A Metaheuristic Approach for an Extension of the Vehicle Routing Problem

Mark van der Broek Niels van der Laan

April 24, 2017

Abstract

We apply metaheuristic search techniques to an extension of the multi-depot vehicle routing problem. The goal of this research is to design an efficient solution approach that finds high-quality feasible solutions for the multi-depot split-delivery vehicle routing problem with inventory constraints and heterogeneous fleet. We use an extended large neighbour search algorithm, which combines destroy and repair operators with local search techniques. Our algorithm finds feasible solutions for a large number of instances in short time. The main contributions of this paper are twofold: we extend and apply an existing metaheuristic approach, and we consider a new extension of the vehicle routing problem.

1 Introduction

We consider the problem of dispatching and scheduling vehicles from multiple depots to serve and deliver goods to clients at different locations. One application is supplying products to supermarkets, i.e. clients, from a number of distribution centres. A client can be served by more than one vehicle, for example, if the client demands a large quantity or if the vehicle capacity is small. Serving a customer requires a given service time, which corresponds to activities such as handling and unloading goods from the vehicle. At each depot, different types of vehicles are available, which differ in speed and travel costs. As an example, think of trucks that vary in size and fuel type. The use of vehicles comes at a cost, corresponding to fuel expenditures and driver wages. The objective is to minimize total costs, while serving all customers and taking into account restrictions such as travel time regulations and inventory constraints.

The main characteristics of the problem we study are multiple depots and the option to split client deliveries. Gulczynski et al. (2011) discuss the relevance of such problems. Applications include waste collection in large cities, multi-warehouse distribution systems, and disaster relief logistics.

In this paper, we outline an efficient metaheuristic approach for the multi-depot split-delivery vehicle routing problem with inventory constraints and heterogeneous fleet (MDSDHVRP-I) that yields high-quality solutions. Solution quality is measured by travel costs and feasibility with respect to all constraints. Based on Hemmelmayr (2015), we implement a metaheuristic algorithm combining a large neighbourhood search (LNS) with

local search techniques. Solutions are sequentially destroyed and repaired to find new candidate solutions. Only promising solutions are further improved using local search operators. A simulated annealing scheme is used to accept non-improving solutions.

The MDSDHVRP-I is an extension of the classical vehicle routing problem (VRP). Caceres-Cruz et al. (2014) discuss the VRP as well as a large number of extensions, including vehicle capacity restrictions, multiple depots, split deliveries, and heterogeneous fleets. Vidal et al. (2013) present an overview of different (meta)heuristic approaches for VRP extensions. Their discussion includes LNS, tabu search, population based methods, and hybrid metaheuristics.

Hemmelmayr (2015) describes several LNS procedures and applies them to an extended VRP with location decisions and periodicity constraints. The main feature of LNS is the use of randomly selected destroy and repair operators to explore large neighbourhoods. In contrast to adaptive LNS, the operator selection probabilities are not based on past performance. Hemmelmayr (2015) demonstrates good performance of the proposed LNS procedures using computational experiments. We extend their analysis to the MDSDHVRP-I.

Split deliveries were introduced by Dror and Trudeau (1989), who showed that considerable cost savings can be obtained by allowing demand splits. More recently, Boudia et al. (2007) proposed a memetic algorithm and Silva et al. (2015) designed an iterated local search heuristic for the split delivery VRP (SDVRP). While Silva et al. (2015) allow for fleet size restrictions, neither they nor Boudia et al. (2007) consider different vehicle types, multiple depots, or route duration restrictions. Both papers propose a number of local search operators designed for split deliveries, that can add and remove splits. We adopt these operators in our approach. An alternative approach for the SDVRP is suggested by Chen et al. (2017), which involves splitting customer demands a priori and solving the resulting capacitated VRP.

Vidal et al. (2012) design a hybrid genetic algorithm for the multi-depot periodic VRP, in which customers have to be served at a given frequency. Next to the multi-depot aspect, other similarities with our setting are customer service times and route duration restrictions. However, our solution approach is different and Vidal et al. (2012) do not consider the option to split deliveries. Salhi et al. (2014) also study the multi-depot aspect. As opposed to Vidal et al. (2012), they consider a heterogeneous vehicle fleet. In their setting vehicles come at a fixed cost, while the number of vehicles of each type is unlimited. In contrast, we ignore fixed vehicle costs and impose fleet size restrictions.

