Erasmus University Rotterdam
Erasmus School of Economics
Operations Research and Quantitative Logistics

April 5, 2023

Tactical Planning for a Capacitated Vehicle Routing Problem with the Purpose of Deriving Delivery Patterns

Supervisor:
Dr. Remy Spliet

Realised by:
Matthijs van der Kaaden *464381*

Second assessor:
Rick Willemsen

Abstract

The *Tactical Planning for a Capacitated Vehicle Routing Problem* (TPCVRP) in this paper has a secondary objective to plan routes in the same period in the same direction. From the final solution, delivery patterns will be derived. The secondary objective is introduced to increase the quality of these patterns. For each period, a pattern is created and all nodes that are visited in that period, will belong to the pattern. An *Adaptive Large Neighborhood Search with Tabu Search* algorithm is used to solve the TPCVRP. The algorithm uses a new operator to repair the solutions. A real-life data set is used to test the algorithm.

Keywords:
Tactical Planning - Pattern Derivation
Capacitated Vehicle Routing Problem

[0] The content of this thesis is the sole responsibility of the author and does not reflect the view of the supervisor, second assessor, Erasmus School of Economics or Erasmus University.

Contents

1 Introduction

Nowadays, many parcel delivery companies offer you a choice between days on which you want your parcel to be delivered. For web shops with a low demand density, it is often not profitable to visit every region everyday. These web shops can increase their profit if they cluster demand of certain regions on certain days. This way, the delivery trucks will be less likely to drive to every corner of the country for only a few deliveries. These geographical clusters of demand will be referred to as zones or regions.

The case in this paper concerns a company in the Netherlands. They have certain delivery patterns, where each pattern consists of a few zones. For example, the pattern on Monday is to visit zone A and C, the pattern on Wednesday is to visit zone A and B. Customers in zone A can then choose to have their package delivered on either Monday or Wednesday, whereas customers in zone B and C can only have their package delivered on Wednesday and Monday, respectively. In their case, there exist five patterns, one for each day of the week. The pattern consists of all zones that are visited on a particular day.

At the moment, the company made the patterns by hand. For example, pattern A consists of all zip codes ranging from 80xx to 85xx. The zip codes are not structured in a logical fashion for this purpose. As a result, some regions are matched with nonadjacent zones or zones that are adjacent, but difficult to reach. Once they are matched, they end up in the same pattern. It would not be logical to visit these regions on the same day. Therefore, this paper will provide a model to see if the efficiency of the routing patterns can be increased.

Based on historical data of deliveries in each region, optimal routes through the regions will be determined. Due to capacity restrictions, not all regions can be served on the same day. To account for this, a *Tactical Planning for a Capacitated Vehicle Routing Problem* (TPCVRP) will be solved. The TPCVRP is the problem of creating routes and assigning these routes to time windows, in this case days.

One of the decisions to be made, is how many days a week a region is to be visited. This can either be once or twice a week. It is preferable to visit regions twice a week, but for regions that are far away from the depot, the driving distance is too big of a cost. After all, the objective of the TPCVRP is to minimize the total driving distance. The TPCVRP does not include this decision, but for a specific instance, this decision can be derived from the solution.

The solution will be obtained with the use of an *Adaptive Large Neighborhood Search with Tabu*

Search (ALNS-TS). Most operators were obtained from literature, one new operator is introduced. This new operator is specifically designed for this type of problem. A secondary objective is introduced for the purpose of bringing routes in the same period closer together. The idea is that this will increase the quality of the patterns.

The solutions will be evaluated with the *Silhouette Coefficient*, which is a measure for the quality of patterns. Different weights for the secondary objective are tried in order to try and find out what the best value is. It was concluded that the best weight is instance dependent.

The remainder of this paper will be structured as follows. In Section 2, previous research concerning this topic is discussed. Then, Section 3 will describe the problem in more detail. In Section 4, the proposed algorithm is discussed and explained. Next, the results will be discussed in Section 5. Then, a conclusion is given in Section 6. Finally, possible further research is discussed in Section 7.

2 Literature Review

The TPCVRP is a variant on the classical *Vehicle Routing Problem* (VRP). The TPCVRP provides a schedule of which routes need to be driven at what time (or day). Not all variants of the VRP consider the tactical planning component. However, research on other variants can still be very useful.

Battarra et al. (2014) provided an algorithm for the clustered VRP. The fact that it is clustered is the same for this research, but they do not consider tactical planning. They propose a Branch and cut algorithm to solve this problem. Izquierdo et al. (2013) also considered the clustered VRP. In addition, they included a capacitated component. They use the classical formulation for the capacitated VRP and then mainly focus on solving a Travelling Salesman Problem (TSP) within each cluster. Sungur et al. (2008) formulated a variant of the capacitated VRP in which they included uncertainty in demand. They used the *Miller-Tucker-Zemlin* formulation (MTZ) of the VRP, and then derived the robust counterpart of the capacity constraints. The uncertain demand is not used in this paper, but the MTZ formulation does represent the problem concerned in this paper.

Baldacci et al. (2011) came up with an exact algorithm for a tactical planning model. For each customer, they made possible schedules as to which days this customer would be visited. The model could then choose one of these schedules. They solve the model using an exact algorithm.

Hernandez et al. (2017) also solve TPCVRP, but they use a tabu search heuristic to get close to the optimal solution. They also use predetermined possible zones for each node. Both of these papers use predetermined schedule possibilities. Francis and Smilowitz (2006) proposed a formulation where the amount of visits is a decision variable. The problem in this paper will not set options for schedules, this will be decided by the model.

