Routing vehicles with inventory constraints

Kees Jan Roodbergen, Arjan S. Dijkstra, Stuart Zhu, Maartje H. Jacobs, Kevin J. Mann,
Willem Scherphof, Friso Kramer, Dion Witteveen, Ruben te Wierik

University of Groningen
Faculty of Economics and Business
P.O. Box 800
9700AV Groningen
<k.j.roodbergen@rug.nl>

Introduction

In this study we investigate the complexities of scheduling deliveries from a number of storage points to locations with demand. The study is inspired by the situation at ITRACT project partner Alliance Healthcare. Products (in the case of Alliance Healthcare: medicines) are available at one or more distribution centers. These products need to go to individual consumers (patients). A number options exist and must be chosen from, which is known as multi-channeling. For example, products may be shipped from a distribution center to a store (pharmacy), where the consumer comes to pick up the product. Or products may be home delivered to consumers from either the distribution center or from a nearby store. Furthermore, Alliance Healthcare is investigating options for self-service unmanned lockers, where patients can pick up medicines.

Numerous arguments play a role in selecting the right channel for getting products delivered. First, costs are an important factor, especially for commercial companies. However, other factor may be equally or more important, depending on the application. For example, if instead of products, we need transportation for people, factors timing and comfort (e.g., fewer stops) are vital. Sustainability of the operation is a crucial concern as well. Sustainable transport options for both people and freight flows are the key factor for economic success. Part of the ITRACT project aims at developing incentivizing and scheduling strategies to influence consumers' demand towards more sustainable and less resource intensive transport options, which can be a great stimulation for users and suppliers and are therefore an important indicator for the success of the models. This is especially true for sparcely populated areas in the North Sea Region. In this context understanding the price of offering certain services, such as a delayed pick-up and/or arrival is key to dynamic pricing of transport features. It is in this area that this paper makes a contribution by creating models that can be used to determine pricing trade-off effects.

More technically formulated, we consider a number of locations with inventories of products (for example, libraries with books, or pharmacies with medicines). And consider customers with known demand for products. A number of vehicles, with given, diverse home bases, are available for transporting products. Vehicle capacity does not impose limits in practice, but there is a maximum travel time per vehicle. The objective is to minimize transportation costs, under the condition that as much demand must be fulfilled as possible. That is, if total supply suffices for total demand, then all demand will be met. This model will thus include home delivery in the most economical way in the scheduling of daily operations.

Three models have been created, programmed and tested for the described situation. These models can be helpful in other work projects of ITRACT in determining additional cost parameters, and to provide insights in the logistics costs of sustainability or service choices.

In Chapter 1, we present an Iterated Local Tabu search. Chapter 2 gives a Hybrid Genetic Search method, and Chapter 3 provides a Variable Neighborhood Search method. The three methods each have their advantages and disadvantages when applied to the problem at hand. The Iterated Local Tabu Search method shows some underperformance when compared to the other two methods, which may be further improved upon in the future by adding a single-route improvement neighborhood, such as *k*-opt, and by tuning parameters on a more extended test set. The hybrid Genetic Search method appears to work well, though computation times limit scalability of the algorithm. The Variable Neighborhood Search algorithm also provides good results. Table 1 provides an overview of results of the three methods. Details on the mathematics, results, literature and experiments can be found in the three chapters.

	Average All Instances	**Average Small Instances**	**Feasible**	**# best solution**
Iterated Local Tabu Search (Chapter 1)	1003.03	769.55	922	65
Hybrid Genetic Search (Chapter 2)	909.73	729.18	984	781
Variable Neighborhood Search (Chapter 3)	977.60	760.37	1000	155

Table 1. Comparison of results of three algorithms for the multi-depot vehicle routing variant.

The first column of Table 1 provides average route length across all instances tested, and thus gives an indicator for overall performance (lower is better). The second column provides the average route length across 310 smallest instances, and gives an indicator for performance across the "easier" instances. The third column provides a counter for the number of times the algorithm succeeded in finding a feasible solution to the problem, where 1000 indicates a perfect score (higher is better). The last column provides a counter for the number of times the algorithm found a solution that is equal to or better than the other two algorithms.

Chapter 1
Iterated Local Tabu Search

M.H. Jacobs, K.J. Mann

1 Introduction

Many companies have to solve problems considering finding a route from one place to another. Consider for example a company which delivers goods to their customers at home, and a waste processing company having garbage trucks picking up garbage at certain places. Or think of a postman who has to deliver the mail. Beforehand, the order in which customers are visited is not given. There are many possible routes these trucks can drive, but often the costs increase as the route becomes longer. Therefore, the problem is to find the shortest route, starting at the company, passing all the customers and then returning to the company. There are also companies with multiple storehouses from where the customers are served. Now the problem becomes more difficult because it is not certain which customer should be served from which storehouse in the optimal way. Another extension of the general problem is that there is not one product that has to be delivered, but that there are multiple products distributed over multiple storehouses.

The Vehicle Routing Problem (VRP) as introduced by Dantzig and Ramser (1959) deals with the design of an optimal set of routes for a fleet of vehicles from a central depot in order to serve a given set of customers. It is one of the most widely studied problems in Operations Research. Many generalizations of the VRP exist, among these is the Multi-Depot Vehicle Routing Problem (MDVRP) in which multiple depots together have to satisfy total customer demand by optimally dividing deliveries and designing the corresponding vehicle routes. Other examples of generalizations of the VRP are the VRP with time windows, where the delivery has to be within a certain time interval; the capacitated VRP, where the vehicles have a limited capacity of cargo space; the periodic VRP, where each customer has to be visited a certain number of times in a horizon of T time units and the VRP with pickup and delivery, where vehicles need to pickup items at some locations which have to be delivered at other locations. Combinations of these generalizations are possible.

As an $\mathcal{NP}$-hard problem (see Lenstra and Kan, 1981), large instances of the VRP and its generalizations are hardly solved to optimality and heuristics are commonly used in practice. There are several recent well performing (meta-)heuristic approaches. Pisinger and Ropke (2007) transform various generalizations of the VRP into a pickup and delivery model and solve it with an adaptive large neighborhood search framework. Variable neighborhood search (Hemmelmayr et al., 2009) uses an iterative improvement algorithm with multiple neighborhoods. Ant colony optimization is based on the food seeking process of ant colonies, where the MDVRP is solved using a virtual central depot which corresponds to the nest, the original depots correspond to the entries of the nest and the customers correspond to food. Yu et al. (2011) developed a parallel improved ant colony optimization heuristic with a coarse-grain strategy. Parallel iterated tabu search is used by Cordeau and Maischberger (2012) for multiple variants of the VRP. The hybrid genetic algorithm from Vidal et al. (2012) uses new population-diversity management mechanisms to allow a broader access to reproduction, while preserving the characteristics of elite solutions.

Common assumptions in literature are unlimited depot supply and a demand for a homogeneous good, whereas in practice it is easy to imagine these assumptions will fail to hold. In this paper we relax these assumptions and develop an iterated local tabu search heuristic for the multi-product MDVRP with limited depot supply and test its performance on a set of 1000 problem instances.

The layout of the article is as follows. In Section 2 we formulate the mathematical model and in Section 3 we will go over our solution approach. The Iterated Local Search Heuristic will be explained in Section 4. After that, in Section 5 the values of the parameters are determined and the results are shown in Section 6. The conclusion can be found in Section 7.

2 Problem Formulation

We define the MDVRP on a complete graph $G = (V, E)$ with the vertex set $V = \{v_1, \ldots, v_t, v_{t+1}, \ldots, v_{t+n}\}$ consisting of t depots and n customers, and the edges $E = \{(v_i, v_j) : v_i, v_j \in V\}$. Associated with the edges is a cost function $c : E \to \mathbb{R}^+$ representing the costs of traversing edge $e \in E$. We represent all vertices by their Euclidean coordinates and correspondingly let $c_{ij} = c(v_i, v_j) = \|v_i - v_j\|$, the Euclidean distance between v_i and v_j. Vertex i has a given demand $Q_{hi} \geq 0$ for product $h \in \{1, \ldots, p\}$ and, as only customers can demand products, $Q_{hi} = 0$ for $i \in \{1, \ldots, t\}$. Furthermore, depot $\ell \in \{1, \ldots, t\}$ has a given limited supply $S_{h\ell} \geq 0$ of product h. We assume that total depot supply is sufficient to meet total customer demands for each product, i.e., $\sum_{\ell=1}^{t} S_{h\ell} \geq \sum_{i=t+1}^{t+n} Q_{hi}$ for all products h. In Figure 1 an instance of the MDVRP is shown.

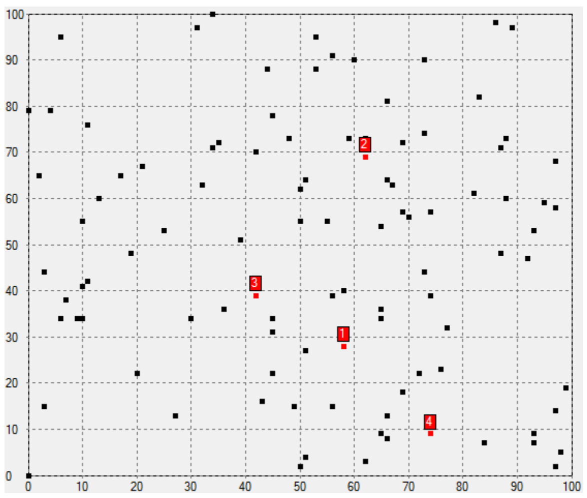


Figure 1: An instance with 4 depots and 100 customers. The x- and y-axes represent the x- and y-coordinates respectively.

Each depot ℓ has a fleet of m_ℓ homogeneous vehicles to deliver products to customers. Each of these vehicles can travel a maximum distance D per day. We require D to be such that any customer can feasibly be visited by a vehicle from at least one depot. We will from now on assume that the number of vehicles per depot is non-binding and that the capacity of these vehicles is large enough to accommodate any delivery. Furthermore, a vehicle route is required to both start and end at the same depot and no other depots can be visited for pick-ups along the way.

To formulate this problem as a mixed integer linear program (MILP), we define the following decision variables:

$$x_{ijk\ell} = \begin{cases} 1 & \text{if vehicle } k \text{ from depot } \ell \text{ visits vertex } j \text{ immediately after vertex } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$b_{hi\ell} = \text{the amount of product } h \text{ supplied to vertex } i \text{ by depot } \ell.$$

We minimize the total travel costs of vehicles. That is,

$$\min \sum_{\ell=1}^{t} \sum_{k=1}^{m_\ell} \sum_{i=1}^{t+n} \sum_{j=1}^{t+n} c_{ij} x_{ijk\ell} \tag{1}$$

subject to

$$\begin{aligned}
\sum_{i=1}^{t+n} x_{igk\ell} - \sum_{j=1}^{t+n} x_{gjk\ell} &= 0 && \forall g, k, \ell && (2)\\
\sum_{j=1}^{t+n} x_{\ell jk\ell} &\leq 1 && \forall k, \ell && (3)\\
z_{hijk\ell} &\leq Q_{hi} x_{ijk\ell} && \forall h, i, j, k, \ell && (4)\\
z_{hijk\ell} &\geq 0 && \forall h, i, j, k, \ell && (5)\\
z_{hijk\ell} &\leq b_{hi\ell} && \forall h, i, j, k, \ell && (6)\\
z_{hijk\ell} &\geq b_{hi\ell} - Q_{hi}\left(1 - x_{ijk\ell}\right) && \forall h, i, j, k, \ell && (7)\\
\sum_{\ell=1}^{t} \sum_{k=1}^{m_\ell} \sum_{j=1}^{t+n} z_{hijk\ell} &= Q_{hi} && \forall h, i && (8)\\
\sum_{i=1}^{t+n} b_{hi\ell} &\leq S_{h\ell} && \forall h, \ell && (9)\\
\sum_{i=1}^{t+n} \sum_{j=1}^{t+n} c_{ij} x_{ijk\ell} &\leq D && \forall k, \ell && (10)\\
\sum_{v_i \in S} \sum_{v_j \in S} x_{ijk\ell} &\leq |S| - 1 && \forall k, \ell, S \subseteq V \setminus \{v_1, \ldots, v_t\}; |S| \geq 2 && (11)\\
x_{ijk\ell} &\in \{0, 1\} && \forall i, j, k, \ell && (12)\\
b_{hi\ell} &\in \{0, \ldots, Q_{hi}\} && \forall h, i, k, \ell && (13)
\end{aligned}$$

Constraints (2) ensure that when a vehicle from a given depot arrives at a customer, it also leaves that customer. Constraints (3) guarantee that each vehicle from a given depot is used at most once. Constraints (4)-(7) model the linearization of the product of $b_{hi\ell}$ and $x_{ijk\ell}$. Constraints (8) ensure that each customers demand for each product is met by the total deliveries. Constraints (9) are in place to limit the total delivery of each product by a given depot, by that depots supply of each product. Limits on route length are imposed through constraints (10). Finally, constraints (11) are standard subtour elimination constraints. The objective function (1) and constraints (2), (3), (10), (11) and (12) are taken from Cordeau et al. (1997), but adjusted to suit this particular problem.

3 Solution approach

VRP is $\mathcal{NP}$-hard, which means there does not exist a polynomial time algorithm to solve it, unless $\mathcal{P} = \mathcal{NP}$. As a generalization of the VRP, the multi-product MDVRP is even more difficult to solve, and thus $\mathcal{NP}$-hard as well. Therefore, we choose a heuristic approach.

To solve instances of the generalized MDVRP, we propose an iterated local tabu search heuristic. Starting from an initial solution, Iterated Local Search (ILS) uses a local search algorithm to search the solution space, and a perturbation algorithm to diversify the search space and prevent the algorithm getting trapped in local optima. The search is further diversified by allowing the local search algorithm to restart its search from previously found solutions. For an elaborate introduction on ILS, see Lourenço et al. (2003).

The local search algorithm used within the ILS is tabu search. Starting from a solution s, the tabu search iteratively moves to the best neighboring solution s' from the set $N(s)$ of all solutions neighboring s. Solutions are evaluated using objective function $f(s)$. Since we allow infeasible solutions during the search, the constraints with respect to the route duration and the delivery of the products are relaxed. Excesses of route duration and deliveries are penalized. The objective is given by $f(s) = c(s) + \alpha d(s) + \beta q(s)$. Here, $c(s)$ is the route duration and α and β are the penalty coefficients. $d(s)$ and $q(s)$ are the excess route duration and excess delivery respectively. They are given by

$$\begin{aligned} c(s) &= \sum_{\ell=1}^{t}\sum_{k=1}^{m_\ell}\sum_{i=1}^{t+n}\sum_{j=1}^{t+n} c_{ij}x_{ijk\ell}, \\ q(s) &= \sum_{\ell=1}^{t}\sum_{h=1}^{p}[(\sum_{k=1}^{m_\ell}\sum_{i=1}^{t+n}\sum_{j=1}^{t+n} z_{hijk\ell}) - S_{h\ell}]^+, \\ d(s) &= \sum_{\ell=1}^{t}\sum_{k=1}^{m_\ell}[(\sum_{i=1}^{t+n}\sum_{j=1}^{t+n}(c_{ij})x_{ijk\ell}) - D]^+. \end{aligned}$$

The penalty coefficients α and β are set to 1 at the start of the tabu search, and they are adjusted throughout the optimization. This is done by multiplying or dividing α and β by $1 + \delta$, depending on whether their corresponding penalty term is positive or zero respectively. We let δ be chosen randomly from an interval at the start of each tabu search phase. This interval is determined in Section 5. Adjusting the penalty coefficients pushes the search towards feasible solutions.