Gulczynski et al. (2011) study the multi-depot split-delivery VRP (MDSDVRP). They propose an integer-programming based heuristic which they apply to instances with at most 250 customers. We consider a different solution approach and apply it to larger problems. Ray et al. (2014) design a metaheuristic approach for the MDSDVRP with the option to open depots at customer locations, which comes at a fixed cost. While they restrict the number of available vehicles, neither they nor Gulczynski et al. (2011) consider a heterogeneous fleet, nor do they consider customer service times or inventory and time travel restrictions.

To our knowledge, we are the first to extend the MDSDVRP in several ways. We include a heterogeneous fleet, customer service times, route duration restrictions, and inventory constraints.

The remainder of this paper is organized as follows. In Section 2, we provide a detailed problem description and an integer linear programming (ILP) formulation of the

MDSDHVRP-I. Section 3 outlines our metaheuristic approach in detail. In Section 4, we apply our algorithm to a number of test instances and report on the results. Experimental insights are provided in Section 5. Finally, Section 6 concludes.

2 Problem definition

The MDSDHVRP-I is described by a complete graph $G(V, E)$, where $V = V_d \cup V_c$. The sets of nodes V_d and V_c contain the m depots and n customers, respectively. There are a total of κ different vehicle types.

Depot j has inventory I_j and at most η_{jk} vehicles of type k. Customer i has positive demand q_i and service time τ_i. Vehicle type k has capacity Q_k, maximum route duration T_k, driving speed v_k and travel cost per unit of distance α_k. Route duration is defined as the sum of driving time and customer service times of customers on the route.

Each vehicle must return to its own depot and intermediate depot visits are not allowed. Vehicles cannot transfer items to other vehicles. A customer can be served by more than one vehicle, possibly originating from different depots. This is illustrated by Figures 1 and 2. If a customer is on multiple routes, their demand can be split, while service times cannot be split.

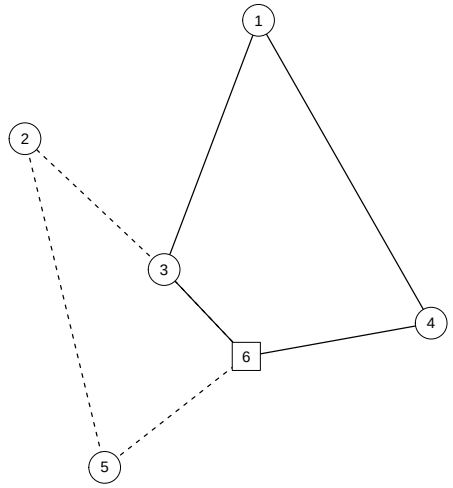


Figure 1: Single-depot split delivery (customer 3).

The objective is to minimize total travel costs while serving all customers and satisfying all restrictions. These restrictions relate to the fleet size, depot inventories, vehicle capacities, and vehicle route durations.

Below, we repeat the meaning of all symbols for convenience.

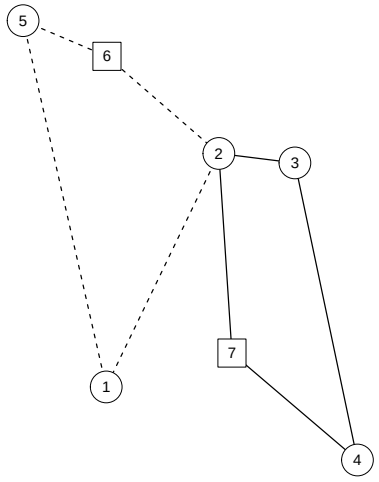


Figure 2: Multi-depot split delivery (customer 2).

m: Number of depots.

n: Number of customers.

κ: Number of vehicle types.

V_d: Set containing depot nodes.

V_c: Set containing customer nodes.

V: Set containing all nodes.

d_{ij}: Euclidean distance between nodes i and j, $\quad i, j \in V$.

I_j: Inventory of depot j, $\quad j \in V_d$.

η_{jk}: Number of vehicle of type k at depot j, $\quad j \in V_d$, $k = 1, \ldots, \kappa$.

q_i: Demand of customer i, $\quad i \in V_c$.

τ_i: Service time of customer i, $\quad i \in V_c$.

Q_k: Capacity of vehicles of type k, $\quad k = 1, \ldots, \kappa$.

T_k: Maximum route duration of vehicles of type k, $\quad k = 1, \ldots, \kappa$.

v_k: Driving speed of vehicles of type k, $\quad k = 1, \ldots, \kappa$.

α_k: Travel cost of vehicles of type k, $\quad k = 1, \ldots, \kappa$.