As the VRP is *NP-Hard*, the problem becomes too difficult to solve quickly. To reduce the complexity of the problem, Choi and Tcha (2007) proposed a column generation algorithm. First, they use the *Dantzig-Wolfe decomposition* to get another formulation, this decomposition is succesful in solving the VRP (Desrochers et al. (1992)). The reformulation is a set covering problem, where the variables are routes and the objective is to minimize the costs of the chosen routes. Every customer should be covered by at least one route. Whenever the column generation does not generate another column (route), the current solution will be checked for feasibility. If it is not feasible (non-integer), a Branch-and-Bound algorithm is used to get an integer solution.

Danna and Pape (2005) propose a Branch-and-Price algorithm to find a feasible solution. The Branch-and-Price is known for providing tight bounds on the optimal solution value, but it struggles with providing these bounds fast (Barnhart et al. (1998)). Thus, they propose to include a local search component in the Branch-and-Price algorithm. This speeds up the algorithm while the solution quality is maintained. Their *Restricted Master Problem* (RMP) is also a set covering problem in which every customer should be visited at least once. Whenever a customer is visited more than once, it is removed from the route from which the removal results in the least decrease in overall costs. This new route is then added as a column. For the pricing problem, Fukasawa et al. (2006) and Danna and Pape (2005) propose a shortest path problem which they solve by dynamic programming.

Another approach to deal with the complexity of the problem, is to use heuristics. Ho and Gendreau (2006) propose to use Tabu search with path relinking. Path relinking is an evolutionary algorithm that improves the solution quality and accelerates the heuristic. First, a reference set is build. Five strategies are proposed to do this, for example, if a solution is at one point the best overall solution, it is added to the reference set. Then, a guiding solution and initial solution are selected (again, five strategies are proposed). The idea is that the guiding solution will introduce its attributes into solutions obtained in the Tabu search. Prins (2002) also propose an interesting local search method. They use an insertion heuristic where they insert customers on certain routes based on the increased costs. They consider a 'regret' as well, which resembles the increased costs if a customer is inserted in its second best option. After this is calculated, the customer with the biggest regret is inserted in its preferred route.

Ribeiro and Laporte (2012) propose an *Adaptive Large Neighborhood Search* (ALNS) to solve a Cumulative Capacitated VRP. The objective in this problem is to minimize the sum of the arrival times at the customers. They compared their ALNS algorithm to a pair of Memetic algorithms. The ALNS showed better computation times and solution values, especially for large instances. They discuss several removal/insertion heuristics, where the algorithm decides which heuristic will be used in what iteration. The decision for the heuristic is based on the performance of the heuristics over the past iterations. Pan et al. (2021b) also used an ALNS algorithm, but their decision for the heuristic was a 'roulette wheel selection'. This means that the decision depends on how many times each heuristic was already used. This research was extended by creating a hybrid algorithm (Pan et al. (2021a)). This algorithm combines the ALNS with a Tabu Search, which will be used in this paper.

3 Problem description

The problem is formulated as follows. Let $G = (V, A)$ be a graph with the set of vertices $V = \{0, 1, ..., n\}$, where $1, ..., n$ correspond to zones and 0 to the depot ($V^* = V \backslash \{0\}$). The set of arcs A is defined as follows, $A = \{(i, j) : i, j \in V\}$. A distance matrix is defined over A with c_{ij} as the distance (driving time) between i and $j \in V$. The objective is to create routes that visit every zone, while minimizing the total driving time.

There is a homogeneous fleet of K vehicles, each with a maximum trip duration Q. The trip duration for the vehicles consists of driving time and service time, if a vehicle visits region i, it has to provide service for a certain time (s_i). The planning is done over a set of periods T. Every vehicle can only drive one route per period and each route starts and ends at the depot. In addition to this, every vehicle has a capacity restriction. The vehicles have a predetermined maximum trip duration. The service time and driving time together can not exceed this maximum trip duration.

To make sure that certain regions can be visited more than once a week, the zones are clustered, every cluster $R \subset V$ contains a subset of nodes. Each cluster R needs to be visited a_R times in the available set of periods. Since the aim is to find optimal delivery patterns, it is preferable that routes on the same day are close to each other such that fluctuations in demand can be caught by nearby routes. To account for this, a secondary objective is included with the aim of bringing routes in the same period closer together.

4 Methodology

This section will describe the used solution method in detail. The solution method is an *Adaptive Large Neighborhood Search with Tabu Search* (ALNS-TS). Tabu Search is a good method to escape from local optima (Cordeau et al. (2001)), while the ALNS is a good option for the exploration of the whole feasible space (Ropke and Pisinger (2006)). These two methods are combined to get the strengths out of both methods and find a good solution to the TPCVRP (Pan et al. (2021a)). Algorithm 1 presents the ALNS-TS in an overview.

The ALNS-TS in this paper uses an adaptive search engine. The performance of each operator is measured, based on the performance, an operator will be chosen more often or less frequently. Let w_i be the performance measure of operator i. Given that there are h heuristics, heuristic i is chosen with probability $\frac{w_i}{\sum_{j=1}^{h} w_j}$. The destroy and repair operators have independent weights. In every iteration, the destroy operator removes γ nodes from the solution, which will be reinserted by the repair operator. That is where the Tabu component is introduced. Whenever a node is removed, the in- and out-going arcs are saved in the Tabu list. A node can not be reinserted in the solution next to a node that it was previously next to.

