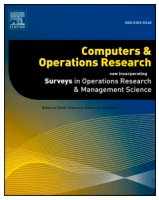
Computers & Operations Research 131 (2021) 105262

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



An iterative matheuristic for the inventory routing problem

Simen T. Vadseth *, Henrik Andersson, Magnus Stålhane

Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Alfred Getz veg 3, 7491 Trondheim, Norway

ARTICLE INFO

Keywords:
Transportation
Inventory routing
Matheuristic

ABSTRACT

The paper considers the inventory routing problem with the Maximum Level replenishment policy. Here, the supplier is in charge of replenishing goods to a number of customers and can decide when, and in what order, these customers should be visited over a defined time period. The goal is to minimize transportation costs and inventory holding costs at both the supplier and the customers. We present a matheuristic that uses a giant tour and simple operators to heuristically create routes that are used in a path-flow formulation. The proposed method iterates between solving a path-flow model with a small set of routes and updating the route set based on the optimal solution from the previous iteration. Computational results on known benchmark instances show that it outperforms state-of-the-art exact methods and heuristics on larger and more difficult instances. It finds the best-known solution on 179 out of 240 larger multi-vehicle benchmark instances, where 178 of them are strictly improving upon the previously best-known solution, and does so in considerably shorter time compared with other methods. In addition, when tested on another set of benchmark instances consisting of 798 smaller instances, the matheuristic finds the optimal solution in 44.7% of the 642 instances with known optima and has an average gap of 1.75% on the others. It also improves the best-known solution of 14 out of 156 open instances.

1. Introduction

With an increasing amount of online retailing and goods delivery, research on efficient routing and inventory control is highly relevant and applicable. Better goods transportation and inventory control can lead to significant savings for companies and potentially lower greenhouse gas emissions. The focus of this paper is on the standard inventory routing problem (IRP) with the Maximum Level replenishment policy, a well-studied problem described in great detail by several authors including Archetti et al. (2017), Coelho et al. (2014), Adulyasak et al. (2014), Desaulniers et al. (2016) and Archetti et al. (2017). The IRP is a part of a business practice called vendor-managed inventory (VMI) where the supplier decides how much quantity of goods it should deliver to each of its customers and when to do so. The entire supply chain may benefit as a result of this comprehensive planning as it can lead to better routing and inventory control, while ensuring that the customers' storage limits are respected. However, the IRP has proven to be a very challenging and computationally hard problem to solve.

Several papers have been written on the standard IRP, and there exist two relevant surveys from the last decade. The first one by Andersson et al. (2010) focused on different applications of the IRP, while the second one by Coelho et al. (2014) studied the methodological aspects. Integrating inventory management and vehicle routing in the scientific literature started with the paper of Bell et al. (1983), and the first exact method on the standard IRP itself was proposed by Archetti et al. (2007). The authors used a branch-and-cut algorithm to solve the single-vehicle IRP and solved instances up to 50 customers with three time periods and up to 30 customers with six time periods to optimality. They also introduced one of the two sets of benchmark instances that most researchers have worked on since. This set of instances consists of 798 small instances for up to five vehicles. The multi-vehicle version of the standard IRP was solved exactly by Coelho and Laporte (2013) and Adulyasak et al. (2014) with branch-and-cut algorithms, while Desaulniers et al. (2016) used a branch-cut-and-price algorithm. Avella et al. (2018) defined a new generic family of valid inequalities for the IRP and solved the problem with a branch-and-cut algorithm using two specific subclasses of the proposed valid inequalities. This method tightened the duality gap of several of the small benchmark instances.

Two additional branch-and-cut algorithms were developed by Guimarães et al. (2020) and Manousakis et al. (2020). The former employed two techniques to find improved primal solutions during the branch-and-cut search. The authors were able to close the duality gap for several instances and found new best-known solutions for 129 instances in the set of small benchmark instances. The latter proposed a two-

* Corresponding author.
E-mail address: simen.t.vadseth@ntnu.no (S.T. Vadseth).

https://doi.org/10.1016/j.cor.2021.105262
Received 20 October 2020; Received in revised form 3 February 2021; Accepted 19 February 2021
Available online 8 March 2021
0305-0548/© 2021 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

commodity flow formulation for the problem and was with the help of a good starting heuristic able to improve the best-known solution of 139 instances in a set of 300 large benchmark instances introduced by Archetti et al. (2012).

None of the exact methods described above have been able to solve larger multi-vehicle instances to optimality. Further, the duality gap can become very large when the size of the instances increases, and often no feasible solution is found within a reasonable time frame when an exact method is used. There is, consequently, a need for good heuristics for the IRP, and several have been developed over the years. Coelho et al. (2012b) developed an adaptive large neighborhood search (ALNS) heuristic for the IRP with transshipment, which the authors also tested on the single-vehicle IRP. An ALNS for the multi-vehicle version has been designed by Adulyasak et al. (2014), who solved instances with two and three vehicles.