2.1 Integer linear programming formulation

For notational convenience, define the fleet set F as

$$F = \{(d, k, p) : d \in V_d, \quad k = 1, \ldots, \kappa, \quad p = 1, \ldots, \eta_{dk}\}.$$

Each element of F corresponds to a particular vehicle. To formulate an ILP for the MDSDHVRP-I, we introduce a five-index binary variable

$$x_{ijdkp}, \quad i, j \in V, \quad (d, k, p) \in F,$$

where $x_{ijdkp} = 1$ if and only if the p-th vehicle of type k originating from depot d travels from node i to j. This idea is based on Salhi et al. (2014), who present a flow-based formulation for a multi-depot heterogeneous fleet VRP using a four-index binary variable. The integer-valued decision variables

$$y_{idkp}, \quad i \in V_c, \quad (d, k, p) \in F$$

represent the delivery made at customer i by the p-th vehicle of type k origination from depot d. This is adapted from the two-index decision variables in the formulation for the MDSDVRP by Ray et al. (2014). This leads to a total of $((m + n)^2 + n)|F|$ decision variables, where $|F| = \sum_{j=1}^{m} \sum_{k=1}^{\kappa} \eta_{jk}$. The number of constraints is linear in $|F|$ and exponential in the number of nodes, because of the subtour elimination constraints.

Our ILP is based on the formulation by Ray et al. (2014), which we extend in several ways. First, we incorporate the heterogeneous fleet aspect. Next to that, we include route duration restrictions and inventory constraints.

$$\min C = \sum_{(d,k,p)\in F} \alpha_k \sum_{i\in V}\sum_{j\in V} d_{ij} x_{ijdkp} \tag{1}$$

Subject to:

$$\sum_{j\in V} x_{ijdkp} = \sum_{j\in V} x_{jidkp}, \quad \forall i \in V, (d,k,p) \in F \tag{2}$$

$$\sum_{i\in V_c} x_{didkp} \leq 1, \quad \forall (d,k,p) \in F \tag{3}$$

$$\sum_{i\in V} x_{id_2d_1kp} = 0, \quad \forall (d_1,k,p) \in F, d_2 \in V_d \setminus \{d_1\} \tag{4}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ijdkp} - \sum_{j\in S} x_{djdkp} \leq |S| - 1, \quad \forall S \subset V : |S| \geq 2, (d,k,p) \in F \tag{5}$$

$$\sum_{(d,k,p)\in F} y_{idkp} = q_i, \quad \forall i \in V_c \tag{6}$$

$$y_{idkp} \leq q_i \sum_{j\in V} x_{jidkp}, \quad \forall i \in V_c, (d,k,p) \in F \tag{7}$$

$$\sum_{k=1}^{\kappa}\sum_{p=1}^{\eta_{dk}}\sum_{i\in V_c} y_{idkp} \leq I_d, \quad \forall d \in V_d \tag{8}$$

$$\sum_{i\in V_c} y_{idkp} \leq Q_k, \quad \forall (d,k,p) \in F \tag{9}$$

$$\frac{1}{v_k}\sum_{i\in V}\sum_{j\in V} d_{ij} x_{ijdkp} + \sum_{i\in V_c}\sum_{j\in V} x_{ijdkp}\tau_i \leq T_k, \quad \forall (d,k,p) \in F \tag{10}$$

$$x_{ijdkp} \in \{0,1\}, \quad \forall i,j \in V, (d,k,p) \in F \tag{11}$$

$$y_{idkp} \in \mathbb{N} \cup \{0\}, \quad \forall i \in V, (d,k,p) \in F \tag{12}$$

Here, equations (2), (5)-(7), and (9) are straightforward adaptations from Ray et al. (2014).

The objective (1) is to minimize the sum of travel costs for each vehicle. Flow conservation is ensured by constraints (2), which state that if a node is entered by a vehicle, it is left by the same vehicle. Constraints (3) and (4) (based on Salhi et al., 2014) impose the restriction that a vehicle cannot make intermediate depot visits. Constraints (5) are the sub-tour elimination constraints. We ensure that all demand is satisfied by (6). Constraints (7) impose the logical restriction that a vehicle can make a delivery to a customer only if the customer is visited by that vehicle. By the assumption that all customers have positive demand, constraints (6) and (7) imply that each customer is visited at least once. The inventory and capacity constraints are enforced by (8) and (9), respectively. Route duration restrictions are represented by (10). The first and second term in (10)

correspond to driving time and service time, respectively. Finally, (11) and (12) restrict the decision variables to binary and integer values.

3 Metaheuristic approach

The MDSDHVRP-I is a generalization of the SDVRP, which is NP-hard (Dror and Trudeau, 1990). We therefore present a metaheuristic approach which combines LNS and local search techniques. The outline of our approach is shown in Algorithm 1, which is adapted from the sequential LNS (LNS-S) algorithm by Hemmelmayr (2015).