To diversify the search space and prevent getting trapped in local optima, solution possessing attributes of recently searched solutions are declared tabu for a number of iterations, unless a certain aspiration criterion is met. This means that these solutions may not be searched during the period that they are tabu, unless their objective meets some threshold. For an elaborate introduction on tabu search, see Glover (1990).

4 Iterated local tabu search heuristic

In this section the ILS heuristic with tabu search is described in depth. The heuristic starts from an initial solution s_0, which is improved before the iterated local search is started. The improved solution is called $\hat{s}$ and if the improved solution is feasible, it is called s^*, which is the best feasible solution found. The iterated local search heuristic uses multiple phases in each iteration: a perturb phase, an improve phase and an acceptance phase. In the perturb phase, solution $\hat{s}$ is perturbed to solution s'. This solution is improved, which results in solution $\tilde{s}$. If the improved solution is accepted, the next perturbation will be done with $\tilde{s}$, otherwise, $\hat{s}$ will be perturbed again. For every improved solution, the objective is compared with the objective of the best feasible solution found. If the objective is lower and feasible, the best feasible solution will be $\tilde{s}$. The improvement phase is the most time consuming phase in the iterated local search. Every improvement phase contains multiple improve iterations. Therefore, the termination of the iterated local search is determined by the number of improve iterations within an improvement phase. The iterated local search heuristic is given in Algorithm 1.

```
$s_0 \leftarrow$ initial solution
$s^* \leftarrow \hat{s} \leftarrow$ improve($s_0$)
while $\neg$ termination do
    $s' \leftarrow$ perturb($\hat{s}$)
    $\tilde{s} \leftarrow$ improve($s'$)
    if accept($\tilde{s}$) then
        $\hat{s} \leftarrow \tilde{s}$
    end
    if $f(\tilde{s}) < f(s^*)$ then
        $s^* \leftarrow \tilde{s}$
    end
end
return $s^*$
```

Algorithm 1: The iterated local search heuristic.

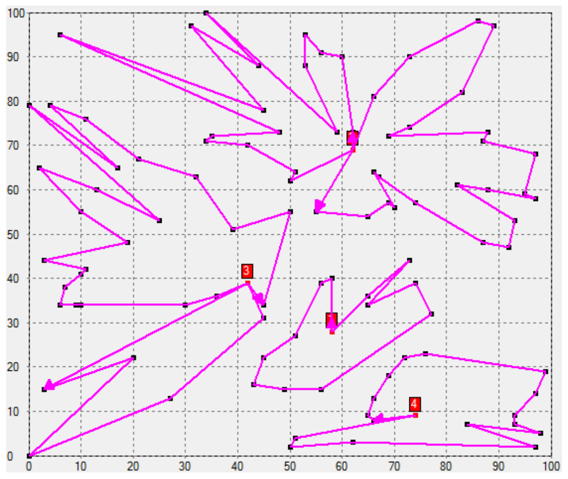


Figure 2: Initial solution of the instance from Figure 1.

4.1 Initial solution

We find an initial solution in which every customer is visited. It is assumed that the initial solution may be infeasible due to excess deliveries with respect to depot supply. As the number of vehicles per depot is assumed to be non-binding, there will no excess route duration in the initial solution. The initial solution is found using the sweep algorithm as first introduced by Wren and Holliday (1972).

First, every customer is assigned to its nearest depot. Second, we find for each depot its nearest customer. Then, for each depot its assigned customers are sorted in non-decreasing order of the angle they make with the depot and the nearest customer of that depot. The sorted customers make a giant tour with the depot. The first vehicle of a depot then goes to the first customer. As we require the initial solution to be feasible with respect to the maximum route length, when a vehicle is at a customer, the vehicle is allowed to go to the next customer in the list if from there it can also feasibly return to the depot. This check is repeated at the next customer in the list. If for some customer route length feasibility is violated, the vehicle returns to the depot. Note that this is possible without exceeding the maximum travel distance due to the preceding check and the assumption that any customer can feasibly be visited by a vehicle from at least one depot. When a vehicle must return to the depot, the next vehicle will serve the customer after the last customer in the list sorted by the angle of the customer. This proceeds until every customer is visited by a vehicle from its assigned depot. A graphical example of an initial solution is shown in Figure 2.

We do not take care of the supply of the depots. For the initial solution, we assume that the depots have an infinite supply of the products and that every customer receives what he demands. However, there is a penalty for excess deliveries in the objective function.

4.2 Improve

The largest part of the ILS heuristic is the improvement phase. Every improvement phase consists of multiple iterations, in which several neighborhoods are searched to improve the solution. Each neighborhood has a certain probability of being searched during a iteration.

4.2.1 Neighborhoods

Following notation as in Cordeau et al. (1997), each solution s is associated with attribute set $B(s) = \{(i, k, \ell)\}$, where attribute $(i, k, \ell) \in B(s)$ means that, in solution s, customer i is served by vehicle

k of depot ℓ. Each solution s has neighborhood $N(s)$. A solution $s' \in N(s)$ can be obtained by the removal from and addition of attributes to $B(s)$. In each iteration of the improvement phase, neighborhood type N_a ($a \in 1, \dots, 5$) is selected and searched for improving solutions with probability p_a, where $\sum_{a=1}^{5} p_a = 1$. We describe these neighborhood types below.

1. Move customer to other depot

   In the first neighborhood N_1, one customer is moved to another depot. That is, for some customer i attribute (i, k, ℓ) is changed to attribute (i, k', ℓ'), where ℓ' is another depot as ℓ and k' can be any truck from the set $\{1, \dots, m_{\ell'}\}$. We write $N_1 = \{(i, k, \ell) \rightarrow (i, k', \ell') \mid \ell \neq \ell'\}$.

   The heuristic uses cheapest insertion to insert customer i in every truck of every depot $\ell' \neq \ell$. In order to decrease the running time of the search, this neighborhood is not searched fully, but rather for some fraction ξ_{N_1} of customers. The customer whose insertion into another route leads to the lowest objective value is moved to its new depot and vehicle, given that this move is not tabu (see Section 4.2.2).

2. Move customer to other vehicle

   In the second neighborhood N_2, one customer is moved to another vehicle within the same depot, that is, $N_2 = \{(i, k, \ell) \rightarrow (i, k', \ell) \mid k \neq k'\}$. Again, cheapest insertion is used to insert customer i into the route of vehicle k' and the neighborhood is searched only for fraction ξ_{N_2} of customers. The move with the lowest objective value is performed, given it is not tabu.

3. Split customer

   For every depot with excess delivery, the customer receiving the largest amount of the product with the highest excess delivery is found. For every such customer, the objective value after the split is calculated, and the split of the customer with the lowest objective value is performed. The split is done as follows. From the original depot, the customer receives half of the original delivery of this product. The other half is delivered by a vehicle from a new depot, which is chosen by cheapest insertion.

   From a route perspective this neighborhood is defined as $N_3 = \{(i, k, \ell) \rightarrow \{(i, k, \ell), (i, k', \ell')\} \mid \ell \neq \ell'\}$.

4. Inter depot interchange

   In the fourth neighborhood N_4, one customer i is interchanged with customer j, which is one of customer i's $\lceil\sqrt{n}\rceil$ nearest neighbors served by the same depot, but which is served by another vehicle. Cheapest insertion is used to insert customer i and j in each others routes. This interchange is performed for fraction ξ_{N_4} of customers in order to reduce running time of the heuristic. The objective is calculated for each interchange, and the interchange with the lowest objective is performed.

   Mathematically, this neighborhood is described as $N_4 = \{\{(i, k, \ell), (j, k', \ell)\} \rightarrow \{(i, k', \ell), (j, k, \ell)\} \mid k \neq k'\}$.

5. Intra depot interchange

   Finally, in the fifth neighborhood N_5, a customer i is interchanged with customer j, but now j is required to be served by another depot. Cheapest insertion is used to insert customer i and j in each others routes. This interchange is performed for fraction ξ_{N_5} of customers in order to reduce running time of the heuristic. The objective is calculated for each interchange, and the interchange with the lowest objective is performed.

   Mathematically, we have $N_5 = \{\{(i, k, \ell), (j, k', \ell')\} \rightarrow \{(i, k', \ell'), (j, k, \ell)\} \mid \ell \neq \ell'\}$.

4.2.2 Tabu list

When the heuristic moves from solution s to solution s' within the the improvement phase, attributes in the set $B(s) \setminus B(s')$ are declared tabu for the next θ iterations. This means that if a customer is moved from depot ℓ and vehicle k, it is not allowed to place this customer back to depot ℓ and vehicle k for the next θ iterations. Only when the aspiration criterion is satisfied, this move is possible.

The length of the tabu list θ will depend on the size of the instance in such a way that θ increases with instance size parameters n, t, p and m, with m denoting the average number of trucks per depot $\sum_{\ell=1}^{t} m_\ell / t$.

4.2.3 Aspiration criterion

A *feasible* solution s with tabu attribute (i, k, ℓ) is approved if the objective value of s is lower than $\sigma_{ik\ell}$, the aspiration level of an attribute. All aspiration levels are initially set to infinity. When the heuristic moves from solution s to feasible solution s', we set $\sigma_{ik\ell} = f(s')$ for all attributes $(i, k, \ell) \in B(s) \setminus B(s')$ for which the objective of s' is lower than the current aspiration level.

4.2.4 Diversification

The tabu search is diversified by augmenting the objective function that is used when evaluating a solution $s' \in N(s)$. That is, when during the tabu search the objectives of a set of neighboring solutions are compared, an augmented objective function is used rather than objective function f itself. The augmentation is by a penalty term that is proportional to the number of times attributes in $B(s') \setminus B(s)$ have been used to move to a neighborhood in previous iterations. Let $\rho_{ik\ell}$ be the number of times attribute (i, k, ℓ) has been added to a solution in previous ILS iterations. Then, the augmented objective function $g(s')$ is defined as follows

$$g(s') = f(s') \left(1 + \zeta \sum_{(i,k,\ell) \in B(s') \setminus B(s)} \frac{\rho_{ik\ell}}{\lambda} \right).$$

Here, ζ is the control parameter for diversification, which is drawn from some continuous uniform distribution at the beginning of each improvement phase and is used to control the influence of the diversification, and λ is the current iteration number.

4.2.5 Stopping criterion

The improvement phase continues until no better feasible solution is found for μ iterations. Although the value of μ needs to be tested still, it depends on η, λ and π, where π is the number of customers which were perturbed in the perturb phase preceding the improvement phase (see Section 4.3). Balancing these three parameters, the heuristic will be allowed longer searches in early improvement phases and in improvement phases succeeding perturb phases with high values of π.

4.3 Perturb

Another part of the iterated local search heuristic is the perturbation of the solution, which main function is to broaden the search space of the heuristic. When a solution is perturbed, a cluster of customers is removed from their current vehicle and depot, and inserted into another route.

First, a random customer is chosen. Then, this customer and its π nearest neighbors are removed from their current routes and all randomly assigned to a new depot and a new vehicle. Using cheapest insertion they are inserted into the new route.

4.4 Accept

After every improvement phase in the iterated local search, the current solution is accepted with probability γ. If the solution is accepted, the next perturbation will be done with this solution. If the solution is not accepted, the previous accepted solution will be perturbed. This means that this solution was already perturbed in the previous perturbation phase.

5 Parameter tuning

We have performed experiments on the ILS in order to find the best values for the parameters. To tune the parameters, we have 100 test instances with different numbers of depots ($2 \leq t \leq 4$), products ($1 \leq p \leq 5$) and customers ($n = 50, 70, 100, 120$). The locations of the depots and customers are randomly generated on a square of $[0, 100] \times [0, 100]$. First, we examined the parameters given in Table 1. In cases where fractional results are obtained for discrete parameters values, results were always rounded up.

Table 1: Parameter on which we perform sensitivity analysis.

Parameter	Definition
θ	Tabu length
δ	Adjustment parameter for α and β
ζ	Control parameter for diversification
μ	Parameter for stopping criterion
π	Perturbation size
γ	Acceptance probability

For each experiment, we did five runs for four different test instances, which are chosen such that they have different characterizations. These are given in Table 2. This set of experiments was done with $\eta = 6n$ iterations, except for instance 100 where we used $\eta = 2n$, in order to decrease running time.

Table 2: Instances on which sensitivity analysis was performed.

Instance	t	p	n
8	2	2	50
20	2	4	50
55	3	1	100
100	4	5	120

5.1 Tabu length θ

Experiments are performed to determine the optimal length of the tabulist. This length is determined at the beginning of every improvement phase. Cordeau and Maischberger (2012) use a random tabu length from the discrete uniform distribution on $[0, \sqrt{nmt}]$. We will compare this interval with $[0, \sqrt{nmpt}]$, since our problem has multiple products, and the optimal tabu length seems to increase as the instance size increases. Furthermore, a fixed tabu length of $4\sqrt{nmpt}$ was tested. The results can be found in Table 3a.

From Table 3a, we see that for instance 8 and 20 the second considered tabu length gives the best results, whereas the first tabu length is better for instances 55 and 100. The last tabu length does not give good results. We choose to use a random tabu length from the interval $[0, \sqrt{nmt}]$. This tabu length is also used in further experiments.

Table 3: Parameter setting of θ and δ.

(a) Mean objective value for different values of θ.

	θ		
Instance	$[0, \sqrt{nmt}]$	$[0, \sqrt{nmpt}]$	$4\sqrt{nmpt}$
8	714	692	731
20	781	760	810
55	1090	1114	1127
100	1560	1575	1656

(b) Mean objective value for different values of δ.

	δ	
Instance	$[0, 1]$	$[0, 0.5]$
8	692	740
20	760	739
55	1114	1060
100	1575	1508

5.2 Adjustment parameter for penalties δ

The penalties of infeasibility are controlled by a random parameter δ. Higher values of δ result in a faster adjustment of the penalty parameters α and β. We tested δ for two continuous uniform distributions from which a value is picked at the beginning of an improvement phase. The intervals of these distributions are $[0, 1]$ and $[0, 0.5]$. Results of the experiments can be found in Table 3b. This table shows that for three of the four instances, the interval $[0, 0.5]$ performs better than the interval $[0, 1]$. Therefore, the interval for δ is set to $[0, 0.5]$ and these values are also used in further experiments.

