










Solving Dynamic Delivery Services Using Ant Colony Optimization

Miguel S. E. Martins¹(✉) , Tiago Coito¹ , Bernardo Firme¹ ,
Joaquim Viegas¹ , João M. C. Sousa¹ , João Figueiredo² ,
and Susana M. Vieira¹ 

¹ IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

{miguelsemartins, tiagoascoito, bernardo.firme,
joaquim.viegas, jmsousa, susana.vieira}@tecnico.ulisboa.pt

² IDMEC, Universidade de Évora, Évora, Portugal

jfig@uevora.pt

Abstract. This article presents a model for courier services designed to guide a fleet of vehicles over a dynamic set of requests. Motivation for this problem comes from a real-world scenario in an ever-changing environment, where the time to solve such optimization problem is constrained instead of endlessly searching for the optimal solution. First, a hybrid method combining Ant Colony Optimization with Local Search is proposed, which is used to solve a given static instance. Then, a framework to handle and adapt to dynamic changes over time is defined. A new method pairing nearest neighbourhood search with subtractive clustering is proposed to improve initial solutions and accelerate the convergence of the optimization algorithm. Overall, the proposed strategy presents good results for the dynamic environment and is suitable to be applied on real-world scenarios.

Keywords: Pickup delivery problem · Ant Colony Optimization · Local Search · Time windows · Dynamic requests

1 Introduction

According to data on *The World's Cities in 2018* United Nations (2018), it is clear that big cities are bound to grow both in size and number. This brings many concerns regarding the already problematic vehicle saturation on urban settlements. Small efficiency increases can have a big impact all-around when applied at a larger scale. This is especially relevant for transportation companies, whose main activity often involves driving and thus requires careful planning not to travel on heavy traffic situations.

This work was supported by FCT, through IDMEC, under LAETA, project UIDB/50022/2020.

The results presented in this article culminate from the development of a model for courier services based on the general demands of a real world scenario. At its core, the problem at hand focuses on transporting goods between two locations while efficiently handling the routing of a vehicle fleet, this is, a Vehicle Routing Problem (VRP). Since it operates on a dynamic environment, it is constructed to handle both changing traffic conditions and insertion of new customer requests over time.

2 Vehicle Routing Problem

The VRP category is a combinatorial optimization and integer programming set of problems. It deals on how to direct a fleet to serve interest points in the most profitable way. The most common problem on combinatorial problems is the large number of possible solutions. Many of these problems are in fact NP-Hard, i.e., there is no guarantee the optimal result can be reached in polynomial computation time Steiglitz (1982).

The origin of this problem can be traced to the Travelling Salesman Problem (TSP), in Flood (1956). Here the goal is to make a single salesman find the round-trip through each and all the cities only once. From here, a broader generalization was made in 1959, the Vehicle Routing Problem (VRP), credited to Dantzig (1959). The focus of the VRP shifts to finding the optimal set of routes for a fleet of vehicles to service a given set of customers. An example comparison between the two problems can be visualized in Fig. 1.

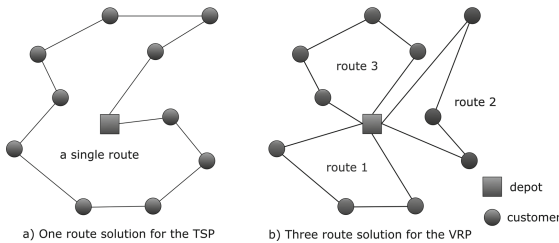


Fig. 1. TSP and VRP example for the same set of points

Many other constraints exist to model different real-life scenarios, many often approached as specialized sub-problems of the VRP Montoya-Torres (2015):

- **Capacitated VRP (CVRP):** customers have specific demands for amount of goods and vehicles have finite capacity, which can be volume or weight;
- **Time Windows:** if locations must be always be visited within a time interval, these are called *hard time windows*. If the visit can be outside the time interval but the solution incurs into a penalty, they are *soft time windows*.

- **VRP with Pickup and Delivery (PDVRP)**: when there is a need to *pickup* an order from a specific location and *deliver* it to another. A *request* is then characterized by the pair pickup-delivery.

If the conditions of a problem instance remain unchanged during the whole problem, it is called *static*. Otherwise it is a *dynamic* problem, which can entail changes in orders, number of vehicles, travel costs, etc. The simplest approach to a dynamic problem is to solve it as static, and restart the optimization if the conditions change. When considering real world applicability, there might not be enough processing time to achieve good solutions between updates. An interesting modification is presented by Ferruci (2014), where the working span is separated into successive time intervals of fixed length, each is solved as a static instance. During this interval, any new requests are buffered for insertion at the end of the time span.