The LNS-S algorithm uses destroy and repair operators to explore large neighbourhoods. Only promising solutions are further explored by means of local search routines. A solution is considered promising if the ratio between its objective value and the best found objective value is below $1 + \theta$. Furthermore, Hemmelmayr (2015) uses simulated annealing (SA) to accept non-improving solutions.

Algorithm 1

1: $s, s^* \leftarrow InitialSolution$
2: **repeat**
3: $\quad s' \leftarrow DestroyAndRepair(s, D, R)$
4: $\quad$ **if** $SimulatedAnnealing(s, s')$ **then**
5: $\quad\quad s \leftarrow s'$
6: $\quad\quad s^* \leftarrow min(s^*, s)$
7: $\quad$ **end if**
8: $\quad$ **if** $f(s) > (1 + \gamma)f(s^*)$ **then**
9: $\quad\quad s \leftarrow s^*$
10: $\quad$ **end if**
11: **until** First stopping criterion is met
12: **repeat**
13: $\quad s' \leftarrow DestroyAndRepair(s, D, R)$
14: $\quad$ **if** $f(s') < (1 + \theta)f(s^*)$ **then**
15: $\quad\quad s' \leftarrow LocalSearch(s')$
16: $\quad$ **end if**
17: $\quad$ **if** $SimulatedAnnealing(s, s')$ **then**
18: $\quad\quad s \leftarrow s'$
19: $\quad\quad s^* \leftarrow min(s^*, s)$
20: $\quad$ **end if**
21: $\quad$ **if** $f(s) > (1 + \gamma)f(s^*)$ **then**
22: $\quad\quad s \leftarrow s^*$
23: $\quad$ **end if**
24: **until** Second stopping criterion is met
25: Return s^*

We extend the LNS-S algorithm in two ways. First, we implement an additional search loop (Algorithm 1, lines 2-11), which does not involve local search techniques. During this loop, solutions are subsequently destroyed and repaired. This is to exploit the stand-alone effectiveness of LNS without spending computation time on local search routines.

Second, to control the search, solution quality is compared with the best found solution at each iteration in both loops. If the relative gap is larger than a threshold γ, the search returns to the best found solution.

During the search, we temporarily accept infeasibilities with respect to the route duration restrictions. We adopt the penalization scheme used by Hemmelmayr (2015). The penalty is proportional by a factor β to the total excess travel time. At each iteration, feasibility of s is checked. If s is infeasible, then β is multiplied by λ, else β is divided by λ. The penalty parameter β must always lie in the interval $[\iota, \nu]$.

In the remainder of this section, we describe the construction of an initial solution, the destroy and repair operators, and the local search procedure.

3.1 Initial solution

An initial solution is constructed in three steps. First, customers are assigned to depots. Since demand splits are allowed, customers can be assigned to multiple depots. Second, customers are assigned to vehicles, again allowing for demand splits. Finally, we face a travelling salesman problem for each vehicle. We then use the cheapest insertion heuristic to arrive at an initial solution.

For the first step, customers are assigned in ascending demand size to the nearest available depot. A depot is available if it has positive remaining inventory. If necessary, demand is split and remaining demand is assigned to the now nearest available depot.

For each depot, a customer-vehicle assignment is constructed by selecting a random seed customer from the assigned customers, which is then assigned to the cheapest available vehicle (with the lowest travel cost α_k). If the vehicle has positive remaining capacity, the customer closest to the seed customer is assigned to the vehicle. Demand is split if necessary to satisfy the capacity constraints. This process of assigning customers to the selected vehicle continues until the vehicle is no longer available due to its capacity restriction. If a vehicle becomes unavailable, we repeat the process for a newly selected seed customer until all customers are assigned to vehicles.

3.2 Destroy and repair operators

Following Hemmelmayr (2015), we use a set of destroy operators. At each iteration, one of these destroy operators is selected at random, according to predefined probabilities. We adopt three destroy operators from Hemmelmayr (2015), namely *related-removal*, *cost-removal*, and *route-removal*. These operators act by removing at most q randomly selected customers from the solution entirely. Customers that are on multiple routes are removed from all their routes.

Related-removal selects a random seed customer and removes the $q - 1$ nearest customers. *Worst-removal* removes the q worst customers from the solution. For each customer, the gain in costs resulting from removing them from the solution is evaluated. To introduce randomness in the customer selection, this cost gain is perturbed by multiplying with a random variate $d \sim U(0.8, 1.2)$. The q customers with the largest perturbed cost gain are removed from the solution. Finally, *route-removal* selects a random non-empty route and removes at most q customers from this route.