After each ϕ iterations, the weights of the operators are updated. The first ϕ iterations, the weights are all the same. The weights of the operators are updated based on the scores they obtained. The score of an operator is updated after it finds a new solution. If the solution is the best solution so far, the score is increased by σ_1. If the solution is better than the current solution, but worse than the best solution, the operator score is increased by σ_2. If the current solution is non-improving, but is accepted either way, the score gets increased by σ_3. If the solution is not accepted, no score is awarded. The scores are updated as follows, let π_i be the score obtained by operator i and o_i the amount of times the operator was chosen in the ϕ iterations. Then,

$$w_i = \begin{cases} w_i & , if \quad o_i = 0 \\ (1-\eta)w_i + \eta\frac{\pi_i}{o_i} & , otherwise \end{cases} \tag{1}$$

where $0 < \eta < 1$ is called the reaction factor. Which controls how quickly the weights are increased if the operators perform well.

It was mentioned above that a solution can be accepted if it is non-improving. Given a current solution s, the probability of this happening for solution s' is $e^{-(v(s')-v(s))/T}$, where T is the current temperature and $v(s)$ is the objective value of solution s. The temperature is multiplied with the cooling rate c after each iteration. This temperature based approach comes from *simulated annealing* (Osman (1993)).

The objective function consists of multiple components. First, the sum of all used arcs. Second, a penalty M for when not all nodes are in the current solution or when a cluster is not visited its minimum amount of times. And third, an award for when a cluster is visited by multiple vehicles in the same period. The award is calculated as follows. For each cluster, every time period has a certain amount of routes that visit the cluster. The time period with the most visits has the quantity of the visits multiplied with α and subtracted in the objective function. The factor that α is multiplied with, will be referred to as the *cluster factor*.

If no new best solution was found for ω iterations, the current solution is reset to the best solution found so far and the value of γ is increased. The value of γ is increased to widen the search for a better solution. If a new best solution is found, the value of γ is reset to its original value. After Ω iterations without finding a new best, after reaching a predetermined time limit or after I iterations in total, the algorithm is terminated.

Section 4.1 will describe what is done before the initial solution is obtained and how it is obtained. Then, Section 4.2 first describes some general properties of the destroy operators and then it provides a more detailed explanation about each of the individual operators. Section 4.3 is structured the same way, but focuses on the repair operators.

4.1 Preprocessing and initial solution

The preprocessing step will create the nodes. Given a data set, data points will be clustered based on their location, in this case based on the zip-code. These clustered data points will be the nodes in the problem. Since the nodes are ordered by location, the nodes are relatively close to their neighbours. For this reason, the nodes are simply inserted in a route until the vehicle reaches its maximum trip duration. A new route is created and again the nodes are inserted until the trip becomes too long. After all nodes are inserted in a route, the routes are assigned to periods. This will result in the initial solution.

Algorithm 1 Pseudocode ALNS Algorithm

```
Initialize solution $S$
$iter, counter = 0$

while Termination criteria not met do
    Select destroy/repair operator, $d$ and $r$ respectively, using the performance based probabilities
    $S$ = Destroy($S, d$)
    Update Tabu List
    $S$ = Repair($S, r$)
    $S$ = AcceptReject($S$, $T$)
    $T = cT$
    Update scores $d$ & $r$
    if $f(S) \leq f(S_{best})$ then
        $S_{best} = S$
        Reset $\gamma$
        $counter = 0$
    end
    else
        $counter++$
    end
    if $iter = \phi$ then
        Update weights and probabilities of the destroy/repair operators (see equation 1)
        $iter = 0$
    end
    if $counter = \omega$ then
        $S = S_{best}$
        Update $\gamma$
    end
    $iter++$
end
Return $S_{best}$
```

4.2 Destroy operators

This subsection will provide an overview of the destroy operators that are used in the ALNS-TS. The (random) cluster removal operators only removes the cluster if all nodes in the cluster can be removed. This depends on whether the total amount of removed nodes would exceed γ if all nodes from the cluster were to be removed. The other destroy operators repeat the removals until the amount of nodes that are not in the solution is exactly γ. Some nodes might have been unable to be reinserted in the iteration before, if that were to be the case, less nodes would be removed in the current iteration.

4.2.1 Worst Removal

This destroy operator removes the nodes with the worst impact on the solution value. A node is considered bad if the increment on the solution with its in- and out- going arcs is high. Also, if the removal of the node would result in that route not visiting its cluster anymore, and in turn, resulting in a reduction of the cluster factor, the increment increases by α.

4.2.2 Random Removal

While the amount of removed nodes is smaller than γ, this method generates three random numbers. The first one for the period, the second for the vehicle and the third for an index in the vehicles' route. The node at this index is removed from the solution.

4.2.3 Shaw Removal

This operator first selects and removes a random node. A group of γ nodes that are most similar to the first node is iteratively selected. The node that is closest to the original node is added to the group first. Then, the node that is closest to any of the nodes already in the group, is added to the group. This continues until the desired group size is reached. If the group size has the desired size, all nodes in the group are removed from the solution.

4.2.4 (Random) Cluster Removal

While the amount of removed nodes is smaller than γ, the random cluster removal selects a random cluster from which all nodes are removed from the solution.

The cluster removal also removes all nodes from within a cluster. But the clusters are not selected randomly, they are selected based on their increment on the solution. The increment of a cluster is defined as the sum of all in- and out- going arcs of all nodes in that cluster. Also, if the amount of visits to a cluster is not the minimum amount, the increment also considers a penalty. This operator does not consider changes in the cluster factor.