2.1 Metaheuristics

Metaheuristics are a common approach when exact methods are inapplicable to large search spaces. At their core, they are a set of general directives that can be adapted into a big variety of problems with little changes to the inner workings of the algorithm. Metaheuristics also have the advantage of exploring large search spaces without getting trapped in local minima by allowing temporary deterioration of the solution. However, these strategies often require long computational times and a careful parameter tuning to provide good solutions. An example of a widely used metaheuristic, and the one used for this paper, is the Ant Colony Optimization (ACO). As many other, this metaheuristic is based a natural behaviour, the foraging process of a colony of ants.

The ACO formulation is convenient to the VRP formulation since ants are assumed to travel on a weighted node graph, visiting every node once and stopping only when returning to the starting point. Each vertex $i \in V$ in the graph represents an interest point (be it pickup, delivery or depot). Each edge $ij \in \mathcal{C}$ has associated two things. The first is *pheromone trail*, τ_{ij} , information left by previous ants (the attractiveness of past visits). The second characteristic of each edge is a cost η_{ij} , translating the effort to transverse that arc (usually distance between nodes), called the *heuristic information*.

Ants *construct solutions* by sequentially deciding which node to visit next, weighting *pheromone trail* information left by previous ants (the attractiveness of past visits) and the *heuristic information* (translating the effort to transverse that arc, usually distance between nodes). The probability of travelling through an edge is then given by Eq. 1.

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \times \eta_{ij}^\beta}{\sum_{j \in \text{feasible set}} \tau_{ij}^\alpha \times \eta_{ij}^\beta} \quad (1)$$

with τ_{ij} being the pheromone trail value and η_{ij} being the heuristic information, both associated with c_{ij} . Parameters α and β are ACO model parameters intended to balance relative weight importance.

Daemon actions is an optional step that includes other actions not based on real ant behaviour, and is further explained in Sect. 2.2. The last step of the algorithm is to *update pheromones*, computed using Eq. 2.

$$\eta_{ij} = (1 - \rho) \times \eta_{ij} + \rho \times f(s_{best}) \tag{2}$$

where ρ is the evaporation coefficient (rate at which pheromone trail values wear off) and $f(s_{best})$ is a value derived from the quality of the best solution s_{best} , commonly called the fitness value.

The decision process is probabilistic since moves with low attractiveness can still be selected, even if less often. To provide a better balance between choosing good moves and exploring less common moves, the *pseudorandom proportional rule* is used, which can be defined as:

- if $q > q_O$, Eq. 1 is used (biased exploration)
- if $q \leq q_O$, next node is dictated by $arg \max_{u \in J_k^i} \{[\tau_{ij}]^\alpha \times [\eta_{ij}]^\beta\}$ (exploitation)

2.2 Local Search

Local search (LS) is intended to exploit the current solution s in search of improvements in the immediate neighbourhood solutions. LS procedures for VRP usually fall into the *edge-exchange* category. Starting from a feasible solution, new solutions are generated by deleting k edges and replacing them with new edges. These connect the same nodes in a different ways, completing the cycle, performing what is known as a *k-exchange*.

A noteworthy local search strategy for the VRP is the *or-exchange* or *or-opt*. Instead of deleting arcs as in *edge-exchange* strategies, a certain chunk of size s from the route, or *slice*, is separated from the rest. The *neighbourhood search space* is composed by all feasible solutions after moving the slice to each position around the original one, up to L steps in every direction. A schema for this can be seen on Fig. 2. Note that when exchanging more than one node, the original travel order of a slice is maintained.

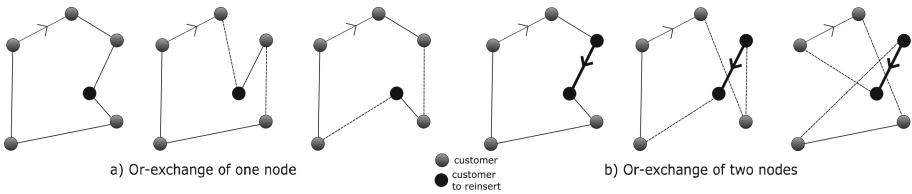


Fig. 2. Or-opt local search strategy for VRP, for $k = 1$ and $k = 2$

3 Proposed Approach

The motivation for this paper is to solve a real world delivery problem. Thus, considering the complex combinatorial challenge it represents, a metaheuristic

algorithm is used to tackle the big solution search space. Ant Colony Optimization is a graph based method, which is an intuitive way to formulate a VRP. To make the algorithm more efficient, the ACO is to be paired with a Local Search to further exploit good solutions by trying small changes and that can lead to big improvements.