5.3 Control parameter for diversification ζ

The control parameter for diversification, ζ, is randomly taken from a continuous uniform distribution at the start of each improvement phase. Four different intervals are tested: $[0, 0.5]$, $[0, 1]$, $[0, 2]$ and $[0, 4]$. The results of the experiments are shown in Table 4a.

Table 4: Parameter setting of ζ and μ.

(a) Mean objective value for different values of ζ.

	ζ			
Instance	$[0, 0.5]$	$[0, 1]$	$[0, 2]$	$[0, 4]$
8	701	740	758	724
20	764	739	778	831
55	1069	1060	1114	1090
100	1507	1508	1739	1767

(b) Mean objective value for different values of μ.

	μ	
Instance	$\sqrt{(\eta - \lambda)\pi}$	$15\sqrt[4]{(\eta - \lambda)\pi}$
8	740	701
20	739	753
55	1060	1060
100	1508	1461

As can be seen in Table 4a, the intervals $[0, 0.5]$ and $[0, 1]$ are performing the best. We choose to use interval $[0, 1]$ for ζ, which is also used in later experiments.

5.4 Parameter for stopping criterion μ

After μ non-improving iterations the improvement phase stops. We give μ a high value in early improvement phases, but let its value decrease towards the end. Cordeau and Maischberger (2012) proposed $\mu = \sqrt{(\eta - \lambda)\pi}$. As we observed that the improvement phase may be too short at the end of the run, we compared this value of μ with $\mu = 15\sqrt[4]{(\eta - \lambda)\pi}$, see Table 4b. We have chosen $\mu = 15\sqrt[4]{(\eta - \lambda)\pi}$, which is used in further experiments.

5.5 Perturbation size π

The perturbation size is picked randomly from a discrete uniform distribution at the beginning of an perturbation phase. We did experiments on two intervals for these distributions: $[0, \sqrt{n}]$ and $[0, 2\sqrt{n}]$. In Table 5a, the results of these experiments are shown.

Table 5: Parameter setting of π and γ.

(a) Mean objective value for different values of π.

	π	
Instance	$[0, \sqrt{n}]$	$[0, 2\sqrt{n}]$
8	701	701
20	753	724
55	1060	1116
100	1461	1388

(b) Mean objective value for different values of γ.

	γ	
Instance	$1 - (\eta/\lambda)^2$	$1 - (\eta/\lambda)$
8	701	690
20	753	786
55	1060	1104
100	1461	1401

When looking at the averages, it is difficult to determine which interval is best. However, we noticed that the results of the experiments with interval $[0, \sqrt{n}]$ had a few bad results, but also good results. We are planning to perform a few experiments, and the best objective will be saved. Therefore, we will use the interval $[0, \sqrt{n}]$.

5.6 Acceptance probability γ

The probability that an improved solution is accepted for the next perturbation phase is high for solutions with a low iteration number, and low for solutions with a high iteration number. Two possible values for γ are $\gamma = 1 - (\eta/\lambda)^2$ and $\gamma = 1 - (\eta/\lambda)$. The results of the experiments with these values are given in Table 5b. As three out of the four instances perform better with $\gamma = 1 - (\eta/\lambda)^2$ than with the other value of γ, we take $\gamma = 1 - (\eta/\lambda)^2$ as the probability of accepting a solution. When we perform further experiments, this value will be used.

5.7 Probabilities of neighborhoods

We have performed experiments on five different settings for the probabilities of the neighborhoods in the improvement phase. Only one of these neighborhoods is explored per improve iteration. The settings are shown in Table 6. We choose three instances to test these settings. They differ in the number of depots, customers and products in order to find good probabilities for all types of instances. For every setting, five experiments are performed and the mean values of these experiments can be found in Table 7.

Table 6: Probabilities of the neighborhoods per improve iteration for five different settings.

Setting	$P(N_1)$	$P(N_2)$	$P(N_3)$	$P(N_4)$	$P(N_5)$
1	0.5	0.25	0.1	0.1	0.05
2	0.025	0.5	0.1	0.1	0.05
3	0.4	0.4	0.05	0.1	0.05
4	0.325	0.325	0.05	0.15	0.15
5	0.45	0.3	0.75	0.125	0.05

As can be seen in Table 7, setting 3 performs best. Therefore, for further experiments, we use the probabilities of the neighborhoods as in setting 3.

Table 7: Mean objective values for different settings of probabilities for the neighborhoods.

Instance	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5
14	768	838	758	810	781
36	959	999	956	1048	984
80	1313	1428	1310	1317	1304
Mean	1013	1089	1008	1059	1023

Within each neighborhood, customers have only a limited probability to be explored. In this way, the running time per neighborhood is reduced and we prevent the algorithm of converging too fast. We have performed experiments on the probabilities within each neighborhood, ξ_{N_a}, with $a \in \{1, \ldots, 5\}$. We have found the following values: $\xi_{N_1} = 0.4$, $\xi_{N_2} = 0.25$, $\xi_{N_3} = 1$, $\xi_{N_4} = 0.25$ and $\xi_{N_5} = 0.25$. Neighborhood N_3 is fully searched since only the customers with highest demand for a product are split.

6 Results

The ILS heuristic was used to solve 1000 instances, each with $\eta = 200000/(nt)$ iterations. Every instance was solved twice, and the best solution was stored. The total number of iterations was chosen to depend on n and t as we wanted all instances to run with fair running times. However, we observed that the running times increase with instance size if the number of iterations was kept fixed. An example of the graphical representation of the routes can be seen in Figure 3.

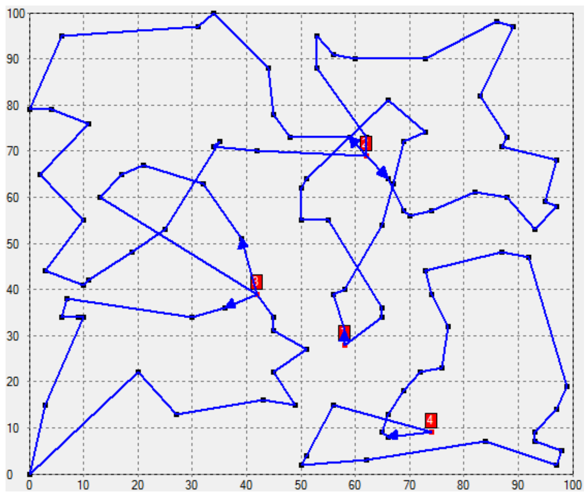


Figure 3: Solution of the instance from Figure 1 as returned by the heuristic.

Table 8 shows the mean objective value of 20 instances with the same number of depots, customers and products. Also, the mean objective value and mean CPU in seconds are given for each combination of depot and customer size. A number of things are observed. First, we notice that instances with more customers have higher objective values. This is obviously what is expected as the vehicles have to visit more customers and the objective value for feasible solutions is the total traveled distance. We also see that the average distance traveled per customer increases as the number of customers decreases. For example, the average total traveled distance for 140 customers is 1325, and hence, per customer 9.47. For the instances with 40 customers, we have an average total travel distance of 656 and per customer 16.39. This is also expected, as the distance between customers decreases as the number of customers increases.

We observe that instances with more depots have a higher objective value than instances with less depots. For example, instances with 80 customers that have 4, 3 or 2 depots have an average objective

Table 8: Mean objective value of 20 instances with same number of depots, customers and products.

		p						
t	n	1	2	3	4	5	Mean	Mean CPU (s)
2	40	624	635	648	670	698	656	63
	60	773	797	804	817	835	805	125
	80	914	898	933	966	977	938	212
3	60	749	793	832	840	917	826	106
	80	882	923	970	963	1036	955	190
	100	1025	1054	1078	1110	1195	1093	263
	140	1223	1252	1363	1360	1419	1324	594
4	80	887	931	967	1022	1131	988	160
	100	1038	1053	1130	1140	1238	1120	240
	140	1201	1288	1312	1380	1452	1327	585

value of 988, 955 and 938 respectively. These averages are over 100 instances with 1 to 5 products. This is not what we expected, since on average, with more depots, there should be a depot closer by to the customer. However, as we set the number of iterations on $200000/(nt)$, the instances with more depots have less iterations than instances with fewer depots and are expected to find worse solution on average. The average running time also shows that the small instances can search longer: 160, 190 and 212 seconds are the average running times for 4, 3 and 2 depots respectively. Furthermore, as depot supply is limited, the argument that instances with more depots are at least as easy as instances with less depots, does not hold.

We see that the objective value increases as the number of products increases, other parameters held constant. For example, the increase in objective value from one to five products for the instances with 40 customers and 2 depots is 10.6%. We also observed that the relative difference in objective value from one product to five products is bigger as the number of depots increases. For instances with 80 customers, the increase in the average objective value for 2, 3 and 4 depots is 6.4%, 14.9% and 21.6% respectively.

Using the sweep algorithm, we find that all initial solutions are infeasible with respect to the supply of the depots. All final solutions obtained by the ILS are feasible. The average objective value (measuring only route duration) of the initial solutions is 1542, whereas the average objective value of the final solutions is 1003. Hence, the final solutions are an improvement in both feasibility and route duration, since these are reduced by 35% on average. We notice that for the instances where a customer is at the same location as a depot, the initial solution has a relatively high objective value, as our current implementation of the sweep algorithm is not robust to this situation. However, the average of the objective values of the initial solutions without these instances is 1520, and of the final solutions without these instances 1000. Hence, for this situation, the average objective is reduced by 34%.

7 Conclusion

A solution approach was presented to solve instances of the MDVRP within reasonable time. In the MDVRP a number of vehicles originating from multiple depots has to be routed to serve a set of customers. We generalize previous research in this area by considering customer demand and depot supply for multiple products. Furthermore, the problem difficulty is increased by assuming that the depot supply for all products is limited.

An optimal solution returns a set of routes in which the total travel distance is minimized, given that all customer demand is met from the depot supplies and no vehicle route exceeds the maximal travel distance for a vehicle. As a generalization of the VRP, which is known to be $\mathcal{NP}$-hard, no known algorithm is in place to find this optimal solution in polynomial time.

Therefore, we choose a heuristic approach to produce feasible solution within reasonable time. The presented approach is an iterated local tabu search algorithm in which an initial solution is repeatedly improved and perturbed in order to increase the solution quality. We find that the average objective value increases as the number of customers increases. Also, the average objective value increases as the number of products increases. This indicates that the problem becomes more difficult with more products. Furthermore, the relative difference of the average objective value of instances with one and five products increases as the number of depots increases.

To our knowledge, the MDVRP with multiple products and limited depot supply has never been studied before and no known results from heuristics are publicly available to compare our heuristic against. Since the problem is $\mathcal{NP}$-hard, we are not able to find the optimal objective value for large instances. Therefore, we have compared our results with our initial solution obtained by the sweep algorithm. This problem is a relaxation of the original problem with respect to the supply of the depot, as all demand is fulfilled. The routes are feasible with respect to the duration. We have found that the average objective value is reduced by 35% compared with the initial solution and all final solutions are feasible whereas they were infeasible for the initial solution.

Even though extensive effort has been put into the setting of various parameters, future research could focus on further improvement of parameter setting to increase solution quality and consistency or decrease processing time. Our research suggests that optimal values of various parameters seem to depend on instance size, but some parameters were left instance independent due to time constraints.

Furthermore, upon graphical inspection of solutions, we observe that in a number of instances routes cross over themselves still. Research focusing on the addition of neighborhoods, and in particular the 2-opt move, will likely yield improving solutions. The inclusion of more sophisticated insertion algorithms, such as the GENI algorithm as proposed by Gendreau et al. (1992), will likely add to a further improvement of solution quality.

References

Cordeau, Jean-François, Michel Gendreau, and Gilbert Laporte (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks 30*(2), 105–119.

Cordeau, Jean-François and Mirko Maischberger (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research 39*(9), 2033–2050.

Dantzig, G. B. and J. H. Ramser (1959). The truck dispatching problem. *Management Science 6*(1), 80–91.

Gendreau, Michel, Alain Hertz, and Gilbert Laporte (1992, November). New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res. 40*(6), 1086–1094.

Glover, Fred (1990). Tabu search: A tutorial. *Interfaces 20*(4), 74–94.

Hemmelmayr, Vera C., Karl F. Doerner, and Richard F. Hartl (2009). A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research 195*(3), 791 – 802.

Lenstra, J.K. and A.H.G. Rinnooy Kan (1981). Complexity of vehicle routing and scheduling problems. *Networks 11*(2), 221–227.

Lourenço, Helena R, Olivier C Martin, and Thomas Stützle (2003). Iterated local search. *Handbook of metaheuristics*, 320–353.

Pisinger, David and Stefan Ropke (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research 34*(8), 2403–2435.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei (2012, May). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Oper. Res. 60*(3), 611–624.

Wren, Anthony and Alan Holliday (1972). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly (1970-1977) 23*(3), 333–344.

Yu, Bin, ZZ Yang, and JX Xie (2011). A parallel improved ant colony optimization for multi-depot vehicle routing problem. *Journal of the Operational Research Society 62*(1), 183–188.

Chapter 2
Hybrid Genetic Search

W. Scherphof, F. Kramer

1 Introduction

In this section we introduce the Multi-Depot Vehicle Routing Problem with Multiple Products problem. In subsection 1.1 we discuss the background of the problem and in subsection 1.2 we discuss the current literature on the topic.

1.1 Background

To stay competitive companies need to continuously improve their productivity, quality, customer responsiveness and lower their costs. When it comes to lowering costs, efficiency and optimization are key terms. Factories, for instance, face the challenging task to find an optimal schedule to process their production. Such problems can be solved with combinatorial optimization.

One of the best known issues in the field of combinatorial optimization is the Vehicle Routing Problem. In the earlier formulation of this problem we consider a set of customers together with a fleet of vehicles. The goal is to find for each vehicle a sequence of customers to visit and deliver some product such that the overall costs are minimized and all customers are visited. Despite the relatively simple setting obtaining an optimal solution turns out to be challenging. In fact, decades of research have passed and still the subject enjoys the attention of many researchers. This is partly due to the relevance of the theory involved in vehicle routing in numerous of different fields, such as mathematics and artificial intelligence. Another reason is the ease at which it can be extended to more special cases. These special cases are more interesting from a practical point of view but obviously grow in complexity and therefore require more scientific attention. One can consider, for example, that a courier company is interested in delivering a product within a certain time window. On the other hand, in waste management it is common to consider pickups as well. Although these examples have a different focus they both rely on the same basic setting the vehicle routing problem provides.

Besides the possibility to append the basic problem with additional constraints, such as time windows, one can increase the size of the problem. A natural extension is to enlarge the customer base. However, other options to consider are the delivery of multiple products or the situation in which the products are stored in different locations or a combination of these. Due to the complexity of vehicle routing problems various ways exist to tackle it. Possible approaches vary from exact algorithms coping with smaller sized problems to heuristics applicable to larger instances. In this paper, we construct an algorithm able to solve the Multi-Depot Multi-Product Vehicle Routing Problem. The produced algorithm is of a genetic kind, i.e., we create a search heuristic that uses the process of natural selection in order to find the optimal solution.

