A live route planning algorithm for UC's NightRide service

Almudena Chapa, Dani Lázaro, Jon Ander Martín

April 25, 2021

Abstract

The Vehicle Routing Problem is a classical problem in discrete optimization. The interest of this problem lies in the multiple variants and applications it has in the real world for the transportation of goods and people. In this work we solve the dynamic version of this problem for a specific application: the NightRide service provided by the University of Cincinnati. Using a combination of smart clustering and Ant Colony Optimization, we have solved the problem and demonstrated the algorithm can outperform the performance of the route planning done by a human operator.

1 Introduction

NightRide is a program within the University of Cincinnati's (UC) Public Safety Department that provides on-demand safe transportation for students, staff and faculty within one mile of UC's main campus[1]. The service uses several vehicles for the purpose. The users can request point to point trips within the aforementioned radius. So far, each vehicle needs two workers to operate: the driver and a copilot that decides the routes based on incoming requests. There is no communication between vehicles, other than a general view of all the pending requests. These facts result in a sub-optimal service, that often leads to long waiting times when demand for the service is high. Therefore, it is of general interest to come up with a system that can optimize and coordinate the routes of each vehicle to improve the quality of the service.

We define the problem to solve as follows. We consider as input a number of customer requests, each with a pick-up and a drop-off point in a metric space. The set of requests is not fixed, and new requests appear during the simulation. We also consider the number of vehicles available to provide the service, and the capacity of each of them k. The goal is to calculate a route that progressively serves all the incoming requests, minimizing its length and the waiting time of the customers.

To bound the scope of this work and fit it to the allotted time frame, we have considered several simplifications and assumptions. Each request will have a single person riding, as opposed to the real service where groups of up to 5 customers can request a ride. The vehicles can have multiple customers with different routes riding at the same time, provided the maximum capacity is not exceeded. We consider the capacity of a vehicle to be $k = 5$. We also limit the number of vehicles to $N = 2$.

All the incoming requests go to a common customer pool shared by all the vehicles in service. However, it is worth noting that each customer requires its pick-up and drop-off locations to be included into the same route. This is not only a constraint, but also an added difficulty since both way-points have influence in the length of the route.

We have divided the resolution of the problem into two tasks. First, based on the two

[1] "NightRide". *University of Cincinnati Public Safety website.* Available at *https://www.uc.edu/about/publicsafety/services/night-ride.html*

way-points and waiting time of each customer, a clustering algorithm allocates a set of $n \leq k$ customers from the pool to a vehicle. Note that with this limitation the capacity of the vehicles can never be exceeded. An Ant Colony Optimization (ACO) algorithm takes the $2n$ way-points of the selected customers and finds a near-optimal route to serve them efficiently.

2 Background

The Vehicle Routing Problem (VRP) is a well-known discrete optimization problem. The VRP involves a set of customers that must be served once, and a fleet of vehicles that depart from a depot, serve the requests, and go back to the depot. The VRP and its many possible variations have been widely studied in the past. [1] shows an extensive literature review on the topic. Another interesting review is presented in [2], with a novel method to solve the multi-depot VRP. One possible variation of the VRP closer to reality is the Dynamic Vehicle Routing Problem (DVRP), where the customers change their demands gradually with time. One way to solve this problem is to divide it into multiple static VRP problems [3]. In the specific problem we are solving, the vehicles do not go back to a depot after serving a number of requests, and every request has two waypoints, both for pickup and delivery. These conditions are shared with the Dial a Ride Problem (DARP) when the capacity of the vehicle k is not equal to 1 [4]. However, to the authors' best knowledge, the problem that combines DVRP and DARP has not been solved yet.

The VRP and all its variations are considered NP-hard problems. There is no known algorithm that can find the optimal solution for every instance of this problem, and heuristic methods are considered a good approach to solving them [5]. The ACO is one possible algorithm to find a near-optimal solution to this kind of problem [6]. Further, the basic ACO can be enhanced by using several methods, such as 2-opt heuristic [3, 5, 7] or a Firefly Algorithm [8].