The central piece of the approach is the *static solver*, which optimizes an existing solution using a hybrid Ant Colony System paired with Local Search. The initial solution is given by an *initial solution constructor* module, as detailed in Sect. 3.2.

The approach taken to solve dynamic instances is based on the presented solution for static problems. Starting from an initial feasible solution, the *static solver* is applied for a limited period of time. This intends to represent the pre-computation of requests already known beforehand. At this point we are at the beginning of the working span and will next repeat the same set of directives until all requests have been serviced:

- Insert new requests buffered during previous interval into the best routes;
- Deploy the best obtained solution after the insertion to the physical vehicles;
- Predict changes to deployed route at current interval’s end-state, namely new vehicle positions and serviced requests;
- While current interval’s end isn’t reached, optimize the end-state prediction;
- Output from the *static solver* the best and the latest found solutions;
- Group these two solutions with the state of the deployed route to form the best selected routes;
- Update vehicle positions, serviced requests and distance matrix.

3.1 Mathematical Formulation

An adaptation from the well-known mathematical formulation of the VRPTW Hasle (2007) is presented, where the goal is to service as efficiently as possible a set of customers requests $\mathcal{R} = \{1, \dots, n\}$. Every request is defined by a pickup and a delivery location, each represented by a unique graph node out of a total $2n$ nodes. The full set of customers to service is given by $\mathcal{C} = \{p_1, d_1, p_2, d_2, \dots, p_n, d_n\}$ where p_r is the pickup node of request r from the subset $\mathcal{P} = \{1, 3, \dots, 2n - 1\} \subset \mathcal{C}$ and d_r is the delivery node of request r from subset $\mathcal{D} = \{2, 4, \dots, 2n\} \subset \mathcal{C}$.

In order to service each customer request we have available k vehicles. The depot location is split into k nodes forming the set of depot nodes $\mathcal{W} = \{1, \dots, k\}$. The mathematical formulation is dimensioned for a graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$, where $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of graph edges representing all travel possibilities between nodes and $\mathcal{N} = \mathcal{W} \cup \mathcal{C}$ represent the graph nodes. $\mathcal{V} = \{1, \dots, k\}$ is the set of homogeneous vehicles. Each vehicle has a fixed hire cost of e_k and a maximum capacity given by $q \geq l_i$, $i \in \{1, \dots, n\}$ where l_i is the load capacity demand for customer i , i.e. how much of a vehicle’s available capacity a request will occupy.

The variable x_{ij}^k is a binary parameter that expresses if a vehicle travels directly from node i to node j . For each arc ij , it takes the value 1 if vehicle

k travels directly from i to j and 0 otherwise. x_{kj}^k represents an arc between a depot node and node j , serviced by vehicle k . Similarly, x_{ik}^k expresses if an arc between a customer node i and the depot is serviced by vehicle k . Each arc is also defined in terms of travel time, t_{ij} specific positive travel cost, c_{ij} , for each arc in \mathcal{A} . The variable s_i^k is the exact time of service at each point i by vehicle k and $[a_i, b_i]$ is the time window specified for node i . Finally, $\mathcal{M} = [m_1, m_2, m_3, m_4]$ is a vector composed by scaling factors, which define the priority of each term in the objective function.

minimize

$$m_1 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}^k + m_2 \sum_{k \in \mathcal{V}} \sum_{j \in \mathcal{C}} e_k x_{kj}^k + \tag{3}$$

$$m_3 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} \max(s_j^k - b_j, 0) x_{ij}^k + m_4 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} \max(a_j - s_j^k, 0) x_{ij}^k$$

subject to

$$\sum_{k \in \mathcal{V}} \sum_{j \in \mathcal{C}} x_{ij}^k = 1, \quad \forall i \in \mathcal{C} \tag{4}$$

$$\sum_{(i,j) \in \mathcal{A}} l_i x_{ij}^k \leq q, \quad \forall k \in \mathcal{V} \tag{5}$$

$$\sum_{p \in \mathcal{P}} x_{hp_n}^k - \sum_{d \in \mathcal{D}} x_{gd_n}^k = 0, \quad \forall h \in \mathcal{N}, \quad \forall g \in \mathcal{N}, \quad \forall k \in \mathcal{V}, \quad \forall n \in \mathcal{R} \tag{6}$$

