AAMAS Project 2018, Group 16 *

Alice Karnsund
alicekar@kth.se

Therese Stålhandske
thesta@kth.se

ABSTRACT

In this report a case of the Vehicle Routing Problem is investigated, relating it to the real-world problem of food delivery in a city. The problem is simulation in both a non-dynamic and dynamic environment, in which the solutions displays similar behavior. Different cost functions are explored, which are found to lead to distinct types of solutions and agent behaviors.

1. INTRODUCTION

Uber Eats is an American online food ordering and delivery company that was started by Uber in 2014. It partners up with restaurants in cities all over the world, including Lisbon. Ordering is done on their website or through their phone application. Meals are delivered to the customer by couriers. Uber eats have several couriers employed that need to be coordinated in order to optimize the delivery.

In this specific project, we approach the problem in two different ways,

1. All orders are known from the start and the goal is to divide the orders between 2, 3 or 4 couriers in such a way as to minimize the total travel distance. Thus they have to collaborate deciding upon who should go where and when. When all these customers are served the task is completed.

2. New orders will be made as the time progresses, thus the agents together need to re-plan and distribute the orders in a new most efficient way, taking the new order/orders into account.

Our main focus in both cases is to minimize the total travel distance and delivery times made by the couriers.

2. THEORY AND IMPLEMENTATION

In this project, we will consider an Uber eats simulation in Lisbon. The solution visualizes couriers making their routes on a map of Lisbon during a certain time period.

*For use with aamas2017.cls

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

2.1 Environment and Problem Definition

The traffic network of Lisbon is visualized as a graph, where the nodes are locations of the restaurants or the customers and the edges are the paths connecting these. In our case, we will connect the nodes by the shortest distance, thus not using the true roads of Lisbon. Every edge will be divided into steps, so-called mid-steps, corresponding to the length of a time-step.

The agents will start out at random and unique selected nodes in the traffic network graph, which are neither customer or restaurant nodes. Thereafter the agents are only allowed to move along any of the predefined edges. To keep the problem to at a reasonable size, we set a couple of constraints,

- The graph consists of a set number of nodes, that are either restaurant, customer or start nodes
- The orders are divided as equally as possible between all the couriers
- Each order can be seen as a pair of nodes consisting of a customer node and a corresponding restaurant node
- The couriers will always drive first to a restaurant and then to the corresponding customer, thus the end nodes will all be customer nodes
- The number of couriers will be either 2, 3 or 4, due to the fact that we have a set number of nodes
- Couriers are not allowed to visit a node that is included in another courier's route
- Couriers are not allowed to visit a node that they have visited before

The couriers know the distribution of possible customers and restaurants at each time step, making the environment **fully observable**. In the second scenario, new orders randomly pop up during the day, making the environment **non deterministic**. Due to the fact that we divide the time period into a set number of discrete steps, there are a limited number of discrete, clearly defined states of the environment, and thus it is **discrete**.

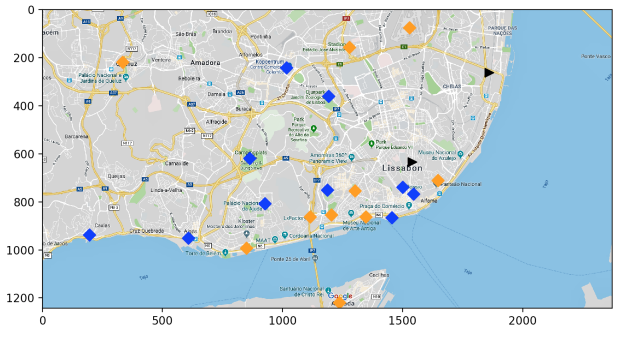

Figure 1: Map over Lisbon where the orange markers corresponds to randomly selected restaurants, the blue corresponds to customers and the black markers are starting nodes. In this case the environment is for 2 couriers.

Since each order is seen as a pair of two nodes, i.e a customer and a restaurant, this problem can be compared to the Multi-depot Multiple Traveling Salesman Problem (MMTSP), a special case of the Traveling Salesman Problem (TSP).

Given a set of nodes, a set of m salesmen that are located on a set of depots and a cost metric. The objective of the MMTSP is to determine a tour for each salesman, such that the total tour cost is minimized and that each node is visited exactly once by only one salesman[1].