4.3 Repair operators

This subsection will describe the operators that are used to repair the solution again. The insertion heuristics also consider the award α per cluster and the penalties for not having the minimum amount of visits to a cluster. The repair operators will keep inserting nodes until there are no nodes left to insert or until there is no feasible spot for the insertion of the remaining nodes.

4.3.1 Greedy Insertion

For each node that is not in the solution, this operator calculates the best option to insert the node. The best option is defined as the smallest increase in the objective value. After every node had its best option calculated, the best option (node) overall is inserted in the solution.

4.3.2 Two Phase Insertion

This operator calculates the best two options for insertion for each node. The difference in the objective value between these options will be referred to as *regret*. The node with the highest regret, is inserted first.

4.3.3 Insert By Zip One

This operator groups the removed nodes based on the first number of their ZIP-code. Then, for each group, a random node from that group is selected. This node will be inserted in its best position (greedy). After this, all nodes from the same group must be inserted in the same period. Given the construction of the cluster factor, the nodes will be inserted in as many different routes as possible.

5 Results

The testing of the algorithm was done on an Intel i5 PC with 2.60 GHz. A good solution should fit well for the purpose of deriving delivery patterns. The most important criterion for that is that routes in the same period, go in the same direction. To check whether the introduced objective function makes this happen, the solutions for α equal to 0 and for solutions with α greater than 0 are compared. If α is set to 0, the problem becomes a TPCVRP which only minimizes the length of the routes.

For the evaluation of the solution, the *Silhouette coefficient* is used (Rousseeuw (1987)). This coefficient is used to check the quality of the derived patterns. It indicates how well a certain node fits in its selected pattern. The value of the coefficient lies between -1 and 1, where positive values indicate a well fitted pattern and negative values indicate there exists a better pattern for that particular node.

Another interesting performance topic is the performance of the *Insert by Zip One* operator in comparison to the other repair operators. To my knowledge, this operator has never been used before in the literature, that is what makes it interesting to see how well it performs.

In Section 5.1, a description of the used instances is given. Then in Section 5.2, the parameter settings are discussed. Next, Section 5.3 introduces the method of evaluation. Section 5.4 provides statistics on the performance of the algorithm. Finally, Section 5.5 compares the performance of the original repair operator to the performance of the repair operators from literature.

5.1 Instances

The algorithm will be tested on real life instances. A data set with approximately 11.000 orders is used. In preprocessing, the orders are clustered based on their ZIP-code. Every order that has the same first three numbers, will be clustered into one node. If the whole data set is used, this will result in 587 nodes after preprocessing. The clusters will in turn exist of all the nodes with the same first number of their ID's, where the ID's will be the three numbers that the preprocessing based the nodes on. For smaller instances, a sample is taken of the original data set.

The largest instance is the complete data set (587 nodes), then instances of 70, 100 and 200 nodes will be generated. For each of the instance sizes, four instances are generated. The algorithm might perform better in high density areas, as the value of α becomes relatively smaller with higher distances between the nodes. To check this, instances with high node density (or short average distance), will be generated. These instances will be generated with 70 and 200 nodes and they will be referred to as $70H$ and $200H$ respectively. Both $70H$ and $200H$ will have two instances, A and B. The difference between A and B for both instances is the position of the depot. For instances A, the depot is in the center of all demand, while for instances B, the depot is decentralized.

For the largest instance, it might be interesting to see what happens when the minimum amount of visits for each cluster is set to one. This gives the algorithm a lot more freedom. This instance with all minimum visits equal to one, will be referred to as *Instance 587 A*. The same instance with time dependent service times will be referred to as *Instance 587 B*.

5.2 Parameter settings

In Table 1, an overview of the parameter settings is given. Ropke and Pisinger (2006) stated that the value of γ has no necessity to be large. The value of γ will start at five percent of the instance size, and gradually increase to a maximum of ten percent. The segment size ϕ, temperature T, cooling rate c, reaction factor η and values for different σ were found to perform well with the settings in the table (Ribeiro and Laporte (2012)). Values of α will range between 0 and 50, where 0 is to see what the solution would be like if there would have been no secondary objective. For high values of α, the solution might focus too much on getting as many routes through a cluster as possible. The time limit for the algorithm was set to eight hours.

Symbol	**Values**
α	[0, 5, 20, 50]
M	10.000
γ	between 5 and 10 percent of the instance size
I	50.0000
ϕ	50
ω	50
Ω	500
T	the objective value of the initial solution
c	0.99975
η	0.01
σ_1	50
σ_2	20
σ_3	5

Table 1: Parameter settings

5.3 Quality evaluation

To evaluate the quality of the solution, the silhouette coefficient $(s(i))$ is used to see if patterns are logical. The patterns are linked to the periods, so if a node is visited in period t, it belongs to pattern t. The coefficient is constructed as follows, for each node i, the average distance to nodes in the same pattern A is calculated $(a(i))$.

$$a(i) = \frac{\sum_{j \in A} c_{ij}}{|A|} \tag{2}$$

Also, for each other pattern $B \neq A$, the average distance to the nodes within that pattern is calculated. The pattern with the lowest average distance is selected to calculate the silhouette coefficient with $(b(i))$. This is done as follows for node i in pattern A:

$$c(i, B) = \frac{\sum_{j \in B} c_{ij}}{|B|} \tag{3}$$

$$b(i) = \min_{B \neq A} c(i, B) \tag{4}$$

With these values for $a(i)$ and $b(i)$, the silhouette coefficient for node i can be calculated as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{5}$$

This will result in a value between -1 and 1, where 1 would indicate a perfect position and -1 would indicate that the node would be better of in another pattern. The average silhouette coefficient over all nodes will return the silhouette coefficient of the solution. In Figure 1, an illustration is given of how the coefficient is calculated.