$$\sum_{j \in \mathcal{C}} x_{kj}^k = 1, \quad \forall k \in \mathcal{V} \tag{7}$$

$$\sum_{i \in \mathcal{V}} x_{ih}^k - \sum_{j \in \mathcal{N}} x_{hj}^k = 0, \quad \forall h \in \mathcal{C}, \quad \forall k \in \mathcal{V} \tag{8}$$

$$\sum_{i \in \mathcal{V}} x_{ik}^k = 1, \quad \forall k \in \mathcal{V} \tag{9}$$

$$x_{ij}^k (s_i^k + t_{i,j} - s_j^k) \leq 0, \quad \forall (i, j) \in \mathcal{A}, \quad \forall k \in \mathcal{V} \tag{10}$$

$$a_i \leq s_i^k, \quad \forall i \in \mathcal{N}, \quad \forall k \in \mathcal{V} \tag{11}$$

$$x_{ij}^k \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}, \quad \forall k \in \mathcal{V} \tag{12}$$

Equation 3 defines the objective function. It is the sum of four different terms, each multiplied a scaling factor from the vector \mathcal{M} . The first term, $m_1 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}^k$, represents the route specific travel costs. It takes into account the sum of each cost, c_{ij} , for all arcs travelled by each vehicle of the fleet, this is where $x_{ij}^k = 1$. The second term, $m_2 \sum_{k \in \mathcal{V}} \sum_{j \in \mathcal{C}} e_k x_{kj}^k$, gives the cost of hiring vehicles. This value is independent of vehicle travelled distance since such costs are already covered in the first term. It sums the one-off cost of hiring each vehicle in the solution by multiplying each vehicle cost e_k by x_{kj}^k , which

represents leaving the depot. This means that for all unused vehicles there is no travel out of the depot, making $x_{kj}^k = 0$ and thus not considering the unused vehicle cost in the objective function.

The third term, $m_3 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} \max(s_j^k - b_j, 0)x_{ij}^k$, considers how late each node visit is. Deliveries have a specified time window. The start of this time window is hard, but the ending of it is soft. This means that a location cannot be visited before the time window starts, but can be visited after it ends. The third term is used to penalize late arrivals, which is visiting a node after its time window has ended. This does not make the solution unfeasible but has a negative impact on the objective function. The third term sums how late all the deliveries were, and due to the max operator each contribution to total lateness is always equal or greater than zero. While this term handles the end time of a time window, the last term handles the start.

Since deliveries can't be made before the specified time window, if a vehicle arrives earlier it must wait. This wait time is penalized in the fourth term, $m_4 \sum_{k \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} \max(a_j - s_j^k, 0)x_{ij}^k$. This is done similarly to the previous term, but now looking at the difference between the arrival time and the time window start. Both of these terms use the max operator, making this objective function non-linear.

The constraint represented in Eq. 4 assures that all customers are serviced only once. Vehicle capacity constraint is represented by Eq. 5. Equation 6 assures the same vehicle services both pickup and delivery nodes of the respective request. The expression 7 defines that there is only one tour per vehicle, which can be empty. Equation 8 ensures that if a vehicle arrives at a customer location it also departs from the mentioned customer location. Equation 9 define all tours' ending location as the depot. For a vehicle to travel directly from i to j Eq. 10 states the arrival time at customer j is such that it allows travelling between i and j . Expression 11 specifies that the early limit of a time window a_i is hard and Eq. 12 denotes the x_{ij}^k variable as binary. Thus, the model is non-linear due to the non-linear *max* operations in Eq. 3, quadratic terms in Eq. 10 and integrality constraints at Eq. 12.

3.2 Initial Solution Constructor

To the presented method is given as input info on the orders to solve and on the fleet to manage. Orders are given as a list of pickup-delivery pairs, their respective locations and time-windows. Regarding the fleet, to the method is given the total number of vehicles, their capacity, travel speed and cost.

The *initial solution constructor* is then used to generate a feasible solution from scratch, to be used as a starting point for the *static solver* module. Since good starting solutions lead to faster convergence, a proposed new strategy based on Nearest Neighbourhood Search (NNS) and subtractive clustering is presented. Customer locations nodes are clustered, and to each clusters a single vehicle is assigned following the NNS heuristic. A comparison between using only NNS or clustered NNS is showed later in Table 1, which supports the decision to use pre-clustering.