1.2 Literature review

The vehicle routing problem (VRP) is mentioned for the first time by Dantzig and Ramser (1959) where a fleet of gasoline trucks is routed between a main terminal and smaller service stations. Since then a wide variety of VRPs emerged together with abundant amounts of literature in this topic. In the recent paper of Vidal et al. (2013) a decent overview of the most researched classes of the VRP is provided, generalized as Multi-Attribute Vehicle Routing Problems. They argue that all variants of VRPs can be classified in the following three categories.

1. Assignment of customers and routes to resources.
2. Sequence choices.
3. The evaluation of fixed sequences.

The Multi-Depot Vehicle Routing Problem, for example, is contained in the Assignment of customers and routes to resources category. Next to that, they evaluate and discuss the concepts of some of the best performing meta-heuristics to gain insights in why they perform so well.

The particular class of VRPs we are interested in basically consists of two other classes, i.e., the Multi-Depot Vehicle Routing Problem (MDVRP) and the Multi-Product Vehicle Routing Problem (MPVRP) variants. The former has received a lot of attention in literature (see, for example, Cordeau et al. (1997) and Pisinger and Ropke (2007)) in contrast to the latter. In case the MPVRP is considered it is often of a load specific nature due to different compartments reserved for different products in each vehicle (Mendoza et al., 2010). The intuition is that perishable goods, such as frozen food, may need certain transport conditions. Despite the few literature the Multi-Product variant is an interesting and relevant class as Savelsbergh and Sol (1998) show by investigating a case study of a large transportation company. However, to the best of our knowledge there is no literature available with respect to the Multi-Depot Multi-Product Vehicle Routing Problem (MDMPVRP). From a practical point of view the MDMPVRP is relevant since it is applicable to, for example, pharmacies and libraries. These types of companies often have multiple locations and many different products to offer.

Since the MDVRP has been extensively researched we use the existing knowledge from that field in our effort to construct a genetic algorithm capable of solving the MDMPVRP. More specifically, we base our algorithm on the paper of Vidal et al. (2012) since their genetic algorithm proves to be fast and able to solve a set of benchmark instances to optimality.

The remainder of the paper is organized as follows. Section 2 states the notation and formal definition of the MDMPVRP together with a description of the assumptions. The properties of the feasibility of a solution are described in Section 3. Section 4 discusses all aspects of the proposed Genetic Algorithm in detail, including a set of experiments to tune its parameters. The quality of the solutions is reviewed in Section 5. Section 6 contains the experimental insights and section 7 contains the conclusion and recommendations for future research.

2 Problem formulation

We formally present the problem and introduce some definitions and notations. The representation is partly based on the model from Vidal et al. (2012).

Parameter		Description
$\underline{D}$	$= \{1, \dots, D\}$	the set of depots
$\underline{P}$	$= \{1, \dots, P\}$	the set of products
$\underline{N}$	$= \{1, \dots, N\}$	the set of customers
V^{CUST}	$= \{v_1, \dots, v_N\}$	the set of customer vertices
V^{DEP}	$= \{v_{N+1}, \dots, v_{N+D}\}$	the set of depot vertices
V	$= V^{CUST} \cup V^{DEP}$	the set of vertices
E		the set of edges
R		the set of routes
c_{ij}		travel cost from $v_i \in V$ to $v_j \in V$
q_{np}		demand of customer n for products p
s_{dp}		supply in depot d of products p
T		maximum travel time per vehicle
x_{ijrd}	$= \begin{cases} 1 & \text{if vertex } v_j \text{ is immediately visited after } v_i \text{ in route } r \text{ starting at depot } d \\ 0 & \text{otherwise} \end{cases}$	
y_{npd}	$= \begin{cases} 1 & \text{if for customer } n \text{ demand for product } p \text{ is supplied by depot } d \\ 0 & \text{otherwise} \end{cases}$	

Table 1: All relevant variables and parameters for the MDMPVRP.

We have a number of depots, D, in which P different products are stored. These inventories are used to meet the known demand of N customers. Then, the MDMPVRP can be defined as follows. Consider a complete graph $G = (V, E)$ with V the set of vertices and E the set of edges. The set V can be divided into two subsets, i.e., $V = V^{CUST} \cup V^{DEP}$, representing the set of customers and depots, respectively. Hence, it follows that $|V| = D + N$. For customer $n = 1, \dots, N$ and product $p = 1, \dots, P$, we face the known demand q_{np}. Similarly, we let s_{dp} denote the supply at depot $d = 1, \dots, D$ of product $p = 1, \dots, P$. If the demand of customer n for product p is supplied by depot d we let $y_{npd} = 1$. Otherwise, we have $y_{npd} = 0$. At each depot an unlimited fleet of homogeneous vehicles is available. Vehicle capacity does not impose limits in practice, but there is a maximum travel time T per vehicle. For each edge $e_{i,j} \in E$ we let c_{ij} represent the travel cost to go from vertex $v_i \in V$ to $v_j \in V$. We assign one vehicle to each route, where we define a route as follows. A route r starts at a depot d situated at $v_d \in V^{DEP}$ and sequentially visits n_r customers located at $v_i \in V^{CUST}, i \in \underline{N}$, and ends in the starting depot d. Note that the set of customers in a route starting at depot d is a subset of the customers assigned to depot d. We assume that for all products the total supply at the depots is sufficient to meet the total demand of the customers, i.e., $\sum_d s_{dp} \geq \sum_n q_{np}$, $p = 1, \dots P$. Furthermore, we have the following additional constraints. For each customer the demand for a certain product cannot be delivered by multiple vehicles. On the other hand, we allow each customer to receive different products from different depots. In addition we impose the restriction that the vehicles starting in depot d cannot visit depot d', where $d \neq d'$. Also, for each product the sum of the demands of all customers assigned to a certain depot cannot exceed the supply at that depot, i.e., $s_{dp} \geq \sum_n y_{npd} q_{np}$, $p = 1, \dots P$, and $d = 1, \dots, D$. The goal is to find the schedule in which for all customers each product is allocated to a depot producing the lowest transportation costs. In other words, we want to minimize transportation costs, under the condition that all demand is met. All parameters and decision variables are summarized in Table 1. The resulting Mixed Integer Linear Program formulation can be found in more detail in Appendix A. When we refer to the distance or transportation costs we mean the traveling time.

3 Properties

Assume we relax the constraint of a maximum travel time per vehicle. Then, we only need at most one vehicle per depot and the total distance corresponding to this setting provides a lower bound. Adding the constraint again could cause the solution corresponding to the lower bound to become infeasible. It is straightforward to see that good solutions are likely to be found on the borders of feasibility. This observation gives rise to the idea to relax the maximum travel time constraint in our algorithm to fruitfully explore the boundaries of feasibility. In order to do so we introduce the concept of a penalized cost for a route which works as follows. Consider a route r starting at depot d and visiting n_r customers, summarized by the sequence $r = v_d, v_1, \dots, v_{n_r}, v_d$. The associated distance is given by $c(r) = c_{dv_1} + \sum_{i=1}^{n_r-1} c_{v_i v_{i+1}} + c_{v_{n_r} d}$. Note that without the maximum travel distance constraint we could have $c(r) > T$. Hence, we consider the penalized cost defined by $\phi(r) = c(r) + \omega \max(0, c(r) - T)$. That is, when the route duration $c(r)$ exceeds the maximum travel time, the difference between $c(r)$ and T is penalized by a factor ω.

4 Solution approaches

In this section we discuss our general approach for solving the MDMPVRP and discuss in detail all the steps.

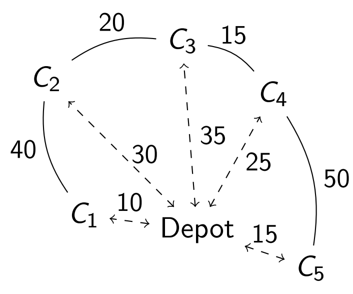
We know that the Traveling Salesman Problem (TSP) is NP-hard (Karp, 1972). Since the MDMPVRP is a generalization of the TSP it is NP-hard as well. Therefore, we try to heuristically solve the MDMPVRP. The meta-heuristic we propose is based on the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) of Vidal et al. (2012). The pseudo code of our meta-heuristic is exactly the same as Vidal et al. (2012) present and is shown in Algorithm 1. Before discussing the steps in more detail we review the general steps of the algorithm. The first step is to randomly create an initial population. Since we relax the maximum travel time constraint each solution can be either feasible or infeasible and we divide the population in two corresponding sub-populations. Then, for each iteration we select two parents, P_1 and P_2, based on their qualifications and generate an offspring C. The offspring C is educated by a local search procedure and added to the sub-population according to its feasibility. When one of the sub-populations has reached the maximum size, we keep the best solutions and delete the remaining from the sub-population under consideration. To avoid our algorithm getting stuck at some local optimum we invoke the diversify procedure as soon as we observe that the best solution does not improve for a number of iterations. This procedure creates a new population by calling the initialization procedure and combines it with the best solutions of the old population.

Algorithm 1 (HGSADC)

```
1. Initialize population
2. while number of iterations without improvement < $It_{NI}$, and time < $Tmax$
    (a) Select parent solutions $P_1$ and $P_2$
    (b) Generate offspring $C$ from $P_1$ and $P_2$ (crossover)
    (c) Educate offspring $C$ (local search procedure)
    (d) if $C$ is infeasible, then insert $C$ into infeasible sub-population.
    (e) if $C$ is feasible, then insert $C$ into feasible sub-population.
    (f) if maximum sub-population size reached, then select survivors
    (g) Adjust penalty parameters for violating feasibility conditions
    (h) if best solution not improved for $It_{DIV}$ iterations, then diversify population
3. Return best feasible solution
```

4.1 Search Space and Initialization

The initial population $\mathbb{S}$ is created by generating 4μ solutions. In the initialization procedure presented by Vidal et al. (2012) solutions are created by



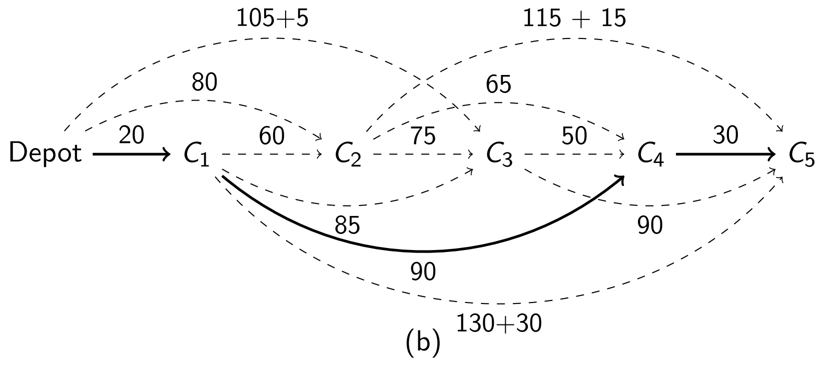


Figure 2: Example of a giant tour representation (a), and its corresponding auxiliary acyclic graph with maximum travel time $T = 100$ and penalty $\omega = 1$ (b)

randomly assigning customers to depots. In turn, routes are formed by randomly choosing a sequence of customers. We deviate from this approach due to the following observation. It is possible that a certain customer is supplied from multiple depots. In a good solution the number of these split deliveries is most likely limited. Next to that, a customer is preferably supplied from a depot located nearby. This is not always possible since in the process of assigning customers to depots supplies will eventually be depleted and customers can no longer be serviced from their preferred depot. To illustrate the consequences we consider the following example of three depots and two customers depicted in Figure 1. Since the distance from Customer 2 to each of the three depots is almost equal, it is not relevant to assign this customer to its nearest depot. The contrary holds for Customer 1 and servicing him from the nearest depot is more important. By considering the standard deviation of each customer with respect to the distances to each depot we retrieve a priority ranking. When it comes to travel time, customers with a high standard deviation, i.e., ranking, are more important to assign to the nearest depot. Then, we randomly determine the length of the ranked sequence and we randomly add the remaining customers. The resulting sequence determines the order in which customers are assigned to the depots. Starting from the nearest depot, a customer is only assigned if all products can be supplied. Any remaining customer will receive deliveries from multiple depots as follows.

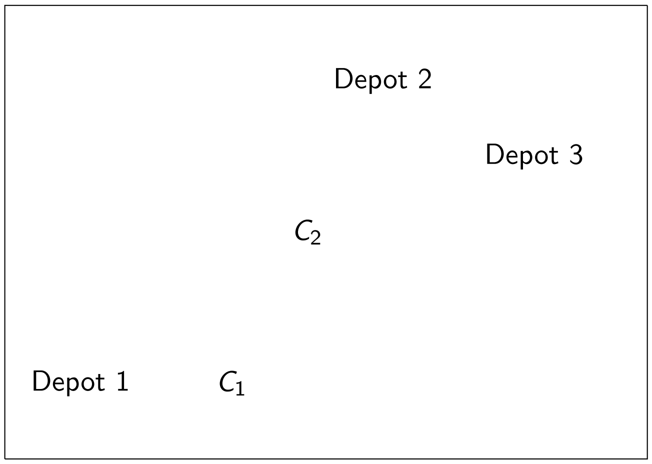


Figure 1: Example illustrating importance of depot allocation

We form a sequence of customers, decreasing in the number of outstanding products. Then, for each customer we validate whether we can assign $P - 1$ products, starting at the nearest depot. If no depot is capable of supplying $P - 1$ products, we repeat this process for $P - 2, \dots, 1$ products. By ranking the customers we avoid our algorithm spending too much time searching in unfavored directions. We are aware that especially for Genetic Algorithms this kind of steering can be risky. However, experiments showed that with less computation time we were able to find better solutions.

After specifying for all customers the depots responsible for delivering each of the products, we construct for each depot d a sequence of n_d customers by applying the nearest neighbor procedure. This sequence is called the *giant tour*, and passes through all assigned customer locations. Using the notation of Vidal et al. (2012) we define the giant tour by $\sigma_1^d, \dots, \sigma_{n_d}^d$, $\sigma_i^d \in V^{CUST}$. It is intuitive that a giant tour has a shorter distance than multiple routes, provided that the sequence of customers is based on the nearest neighbor procedure. Then, with probability PR_E the solution is educated and inserted in the appropriate sub-population according to its feasibility. To obtain all different routes starting from depot d we combine the giant tour and the penalized cost function. In order to do so we use the same algorithm as Vidal et al. (2012), named *Split* which was introduced by Prins (2004). This algorithm translates a giant tour into a graph $G_d = (V_d, E_d)$ where V_d is the set of vertices containing depot d and all its assigned customers. The set of edges E_d consists of two subsets. The first subset contains the edges $e_{n,n+1}$, $v_n \in V_d$ linking each customer to its immediate predecessor. The second subset consists of the edges $e_{d,n} \in E_d$ for all the n_d customers in the giant tour and represents a direct route between each customer and depot d. We assign to each edge $e_{i,j} \in E_d$ a weight equal to the Euclidean distance between vertices v_i and v_j. An illustration of such a graph is given in Figure 2 (a). This graph can be transformed to an auxiliary acyclic graph, H, containing all possible routes r that cover all customers in the giant tour. The edges in graph H have weights equal to the penalized cost, see Figure 2 (b) for an example with maximum travel time $T = 100$ and penalty $\omega = 1$. Finding the shortest path in graph H results in the route delimiters. We solve the shortest path with Dijkstra's algorithm (Dijkstra, 1959). A more detailed description can be found in the e-companion of Vidal et al. (2012).