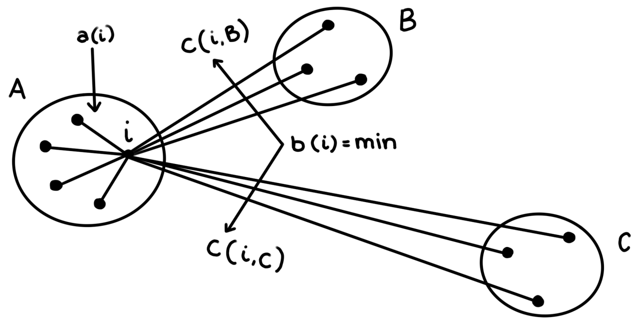


Figure 1: Illustration of how the silhouette coefficient is calculated.

5.4 Solution values

In Table 2, the average silhouette coefficients over all nodes can be found. The instances with 70 and 100 nodes had the algorithm terminate because the maximum amount of iterations were done. For the larger instances, the time limit was reached every time. For the instances with $n = 200$, between 7.000 and 15.000 iterations were performed. Only 1000 iterations were done for the largest instance. As the allocation of routes to periods is not influenced by the secondary objective for α equal to 0, it seems that the derived patterns can be relatively good by coincidence.

From Table 2, it becomes clear that for the instances with size $n = 70$ and $n = 100$, the average

n	α	**Silhouette coefficient**				**Avg. coeff.**	**Avg. sec**
		Instance 70 A	*Instance 70 B*	*Instance 70 C*	*Instance 70 D*		
	0	0.189	0.129	0.247	0.195	0.190	5333
	5	0.142	0.254	0.294	0.205	0.224	4544
70	20	0.206	0.253	0.325	0.131	0.229	6032
	50	0.343	0.205	0.158	0.263	0.242	5635
		Instance 100 A	*Instance 100 B*	*Instance 100 C*	*Instance 100 D*		
	0	0.213	0.134	0.151	0.024	0.131	12351
	5	0.116	0.072	0.121	0.245	0.139	16586
100	20	0.234	0.171	0.074	0.204	0.171	9327
	50	0.181	0.145	0.219	0.218	0.191	10564
		Instance 200 A	*Instance 200 B*	*Instance 200 C*	*Instance 200 D*		
	0	0.189	0.100	0.150	0.190	0.157	28800
	5	0.127	0.201	0.231	0.038	0.149	28800
200	20	0.228	0.289	0.131	0.216	0.216	28800
	50	0.076	0.225	0.198	0.118	0.154	28800
		Instance 587 A	*Instance 587 B*				
	0	0.175	0.193			0.184	28800
	5	0.221	0.180			0.201	28800
587	20	0.242	0.228			0.235	28800
	50	0.227	0.156			0.192	28800
		Instance 70H A	*Instance 70H B*				
	0	0.280	0.128			0.204	5936
	5	0.259	0.074			0.167	7019
70H	20	0.300	0.094			0.197	8340
	50	0.302	0.084			0.193	4252
		Instance 200H A	*Instance 200H B*				
	0	0.284	0.064			0.174	28800
	5	0.274	0.050			0.162	28800
200H	20	0.260	0.179			0.220	28800
	50	0.149	0.119			0.134	28800

Table 2: The average silhouette coefficients and running times for each instance and each α

silhouette coefficient increases with an increase in α. The intention for which α was introduced, seems to have its desired effect according to the averages. However, each individual instance fails to show this pattern. This makes it difficult to draw hard conclusions from the table for these instance sizes.

For the instances with size $n = 200$, there is no clear pattern on either average or individual level. As the allocation of routes to periods is not influenced by the secondary objective for α equal to 0, the other values for α are being focused. If instance 200 C is disregarded, a pattern arises. The coefficient is low for an α of 5, then higher for α equal to 20 and then it becomes lower again for α equal to 50. Only for instance 200 C, this is not the case.

In Section 5.2, it was stated that values of α can become too high. As the algorithm might switch its focus solely to getting as many routes as possible through a certain cluster, while sacrificing a lot of route efficiency. For the instances with $n = 200$ and $n = 587$, the best value for α is 20. With a value of 50, the patterns became worse (on average and individually). This was mainly due to the fact that a few nodes were visited on illogical routes, which put these nodes in a completely wrong pattern. Since the algorithm terminated due to the time limit, it is possible that this was not yet the optimal solution.

For $n = 587$, *Instance 1* is an instance for which all minimum visits are set to one, where *Instance 2* has service time dependent minimum visits. The table shows that the patterns lose quality when there are minimum amounts of visits imposed. This makes sense, as the secondary objective is introduced to get as many routes as possible in the same period through the same cluster. This is exactly the opposite of making sure a certain cluster is visited in more than one period. Since there can also be some overlap in the patterns, the derived patterns might imply multiple visits per time horizon already. However, there still remains a choice to be made: customer satisfaction vs. pattern quality.