To mitigate the negative effects of clustering locations, and not time availability, the maximum number of allowed vehicles is always used. This helps reducing lateness and excessive waiting times in the initial solutions, even if at the cost of using more vehicles. To do so, an heuristic starts with a very low *cluster influence range* parameter and subtractive clustering is applied on the midpoint of each pickup-delivery pair. Having an extremely low *cluster influence range* results in each midpoint being a cluster centre, i.e, it asks for as many vehicles as there are pickup-delivery pairs. If this parameter creates a solution with more vehicles than allowed by the problem, the parameter is increased and the process repeated until the maximum number of vehicles is in use.

3.3 Static Solver

The *static solver* combines ACO and Local Search. The overall logic can be seen on Fig. 3. Starting from a feasible solution, module specific variables are initialized. The *pheromone trail matrix* is also here generated. It is a $Q \times Q$ matrix, where Q is the length of \mathcal{N} , and it is initialized uniformly at the value $\tau_0 = 0.5$.

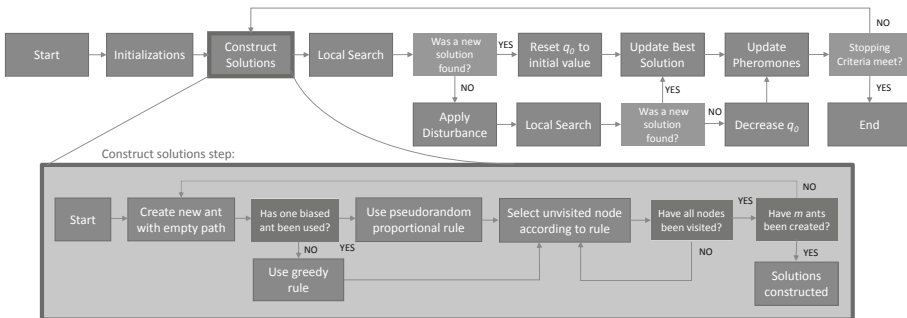


Fig. 3. Static solver, with the module *Construct Solutions* further detailed

Pheromone limits are dynamic and depend on the quality of the current best solution, meaning they will be updated every time a new global best solution, s_{best} , is found. Pheromone trail matrix limits are computed according to the following equations Gambardella (2015):

$$\tau_{max} = \frac{1}{\rho \times f(s_{best})} \tag{13}$$

$$\tau_{min} = \frac{\tau_{max}}{2Q} \tag{14}$$

The ant behaviour is guided by a pseudorandom proportional rule, as explained at Sect. 2.1. One *biased ant* is also used each cycle, where the value of q_0 is set to 1 making this ant greedily select the most attractive move.

At each cycle, the *static solver* can work either with or without the *local search* module. If no new solutions have been found for more than LS_{limit} cycles, as defined in Eq. 15, LS is used. This is done to let the algorithm use ACO to explore the search space without the overhead of the LS as long as solution improvements are being regularly found.

$$LS_{limit} = \frac{\log(\frac{1}{2Q})}{\log(1 - \rho)} \tag{15}$$

When the iterative cycle starts, m new ants are generated using the *new ant* generator module. If an improving solution is found, both s_{best} and pheromone matrix limits are updated. After all ants are computed, the *local search* method is applied to the most promising ants on the top ant fitness list from current iteration (size specified by *top_ants_number*) until no improvement is found, and are compared with the global best ant, s_{best} .

If no best solution was found up to this point on the current cycle, a disturbance is induced on s_{best} ant, and to the disturbed solution s_{new} the *local search* method is applied. The number of new solutions generated by the disturbance method is given by *perturbed_ants_number* times. Every s_{new} is saved on *exchange_memory* to avoid applying a local search on equal solutions and save valuable computational time. The disturbance introduced can either be a *shift*, where a pickup delivery pair is moved from on vehicle to another, or a *switch*, where one pair of each vehicle switch places.

When eventually the time limit is surpassed for the *static solver* module, it stops running iteratively and outputs the best found solution as result.

3.4 Dynamic Solver

To handle the dynamic changes over time, the working horizon is divided into successive intervals of length $time_{ss}$. During each interval, the working conditions (time and distances matrix) will remain unchanged and any new requests appearing during this interval will be buffered, i.e., saved for later insertion. When the end of the interval is reached, time and distance matrix are updated and new requests inserted in the already existing routes. This way, during the length of the interval, the problem instance does not change and the *static solver* method can be applied.