Now we have for each solution s a set of routes, $R(s)$. The total distance associated with this solution, $\phi(s)$, equals the sum of the penalized cost of all routes, i.e., $\phi(s) = \sum_{r \in R(s)} \phi(r)$. Observe that if the maximum travel distance constraint is violated we find $\sum_{r \in R(s)} \phi(r) > \sum_{r \in R(s)} c(r)$ and the solution is infeasible. Then, the total population is divided in two subsets corresponding to feasible and infeasible solutions.

	Depot 1			Depot 2		
customers	5	2	1	4	1	3
product 1	2	6	4	0	0	1
product 2	1	2	0	2	4	0

Table 2: Chromosomes Representation

4.2 Solution Representation

A solution is characterized by the product allocation of each customer over the depots, the sequence of customer visits, and the penalty used for exceeding the maximum travel distance. According to Vidal et al. (2012) the representation of routes from the same depot as a giant tour is useful as it provides the means to use simple and efficient crossover procedures working on permutations. Hence, an individual s is represented as a set of two chromosomes. The first set corresponds to the giant route chromosome and contains the sequence $\sigma_1^d, \dots, \sigma_{n_d}^d$ of customers per depot d. The second set reflects the customer chromosome, which contains for each customer n assigned to depot d the demand for product p supplied by that depot. In Table 2 an example is given with two depots, five customers, and two types of products p_1 and p_2, resulting in two giant tours with the customer sequence and the demand of these customers supplied by each depot. To obtain the set of routes associated with this solution we apply the *Split* algorithm with the penalty ω as input.

4.3 Evaluation of Individuals

In a given population some individuals are more relevant than others in terms of genetic information. Therefore, we apply an evaluation function to assign a certain fitness value to each solution. Vidal et al. (2012) mention that such an approach is generally myopic with respect to the possible impact of the evaluation and selection processes on the diversity of the population. They determine a fitness measure that addresses both the diversity contribution and the costs.

As it turns out, this approach is very time consuming for our problem and we choose to base our fitness measure only on the penalized distance of a solution.

4.4 Parent Selection and Crossover

Offspring solutions are created according to Algorithm 2, where Step 1 to Step 3 are based on the paper by Vidal et al. (2012). For each parent we randomly select, with uniform probability, two candidates from the entire population and keep the one with the shortest penalized distance. The remainder of Step 1 basically forms an initialization for the inheritance procedure in Step 2. Offspring can inherit genetic information from a parent in numerous proportions. As a consequence Vidal et al. (2012) mention the possibility to create a solution which is to a large extent similar to one of the parents. This enables the fine-tuning of solutions. On the other hand, a more balanced mixture of genetic information from both parents provides the opportunity to enlarge the search space. Compared to Vidal et al. (2012) we have the additional constraints with respect to the demand and supply of all the different products. This implies we have to be careful when assigning the inherited products of a customer to a depot. Since it is most efficient for each customer to have all products supplied by a single depot we try to achieve this as follows. When a parent passes on a certain customer from a depot we assign its entire outstanding demand to that depot. If for at least one product the supply is inadequate to meet the demand, the customer is not inherited at this point. By doing so, we save room for customers for which all products can be delivered by the depot under consideration. It follows that after a first attempt of passing on genetic information we may still have some customers for which not all products can be assigned to a single depot. Furthermore, due to the random nature of the genetic material it is possible that some customers are not inherited at all. To complete the allocation we validate whether a depot is able to supply P products, starting from the customer with the most unassigned products. While we still have unmet demand we repeat this for $P-1, \ldots, 1$ products.

Algorithm 2

```
Step 1: Inheritance rule
    1. Pick two random numbers between 0 and $D$ according to a uniform distribution. Let $n_1$ and $n_2$ be, respectively, the smallest and the largest of these numbers;
    2. Randomly (uniform distribution) select $n_1$ depots to form a set $\lambda_1$;
    3. Randomly (uniform distribution) select $n_2 - n_1$ depots to form a set $\lambda_2$;
    4. The remaining $D - n_2$ depots make up the set $\lambda_{mix}$.
Step 2: Inherit data from $P_1$
    1. **for** each depot belonging to set;
    2. $\lambda_1$: Copy the sequence of customer visits from $V_d(P_1)$ to $V_d(C)$;
    3. $\lambda_{mix}$: Randomly (uniform distribution) select two chromosome-cutting points $\alpha$ and $\beta$; copy the $\alpha$ to $\beta$ substring of $V_d(P_1)$ to $V_d(C)$.
Step 3: Inherit data from $P_2$
    1. **for** each depot $d \in \lambda_2 \cup \lambda_{mix}$, selected in random order, consider each customer visit $n$ in $V_d(P_2)$;
    2. **if** the supply of the depot is sufficient **then** copy it at the end of $V_d(C)$.
Step 4: Complete customer services
    1. Create a sequence of customers with positive demand, decreasing in the number of unassigned products;
    2. **for** each customer;
    3. order the depots with respect to the distance;
    4. sequentially try to place $P$ products in the depots;
    5. **if** the supply of the depot is sufficient **then** copy it at the end of $V_d(C)$;
    6. **while** not all customers are allocation **repeat** (1) - (5) **for** $P = P, \ldots, 1$.
```

4.5 Education

A standard component of a Genetic Algorithm is mutation which aims to increase the variety in genetic information in the population. In our algorithm we include a mutation in the form of education. This education procedure is more radical than standard mutation since it tries to improve offspring. We base our education on Vidal et al. (2012) in which nine different local-search procedures are implemented consisting of different versions of *insertions*, *swaps*, and *2-opt*. Since we face customers with demand for multiple products it is more difficult to, for example, swap customers between depots. In fact, in most cases this results in infeasible solutions. To overcome this problem we introduce an additional check that validates, before swapping customers, whether their demands match the inventories in the corresponding depots. Next to that, we observed that some of the procedures used by Vidal et al. (2012) consumed relatively much computation time without increasing solutions significantly and are therefore not suitable for our MDMPVRP. Then, our education consists of the following four procedures. The first procedure is the simple local-search algorithm two-opt. Second, we have an insertion procedure that randomly inserts customers in routes starting from the same depot only if this results in a feasible solution with a lower objective value. Furthermore, we have two swap moves. The first swap move interchanges two customers assigned to the same depot whereas the second switches customers assigned to different depots within a certain neighborhood. Again, these moves are only performed if this results in a feasible solution with a lower objective value. For all moves the following holds. Only if all customers have been considered consecutively and no improvement has been found the procedure stops. Then, the education procedure, given by Algorithm 3, consists of performing all four moves in random order. Only if all moves have been consecutively performed without improvement the education stops.

Algorithm 3

1. **while** no improvement has been found randomly perform:
    (a) two-opt;
    (b) insertion within depot;
    (c) swap within depot;
    (d) swap between depots;
2. **if** not all moves have been performed consecutively **then** go to 1.

4.6 Population Management

The population consists of two sets corresponding to feasible and infeasible solutions. The number of elements in each set depends on the size of the penalty ω. If, for example, $\omega = 0$, all solutions are represented by their giant tours and the set of infeasible solutions is relatively large. On the other hand, if $\omega = \infty$, the set of infeasible solutions is empty. Hence, the number of feasible solutions is increasing in the size of the penalty. To avoid convergence to either subset we dynamically adjust the penalty parameter during the execution of the main algorithm. The aim is to get a solution space with a reference proportion, ξ^{REF}, of feasible solutions. We monitor the current feasibility proportion by considering the ratio of feasible solutions after each 100 generated offspring, denoted by ξ^{T}. The following adjustment is then performed every 50 iterations:

- **if** $\xi^T \leq \xi^{REF} - 0.05$, **then** $\omega = 1.2 \cdot \omega$;
- **if** $\xi^T \geq \xi^{REF} + 0.05$, **then** $\omega = 0.85 \cdot \omega$.

We impose a maximum size on both sub-populations of $\lambda + \mu$. As soon as one of the populations has reached the maximum size we select the best μ unique solutions with respect to their distance to proceed to the next generation.

4.7 Parameter tuning

In the previous sections many parameters have been introduced and discussed. The performance of the main algorithm obviously depends on the values of these, possibly correlated, parameters. Due to limited time it was not possible to test all possible combinations. Therefore, we considered those parameters we expected to most significantly influence the performance in terms of solution quality and computation

Parameter	Parameter name	Range				Final parameter
μ	Population Size	50				50
λ	Number of offspring in a generation	100				100
PR_E	education rate	0	0.01	0.1	0.3	0.05
It_{NI}	number of iterations	100	200	500		500
It_{DIV}	Diversify iterations	∞				∞
ξ	Reference proportion of feasible individuals	0.3				0.3
ω	penalty	0.1	1	10		10

Table 3: Tuning Results

time. This selection is based on gained experience while testing the algorithm and on insights provided by Vidal et al. (2012). Table 3 shows all parameters used in the algorithm and their final value. The tested parameters are the number of offspring, It_{NI}, the education rate, PR_E, and the starting penalty, ω. The parameters that were not tested are the population size μ, the number of offspring in a generation, λ, the number of iterations before diversification, It_{DIV}, and the reference proportion of feasible individuals, ξ^{REF}. Vidal et al. (2012) use $\mu = 25$ to create an initial population size of 100 individuals. Since the initialization procedure does not require significant amounts of computational time and to ensure a varied population we set μ equal to 50. The population size and the number of offspring in a generation, λ, are related to each other since their sum is the maximum sub-population size $\mu + \lambda$, λ was set to $\lambda = 100$, based on Vidal et al. (2012). The education rate PR_E was tested for four different values, $PR_E = 0$, which means that no offspring is educated, $PR_E = 0.01$, $PR_E = 0.1$, and $PR_E = 0.3$. Vidal et al. (2012) use an education rate of 1 implying that all offspring are educated with local search procedures. Since these local search procedures are very time consuming, especially for the larger instances, we did not test the algorithm for higher rates than $PR_E = 0.3$. We tested these rates on 100 different instances with the number of customers ranging from 50 to 120, the number of depots varying from 2 to 5 and the number of different products ranging from 1 to 5, and a maximum travel time of 285. The depots and customers contained in the set V are situated in the Cartesian plane such that $v_i \in [0, 100] \times [0, 100]$. We found that the computation time linearly increased in the education rate. Since higher rates did not improve solutions significantly we chose a value of $PR_E = 0.05$ to find higher quality solutions without increasing computation time too much. The number of iterations It_{NI} equals the total number of generated offspring and was tested for the values 100, 200, and 500. Using $It_{NI} = 500$ resulted in the best solution quality while the computation time was still acceptable. Vidal et al. (2012) use $It_{DIV} = 0.4 It_{NI}$, where $It_{NI} > 10^4$. Since the diversify procedure only makes sense after a lot of iterations we decided to never diversify a population. To search on the boundaries of feasibility, the reference proportion of feasible individuals in the population was set to $\xi = 0.3$. For the starting penalty, ω, we tested three different values such that $\omega \in \{0.1, 1, 10\}$. The test showed that for smaller instances $\omega = 0.1$ performed better while for larger instances $\omega = 10$ was preferred. The final value was set to $\omega = 10$ to make the algorithm better suitable for larger instances.

5 Solution Quality

Since the particular setting of the MDMPVRP we consider is not discussed earlier in literature, there are no test instances to benchmark the quality of our algorithm. Hence, we used a set of 1000 self generated instances. We varied the number of products from $P = 1$ to $P = 5$, the number of depots from $D = 2$ to $D = 5$, and the number of customers from $N = 50$ to $N = 140$. The depots and customers contained in the set V are situated in the Cartesian plane such that

			Initialization (average)		GA (average)	
Instances	Depots	Customers	Distance	Comp. time (s)	Distance	Comp. time (s)
1-100	4	140	1545.11	30.59	1284.95	2649.03
101-200	3	140	1474.10	44.51	1231.36	4150.95
201-300	4	100	1355.96	15.80	1120.62	727.81
301-400	3	100	1272.30	20.58	1063.72	1067.62
401-500	4	80	1186.86	10.21	988.17	327.00
501-600	3	80	1124.57	12.98	936.97	464.70
601-700	2	80	1051.93	20.62	889.71	849.95
701-800	3	60	975.97	7.86	825.28	201.47
801-900	2	60	908.33	11.19	782.45	302.83
901-1000	2	40	749.81	5.41	653.57	93.43

Table 4: Solutions and computation time

$v_i \in [0, 100] \times [0, 100]$. The maximum travel time of all instances is 285. In Table 4 the features of the instances are summarized. We have 10 groups, each of size 100, with a different number of depots and/or number of customers. For each group the number of products increases by one every 20 instances. That is, instances 1-20 are all with 1 product, instances 21-40 all have 2 products and instances 81-100 all have 5 products. The algorithm was implemented in R on an Intel(R) Core Duo CPU computer with 2.33GHz clock.