Our repair operator is also based on Hemmelmayr (2015), who introduces the *greedy-insertion-perturbation* operator. Customers are reinserted in the solution in a random

order. For a given customer all possible insertion options are evaluated. The insertion costs are perturbed by a random variate $d \sim U(0.8, 1.2)$. Based on these perturbed insertion costs, the customer is inserted using the cheapest option. We adapt this operator to the split delivery situation by splitting demand if necessary to maintain feasibility with respect to the inventory and capacity restrictions. If demand is split, the remaining customer demand is updated and a new random customer is selected.

3.3 Local search procedure

Our local search procedure makes use of seven neighbourhood search operators, labelled $R_1, \ldots, R_7$. These operators are applied sequentially, starting with R_1, based on a best-improvement scheme. If an improvement is found, the search returns to R_1. If R_7 does not find an improvement, *LocalSearch* terminates. This idea is based on Salhi et al. (2014), who use six operators. The local search operators we use are

R_1: *Swap(1,1)**,

R_2: *SplitRemoval*,

R_3: *2-opt intra-route*,

R_4: *1-insertion intra-route*,

R_5: *Swap(1,1) intra-route*,

R_6: *1-insertion inter-route*, and

R_7: *2-insertion intra-route*.

Following Salhi et al. (2014), the ordering of these operators is based on their complexity and experimental results.

The *Swap(1,1)** operator (Boudia et al., 2007) considers two customers i and k in different routes r_1 and r_2 with unequal delivery quantities y_i and y_k. If $y_i > y_k$, then customer k in route r_2 is replaced by customer i with delivery quantity y_k. The delivery quantity of customer i in route r_1 changes to $y_i - y_k$ and customer k is inserted in route r_1 before or after customer i. For illustration, see Figure 3.

SplitRemoval is inspired by the *RouteAddition* operator proposed by Silva et al. (2015), which tries to reduce customer demand splits. *RouteAddition* removes customers that are on multiple routes from two routes and creates three new routes. Instead, *SplitRemoval* removes a customer from all their routes and considers all reinsertion options that do not involve demand splits.

The operators R_3, R_4, R_5, and R_7 act on a single route. When applied to a solution, the best improvement is found for each route. R_3 is the well-known *2-opt* operator proposed by Croes (1958), which removes and replaces two non-consecutive arcs from the solution. *1-* and *2-insertion intra-route* relocate a customer or pair of consecutive customers within their route. The fifth operator, *Swap(1,1)*, tries to improve the solution by swapping two customers.

Finally, *1-insertion inter-route* attempts to relocate a customer from one route to another, while maintaining feasibility with respect to inventory and capacity restrictions. Note that we consider both within- and inter-depot relocations. We remark that this operator may remove customer demand splits.

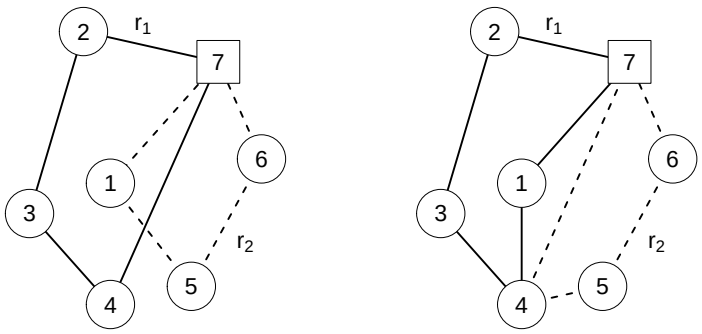


Figure 3: *Swap*(1,1)*-move between customers 1 and 4 ($y_4 > y_1$).

4 Computational results

We used a total of 513 instances in a benchmark set to find good parameter values and analyse the performance of our algorithm. These instances vary in size, with the number of depots ranging from 2 to 50, in between 7 and 1839 customers and at most 3 vehicle types. Our method was implemented using C++ and executed on an Intel Core i7 6700HQ at 2.4GHz. The resulting solutions for the benchmark instances can be found in a separate file.

4.1 Parameter settings

We evaluated the performance of our method for various parameter settings using a subset of the available instances. For these experiments, we set the time limit to 600 seconds for both loops. We use an exponential cooling scheme for the SA-based acceptance decision, that is, at each iteration the temperature is multiplied by the cool-down rate. If the solution has improved the temperature is reinitialized to a percentage of the objective value. Table 1 shows average costs for a range of parameter values. In the final configuration, we set $\theta = 0.03$ and $\gamma = 0.05$. At each improvement, we reinitialize the temperature to 0.01% of the objective value, and we use a cool-down rate of 0.96.