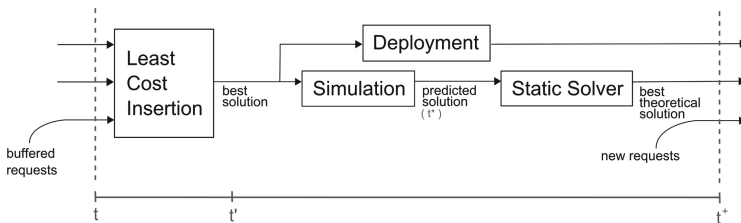


Fig. 4. Cycle of the dynamic solver

After running the static solver to generate a solution from scratch, with the currently known locations, an iterative cycle as represented on Fig. 4 is entered. Denoting the start of the interval as T , the best solutions obtained in the previous optimization interval serve as input to the *least cost insertion* module, which adds any new requests buffered during the previous cycle and outputs the best found solution. This is the solution to be deployed to the physical vehicles, $\dot{s}_{deployed}$, at time T' , which would start to travel immediately accordingly to this route, ignoring any previous orders. Simultaneously to this vehicle deployment, another module called *end-state simulator* predicts where the vehicles will be at the end of the current time interval. This predicted route, \dot{s}^* , is used as input to the *static solver*, which will try to find improvements to this solution for $t_{ss} - (T' - T)$ minutes.

When the end of the time interval T^+ is reached, the *static solver* module is stopped. The best solution from the *static solver*, now \dot{s}_{best} , is grouped with the previously deployed solution, \dot{s}_{best} , and together they serve as input to next cycle's *least cost insertion model*, at T^+ . All requests made available between T and T^+ are now introduced into the routes. The cycle is repeated until no more requests need to be serviced.

When solving dynamic instances, some changes are needed in the *static solver* module to account for requests that have already been partially serviced, this is, whose pickup has already been serviced in the real world and thus is irreversibly tied to a vehicle. This means that the *new ant* module must also account for vehicle history when constructing the feasible node list. When generating new solutions with the local search strategy, any time a feasibility check is done it needs to also take into account nodes outside the current planned route but which are on the vehicle history.

To account for changes between the predicted end-state environment and the real environment at T' , another module is needed to check if at the interval's end the predicted state matches the real state and fix anything needed accordingly. The module *vehicle position update* is responsible for predicting where the state of the system will be at the end of the interval. It works by letting the vehicles follow their current routes until the time of end of interval, using new generated distance and times matrices. For the time being, to these new distance and times matrices random noise is added using the module *vehicle noise*. The *vehicle position update* is also used to re-calculate a predicted route at the end of the time interval, this time with the correct distance and times matrices.

4 Results

On the following section the obtained results are detailed. Before reporting on static and case study performance, parameter tuning and algorithm modifications from previous sections are justified.

The static benchmarks tested are part of the 100 customer group PDPTW instances available at SINTEF (2016). How pickup/delivery locations are distributed geographically can be seen in the file name prefix: *lc* files are clustered

geographically; lr are randomly distributed; lcr mix previous two. Also, file prefixes ending in 1 have short and overlapped time windows and have many vehicles available, while files ending in 2 have long and spaced time windows and few available vehicles.

4.1 Initial Solution Construction

To compare implementing the construction method with and without pre-clustering the customer nodes, Table 1 was generated. Average values of 20 min runs are presented for both options, showing the following values with and without clustering: fitness of initial solution; time until solution with no lateness; best solution fitness after 20 min. The best values in each case are highlighted in bold.

Table 1. Comparing initial solution construction using Nearest Neighbourhood Search with and without pre-clustering with the best values for each comparison in bold

File name	Strategy	Initial Fitness	Time (min)	Final Fitness
lc101	Normal	188.58	0	182.89
	Clustering	190.49	0	182.89
lc201	Normal	9.97E+09	8.04	437.11
	Clustering	5.53E+09	2.41	310.02
lr101	Normal	7.08E+09	2.54	348.73
	Clustering	9.67E+09	2.82	355.91
lr202	Normal	182.89	0	182.90
	Clustering	190.49	0	182.90
lcr101	Normal	8.66E+09	6.42	421.25
	Clustering	3.98E+09	2.19	309.89
lcr201	Normal	7.33E+09	3.57	367.03
	Clustering	1.06E+10	2.60	359.92

Comparing the approach with and without clustering, no consistent improvement can be found for initial fitness and time until a solution with no lateness. However there is almost always a clear fitness improvement after running for 20 min. Pre-clustering the requests and then applying a NNS to each cluster will be the strategy used in all other sections.