3 Methods

In this section we will introduce the algorithms used to solve the two tasks mentioned in Section 1.

In the dynamic problem, customer requests change gradually with time. Directly using an ACO algorithm to solve the problem would entail several issues. If the ACO is executed every time there is a new customer in the pool, the planned route changes every few simulation steps. This means that the vehicles never complete all the computed routes and they are not exploiting the benefits of route planning with the ACO. This approach is also computationally expensive. That is why in our methodology we divide the dynamic problem in multiple static problems [3]. In turn, the static problem is divided into two tasks: customer allocation and route planning with ACO. A new static problem is solved every time a vehicle finishes serving its current route.

3.1 Customer allocation

This algorithm takes as input the information from all the customers in the pool and allocates $n \leq k$ to a vehicle. The information contains the location of the pickup and the drop-off in a metric space. To account for the waiting time, we add the time that customers have been in queue to this information. These parameters are fitted into a vector of the form

$$\boldsymbol{x}_{customer_i} = \begin{bmatrix} x_0 & y_0 & x_f & y_f & t_{wait} \end{bmatrix}^T \qquad (1)$$

where x_0 and y_0 denote the coordinates of the pickup point, x_f and y_f the drop-off point, and t_{wait} represents the waiting time.

The vector of each of the vehicles is built as follows,

$$\boldsymbol{x}_{vehicle_j} = \begin{bmatrix} x_v & y_v & x_v & y_v & t_{longest} \end{bmatrix}^T \qquad (2)$$

where x_v and y_v denote the coordinates of the vehicle when this algorithm is executed. $t_{longest}$ is the highest waiting time of all the customers in the pool. The choice for the “drop-off point” of the vehicle is arbitrary, and helps to select customers closer to the vehicle. Setting the "waiting time" of the vehicle to the highest one of the customers in the pool makes the algorithm give more weight to the customers that have been in queue the longest.

The algorithm calculates the euclidean distance between $\boldsymbol{x}_{vehicle_j}$ and all the customers in the pool, $\boldsymbol{x}_{customer_i}$. Then, selects the n lowest distances, allocates those customers to the vehicle, and removes them from the pool.

3.2 Ant Colony Optimization

This algorithm considers a set of m ants, each of which represents a vehicle. All of them start at the node corresponding to the initial position of the vehicle and visit the way-points assigned to the vehicle until all of them have been visited once. Being n the number of customers assigned to each vehicle, the route consists of $2n + 1$ way-points: one pick-up and drop-off location per customer and the initial position of the vehicle. For a given number of iterations *n_iteration* every ant generates a route over $2n$ steps by choosing a new way-point at each step.

In this DVRP, it must be considered that drop-off locations cannot be visited before pick-up locations. Therefore, to overcome this constraint, we have defined an "illegal" list for each ant, which contains the drop-off way-points whose respective pick-up locations have not been visited yet. At each step, any arc defined by the actual position of the ant and a point in its "illegal" list will be considered an "illegal" arc. Note that "illegal" lists depend on which nodes have been visited so far and can be different for each ant. This means that at the same time an arc may be illegal for one ant but not for another. Consequently, an ant could choose a no longer illegal arc for itself, and thus deposit pheromone on it, which in the next step could still be illegal for another ant.

Since next way-points are chosen based on pheromone and distance information, to avoid selecting "illegal" paths, before choosing any next way-point, m copies of the current pheromone information are done. Each copy is used for one ant, and in it the pheromones corresponding to all the "illegal" arcs of that ant are set to zero. This way, if each ant uses its pheromone copy for the decision, "illegal" arcs will have zero probability of being chosen.