2.2 Algorithm

To find the set of routes that will minimize the total distance traveled by the couriers we made use of a so-called Ant Colony Optimization Algorithm (ACO). This is a metaheuristic algorithm that uses a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. Artificial Ants stand for multi-agent methods inspired by the behavior of real ants. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions [4].

In every iteration, the ants explore the graph and lay pheromones on edges that they find attractable, in this way they will communicate to other ants saying that this is a good path. Thus for the later iterations, some specific edges will contain a lot more pheromones than others, leading to the ants choosing shorter and shorter paths.

When considering our multiple courier case, each courier can be seen to have an ant "working for it" trying to find the best route for that courier, during a fixed number of iterations. The best routes found by the ants will be the routes for the couriers. To make sure that each courier has a unique route, we set constraints saying that the ants in the current iteration are not allowed to visit nodes that in the same iteration has been visited by other ants. They still have to visit a predefined number of nodes, corresponding to almost equal division of orders between the couriers. If an ant is currently on a restaurant node, the ant then has to travel to the corresponding customer node and thereafter, they can only go to a new restaurant.

For this algorithm we investigated two different cost functions, C_{sum} and C_{max} formalized as,

- $C_{sum} = \sum_i T(A_i)$
- $C_{max} = maxT(A_1), T(A_2), \ldots, T(A_m)$

where $T(A_i)$ is the route distance for ant i. C_{sum} is then the total traveled distance of all ants in that iteration. C_{max} is the longest route of an ant in an iteration.

In our case, we investigate two different approaches to the problem, an environment with predefined orders and a dynamic environment, presented in sections 2.2.1 and 2.2.2 respectively.

2.2.1 *Scenario 1: Predefined orders*

First, we consider the case where all orders are known from the start. Thus given the information of where the customers and their corresponding restaurants are, we want to distribute the orders to the couriers in a way that minimizes the total travel distance. As was mentioned above, during the iterations to find the optimal routes, it is the ants that are doing the traveling. In each iteration, the ants will have to select their next node to visit according to the constraints mentioned for the couriers. They are not allowed to move to just any node from a current one. The next node an ant will choose is due to a probabilistic selection.

For example, say that an ant "Ant k" is at a customer node A, this node is connected to all the other nodes in the graph, $[B, C, D, E, F, G, H, I, J, K, L]$. But in this iteration, the nodes $[C, D, J]$ has already been visited by other ants and will, therefore, have transition probability, $P_{AC,AD,AJ} = 0$ for Ant k, meaning that it cannot choose any of these. Also, say that the nodes $[F, G, I]$ are customer nodes, and since Ant k already is at a customer node, these nodes will also have transition probability, $P_{AF,AG,AI} = 0$. If Ant k has been at node B before in this iteration this node will also have $P_{AB} = 0$. Thus Ant k can only choose to go to one of the nodes, $[E, H, K, L]$. The transition probability for each of these nodes is calculated according to,

$$P_{ij} = tau_{ij}^{\alpha} \times eta_{ij}^{\beta}$$

where in this case $i = A$ and $j \in [E, H, K, L]$. tau is a matrix holding the pheromone level at each edge. eta is a matrix holding the desirability of each edge, where each element is calculated as $eta_{ij} = \frac{1}{d_{ij}}$, d_{ij} is the distance between node i and j. The parameters, α and β determines the relative influence of pheromone versus distance ($\alpha, \beta > 0$).

Thereafter, given $P = [0, 0, 0, P_{AE}, 0, 0, P_{AH}, 0, 0, P_{AK}, P_{AL}]$, the new restaurant node will be selected according to the following function:

```
Roulette(P)
    r = random number between 0 and 1
    C = cumulative sum of vector P
    new node is the one corresponding to
    the first appearance in P where r <= C
```

After the tours for every ant in an iteration is found, the tau matrix will be updated, so the new pheromone levels are passed on to the next iteration. Thus providing information about the routes done by the previous ants. The update is done using the following equation,

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \triangle\tau_{ij}^{k} \quad (1)$$

$$\triangle\tau_{ij}^{k} = \left\{ \begin{array}{ll} \frac{Q}{L_k}, & \text{if ant k travels on arc (i,j)} \\ 0, & \text{others} \end{array} \right\} \quad (2)$$

Where Q is a constant and L_k is the k^{th} ant's total tour distance. The ρ parameter is coefficient of evaporation which has the value $\rho \in (0, 1)$, such that $(1-\rho)$ is the evaporation rate[2].