4.2 Static Solver Modifications

On Table 2 we can see different runs for various modifications of the parameter q_0 . It was noted that a high q_0 value is especially valuable at the start of the algorithm, but might negatively impact the search for better solutions once we are closer to the optimal solution. With this in mind, the idea of iteratively decreasing q_0 value with each iteration where a new solution isn't found is tested

in the problem at hand. For this formulation, the best combination found was using $q_0 = 0.9$ initially and decreasing it by 20% each iteration with no better solution found.

Next, in Table 3, we see a comparison between having or not a biased ant in each cycle, this is, single ant with $q_0 = 1$ independently of previous q_0 values of decreasing factor. Since this change has a positive impact on the used formulation, it was used for all the following tests.

4.3 Static Instances

Table 4 presents the average values for the error tables per type of file. Clustered data behaves differently from the others files as it manages to always reach the optimal number of vehicles for type 2 files. While for type 1 it does not reach the optimal value for all of them, it reaches a lower distance than the given by the optimal. For the other two the conclusions are similar, with average vehicle number error of 2 or less. Distance does fall below the optimal value as with the clustered files, but instead has an average error around 20%.

Table 2. Different tests done on the pseudorandom proportional rule parameter, q_0

File ID	q_0	Multiplier	Vehicles	Total Distance
18	0	-	24	1960.55
	0.9	-	24	1849.61
	0.9	0.9	23	1995.27
	0.9	0.8	23	1878.41
	0.8	0.5	24	1895.69
23	0	-	15	1440.08
	0.9	-	15	1450.43
	0.9	0.9	16	1555.86
	0.9	0.8	14	1338.17
	0.8	0.5	14	1417.95

Table 3. Test runs with and without biased ant

File	Vehicles		Distance	
	Normal	Biased ant	Normal	Biased ant
lr101	23.33	20.70	1875.33	1775.71
lr109	13.33	13.20	1463.15	1376.57
lr205	4.00	3.60	1224.59	1190.37
lrc101	14.33	13.20	1728.43	1679.72
lrc102	6.00	5.20	1756.58	1763.81

Table 4. Average distance from optimal solutions for static solver

File	Vehicle				Distance			
	Mean	%	Best	%	Mean	%	Best	%
lc1	0.31	3.46	0.22	1.39	-41.49	-4.06	-23.79	-2.00
lc2	0	0	0	0	5.38	0.91	0.96	0.16
lr	0.18	1.83	0.12	1.31	-19.44	-1.72	-12.14	-0.98
lr1	1.39	12.99	0.67	6.57	131.87	11.33	58.50	4.99
lr2	1.05	40.54	0.55	21.96	329.41	35.31	168.63	18.15
lr	1.23	26.17	0.61	13.93	226.34	22.80	111.17	11.29
lcr1	2	16.62	1.5	12.42	161.70	7.34	98.89	2.84
lcr2	1.35	38.51	1	29.16	371.76	28.86	226.77	15.66
lc	1.68	28.93	1.25	21.83	266.73	17.86	162.83	9.07

4.4 Dynamic Case Study

The presented case study is based on food service distribution centres and its most common design constraints. For this example, real customer requests from a typical day of a distribution company are detailed regarding real order hour and location. This specific example models the distribution services of one restaurant.

All pickups happen at the depot location, the restaurant, and it is assumed that after a customer makes an order, the delivery time window starts in 45 min and lasts for 15 min. For this implementation, the distance between real world locations is computed using the Haversine formula. However, the created model can work with any matrix giving the distances and travel times between nodes, for example given by the *Google Maps Distance Matrix API*.

First, the data is processed by the *static solver* module for 30 min, similarly to the approach on the static benchmarks. This will give a solution to be considered as the optimal when performing any dynamic tests. The case study data is similar to the benchmarks in size, with 47 requests to service. In terms of scheduling horizon, the case study matches the type 2 files. Visualization of a solution found can be seen on Fig. 5 a).

For the static run, 2 vehicles were able to service all requests without lateness for a total of 51.4km travelled. For the dynamic solution, with a $t_{sim} = 15$ min and a $t_{look} = 45$ min the solution obtained is represented in Fig. 5 b). It uses 4 vehicles instead of 2 and has a total travelled distance of 68.6 km.