The equations to calculate the probability of an ant taking an arc (i, j) are extracted from [9]. Two probabilistic rules are defined to select the next destination. A random number q between [0,1] is generated to decide which one to apply. If $q \leq q_0$, the ant will use

$$p_{ij}^k = \begin{cases} 1 & \text{if } j = \arg\max a_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Otherwise, it will use

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum_{l \in \mathcal{N}_i^k} a_{il}(t)} \tag{4}$$

where

$$a_{ij}(t) = \frac{[\tau_{ij}(t)]\,[\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i} [\tau_{il}]\,[\eta_{il}]^\beta} \tag{5}$$

p_{ij}^k represents the probability of an ant k choosing a path (i, j), $\mathcal{N}_i^k$ the neighbors of node i not visited by ant k and $\mathcal{N}_i$ all the neighbors of node i, τ_{ij} the pheromone concentration on arc (i, j), and η_{ij} the inverse of the length of arc (i, j).

Once the next way-point is chosen, if it was a pick-up location, its respective drop-off location is removed from the list of "illegals". On the other hand, the pheromone of the chosen arc is updated using

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0 \tag{6}$$

where ρ represents the evaporation rate and τ_0 is the fixed pheromone increment.

Finally, after all ants have constructed their routes, a "daemon" adds pheromone over the arcs included in the shortest route found according to

$$\tau_{ij}(t + 1) = (1 - \rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{7}$$

where

$$\Delta\tau_{ij}(t) = \begin{cases} 1/L^+(t) & (i, j) \in T^+(t) \\ 0 & (i, j) \notin T^+(t) \end{cases} \tag{8}$$

and L^+ represents the length of the shortest route, T^+.

3.2.1 Parameters

Equations 3 through 8 show the large number of parameters that this algorithm has. These parameters greatly affect the behavior and performance of the algorithm. Using [10, 11] as reference, we have done several test runs to find the parameters that suit our problem the best. The final values are $n\_iterations = 10$, $m = 20$, $q_0 = 0.2$, $\rho = 0.2$, $\beta = 6$ and $\tau_0 = 1/65$.

4 Results

The objective of this work is to improve the performance of NightRide with respect to the current state, in which a human operator in each of the vehicles plans the customers to serve next. Therefore, the results shown here compare the performance of our algorithm with the performance of a planning done by a human over the same list of customers.

The results have been obtained in a simulation environment specifically created for this work. The environment is formed by a 20×20 bi-dimensional space where customers are located and vehicles can move (Figure 1). In each step, the vehicles move a fixed distance v towards the next way-point in their current route. If they reach a way-point, they pick up or drop off a customer and proceed to the next way-point. If the way-point they reach is the end of their current route, they execute the customer allocation and ACO algorithms to calculate the next route. Every *customer_frequency* steps a new random customer appears and is added to the pool. The pseudo-code of the environment is shown in Algorithm 1.

```
initialize vehicles;
initialize customer pool;
for t in T iterations do
    if mod(t,customer_frequency) = 0 then
        add customer to pool;
    end
    for j vehicles do
        move 1 step towards current destination;
        if destination reached then
            set new destination;
            if end of current route then
                new customers ← customer allocation algorithm;
                new route ← ACO;
            end
        end
    end
    for i customers do
        $t_{wait} \leftarrow t_{wait} + 1$;
    end
end
```

Algorithm 1: Simulation environment. The values of *customer_frequency* and the speed of the vehicles have to be balanced to keep a steady-state of customers and avoid an empty pool.

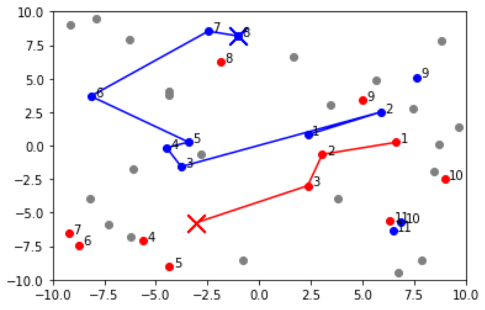

Figure 1: Snapshot of the simulation environment. Each vehicle (X), the route it is following, and the way-points allocated to it is represented with a different color. The grey points are the rest of the customers' way-points in the pool.