The complexity of the IRP has led most researchers to integrate mathematical programming techniques into their heuristics. These types of heuristics, regardless of the type of problems they are applied to, have come to be known as *matheuristics*. A definition of what a matheuristic is was proposed by Boschetti et al. (2009) as: "Matheuristics are heuristic algorithms made by the interoperation of metaheuristics and mathematical programming techniques". This is also the definition used by Archetti and Speranza (2014) in their survey on matheuristics for routing problems. The authors classified matheuristics in three categories: Decomposition approaches, improvement heuristics and branch-and-price/column generation-based approaches. They also show that matheuristics are used to a large extent on problems similar to the IRP, e.g. the vehicle routing problem (VRP), the production routing problem and the location routing problem.

To the authors' knowledge, the first matheuristic used to solve the standard IRP was developed by Archetti et al. (2012). It is a hybrid heuristic that combines tabu search with the solution of mixed integer programs (MIP) and can be classified as an improvement heuristic. The authors studied the single-vehicle case and, as previously mentioned, released the second set of benchmark instances that most researchers use today. The same authors extended the method for the multi-vehicle version in Archetti et al. (2017). It is both a decomposition and an improvement search, combining a tabu search heuristic with solving MIPs. The solutions of 92% of the larger multi-vehicles instances were improved. However, many of these solutions were further improved by Chitsaz et al. (2019) with their three-phase decomposition matheuristic which relies on the iterative solution of different subproblems. Although designed for the assembly routing problem, the algorithm was able to find new best-known solutions for 194 out of 300 large instances for the IRP. Another matheuristic was proposed by Alvarez et al. (2020). They developed a hybrid heuristic, combining an iterative local search metaheuristic and two mathematical programming components, to solve the IRP with perishable products. The authors also tested the algorithm on the standard IRP and improved the best-known solution for a few smaller instances. An additional matheuristic for the IRP was presented by Diniz et al. (2020). Here, the authors combined an iterative local search with a randomized variable neighborhood descent, and were able to find and improve the best-known solutions of several small benchmark instances.

As seen in the paragraphs above, there are several exact methods and heuristics designed for, and applied to, the IRP. Smaller instances of the IRP can be solved to optimality by exact methods, while heuristics outperform exact methods on larger instances. Even though there exists heuristics for the IRP, there is still room for improvement, especially when it comes to larger instances. Many of the proposed solution methods cannot be applied to the largest instances due to the computational complexity, and those that can, suffer from long computing times.

The purpose of this paper is to present a new matheuristic to solve large instances of the IRP in shorter computing times. The matheuristic iteratively solves an exact mathematical model with a limited number of routes. The set of routes used in the first iteration is generated from a giant tour, and is modified by different operators between each iteration. It is tested on known benchmark instances for the IRP and finds new best solutions on 178 out of 240 instances with multiple vehicles, and a further seven new best-known solutions for the single-vehicle case. These improved solutions have also been found in only a small fraction of the time spent by other heuristics in the literature.

The remainder of the paper is organized as follows. In Section 2, the standard IRP is defined and presented mathematically, while our matheuristic is presented in detail in Section 3. Our computational results are reported in Section 4 and concluding remarks are presented in Section 5.

2. Problem definition and formulation

The inventory routing problem concerns the repeated distribution of goods from a supplier to a set of customers over a given planning horizon. We formulate this problem on a graph $G(\mathcal{N}, \mathcal{A})$ where $\mathcal{N}$ is a set of nodes $\mathcal{N} = \{0, 1, \ldots, N\}$ consisting of N customers and a supplier denoted 0. We also introduce $\mathcal{N}' = \{1, \ldots, N\}$ as the set of customers. The set of arcs $\mathcal{A}$ defines movements between each pair of nodes. The problem is defined over a time horizon $\mathcal{T} = \{0, 1, \ldots, T\}$ and we also introduce the set of planning time periods $\mathcal{T}' = \{1, \ldots, T\}$.

In each time period, V vehicles with capacity Q can be used to deliver the goods. There is a driving cost C_{ij}, associated with each arc $(i.j) \in \mathcal{A}$. Customer i has a known demand for the commodity, R_{it}, in each time period t and a maximum, U_i, and minimum, L_i, inventory capacity. The supplier produces R_{0t} units of the commodity at the beginning of each time period t. Both the supplier and the customers have an inventory holding cost C_i^H per unit of commodity at the end of each time period. Each customer can only be visited once per time period. The problem consists of minimizing the transportation and inventory costs of the entire supply chain while making sure that no stock-outs occur.