Table 2 shows the objectives that result from changing q, the number of customers to be removed by the destroy operators, in both loops. Based on these results, we set q to at most 5% of the customers in the first loop and 10% in the second loop. For small instances, these numbers are inflated slightly. The destroy operators *related-removal*, *cost-removal*, and *route-removal* are selected randomly with equal probabilities in the first loop. In the second loop, we use probabilities $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{6}$, respectively.

For the penalty structure, we set $\beta = 100$, $\lambda = 1.1$, and $[\iota, \nu] = [100, 10^5]$. This choice of parameters ensures that the search frequently returns to feasible solutions, while temporary infeasibilities are allowed.

The first search loop is exited after 2×10^6 iterations. The second loop is computationally more expensive and is exited after 5×10^5 iterations. For the final run, we imposed a time limit of 1800 seconds for both loops, limiting total computation time to at most one hour. For 60 large instances, we increased the time limit to six hours.

Table 1: Experiments: general parameters.

Parameter	Value	Cost (average)
θ	0.00	47645
	0.01	47541
	0.03	47210
	0.05	47389
γ	0.01	47331
	0.03	47293
	0.05	47210
	0.07	47479
Initial temp.	0.005%	48778
(perc. of	0.01%	47210
solution cost)	0.05%	47329
Cool-down	0.90	48247
rate	0.93	48094
	0.96	47210
	0.99	48113

Average costs over ten instances.
θ is the threshold parameter for the *LocalSearch* procedure.
The search returns to the best found solution if the relative gap exceeds γ.

4.2 Performance analysis

Our algorithm was able to find feasible solutions for all available instances in short time, that is, well under 10 seconds. For most instances, solutions are feasible within 50 iterations. The solution quality cannot be benchmarked against the current literature. Nevertheless, we can provide insights into the important elements of our algorithm using computational experiments.

Figure 4 shows the objective value of the initial feasible solution, the solution after the first search loop, and the final solution for ten different instances. This figure demonstrates the stand-alone effectiveness of LNS; a large improvement over the initial solution is obtained using only destroy and repair techniques. We remark that the second search loop is most effective for large instances and long runs.

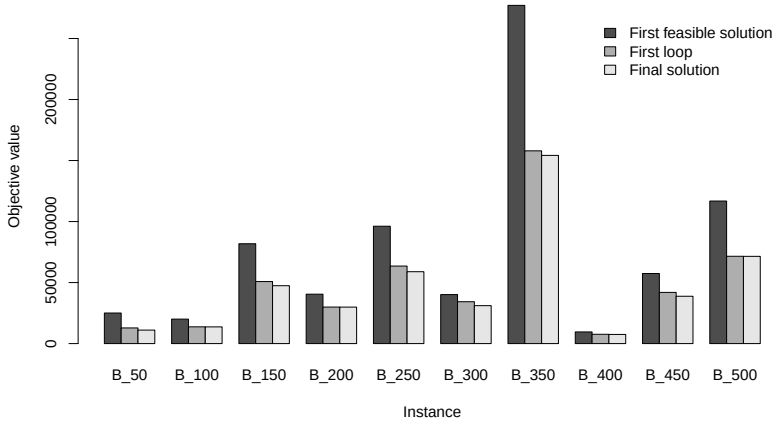
Table 3 shows the effectiveness of all operators we used. We applied our method to ten instances, with and without a given operator. We impose a time limit to ensure a fair comparison. Table 3 reports the average change in the costs. A positive number indicates that the operator is effective. The destroy operator *related-removal* is especially effective,

Table 2: Experiments: destroy operators parameters.

	q (second loop)			
q (first loop)	5%	10%	15%	20%
3%	47888	47786	47910	47858
5%	47446	47210	47386	47305
10%	48840	49919	49604	49498
15%	50225	50682	51193	50376

Average costs over ten instances.
q is the number of customers removed by a destroy operator (expressed as a percentage of the number of customers).

Figure 4: Intermediate solution quality.



as opposed to *cost-removal*. For the local search operators, only *2-opt intra-route* and *1-insertion inter-route* are effective.

The reason that most local search operators are not effective at improving solution quality is that they are time-consuming relative to the destroy operators. Table 3 also shows the computation times and the average improvement in objective value for each local search operator. The *SplitRemoval* operator yields the biggest improvement, followed by *2-insertion intra-route* and *swap(1,1)**. These operators may become more effective as the number of iterations increases and the stand-alone effectiveness of LNS diminishes.