To benchmark the performance of our algorithm we compare our results with a more simple and more straightforward heuristic. Since the initialization procedure of our Algorithm randomly assigns customers we can consider the initialization as a naive heuristic. Obviously, the crossover procedure should enhance the solution quality. So, it is interesting to see the difference in solution quality between the initialization and crossover procedure. To inspect the performance of the algorithm the travel distance is stored for each generated individual. Consider, for example, instance 401 with 4 depots, 80 customers and 1 product. The first 200 individuals are generated in the initialization and individuals 201-700 are offspring, see Figure 3. This Figure displays both feasible and infeasible solutions. Note that feasibility depends on the current value of the penalty. It is clear that almost all offspring have a higher solution quality than the individuals created in the initialization and that the solution quality stabilizes quickly. When we consider instance 500, see Figure 4, where customers demand at most 5 different products instead of 1, we observe the following. Again, the solution quality of the offspring is higher than the individuals created in the initialization. However, contrary to the convergence observed in Figure 3, we do not identify a smooth converging sequence. For both instances we see that the offspring are better than the initial individuals. This holds for all other instances as well, which indicates that creating offspring makes sense in terms of solution quality. Now, consider

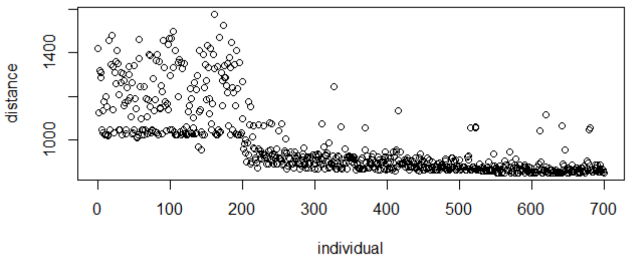


Figure 3: Distance per individual for instance 401, consisting of 4 depots, 80 customers and 1 product.

the solutions of the entire set of instances. For each instance the algorithm returns a feasible solution with the lowest distance for the current population. These

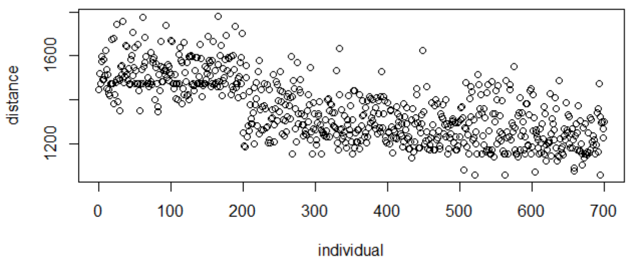


Figure 4: Distance per individual for instance 500, consisting of 4 depots, 80 customers and 5 products.

distances are presented in Figure 5. We observe that the travel distance decreases, on average, with each group of 100 instances, which is intuitive since the number of customers and/or depots decreases every 100 instances. Another observation is that within each group the travel distance increases. This is a natural consequence of increasing the number of different products every 20 instances, resulting in more complex and longer routes. The same pattern holds for the computation times, given in Figure 6. What immediately stands out is that the computation time of the first 200 instances is extremely high, ranging from 20 minutes to almost 3 hours. This is due to the education procedure which is extremely time consuming for larger instances. In the set of instances we used for parameter tuning we did not encounter these extreme computation times. One explanation is that in the tuning set the largest number of customers was 100 whereas in the test set we considered instances up to 140 customers. By considering no education, we were able to solve each instance within 20 minutes, see Section 6. The computation time for the other instances is on average 504 seconds. For each group of instances the average best initial solution, the average computation time, the final solution, and the computation time are given in Table 4. For the initialization it seems that for instances with the same number of customers but with different number of depots the computation time is less when there are more depots, which is a counter intuitive result. A possible explanation is that since it is harder for the algorithm to find a feasible combination of customers per depot when there are less depots, we find more split customers. When we look at the computation times for the entire algorithm we see the same pattern as for the initialization procedure. This can be explained by the same reason as for the initialization since the crossover procedure assigns left-over customers to depots in the same manner as in the initialization procedure.
All the results can be found in the accompanying excel file.

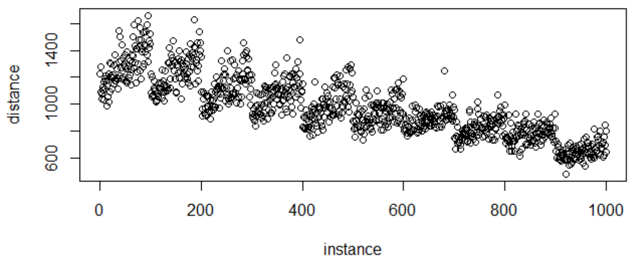


Figure 5: Distance per instance.

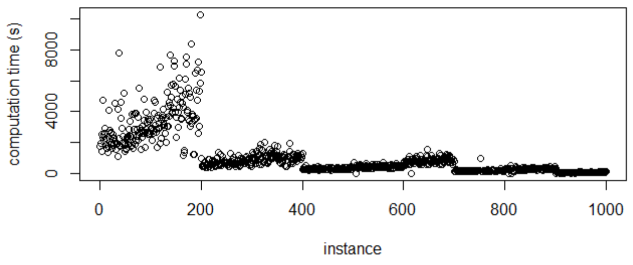


Figure 6: Computation time per instance.

6 Experimental insights

The computation time was discussed in Section 5. For some instances we encountered extremely high computation times, up to almost 3 hours. Most likely this was due to the education procedure. To test the impact of the education procedure on the solution quality and the computation time we recalculated these values without the education procedure, i.e., $PR_E = 0$,

	With education		Without education	
Instance	Distance	Comp. Time	Distance	Comp. Time
198	1399.61	10269.96	1386.20	1172.40
181	1353.39	8338.95	1350.50	1182.44
139	1187.43	7664.70	1186.03	1123.08
170	1386.74	7551.63	1525.70	1299.4
148	1309.61	7271.61	1321.29	1099.7
195	1284.56	7204.31	1311.57	1089.28
171	1297.68	7103.85	1270.07	1191.28
147	1248.31	6950.35	1286.81	1416.47
120	1110.21	6871.47	1150.42	1252.74
192	1457.37	6653.20	1439.14	1113.00

Table 5: Comparison solution quality and computation time with $PR_E \in \{0, 0.05\}$

for the instances with a computation time over one hour. In Table 5 the first 10 worst case instances, in terms of computation time, are given together with their initial distance and computation time and their new distance and computation time. Without much change in distance the computation time reduces significantly with a factor 6 on average. Also, we see that some solutions are even better than the original solutions. Another parameter that influences the computation time is the rate at which the penalty ω is updated. Currently, the penalty is updated every 50 offspring. As discussed in 4.6, the infeasible population or the entire population has to be updated when the ratio of feasible solutions ξ^T is, respectively, lower or higher than the reference ratio ξ^{REF}. Updating the (sub)population is relative costly in terms of computation time. We computed instance 500 again but now with $It_{NI} = 10,000$ iterations and recorded the penalty change. The returned solution now has an objective value of 1034 instead of 1055 found for the old solution. In Figure 7 we see that in contrast to Figure 4 the distance corresponding to the best solution becomes lower when increasing the number of iterations. Again, note that there are also infeasible solutions in the population. In Figure 8 we see that the penalty ω converges from its starting value of $\omega = 10$ to around $\omega = 0.1$ and stabilizes after 28 updates. This means that the population needs $28 \cdot 50 = 1400$ iterations to converge to its preferred ratio of feasibility ξ^{REF}. In this run only 155 updates occurred, this means that for 45 penalty checks the penalty was inside the preferred range of $\xi^{REF} \pm 0.05$. This implies that the instances

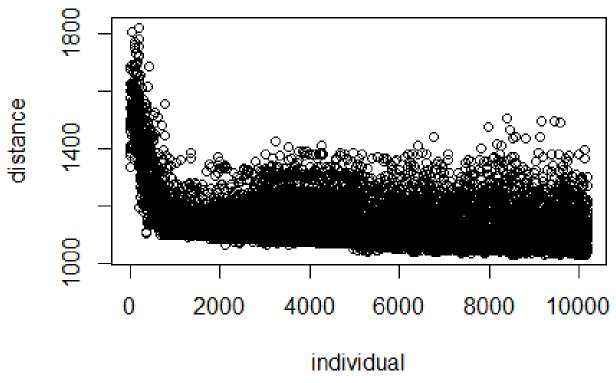


Figure 7: Distance per individual for instance 500 with 10,000 iterations.

used for parameter tuning were not representative for the benchmark set of instances since the education procedure requires more time for larger instances. To save computation time the penalty ω could be adjusted right after the initialization phase to its right value to save the computation time for the convergence of the penalty. Furthermore, we see that increasing the number of iterations It_{NI} results in better offspring.

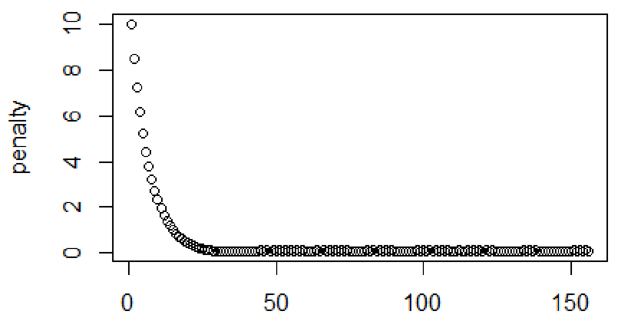


Figure 8: Penalty adjustment behaviour for instance 500.

7 Conclusion

We proposed a new Genetic Algorithm to solve various instances for the Multi-Depot Multi-Product Vehicle Routing Problem (MDMPVRP). In this problem we consider a set of customers with known demand for multiple products. The supplies are distributed over multiple depots and are to be transported by an unlimited fleet of vehicles. Vehicle capacity does not impose limits in practice, but there is a maximum travel time T per vehicle. Due to the lack of available literature in the Vehicle Routing Problem with multiple products we used current state-of-the-art meta-heuristics to construct a Genetic Algorithm capable of solving the MDMPVRP. The Algorithm allows a relaxation of the maximum travel time constraint to find solutions on the borders of feasibility. In order to do so, we use a penalty factor to punish infeasible solutions and control the ratio of feasible solutions in the entire population. Since we could not compare results with literature benchmarks, we randomly created various instances. These instances differ in the number of customers, depots, and products. Results showed that our algorithm significantly performs faster in case fewer depots are considered. On the contrary, the number of different products did not influence results in terms of running time. If more than 100 customers are considered updating penalized costs and the education procedure turned out to be too time consuming. Therefore, we suggest to consider alternative ways of applying mutation, coping with the referenced proportion of feasible solutions, or to find faster ways to educate solutions. Making the algorithm faster would allow us to increase the number of offspring and thereby find better solutions. This would also result in other values for the parameters since at this point the computation time constrains the parameter values and thus the potential of the algorithm. As an alternative to solving the MDMPVRP, a change in inventory policies may be considered. That is, it could be possible that higher inventory levels at each depot ensure that the problem can be solved as a MDVRP, which results most likely in shorter routes. Hence, one can consider the trade off between additional costs associated with increased inventory and lower routing costs due to less split customers. Therefore, one could also consider a more generalized version of the MDMPVRP in which the inventory policy per depot is also considered, or at least compare the results of the two approaches independently.

References

Cordeau, Jean-François, Michel Gendreau, and Gilbert Laporte (1997). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks 30*(2), 105–119.

Dantzig, G. B. and J. H. Ramser (1959). The truck dispatching problem. *Management Science 6*(1), pp. 80–91.

Dijkstra, Edsger W (1959). A note on two problems in connexion with graphs. *Numerische mathematik 1*(1), 269–271.

Karp, Richard M (1972). *Reducibility among combinatorial problems*. Springer.

Mendoza, Jorge E., Bruno Castanier, Christelle Gueret, Andres L. Medaglia, and Nubia Velasco (2010). A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands. *Computers & Operations Re-*

search 37(11), 1886 – 1898. Metaheuristics for Logistics and Vehicle Routing.

Pisinger, David and Stefan Ropke (2007). A general heuristic for vehicle routing problems. *Computers & operations research 34*(8), 2403–2435.

Prins, Christian (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research 31*(12), 1985–2002.

Savelsbergh, Martin and Marc Sol (1998). Drive: Dynamic routing of independent vehicles. *Operations Research 46*(4), 474–490.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research 60*(3), 611–624.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research 231*(1), 1 – 21.

A Appendix

The Multi-Depot Multi-Products Vehicle Routing Problem as described in Section 2 can be formulated by the following Mixed Integer Linear Program.

Sets

V	set of vertices
E	set of edges
R	set of routes
$\underline{N}$	set of customers
$\underline{D}$	set of depots
$\underline{P}$	set of products

Parameters

c_{ij}	$v_i, v_j \in V$	cost of traveling edge $e_{ij} \in E$
q_{np}	$n \in \underline{N}; p \in \underline{P}$	demand of customer n for product p
s_{dp}	$d \in \underline{D}; p \in \underline{P}$	supply in depot d of product p
T		maximum travel time

Variables

x_{ijrd}	$v_i \in V;\ v_j \in V;\ r \in R;\ v_d \in V^{DEP}$	v_j is immediately visited after v_i in route r starting at depot d
y_{npd}	$n \in N; d \in D; p \in P$	link demand of customer n for product p to depot d

Objective

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V} \sum_{r \in R} \sum_{d \in V^{DEP}} c_{ij} x_{ijrd} \tag{1}$$

Constraints

$$s_{dp} - \sum_{v_n \in V^{CUST}} y_{npd} q_{np} \geq 0 \qquad v_d \in V^{DEP}; p \in P \tag{2}$$

$$\sum_{v_d \in V^{DEP}} y_{npd} = 1 \qquad v_n \in V^{CUST}; p \in P \tag{3}$$

$$\sum_{v_d \in V^{DEP}} x_{id'rd} = 0 \qquad v_d, v_{d'} \in V^{DEP}; d \neq d'; r \in R \tag{4}$$

$$\sum_{v_i \in V} x_{ijrd} - \sum_{v_j \in V} x_{jird} = 0 \qquad v_j \in V; v_d \in V^{DEP}; r \in R \tag{5}$$

$$\sum_{v_i \in S} \sum_{v_j \in S} x_{ijrd} \leq |S| - 1 \qquad S \in V^{CUST}; |S| \geq 2; v_d \in V^{DEP}; r \in R \tag{6}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} c_{ij} x_{ijrd} \leq T \qquad v_d \in V^{DEP}, r \in R \tag{7}$$

$$x_{ijrd} \in \{0, 1\} \qquad v_i \in V; v_j \in V; v_d \in V^{DEP}; r \in R \tag{8}$$

$$y_{npd} \in \{0, 1\} \qquad v_n \in V^{CUST}; p \in P; v_d \in V^{DEP} \tag{9}$$

By adding constraint 2 we guarantee that at each depot the demand for the products of the customers assigned to that depot can be supplied. Constraint 3 makes sure that for each customer all different products are covered by exactly one depot. Furthermore, constraint 4 makes sure that a route starting from a certain depot cannot visit any other depot. The constraint 5 guarantees flow conservation. Sub-tours are eliminated by including constraint 6. The constraint 7 makes sure that routes cannot exceed the maximum travel distance.

Chapter 3
Variable Neighborhood Search

D. Witteveen, R. te Wierik

1 Introduction

There are many practical examples of vehicle routing problems where inventory plays a role. Consider pharmacies that deliver medicines to their clients, or libraries that deliver books at home. In these examples, planners are not only concerned with the route length, they need to consider the inventory level as well.

These type of problems can be seen as an extension of the classical vehicle routing problem. A vehicle routing problem concerns a number of customers and a number of vehicles that all start from one depot. Routes need to be determined such that each customer is visited by one vehicle. Now consider there to be multiple depots, each with a number of vehicles. Furthermore, consider there to be multiple types of product. The depots have a limited amount of inventory and the customers have a given demand for each product. The customers must receive the products they demand, but each depot cannot deliver more than it has in stock. An important difference is that customers can now receive their products from more than one vehicle, we call this split deliveries. These problems can be referred to as multi-depot vehicle routing problems with inventory limitations (MDVRPI).

In this paper, we provide a heuristic approach for solving these multi-depot vehicle routing problems with inventory limitations. We provide an algorithm based on variable neighborhood search (VNS), which is a form of local search where we systematically alternate between different types of neighborhoods. We give a detailed description of our implementation of VNS to solve the MDVRPI. Furthermore, we test our algorithm on a set of randomly generated test instances. For comparison, we also provide solutions based on a very simple method. We compare objective values and computation times.