For the instances with $n = 70H$ and $n = 200H$, it is difficult to find any patterns. As stated before, α can become too high. For instances with low average distances, this logically happens a lot faster. For both $70H$ and $200H$, instances A performs a lot better. As stated in Section 5.1, instances B have a decentralized depot. With a decentralized depot and high demand density, the algorithm does not perform well. If the depot is in a central position, the silhouette coefficients are higher, but it is still difficult to draw conclusions from this part of the table. Again, there does not seem to be a pattern in the values of the coefficients.

In Table 3, the objective values of the solutions can be found. It is expected that the objective value decreases when α increases. Given two exact copies of a solution for different α, a higher α would provide a lower objective value. From the table, it becomes clear that most of the values match this expectation. There are four exceptions, for instances *100 D*, *587 A*, *70H A* and *200H B* the objective value increases when α is increased from 0 to 5. It can be concluded that the algorithm did not find the optimal solution yet.

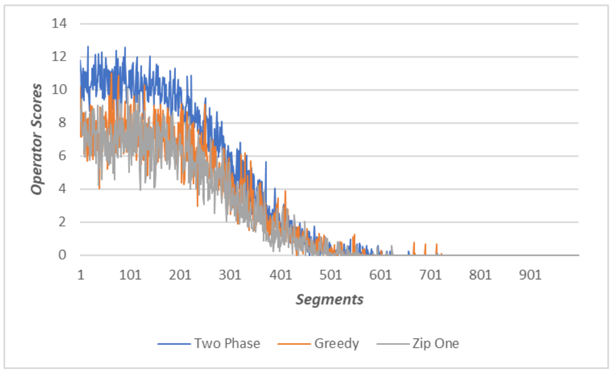
n	α	**Objective Values**				**Avg. obj.**
		Instance 70 A	*Instance 70 B*	*Instance 70 C*	*Instance 70 D*	
70	0	2.138	2.314	2.462	2.150	2.266
	5	2.071	2.258	2.373	2.039	2.185
	20	1.842	2.083	2.140	1.826	1.973
	50	1.315	1.476	1.671	1.352	1.454
		Instance 100 A	*Instance 100 B*	*Instance 100 C*	*Instance 100 D*	
100	0	3.317	3.341	3.311	3.138	3.277
	5	3.287	3.278	3.055	3.199	3.205
	20	2.954	2.943	2.728	2.718	2.836
	50	2.281	2.210	2.225	2.205	2.230
		Instance 200 A	*Instance 200 B*	*Instance 200 C*	*Instance 200 D*	
200	0	5.672	6.016	5.770	5.841	5.825
	5	5.445	5.907	5.662	5.671	5.671
	20	5.197	5.566	5.383	5.310	5.364
	50	4.353	4.636	4.519	4.421	4.482
		Instance 587 A		*Instance 587 B*		
587	0	17.163		17.336		17.250
	5	17.369		17.287		17.328
	20	16.438		16.543		16.491
	50	15.394		15.293		15.344
		Instance 70H A		*Instance 70H B*		
$70H$	0	1.831		3.102		2.467
	5	1.843		2.992		2.418
	20	1.715		2.912		2.314
	50	1.543		2.867		2.205
		Instance 200H A		*Instance 200H B*		
$200H$	0	4.186		6.966		5.576
	5	4.132		7.130		5.631
	20	3.981		6.752		5.367
	50	3.591		6.562		5.077

Table 3: The objective values for each instance and each α

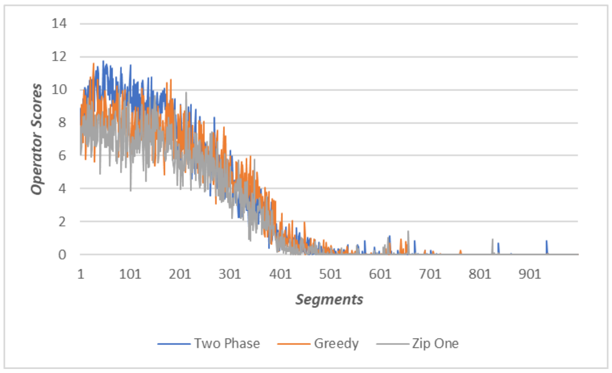
5.5 The Insert By Zip One operator

To see how this operator performs, the scores obtained through the iterations will be compared with the scores of the other repair operators. Since the design of this operator was specifically created for the purpose of α, it is interesting to see how different values of α affect the performance of this operator.

It is also interesting to see how the operator performs in different instance sizes. As the operator is focused on clusters, it might perform better for instances with large clusters. Also, some operators might perform relatively better in early or later stages of the algorithm, for that reason, the average score over the iterations can be found in Figures 2-5. The performances are compared between $\alpha = 0$ and the best α for each instance size. On the vertical axes, the scores obtained through the iterations can be found (as described in Section 4).