We also evaluated the effectiveness of other elements of our method. Table 4 reports the change in objective for several changes to our method. First, basing the acceptance decision on whether the solution has improved instead of SA leads to an increase in objective value, demonstrating the effectiveness of SA. Second, placing very high penalties on constraint violations deteriorates solution quality. This shows that temporarily allowing infeasible solutions during the search leads to an improvement in the final objective

Table 3: Effectiveness operators.

Operator	Change in objective (%)	Computation time (μs)	Average gain
Destroy operators			
Related-removal	4.52	949	
Cost-removal	-0.64	1427	
Route-removal	0.73	598	
Local search operators			
*Swap(1,1)**	-0.24	88868	133.1
SplitRemoval	-0.50	2475	176.0
2-opt intra-route	1.48	365	68.6
1-insertion intra-route	-0.15	1456	43.1
Swap(1,1) intra-route	-0.19	474	55.1
1-insertion inter-route	0.01	11572	76.0
2-insertion intra-route	-0.02	1376	151.7

Average change in objective value over ten instances if operator is not used. Time limit: 600 seconds for both loops. Computation times and gains averaged over instances.

Table 4: Effectiveness general elements.

Change	Change in objective (%)
Improvement-based acceptance decision	0.95
Severe penalty structure	1.48
No destroy and repair loop	3.84

Average change in objective value over ten instances.

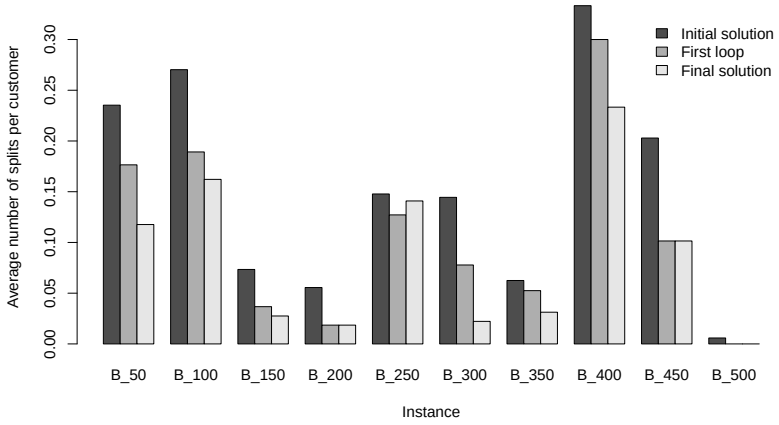
value. Finally, if we only use the second search loop, as in the original LNS-S algorithm by Hemmelmayr (2015), solution costs increase considerably. This is a result of the standalone effectiveness of LNS and the relatively high computation times of the local search operators.

Figure 5 shows how the average number of demand splits per customer changes as the algorithm progresses. Note that the initial solution has many demand splits, which is likely a result of the greedy nature of the construction heuristic. The number of splits is reduced by the destroy and repair operators and further reduced during the second loop.

5 Experimental insights

In this section, we try to identify the factors underlying demand splits. We do so by analysing the solutions generated by our method for the benchmark instances. Intuitively, splits are necessitated by large customer demands and tight constraints pertaining to delivery quantities. The results in this section confirm this intuition.

Figure 5: Average number of splits.



Of the 513 benchmark instances, 245 are tight in the inventory or vehicle capacity constraints. We compute the ratio of the sum of all customer demand quantities to the sum of all inventory sizes as a measure of tightness. We use the adapted inventory level, which is defined as

$$I'_j = \min \left\{ I_j, \sum_{k=1}^{\kappa} \eta_{jk} Q_k \right\}$$

for a depot j. This definition also incorporates vehicle capacity constraints, which are relevant only if the corresponding inventory constraint is redundant.

Figure 6 shows for each benchmark instance the average number of splits per customer and the tightness of the inventory constraints as defined above. The figure shows a non-linear positive relationship. Demand splits appear to become more important when the demand to inventory is larger than 0.90.

Customer demand size is also an important factor driving demand splits. Figure 7 shows a clear positive relationship between mean demand size and the average number of splits per customer.

6 Conclusion

We have studied for the multi-depot split-delivery vehicle routing problem with heterogeneous fleet and inventory restrictions. Furthermore, we included customer service times and route duration restrictions. We have extended a metaheuristic approach proposed by Hemmelmayr (2015) and applied it to a set of benchmark instances.

The metaheuristic approach we used combines large neighbourhood search techniques with local search operators to find good solutions. We repeatedly apply destroy and repair operators to improve the solution quality. In the final phase of our method, promising solutions are further explored by means of a local search routine. The acceptance decision

Figure 6: Scatterplot demand splits vs. demand to inventory ratio.