Table 1 shows the comparison of various metrics when the route planning is done by the algorithm and manually. The human planners designing manual routes were given a set of $k = 5$ customers previously chosen by the customer allocation algorithm. Therefore, this table compares only the route planning, not the customer allocation. The algorithm outperforms the manual planning in 8 out of 10 simulation runs. The algorithm is able to reduce the mean waiting time of customers (t_{mean}) up to a -21.5% with respect to the manual planning. The maximum difference in the cases in which the algorithm in-

Method	t_{mean}	D_1	D_2	n_{total}
Algorithm	84.73	348.8	327.6	55
Manual	87.50	352.8	368.0	60
Algorithm	75.58	334.3	366.8	60
Manual	81.67	361.2	399.5	60
Algorithm	65.67	361.1	343.0	60
Manual	85.67	352.9	402.1	60
Algorithm	66.31	351.7	365.9	65
Manual	84.50	356.0	365.9	60
Algorithm	79.17	371.2	366.3	60
Manual	76.08	352.2	364.9	60
Algorithm	73.83	369.8	355.7	60
Manual	77.75	360.2	399.4	60
Algorithm	60.00	385.0	365.3	65
Manual	70.08	364.8	376.3	60
Algorithm	62.92	333.1	352.7	60
Manual	68.92	337.3	354.5	60
Algorithm	81.17	368.5	378.6	60
Manual	84.64	337.2	345.4	55
Algorithm	74.58	352.6	373.9	60
Manual	71.25	375.0	365.6	60

Table 1: Comparison of results of the algorithm and solutions obtained manually over 400 simulation time-steps. t_{mean} is the mean waiting time of the customers (in simulation time-steps), and D_1 and D_2 the total distances traveled by the vehicles 1 and 2, respectively. n_{total} represents the number of customers served during the simulation time. Each Algorithm-Manual simulation pair has been obtained fixing the random generation of customers. The manual solutions have been designed by the authors of this work and two independent, external, test subjects.

creases the mean waiting time is +4.1%. There are no significant differences in the total distance covered by the vehicles. This can be explained with the fact that the vehicles move at a constant speed and the total simulation time-steps is a fixed parameter. The simulation environment is ideal, so no delays are produced by traffic or customer pick-up or drop-off times. Therefore, the vehicles are expected to cover a similar distance in all the simulation runs. There are very little differences in the number of customers served during the simulations. This indicates there is not a big improvement in the route planning, since more efficient routes would allow the vehicles to serve more customers in the same simulation time.

Method	t_{mean}	t_{std}	L_{mean}	L_{std}
Clustering	59.80	2.21	55.84	1.31
FIFO	76.86	1.77	61.03	1.35

Table 2: Comparison of results when using the clustering algorithm and the FIFO criterion for customer allocation in the simulation. Each simulation environment has been run 300 times and results have been averaged. t_{mean} is the average of the mean waiting times of the customers (in simulation time-steps), L_{mean} is the mean length of the all the routes traveled by both vehicles, and t_{std} and L_{std} their respective standard deviations. The random generation of customers in both environments has been equally fixed.

To see to what extent the classifier is helping in the route planning, we have also compared the clustering algorithm and the human choosing criterion. The latter has been assumed to follow a ≪first in, first out≫ (FIFO) criterion serving the oldest $k = 5$ customers in the pool. Table 2 shows the comparison in the mean waiting times and mean route lengths when both customer allocating strategies are used.

At first glance, one might think that picking up the longest waiting customers would reduce the average waiting time. However, by not considering their pick-up and drop-off locations, routes become more inefficient, which ultimately results in longer average waiting times. A 9.29% increase in average route length converts to a difference in wait time of +28.53%. This demonstrates the effectiveness of the proposed clustering algorithm.