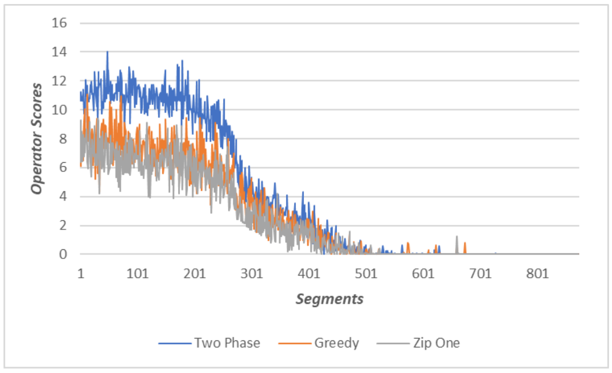


(a) $\alpha = 0$

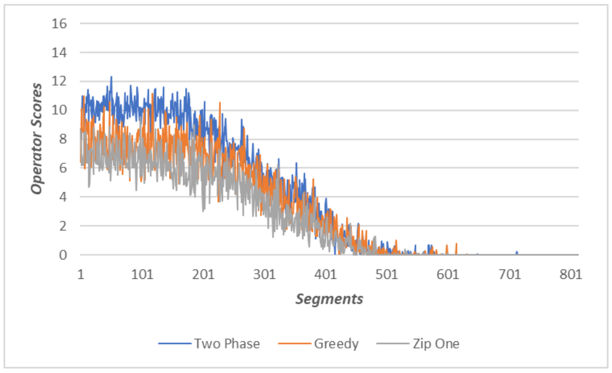


(b) $\alpha = 50$

Figure 2: The average scores over the segments for n=70

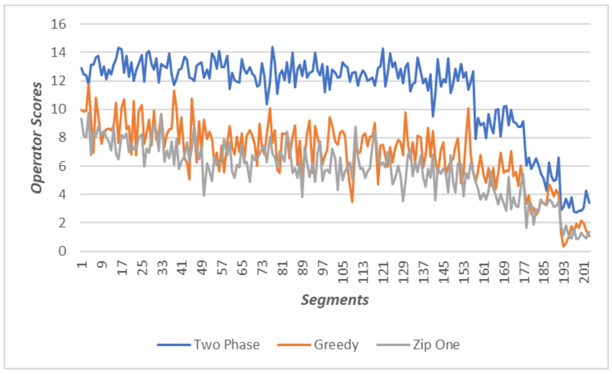


(a) $\alpha = 0$

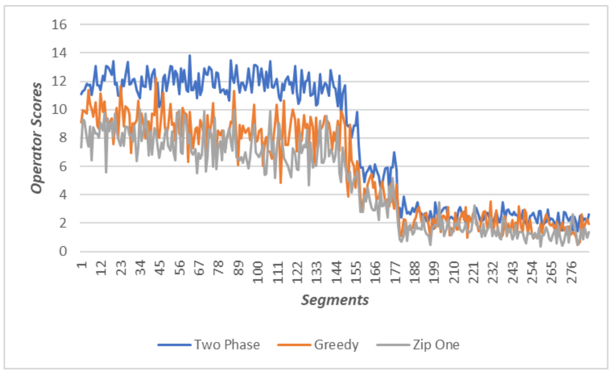


(b) $\alpha = 50$

Figure 3: The average scores over the segments for n=100

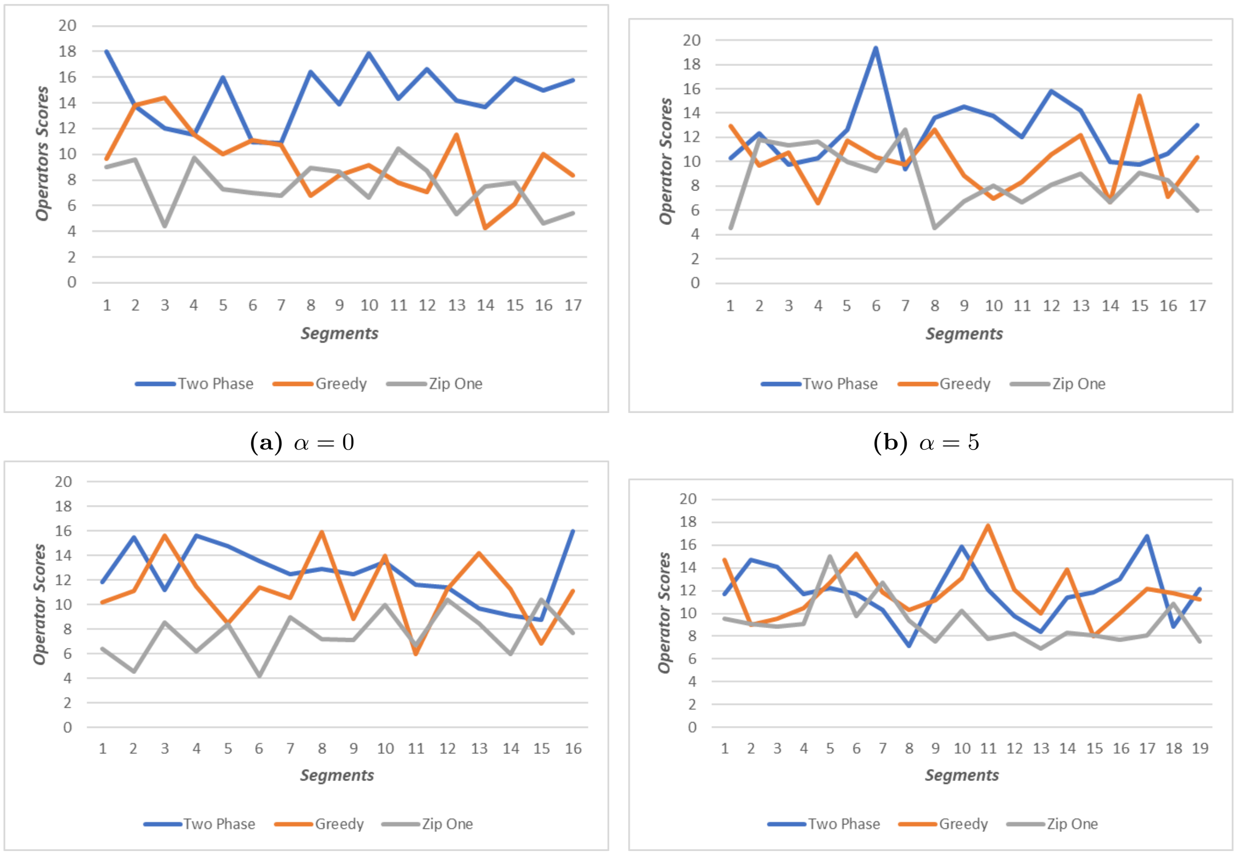


(a) $\alpha = 0$



(b) $\alpha = 20$

Figure 4: The average scores over the segments for n=200



(a) $\alpha = 0$ **(b)** $\alpha = 5$

(c) $\alpha = 20$ **(d)** $\alpha = 50$

Figure 5: The average scores over the segments for n=587

In Figure 2 and 3, the obtained scores do not differ much for the operator for different values of α. In Figure 4, the score is higher for an α equal to 20. Also, in Figure 5, the best performance of the operator is observed for an α equal to 50.

For the smaller instances, the value of α does not matter for the performance. For the larger instances however, a higher value for α seems to increase the performance. The fact that the performance does not change for the smaller instances might be caused by the size of the clusters. The amount of clusters stays the same, which decreases the density of the clusters. Therefore, it is less likely that a lot of nodes from the same cluster are removed and left to be inserted by this operator. If that is the case, the operator starts to look like the greedy insertion heuristic.

6 Conclusion

This Section will discuss the final conclusions. For instances with less than 200 nodes, the average silhouette coefficient increases with an increase in α. However, on an individual level, there is no clear pattern in the performance of the algorithm. It is therefore hard to derive conclusions from these results.

For the instances with 200 or more nodes, if the values for α equal to 0 (in Table 2) are disregarded, there seems to be a pattern that goes up and down again for an increasing α (instance $200C$ excluded). The coefficient is best for α equal to 20, with a lesser performance for an in- or decreased α. From this it can be concluded that the value for α can be too high or too low.

The high density instances ($70H$ and $200H$) did not produce clear patterns in the silhouette coefficients either. This might be because the value of α became too large too quickly. It is possible that with more tuning of α, there would have been more clear results.

The algorithm does not perform well when there are minimum amounts of visits imposed for each cluster. This works in opposite direction of the secondary objective. This algorithm is not recommended for companies for which customer satisfaction is high in regard, because the company is likely to want to visit customers multiple times a week. If the main problem is to be able to process all orders, this algorithm can provide decent patterns.

7 Discussion

For future research, it might be interesting to see if the value for α can be determined based on the data set. This paper concluded that the decision for α needs to be taken with care. So it is interesting to find a way to support the decision making.