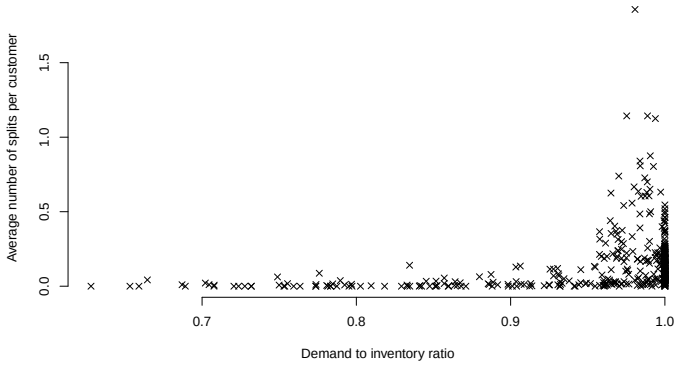
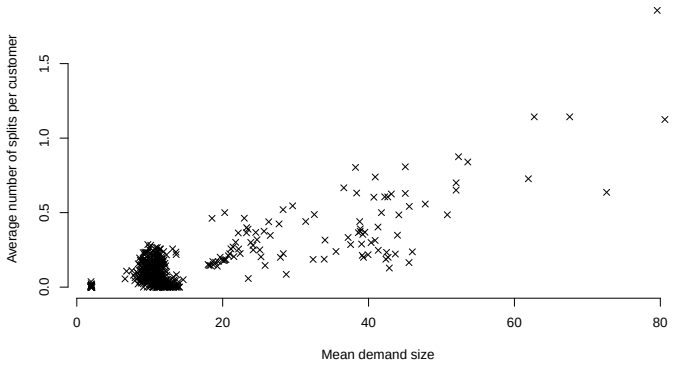


Figure 7: Scatterplot demand splits vs. mean demand size.



is based on simulated annealing, which we extend with a control mechanism to control the search.

We found feasible solutions for all benchmark instances in short time and we were able to book considerable improvements over the initial feasible solutions. Our results demonstrate the stand-alone effectiveness of large neighbourhood search techniques. Other important elements of our approach are the simulated-annealing acceptance decision and the increasing penalty structure.

Based on our final solutions for the benchmark instances, we confirm the intuition that demand splits are especially relevant if capacity and/or inventory restrictions are tight. Furthermore, mean demand size is an important driver for the number of demand splits.

There are many possible extensions of our work. These include incorporating time window restrictions and/or multiple commodities to enhance the practical value of our

results. Another extension is to include the option to open depots at given locations for a fixed cost. With regards to the metaheuristic approach we used, one may consider including adaptive elements in the large neighbourhood search to obtain better results.

References

Boudia, M., C. Prins, and M. Reghioui (2007). An effective memetic algorithm with population management for the split delivery vehicle routing problem. In T. Bartz-Beielstein, M. J. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels (Eds.), *Hybrid Metaheuristics. HM 2007. Lecture Notes in Computer Science, Vol. 4771*, pp. 16–30. Springer Berlin Heidelberg.

Caceres-Cruz, J., P. Arias, D. Guimarans, D. Riera, and A. A. Juan (2014). Rich vehicle routing problem: Survey. *ACM Computing Surveys 47*(2), 1–28.

Chen, P., B. Golden, X. Wang, and E. Wasil (2017). A novel approach to solve the split delivery vehicle routing problem. *International Transactions in Operational Research 24*, 27–41.

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

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

Dror, M. and P. Trudeau (1990). Split delivery routing. *Naval Research Logistics 37*(3), 383–402.

Gulczynski, D., B. Golden, and E. Wasil (2011). The multi-depot split delivery vehicle routing problem: An integer programming-based heuristic, new test problems, and computational results. *Computers and Industrial Engineering 61*(3), 794–804.

Hemmelmayr, V. C. (2015). Sequential and parallel large neighborhood search algorithms for the periodic location routing problem. *European Journal of Operational Research 243*, 52–60.

Ray, S., A. Soeanu, J. Berger, and M. Debbabi (2014). The multi-depot split-delivery vehicle routing problem: Model and solution algorithm. *Knowledge-Based Systems 71*, 238–265.

Salhi, S., A. Imran, and N. A. Wassan (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers and Operations Research 52*, 315–325.

Silva, M. M., A. Subramanian, and L. S. Ochi (2015). An iterated local search heuristic for the split delivery vehicle routing problem. *Computer Operations Research 53*, 234–249.

Vidal, T., T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research 60*(3), 611–624.

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