The implemented Ant Colony algorithm is shown below,

```
Initialize the graph, ACO parameters,
and orders per ant
Give a unique start node to each ant
For a number of iterations,
    Create ants representing the couriers
    For every ant
        While ant not visited all it's
        number of nodes
            Transition probability process,
            find route according to above
    Compute total travel distance for all
    ants, i.e. calculate the cost
    If this cost is less than in previous
    iterations
        Save the ants routs
    For each edge
        Update pheromone (cooperation part)
Return best routes, corresponding to
smallest cost
```

In the above algorithm, in each iteration the routes for the ants are found first for the first ant and then for the second ant and so on, thus when coming to the last ant, it has no option of selecting orders. He simply has to settle for the leftovers. This will, therefore, lead to that the earlier ants will have shorter routes and the last one will most likely have a much longer one. But this division does also depend on from which node each ant starts. The ants, in this case, are cooperating to find the total shortest route for Uber as a company.

2.2.2 *Scenario 2: Dynamic Environment*

The deterministic environment is created such that at each mid-step there is a predetermined probability of a new order being made. This changes the environment to the extent that a new delegation of orders has to be made to minimize the total traveling distance. This is done by re-planning the tour with ACO, as described above, under the new conditions [3]. Each courier is designated to finish its current assigned order but may receive a new route to perform when the current order has been delivered. For simplification, the remaining distance from a courier to the current customer is not considered when delegating new tours. Further, there is no distinction between old or new orders.

3. RESULTS

3.1 Scenario 1: Predefined orders

Using ACO it is possible to use different heuristic functions, which will generate distinct solutions. When minimizing the total route cost for all courier, C_{sum}, we are given a solution with a few long routes and a few shorter, as seen in Figure 2. When minimizing the maximum cost for each individual courier, C_{max}, there is a greater independence between the agents, and no route is too long, which often result in a much higher total distance traveled, as seen in Figure 3.

Parameter settings are studied, as it controls the amount of collaboration between the ants. Figure 4 show that there is a dependence between the evaporation rate and the total route cost. A lower evaporation rate initially led to increased cooperation, resulting in a lower total cost, but further a low evaporation rate causes the optimization to get stuck in the local minimum, increasing the total cost.

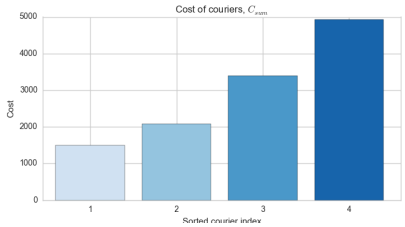

Figure 2: Route cost for each courier using cost function C_{sum}. Number couriers = 4, number of orders = 10

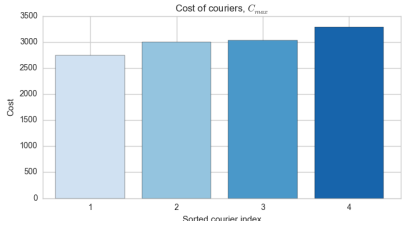

Figure 3: Route cost for each courier using cost function C_{max}. With this cost function, the algorithm does not care to minimize the cost of the routes with cost below the maximum.

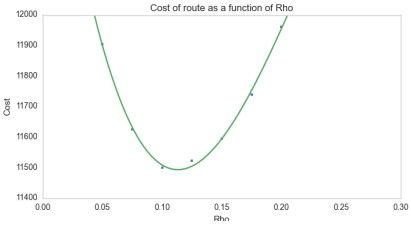

Figure 4: Average Route cost for each courier using cost function C_{sum} with varying evaporation ρ

3.2 Scenario 2: ACO in a dynamic environment

In the dynamic environment, the routes are re-planned when a change in the environment occurs. Using the different cost functions, the prioritization of the new delegated nodes are also different. The two different cost functions, presented above, are used and the result is presented in figures 5 and 6. The planning is done with 4 couriers and 10

orders, with a $P(neworders|t < 50) = 0.1$ at each mid-step. Similar behavior as in the scenario 1 is displayed.