Since the patterns only need to be changed when there is a big change in the customer base, the algorithm does not have to run that often. For this reason, it is possible to let the algorithm run for much longer than eight hours. The resulting patterns will be used until there is a big change in the customer base again. Given that the instances with 200 nodes already terminated because the time limit was exceeded, it might be interesting to see what the algorithm could achieve with a longer running time for the largest instance ($n = 587$).

Another possibility to do more research on, is the performance of the introduced repair operator. The average score of the operator was not as high as the other repair operators, but the operator might be good at escaping local minima. There was no data available to test this in this paper. It is also possible to make the insertion procedure of the Insert By Zip One operator a regret insertion instead of greedy.

Bibliography

Baldacci, R., Bartolini, E., Mingozzi, A., and Valletta, A. (2011). An exact algorithm for the period routing problem. *Operations research*, 59(1):228–241.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329.

Battarra, M., Erdoğan, G., and Vigo, D. (2014). Exact algorithms for the clustered vehicle routing problem. *Operations Research*, 62(1):58–71.

Choi, E. and Tcha, D.-W. (2007). A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(7):2080–2095.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8):928–936.

Danna, E. and Pape, C. L. (2005). Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In *Column generation*, pages 99–129. Springer.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354.

Francis, P. and Smilowitz, K. (2006). Modeling techniques for periodic vehicle routing problems. *Transportation Research Part B: Methodological*, 40(10):872–884.

Fukasawa, R., Longo, H., Lysgaard, J., Aragão, M. P. d., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511.

Hernandez, F., Gendreau, M., and Potvin, J.-Y. (2017). Heuristics for tactical time slot management: a periodic vehicle routing problem view. *International transactions in operational research*, 24(6):1233–1252.

Ho, S. C. and Gendreau, M. (2006). Path relinking for the vehicle routing problem. *Journal of heuristics*, 12(1):55–72.

Izquierdo, C. E., Rossi, A., and Sevaux, M. (2013). Modeling and solving the clustered capacitated vehicle routing problem. In *Proceedings of the 14th EU/ME Workshop*, pages 110–115.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451.

Pan, B., Zhang, Z., and Lim, A. (2021a). A hybrid algorithm for time-dependent vehicle routing problem with time windows. *Computers & Operations Research*, 128:105193.

Pan, B., Zhang, Z., and Lim, A. (2021b). Multi-trip time-dependent vehicle routing problem with time windows. *European Journal of Operational Research*, 291(1):218–231.

Prins, C. (2002). Efficient heuristics for the heterogeneous fleet multitrip vrp with application to a large-scale real case. *Journal of Mathematical Modelling and Algorithms*, 1(2):135–150.

Ribeiro, G. M. and Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & operations research*, 39(3):728–735.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Sungur, I., Ordónez, F., and Dessouky, M. (2008). A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. *Iie Transactions*, 40(5):509–523.