5 Conclusions

The shared Dial a Ride Dynamic Problem that we have proposed is hard to be solved efficiently by one person and usually results in a suboptimal service. To improve the efficiency of this type of service, inspired by the University of

Cincinnati's NightRide, an algorithm that coordinates the service of different vehicles has been proposed. Results have shown that it outperforms a human by hand's planning.

It should be noted that the main difficulty of the problem is that each customer involves the addition of two nodes to the route. As the pool of customers gets larger, it becomes harder to evaluate at first sight the influence of incorporating two nodes and therefore, which customers to allocate to each vehicle. Nonetheless, once this problem is solved and each driver knows which customers to serve, the difficulty of the problem is considerably reduced from the point of view of a person, since there are only 10 nodes to consider. Therefore, much of the success of our algorithm is due to the strategy taken for the allocation of customers.

It is true that ACO generally provides more efficient routes than those planned by hand. However, since once given the 10 waypoints, an individual can also make considerable good routes by hand, the slight improvement in customer waiting times rarely allows more customers to be served in the same simulation time. If the number of nodes in the route was increased, which is equivalent to consider higher capacity vehicles such as buses, the human operator would experience greater difficulties in designing better routes. In that case, one would expect greater favorable differences for the ACO.

These are the first tests developed to make a first evaluation of the performance of the algorithm. Therefore, a very simplified simulation environment has been chosen, which is far from reality. Once we have demonstrated that we have found an efficient algorithm for solving the ride-sharing problem, future work will consist of incorporating this base in a more realistic environment. This would involve, for example, considering the road network to define distance information or including traffic conditions in the generation of routes. The limitation of 1 customer per request should also be eliminated before implementing this procedure in the real service.

References

[1] Çağrı Koç, G. Laporte, and İlknur Tükenmez, "A review of vehicle routing with simultaneous pickup and delivery," *Computers & Operations Research*, vol. 122, 2020.

[2] Y. Li, H. Soleimani, and M. Zohal, "An improved ant colony optimization algorithm for the multi-depot green vehicle routing problem with multiple objectives," *Journal of Cleaner Production*, vol. 227, pp. 1161–1172, 2019.

[3] H. Xu, P. Pu, and F. Duan, "Dynamic vehicle routing problems with enhanced ant colony optimization," *Discrete Dynamics in Nature and Society*, vol. 2018, pp. 137–172, 2018.

[4] M. Charikar and B. Raghavachari, "The finite capacity dial-a-ride problem," in *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pp. 458–467, Nov 1998.

[5] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Advanced Engineering Informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[6] B. Bullnheimer, R. Hartl, and C. Strauss, "An improved ant system algorithm for the vehicle routing problem," *Annals of Operations Research*, vol. 89, pp. 319–328, 1999.

[7] B. Yu, Z.-Z. Yang, and B. Yao, "An improved ant colony optimization for vehicle routing problem," *European Journal of Operational Research*, vol. 196, no. 1, pp. 171–176, 2009.

[8] R. Goel and R. Maini, "A hybrid of ant colony and firefly algorithms (hafa) for solving vehicle routing problems," *Journal of Computational Science*, vol. 25, pp. 28–37, 2018.

[9] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete opti-

mization," *Artificial Life*, vol. 5, pp. 137–172, 04 1999.

[10] J. C. Molina, J. L. Salmeron, and I. Eguia, "An acs-based memetic algorithm for the heterogeneous vehicle routing problem with time windows," *Expert Systems with Applications*, vol. 157, p. 113379, 2020.

[11] D. Gaertner and K. Clark, "On optimal parameters for ant colony optimization algorithms," *Proceedings of the 2005 International Conference on Artificial Intelligence*, vol. 1, pp. 83–89, 01 2005.