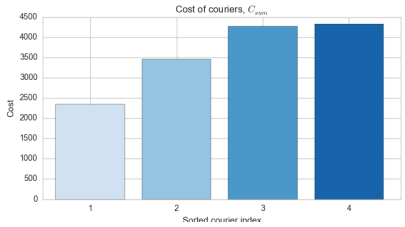

Figure 5: Route cost for each courier using cost function C_{sum}. Number of couriers = 4, number of initial orders = 10

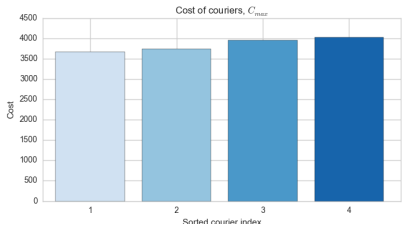

Figure 6: Route cost for each courier using cost function C_{max}. Number of couriers = 4, number of initial orders = 10

When using C_{max}, the new orders tend to be delegated to the courier that has the shortest planned route. As shown in Table 1, this leads the simulation, on average, to end earlier.

Cost function	Average simulation time (mid-steps)
C_{sum}	119 $\pm$5
C_{max}	103 $\pm$4

Table 1: Average time for a simulation to finish, time measured in mid-steps. Initial orders = 10, couriers = 4 and $P(neworders|t < 50) = 0.1$

.

4. DISCUSSION AND CONCLUSIONS

For simplification of the problem, we put some restrictions on the environment to make the implementation easier. For instance, it is assumed that edges exist in between every node. This might not completely be a realistic assumption since not all locations are connected via one road. However, it still provides a good approximation of the real solution. A further restriction was that the couriers could only carry one order at the same time, by removing this restriction it is likely that the optimal solution would be less costly. For the dynamic environment, new orders are randomly made, but each individual customer time is not considered. This means it is possible for a new order to be delivered before an already existing one. Making the individual customer wait a long time. In a more realistic scenario, this would be of importance. If a customer has been waiting for too long, they might cancel the food. Therefore an extension of the problem would be to take the waiting time into account as well as adding a probability of each order to disappear when a certain time threshold has been exceeded.

For the agents to pursue decision with a level of independence, we only let the agents know where a previous agent already has been. If each courier only tries to optimize its own distance, it will only consider its own decisions, independent of what the other agent does.

Further, some interesting notes can be made about the choice of the cost function. Minimizing the max travel distance, using C_{max}, ensures that the distance of the longest route will not differ too much from the others, which will minimize the maximum time any customer have to wait for delivery. The drivers show a more independent and competitive behavior, as each courier are working for minimizing its own tour length. Although this might seem like a good or even sufficient parameter to minimize, it has the obvious downside that no further optimization is made and the total cost increases with C_{max} compared to C_{sum}. On the other hand, minimizing the sum, C_{sum}, may leave some customers to wait longer, which of course could lead to unhappy customers.

For extension and improvement of the algorithm, a mix between the two cost function might result in an agent-behavior that both looks after its own interest along with the main goal to minimize the total cost. It would make sure that no single order is delivered very late, while also minimizing the overall delivery time. An alternative approach to this would be that the ants simultaneously selects their next node, thus leading to a more ”fair” distribution of the travel distance. This way can be seen as a more competitive strategy since every ant wants to travel a distance as short as possible.

In summary, we have shown that ACO is a sufficient algorithm to solve a multiple non-deterministic pickup and delivery problem. Using different cost functions determines the level of independence of the agents and produce slightly different results.

REFERENCES

[1] neos Guide. Multiple traveling salesman problem (mtsp). *https://neos-guide.org/content/multiple-traveling-salesman-problem-mtsp*, Access: May 28, 2018.

[2] T. Ramadhani, G. F. Hertono, and B. D. Handari. An ant colony optimization algorithm for solving the fixed destination multi-depot multiple traveling salesman problem with non-random parameters. *American Institute of Physics*, pages 3–4, 2017.

[3] S. Russel and P. Norvig. *Artificial Intelligence - A mordern Approach*. Pearson Education, Upper Saddle River, New Jersey, 2010.

[4] Wikipedia. Ant colony optimization algorithms. *https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms*, Access: May 28, 2018.