plan, one vehicle would again do most of the job, and the other vehicles would deliver only one or two tasks. This trend is probably a result of the fact that the vehicles have fairly large capacity (30), one vehicle can carry up to ten tasks, and if it moves anyway it wouldn't be very costly for it to pick up most of the tasks. If we reduce the capacity of the vehicles, when can see that multiple vehicles get involved in the optimal plan.
The complexity of the algorithm is dominated mostly by the number of tasks. The complexity of one iteration depends on the number of neighbours generated, which for t tasks and n vehicles would be at most $t(t-1)/2+nt$, that can be represented with $O(t^2)$ (if we assume that n is smaller that t, which is true in general).
The cost of the best plan that is being computed is usually somewhere between 15000 and 20000.

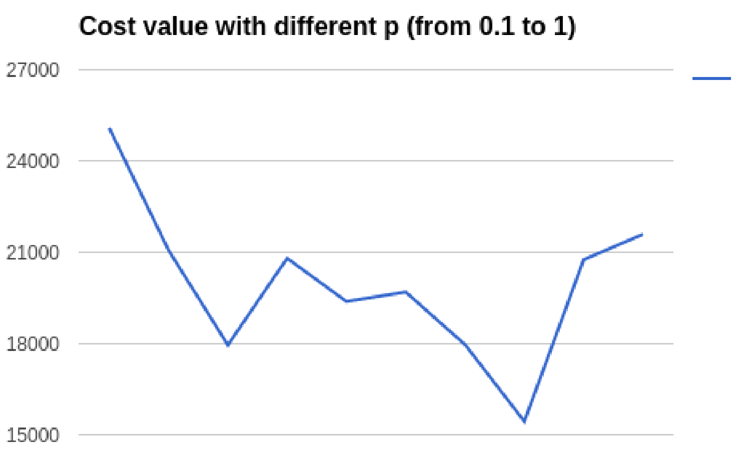

Figure 1: The change of the cost with respect to p.

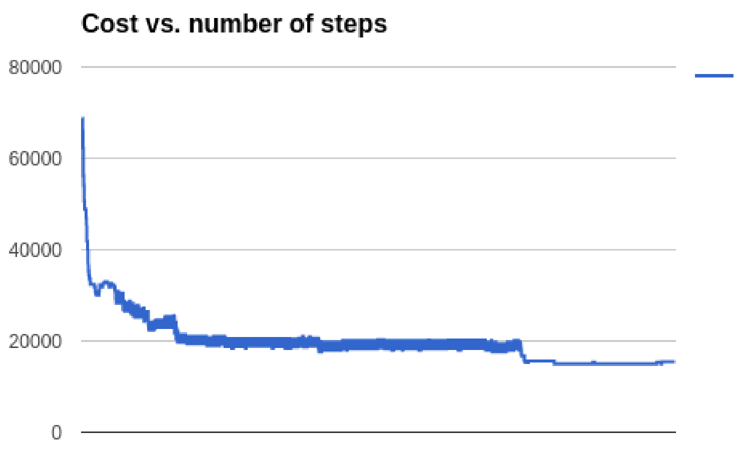

Figure 2: The improvement of the cost after 10000 interactions

## 5 Code

A plan is represented with the class Solution, and its main variable is a Map, that has an entry for every vehicle with a list of its actions. The actions are represent with the class ActionWrapper, that has two subclasses PickUpWrapper and Delivery Wrapper. These classes besides the attributes describing the task, also store a pointer to the counter part (pickup or delivery).