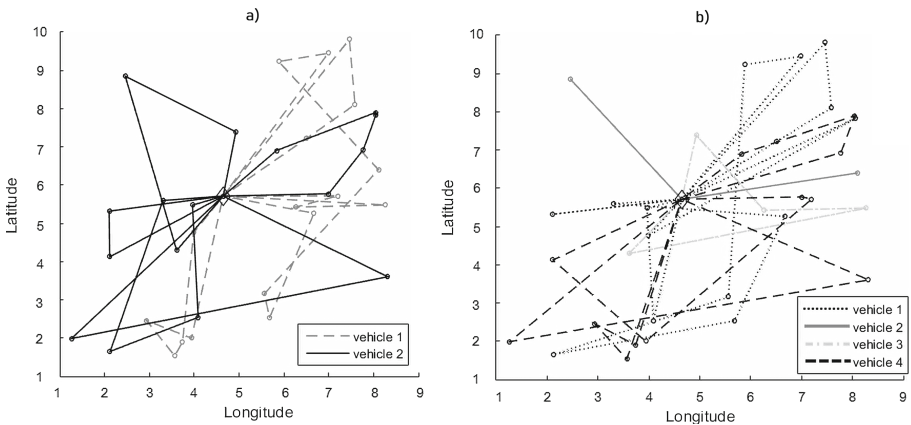


Fig. 5. Comparison of static and dynamic solutions for the case study

5 Conclusions

The main objective was accomplished by defining an algorithm able to solve the original problem. Further, the proposed approach is suitable for implementation

in a real world environment, being able to deal with the tight time windows available to solve such heavily constrained problems. Considering the benchmark problems where the data is not clustered, the proposed approach does not match the competition when comparing with other multi-vehicle pickup delivery problems, except on a few instances. However under the case study presented, the proposed approach is able to give satisfactory solutions within the given time constraints, generating solutions without any delays in the delivery time windows.

In general, the proposed approach shows a good performance in the validation benchmarks. The introduction of the initial clustering step improved the overall results of the proposed approach, only little improvements are needed to consider them competitive with other approaches from the state of the art. The developed strategy for initial solution construction seems very promising and worth exploring further. The hybrid approach improves the solution quality, with the inevitable cost of extra computational time mainly due to the LS module.

5.1 Future Work

The proposed model can be improved by adding the ability to handle different types of vehicles and more advanced waiting strategies. A first attempt to improve the algorithm would be to further improve the Local Search method, namely adding more diversity to different types of solution disturbance. As for the dynamic approach, it would be mandatory for a model applied to a real case to refuse new requests if they will badly influence the already accepted routes.

References

- Bianchi, L.: Ant colony optimization and local search for the probabilistic traveling salesman problem: a case study in stochastic combinatorial optimization. Ph.D. thesis (2006)
- Braekers, K., Ramaekers, K., Nieuwenhuyse, I.V.: The vehicle routing problem: state of the art classification and review. *Comput. Ind. Eng.* **99**, 300–313 (2016)
- Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. *Oper. Res.* **6**(1), 80–91 (1959)
- Ferrucci, F., Bock, S.: Real-time control of express pickup and delivery processes in a dynamic environment. *Transp. Res. Part B: Methodol.* **63**, 1–14 (2014)
- Flood, M.M.: The traveling-salesman problem. *Oper. Res.* **4**, 61–75 (1956)
- Gambardella, L.M.: Coupling ant colony system with local search. Ph.D. thesis (2015)
- Hasle, G., Lie, K.-A., Quak, E.: Geometric modelling, numerical simulation, and optimization: applied mathematics. SINTEF (2007)
- Johnson, D.S., Mcgeoch, L.A.: The traveling salesman problem: a case study in local optimization. In: *Local Search in Combinatorial Optimization*, pp. 215–310 (1997)
- Li, H., Lim, A.: A Metaheuristic for the Pickup and Delivery Problem with Time Windows (2001)
- Mitrović-Minić, S., Krishnamurti, R., Laporte, G.: Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transp. Res. Part B: Methodol.* **38**(8), 669–685 (2004)

- Montoya-Torres, J.R., López, J., Nieto, S., Felizzola, H., Herazo-Padilla, N.: A literature review on the vehicle routing problem with multiple depots. *Comput. Ind. Eng.* **79**, 115–129 (2015)
- SINTEF Applied Mathematics: Transportation Optimization Portal - TOP (2008). <https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/>
- Steiglitz, K., Papadimitrou, C.H.: *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola (1982)
- United Nations: *The World's Cities in 2018 - Data Booklet*. Department of Economics and Special Affaris, Population Division (2018)