To give a overview of the relevant literature surrounding this problem, we start with the classical VRP. The classical VRP was first introduced by Dantzig and Ramser (1959). Since then, many studies have focused on one of the extensions or variants of the classical VRP. Overviews of the many VRP variants are given by Toth and Vigo (2002), Golden et al. (2008), Vidal et al. (2013). A part of the extensions to the VRP include multiple depots. Variants with multiple depots and the possibility of split deliveries are first introduced by Dror and Trudeaut (1989). For an overview of the VRPs with split deliveries, see Archetti and Speranza (2012). As mentioned earlier, as to our knowledge no variants that exist in current literature have considered inventory limitations.

Lenstra and Rinnooy Kan (1981) showed that the VRP and most of its variants are NP-hard problems. At the end of Section 2 we will show that the MDVRPI is NP-hard by giving a polynomial sized reduction from VRP. Since most of the VRP variants are NP-hard, it is very common to use a heuristic approach to solve the problem. We focus on one heuristic approach in particular, called variable neighborhood search. VNS is a local search method where we systematically alternate between different types of neighborhoods, it was first introduced by Mladenović and Hansen (1997). A few years later, Polacek et al. (2004) used a VNS based algorithm to solve a VRP variant with multiple depots and time windows. Salhi et al. (2013) used a VNS based algorithm for solving a VRP variant with multiple depots and a heterogeneous fleet, their algorithm found better solutions for 23 out of 26 problem instances published in the literature. It was their work that inspired us to use a VNS based algorithm for the MDVRPI.

The remainder of this paper is organized as follows. A formal description of the problem is given in Section 2. The variable neighborhood search algorithm is described in detail in Section 3. Section 4 then gives the computational results of using the algorithm on a number of randomly generated instances of the MDVRPI. The conclusions of this paper and suggestions for further research are presented in Section 5.

2 Problem formulation

In this section, we give a formal definition of the problem, including all the assumptions made and the integer linear programming formulation.

Consider a set of depots and a set of customers, with known distances between each of them. Furthermore, consider a number of products, each depot has a given amount of inventory of these products and each customer has a given amount of demand for these products. Each depot has vehicles available for transporting the products from the depot to the customers. The objective is to minimize the transportation costs under the conditions that all demand must be fulfilled, each depot cannot deliver more products than it has in stock, and all routes should not exceed a given maximum length. One assumption that we make in the problem definition, is that the total inventory is enough to fulfill the total demand. In addition, it is allowed to have split deliveries, which means that a customer can be served from more than one depot. The task is to select the amount that each depot delivers to each customer and to select the routes that accompany this delivery plan.

We make three main assumptions. First, we assume that each depot has an unlimited amount of vehicles available. Second, we make the assumption that each vehicle has an unlimited capacity. These assumptions are reasonable when we consider the practical examples of the pharmacy and the library. In these examples, vehicle capacity and amount of vehicles does not impose limits in practice. Last, we assume that the transportation costs are proportional to the sum of route lengths, i.e. we do not consider any fixed costs for using a truck.

Before giving the integer linear programming formulation (ILP) of the MDVRPI, we introduce the following variables.

N :	the total number of customers,
M :	the total number of depots,
L :	the total number of products,
Q_{ip} :	customer i's demand for product p,
I_{kp} :	inventory level of product p at depot k,
D_{ijk} :	distance from customer i to j when starting from depot k,
L :	maximum route length,
R_k :	the number of routes originating from depot k,
a_{ikp} :	amount of product p delivered from depot k to customer i,
x_{ijrk} :	binary variable equal to 1 if edge i-j is in route r from depot k, equal to 0 otherwise.

Note that the distances, given by D_{ijk}, only differ among depots, that is when $i = 0$ or $j = 0$. When $i \geq 1$ and $j \geq 1$, the distances D_{ijk} do not depend on the depot k. That is, only the distances from depot to customers and vice versa change, the distances between customers remain the same, regardless of what depot the route starts in.

The ILP formulation is given by

$$\min \sum_{k=1}^{M} \sum_{r=1}^{R_k} \sum_{i=0}^{N} \sum_{j=0}^{N} D_{ijk} \cdot x_{ijrk}, \tag{1}$$

subject to

$$\sum_{i=0}^{N} x_{ijrk} = \sum_{i=0}^{N} x_{jirk} \qquad \forall\ j = 0,..,N,\ r = 1,..,R_k,\ k = 1,..,M, \tag{2}$$

$$\sum_{i=0}^{N} \sum_{j=0}^{N} D_{ijk} \cdot x_{ijrk} \leq L \qquad \forall\ r = 1,..,R_k,\ k = 1,..,M, \tag{3}$$

$$a_{jkp} \leq \sum_{r=1}^{R_k} \sum_{i=0}^{N} x_{ijrk} \cdot Q_{jp} \qquad \forall\ j = 1,..,N,\ k = 1,..,M,\ p = 1,..,L, \tag{4}$$

$$\sum_{k=1}^{M} a_{ikp} = Q_{ip} \qquad \forall\ i = 1,..,N,\ p = 1,..,L, \tag{5}$$

$$\sum_{i=1}^{N} a_{ikp} \leq I_{kp} \qquad \forall\ k = 1,..,M,\ p = 1,..,L, \tag{6}$$

$$\sum_{r=1}^{R_k}\sum_{i=0}^{N} x_{ijrk} = 1 \qquad \forall\, j = 0,..,N,\; k = 1,..,M, \tag{7}$$

$$\sum_{v_i \in S}\sum_{v_j \in S} x_{ijrk} \leq |S| - 1 \qquad \forall\, r = 1,..,R_k,\; k = 1,..,M; \quad S \subseteq V \setminus \{0\}; |S| \geq 2, \tag{8}$$

$$R_k \in \mathbb{N} \cup \{0\}, \qquad \forall\, r = 1,..,R_k,\; k = 1,..,M, \tag{9}$$

$$a_{jkp} \in \mathbb{N} \cup \{0\}, \qquad \forall\, j = 0,..,N,\; k = 1,..,M,\; p = 1,..,L, \tag{10}$$

$$x_{ijrk} \in \{0,1\}, \qquad \forall\, i,j = 0,..,N,\; r = 1,..,R_k,\; k = 1,..,M. \tag{11}$$

The objective function (1) is given by the sum of lengths of all routes from all depots. Constraints (2) represent the flow conservation constraints, stating that any vehicle that arrives at a customer must also leave that customer and any vehicle that leaves a depot must also return to that depot. Constraints (3) make sure that the length of each route does not exceed the maximum route length. The constraints given by (4), (5), and (6), deal with the delivery of products to customers. Constraints (4) make sure that the amount delivered from a depot to a customer can only be positive when that customer is visited in one of the routes originating from that depot. When the customer is not visited in any of the routes from that depot, the right hand side is equal to zero and the amount delivered is also forced to be zero. When the customer is visited in at least one of the routes from that depot, the right hand side is at least equal to the demand of that customer and therefore does not restrict the amount delivered. Constraints (5) make sure that the total amount delivered to a customer is exactly equal to the demand of that customer. Constraints (6) make sure that the total amount that a depot delivers to customers does not exceed the inventory of that depot. Constraints (7) tighten the formulation by not allowing customers to be visited by more than one route from the same depot. When a depot delivers a positive amount to a customer, that customer must be in at least one of the routes from that depot. However, since the capacity of the vehicles is not an issue, we only need the customer to be in one of the routes from that depot, a customer being in more than one route would always be superfluous. Constraints (8) are standard subtour elimination constraints. Finally, constraints (9), (10), and (11), define the range of our decision variables.

The MDVRPI is an NP-hard problem, which we will show by a reduction from the classical VRP. As mentioned before, the VRP is a known NP-hard problem (Lenstra and Rinnooy Kan, 1981). We claim that the VRP is polynomial-time reducable to the MDVRPI. Consider any instance of VRP with N customers. We could formulate it as a MDVRPI with N customers, one depot and one product. Let us give each customer a demand of one unit and give the depot an inventory of N units. Note that the depot has enough inventory to deliver to all N customers, hence inventory limitations do not play a role anymore. We have now reduced the instance of the VRP to an instance of the MDVRPI, where the routes in the solution of the MDVRPI directly correspond to the routes in the solution of the VRP.

3 A variable neighborhood search algorithm

In this section an approach to solve the MDVRPI is described. The algorithm consists of several heuristics. First, the general structure of the heuristic is explained. Then, the heuristics are explained in more detail and, finally, all parameters and their values are discussed.

3.1 Heuristic structure

Figure 1 shows the pseudo code of the algorithm. The algorithm starts with an initialization phase, that is described in Section 3.2.1. Then the algorithm loop (*ALG_loop*) starts for *Max_ALG* iterations and it consists of two parts: the variable neighborhood search and a diversification procedure.

The variable neighborhood search starts with "shaking" the current solution after which several local search heuristics are applied. The heuristics used in the variable neighborhood search are explained in more detail in Sections 3.2.2 and 3.2.3. If the variable neighborhood search finds an improvement, it is repeated with the same shaking heuristic for a maximum of *Max_count* times. If it does not find an improvement, or when the same shaking heuristic is used for *Max_count* times, the variable neighborhood search is repeated with the next shaking heuristic, until all shaking heuristics have been used.

At the end of each loop of the variable neighborhood search, it is checked whether the current solution is better then the best known solution or not. Finally, the diversification procedure, as described in Section 3.2.4, is applied to search in a wider range of the feasible region.

```
Initialization;
ALG_loop = 0;
while ALG_loop <= Max_ALG do
 |     ALG_loop = ALG_loop + 1;
 |     k = 1;
 |     Count = 0;
 |     while k <= Max_k do
 |      |     Count = Count + 1;
 |      |     Shaking(k);
 |      |     Insert10_interroute;
 |      |     Swap_interroute;
 |      |     Two_opt;
 |      |     Swap_intraroute;
 |      |     Insert10_intraroute;
 |      |     if Improvement of Solution then
 |      |      |     if Count == Max_count then
 |      |      |      |     k = k + 1;
 |      |      |     end
 |      |     else
 |      |      |     k = k + 1;
 |      |     end
 |     end
 |     if Solution better than BestFound then
 |      |     BestFound = Solution;
 |     end
 |     Diverse;
end
Return BestFound
```

Figure 1: Pseudo code of the algorithm

3.2 Detailed heuristics

In this section we will describe the heuristics in the order of use by the algorithm.

3.2.1 Initial solution

The initialization phase assigns the customers in order of relative closeness to their nearest depot. Customers that are not assigned to their nearest depot are considered *borderline customers*. The relative closeness is measured by the distance to the nearest depot divided by the distance to the second nearest depot. Customers are only assigned to their nearest depot if the distance to the nearest depot divided by the distance to the second nearest depot is smaller than *Epsilon*. If a depot's inventory restriction does not allow a customer to receive all his demand from that depot, the customer will receive only the demand that can be met and for the remaining demand the customer will be considered a borderline customer.

One initial route per depot is created using the nearest neighbor algorithm. If the distance of this route is larger than the maximum allowed distance *Max_Dist*, the route will be split into several routes. These splits are made at the point where the initial route cannot reach the next customer without exceeding the distance restriction. Then, the borderline customers are inserted in a route of the nearest depot that can meet their demand using minimal insertion. If a borderline customer cannot be inserted in any of the routes of the depot, a new route is created.

3.2.2 Shaking heuristics

Three heuristics are used to 'shake' the current solution (so *Max_k*=3) and search a wider range of solutions in the feasible region. We will concisely describe, in order of use, the *1-1 interchange*, the *2-0 shift* and the *perturbation*. If any of the shaking heuristics cannot find a feasible shaking, the same heuristic is tried again for a maximum of *Max_Shake* times. The shaking heuristics immediately found feasible shakes most of the times. However, in the case when the shaking heuristic did not find a feasible shake immediately, it was able to find one within very few iterations. So, *Max_Shake*=4 was considered enough.

The *1-1 interchange*: this heuristic tries to randomly interchange a customer from one depot with a customer from another depot. The interchange is limited to the routes the customers originate from. First a random customer is picked from a random route. Then all possible routes from other depots are checked in a random order if they have a customer that can be interchanged with the first customer. If none of the routes of the other depots have a customer that can be interchanged, the routes from the same depot are considered.

The *2-0 shift*: in a random route a random pair of adjacent customers is selected. Then it is tried to move them to a route of a random other depot. If this is not possible due to inventory or route length restrictions, it is tried to move the pair to another route of the same depot.

The *perturbation*: the main idea for this heuristic is from Salhi and Rand (1987). A random customer from a random route is selected and is inserted in a route from another depot without considering inventory limits using cheapest insertion. Then, a random customer is picked from all customers served by that depot for which removal from that depot makes that the inventory constraints are not violated. For this customer we check if this customer can be inserted in a route from other depots without violating inventory and route length constraints.

In the case of only two depots, the perturbation is very similar to the 1-1 insertion. So, in order to save some computation time, the 1-1 insertion is used again instead of the perturbation.

3.2.3 Local search heuristics

Our algorithm uses seven local search heuristics, which are descried in order of appearance in the local search: *1-insertion interroute (inter- and intradepot*, *swap interroute (inter- and intradepot*, *2-opt*, *swap intraroute* and *1-insertion intraroute*. This order of local search neighborhoods is chosen based on the level of complexity of the neighborhoods. Salhi and Sari (1997) showed this order to be an effective one. If a local search heuristic improves the solution, it is repeated until no further improvement is made or when it reaches a maximum number of loops. This maximum number of loops is set equal to 150 for all local search heuristics, due to small running times and the fact that this maximum is rarely met.

1-insertion interroute (inter- and intradepot): every customer in every route is systematically checked for removal from that route and insertion in any other route. The *interdepot* version only considers routes from other depots and therefore has to check for inventory restrictions as well. The *intradepot* only considers other routes from the same depot. For both versions, the best insertion is chosen among all feasible options.

Swap interroute (inter- and intradepot): every customer in every route is systematically checked for a swap with another customer from an other route. As with the 1-insertion, the *interdepot* version only considers routes from other depots and has to check inventory restrictions in both depots. The *intradepot* only considers other routes from the same depot. For both versions, the best insertion is chosen among all feasible options.

2-opt: this local search heuristic was first proposed by Croes (1958). Two edges are taken from a route and replaced by two other ones, such that the result is a route again. This is evaluated for *Num_2opt* random combinations of two edges. The combination with the greatest improvement is chosen to be the new solution. By empirical testing we found that the following setting for *Num_2opt* gives a good balance between the chance of finding improvements and computation time: *Num_2opt* $= \min\{800, c^2\}$, where c is the number of customers in the route.

Swap intraroute: is similar to the swap interroute, but considers only swaps within the same route.

1-insertion intraroute: is similar to the 1-insertion interroute, but considers only insertions within the same route.

3.2.4 Diversification

When the local search has finished, it is checked whether it yielded a better solution than the current best solution. Then, to search in a broader range of the feasible region, a diversification procedure is applied.

The diversification procedure is used to change each of the routes. Therefore, for every depot one giant tour is created by concatenating all its routes. Then, a pair of adjacent customers (c_1, c_2) is chosen which both were not the first or last customer of a route. For each of the customers a new giant tour is created, using these customers as the first customers of the tour. So, one tour is given by $\{c_2, c_2 + 1, \ldots, c_1 - 1, c_1\}$ and the other tour is given by $\{c_1, c_1 - 1, \ldots c_2 + 1, c_2\}$. Then, these tours are split in new routes if necessary in the same way as described in Section 3.2.1. Finally, the set of routes with the smallest total distance is selected.

3.3 Parameter tuning

A test set of one hundred instances was used for parameter tuning. The characteristics of these instances are given in Table 1.

Table 1: Overview of one hundred test instances. Five instances were used for each combination of number of depots, products and customers.

Number of customers	Number of products	Number of depots
50	1,2,3,4,5	2
70	1,2,3,4,5	3
100	1,2,3,4,5	3
120	1,2,3,4,5	4

Initial solutions were computed for several values of *Epsilon*. In fifty of the hundred instances, the best initial solution was given for *Epsilon* = 1, that is, borderline customers are not used in the first place. The other fifty instances have an average best value of *Epsilon* of 0.824. The value of 0.9 for *Epsilon* appears to be best in thirty cases and, since it is almost equal to the average value of optimal *Epsilon*, we take 0.9 as the optimal setting for *Epsilon*.

Using a test run with a very large value for *Max_ALG* (180) we determined for what amount of loops the algorithm stopped finding better solutions regularly, that is when no 1% decrease in the objective value was found for 15 iterations. The results differed a lot among the instances and therefore *Max_ALG* is made problem dependent. It is given by $3(D^2) + 4P$, where D is the number of depots and P is the number of problems.

4 Results

The algorithm was written in Matlab and the experiments were run on a computer with an Intel Core 2 Duo E6550 processor with 2.33 GHz and 2 GB RAM. One thousand instances were run and their characteristics are given in Table 2.

Since there is no literature on this problem, the performance of the algorithm is compared with a simple heuristic, namely our initial solution. Tables 3 and 4 give a summary of the results obtained by the algorithm and the simple method, respectively. Each of the objective values and computation times is an average over twenty

Table 2: Overview of one thousand instances. Twenty instances were used for each combination of number of depots, products and customers.

Number of customers	Number of products	Number of depots
40	1, 2, 3, 4, 5	2
60	1, 2, 3, 4, 5	2, 3
80	1, 2, 3, 4, 5	2, 3, 4
100	1, 2, 3, 4, 5	3, 4
140	1, 2, 3, 4, 5	3, 4

instances with that set of characteristics. The average computation times of the algorithm range from half a minute for the smallest problems to ten and a half minutes for the largest problems, which is fast enough for most real world applications.

Table 3: Average objective values and computation times in minutes of the algorithm

Products:		1		2		3		4		5	
Customers	Depots	Obj	CPU	Obj	CPU	Obj	CPU	Obj	CPU	Obj	CPU
40	2	599	0.45	618	0.68	629	0.52	655	0.73	678	0.97
60	2	739	0.72	758	0.73	765	1.27	779	1.55	790	1.07
60	3	710	1.28	739	1.38	774	2.00	781	1.57	834	2.15
80	2	860	1.08	857	1.20	867	1.58	883	1.20	903	1.85
80	3	843	1.98	854	1.93	890	2.12	884	3.65	927	2.85
80	4	810	3.78	847	4.10	865	3.62	903	3.02	954	3.73
100	3	926	3.03	954	2.48	1024	4.05	1000	3.30	1039	4.15
100	4	923	10.27	950	4.30	997	5.23	1002	4.92	1056	4.65
140	3	1096	3.57	1119	4.42	1186	5.85	1187	9.35	1246	7.85
140	4	1078	7.80	1113	10.28	1154	7.75	1202	7.18	1241	9.17

Table 4: Average objective values and computation times in seconds of the simple method

Products:		1		2		3		4		5	
Customers	Depots	Obj	CPU	Obj	CPU	Obj	CPU	Obj	CPU	Obj	CPU
40	2	911	0.01	944	0.01	976	0.01	1035	0.01	1067	0.01
60	2	1110	0.01	1238	0.01	1284	0.01	1230	0.01	1247	0.01
60	3	1168	0.02	1319	0.02	1425	0.02	1351	0.02	1493	0.02
80	2	1317	0.01	1373	0.01	1440	0.01	1406	0.01	1498	0.02
80	3	1481	0.02	1538	0.02	1629	0.02	1541	0.02	1698	0.02
80	4	1518	0.02	1658	0.02	1704	0.02	1842	0.02	1900	0.02
100	3	1633	0.02	1679	0.02	1902	0.02	1961	0.02	1911	0.02
100	4	1705	0.02	1889	0.02	2036	0.02	1977	0.02	2209	0.03
140	3	1911	0.02	1984	0.02	2217	0.03	2330	0.03	2374	0.03
140	4	1917	0.03	2211	0.03	2317	0.03	2490	0.03	2440	0.03

Clearly, the objective values of the algorithm increase when the number of customers and products increase. For the number of depots this cannot be said so easily: for problems with fewer products the objective value decreases when there are more depots, for problems with more products the objective value can increase when there are more depots. This is caused by the fact that the distribution of the inventory is more restrictive when there are more products.

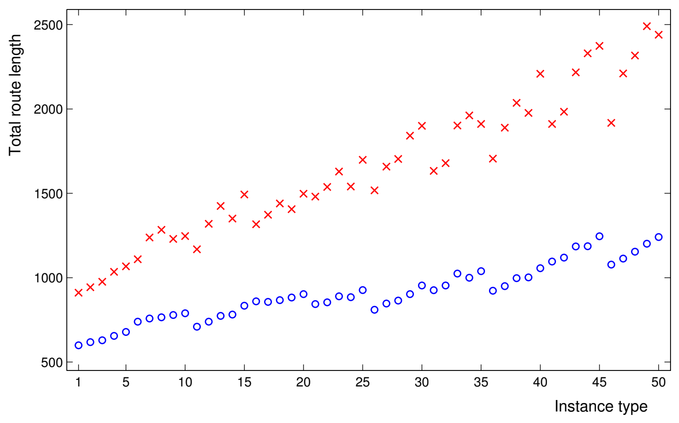


Figure 2: The objective values of the algorithm (blue circles) compared to the simple method (red crosses) for all 50 instance types. The numbering of the instance types follows the same order as in Tables 3 and 4, instance types 1 to 5 correspond to the first row, instance types 6 to 10 correspond to the second row, etc.

The results are also depicted in Figure 2. On average the algorithm reduces the objective value with 42,7%. The reduction was at least 14,5% and at most 63,9%.

To illustrate the improvement of the algorithm we depicted the solution of both methods for one of the instances. Figure 3 shows the initial solution of instance 701 whereas Figure 4 shows the solution of the algorithm. This instance has one product, three depots and sixty customers.

The initial solution was found in 0.02 seconds and has objective value 1242,72. The solution has five routes; one from depot 1 and two from the other depots. It is clearly not optimal, since routes cross themselves. Note that the upper left customer is visited twice; once by the route from depot 1 and once from a route from depot 3.

The algorithm found the solution with objective value 673,10 in 83,93 seconds. It only has one route per depot and no crossing routes at all. Optimality cannot be proven, but it cannot be declared non-optimal upon inspection.

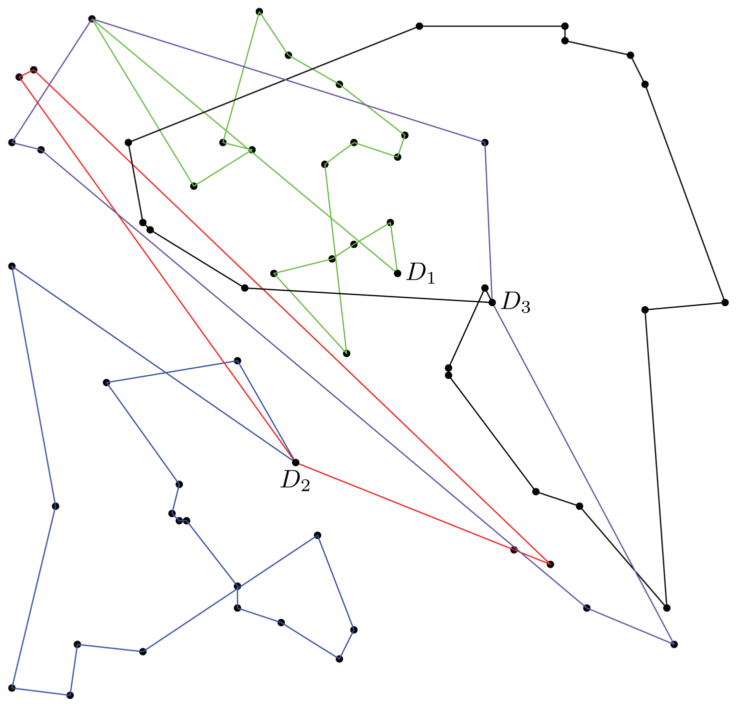


Figure 3: Initial solution of instance 701, with objective value 1242,72

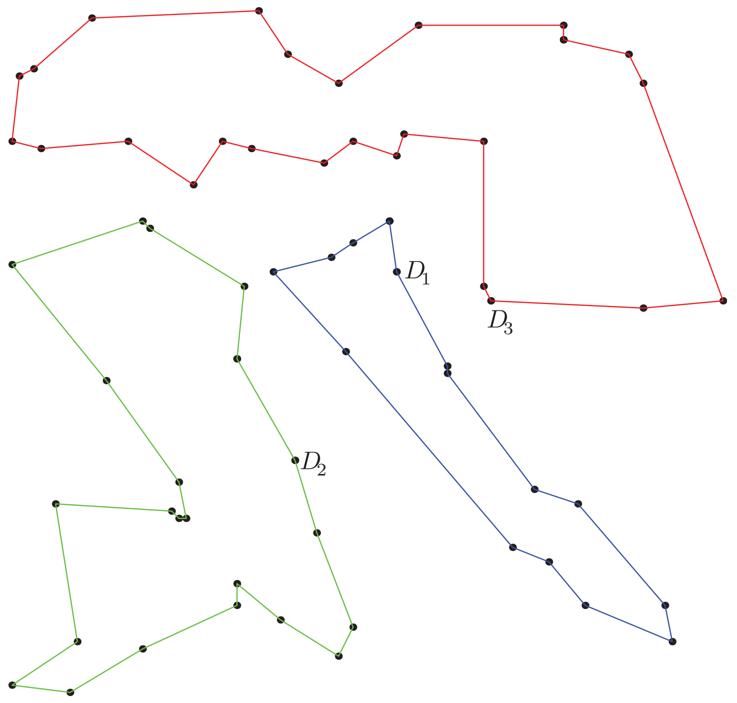


Figure 4: Algorithm solution of instance 701, with objective value 673,10

5 Conclusions

We studied the multi-depot vehicle routing problem with inventory limitations. This extension to the classical vehicle routing problem considers multiple depots, a limited inventory at each depot and a known demand of each customer. Furthermore, the problem considers multiple products and it allows for the possibility of split deliveries. The goal is to minimize the total route length, while making sure that all demand is satisfied. The problem was formulated mathematically and shown to be NP-hard. The presented algorithm makes use of variable neighborhood search with three neighborhoods and seven local search heuristics. A diversification procedure is used to search in a broad area of the feasible region.

The algorithm was applied to one thousand instances of different sizes. The average computation times range from half a minute for the smallest problems to ten and a half minutes for the largest problems, which is fast enough for most real world applications. The results are compared with the initial solution, which is in itself a simple heuristic to solving this problem. The proposed algorithm based on variable neighborhood search produces solutions with a 42,7% average decrease in route length when compared to the simple heuristic.

The quality of the solutions has not been bench-marked against other solutions, since no heuristic for this problem has been developed before. Furthermore, there is no information about the quality of our solution compared to the optimal solution. Therefore, a suggestion for further research is to determine good lower and upper bounds on the optimal solution or to completely solve the problem to optimality. This is even useful for only a part of the instances, i.e. the smaller instances, since that will still give a better understanding of the quality of the algorithm presented in this paper.

A second suggestion for further research is to adapt the algorithm such that it can also handle practical problems where our three simplifying assumptions do not hold. The adaptations could include the possibility of a limited vehicle fleet at each depot, a maximum capacity for each vehicle, or to incorporate the fixed cost per vehicle in the objective function.

A third suggestion is to consider a problem where we incorporate the inventory policy in the decision process. In the multi-depot vehicle routing problem with inventory limitations, it is often the case that not all customers can be served by their nearest depot. These customer will have to be served by a more distant depot and that will result in more costly routes. When the inventory level at each depot is higher, more customers can be served by their nearest depot. The idea is to compare the added costs when serving from a more distant depot to the added costs of having more products in stock. Taking both factors into account, an inventory policy and delivery plan could be determined such that total costs are minimized.

References

Archetti, C., Speranza, M., 2012. Vehicle routing problems with split deliveries. International transactions in operational research 19 (1-2), 3 – 22.

Croes, G. A., 1958. A method for solving traveling-salesman problems. Operations Research 6 (6), 791 – 812.

Dantzig, G. B., Ramser, J. H., 1959. The truck dispatching problem. Management Science 6 (1), 80 – 91.

Dror, M., Trudeaut, P., 1989. Savings by split delivery routing. Transportation Science 23 (2), 141 – 145.

Golden, B. L., Raghavan, S., Wasil, E. A., 2008. The Vehicle Routing Problem: Latest Advances and New Challenges: latest advances and new challenges. Vol. 43. Springer.

Lenstra, J. K., Rinnooy Kan, A., 1981. Complexity of vehicle routing and scheduling problems. Networks 11 (2), 221 – 227.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24 (11), 1097 – 1100.

Polacek, M., Hartl, R. F., Doerner, K., Reimann, M., 2004. A variable neighborhood search for the multi depot vehicle routing problem with time windows. Journal of heuristics 10 (6), 613 – 627.

Salhi, S., Imran, A., Wassan, N. A., 2013. The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. Computers & Operations Research.

Salhi, S., Rand, G. K., 1987. Improvements to vehicle routeing heuristics. The Journal of the Operational Research Society 38 (3), 293 – 295.

Salhi, S., Sari, M., 1997. A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. European Journal of Operational Research 103 (1), 95 – 112.

Toth, P., Vigo, D., 2002. An overview of vehicle routing problems. The vehicle routing problem 9, 1 – 26.

Vidal, T., Crainic, T. G., Gendreau, M., Prins, C., 2013. Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. European Journal of Operational Research 231 (1), 1 – 21.