2.1. A path-flow formulation

The proposed path-flow formulation requires some additional notation. The set $\mathcal{R}$ contains all routes. A route is a Hamiltonian cycle through a subset of the nodes including the supplier. Introducing A_{ijr} as 1 if route r traverses arc $(i.j)$, and 0 otherwise, the cost of route r can be defined as $C_r^T = \sum_{(i,j)\in\mathcal{A}} C_{ij} A_{ijr}$. The variable λ_{rt} is 1 if route r is used by a vehicle in time period t, and 0 otherwise. The amount of commodity delivered at node i in time period t is denoted q_{it} and the inventory level at node i at the end of time period t is denoted s_{it}. The inventory at node i at the beginning of the planning horizon is represented by I_{i0}. Finally, let l_{ijt} be the flow of commodity on arc (i,j) in time period t. With this notation, the model can be formulated as follows:

$$\min \sum_{i\in\mathcal{N}} \sum_{t\in\mathcal{T}'} C_i^H s_{it} + \sum_{r\in\mathcal{R}} \sum_{t\in\mathcal{T}'} C_r^T \lambda_{rt} \tag{1}$$

$$s_{i0} = I_{i0} \quad i \in \mathcal{N} \tag{2}$$

$$s_{0t} - s_{0(t-1)} - R_{0t} + \sum_{i\in\mathcal{N}'} q_{it} = 0 \quad t \in \mathcal{T}' \tag{3}$$

$$s_{it} - s_{i(t-1)} + R_{it} - q_{it} = 0, \quad i \in \mathcal{N}', t \in \mathcal{T}' \tag{4}$$

$$L_i \leqslant s_{it} \leqslant U_i \quad i \in \mathcal{N}, t \in \mathcal{T} \tag{5}$$

$$s_{i(t-1)} + q_{it} \leqslant U_i \quad i \in \mathcal{N}', t \in \mathcal{T}' \tag{6}$$

$$\sum_{j\in\mathcal{N}} l_{jit} - q_{it} - \sum_{j\in\mathcal{N}} l_{ijt} = 0 \quad i \in \mathcal{N}', t \in \mathcal{T}' \tag{7}$$

$$l_{ijt} - Q\sum_{r\in\mathscr{R}} A_{ijr}\lambda_{rt} \leqslant 0 \quad (i,j) \in \mathscr{A}, t \in \mathscr{T}' \tag{8}$$

$$\sum_{r\in\mathscr{R}}\sum_{j\in\mathscr{N}} A_{ijr}\lambda_{rt} \leqslant 1 \quad i \in \mathscr{N}', t \in \mathscr{T}' \tag{9}$$

$$\sum_{r\in\mathscr{R}} \lambda_{rt} \leqslant V \quad t \in \mathscr{T}' \tag{10}$$

$$\lambda_{rt} \in \{0,1\} \quad r \in \mathscr{R}, t \in \mathscr{T}' \tag{11}$$

$$q_{it} \geqslant 0 \quad i \in \mathscr{N}', t \in \mathscr{T}' \tag{12}$$

$$l_{ijt} \geqslant 0 \quad (i,j) \in \mathscr{A}, t \in \mathscr{T}' \tag{13}$$

The objective function (1) minimizes the transportation and inventory holding costs over the entire planning horizon, while constraints (2) set the starting inventory level at each node. The inventory balance at the supplier and the customers are taken care of by constraints (3) and (4). Moreover, the upper and lower limits on the inventory level at each node are handled by constraints (5) and (6). Constraints (7) ensure that the flow of goods out of a node is equal to what comes in except for the amount that is delivered. Constraints (8) make sure that the flow on an arc does not exceed the vehicle capacity, while constraints (9) state that two routes that visit the same node are not used in the same time period. With constraints (10) we make sure that the maximum number of available vehicles is not exceeded. Moreover, constraints (11) state that a route is either used or not while constraints (12) and (13) impose non-negativity for the other variables.

2.2. Valid inequalities

The following valid inequalities, first introduced by Coelho and Laporte (2014) for the IRP, have been adapted for the path-flow formulation and added to the model.

$$\sum_{r\in\mathscr{R}}\sum_{t'=t_1}^{t_2} A_{ir}\lambda_{rt'} \geqslant \left\lceil \frac{\sum_{t'=t_1}^{t_2} R_{it'} - U_i}{min\{Q, U_i\}} \right\rceil \quad i \in \mathscr{N}', t_1, t_2 \in \mathscr{T}', \ t_2 \geqslant t_1 \tag{14}$$

$$\sum_{r\in\mathscr{R}}\sum_{t'=t_1}^{t_2} A_{ir}\lambda_{rt'} \geqslant \frac{\sum_{t'=t_1}^{t_2} R_{it'} - s_{i(t_1-1)}}{min\{Q, U_i, \sum_{t'=t_1}^{t_2} R_{it'}\}} \quad i \in \mathscr{N}', t_1, t_2 \in \mathscr{T}', \ t_2 \geqslant t_1 \tag{15}$$

Constraints (14) state that if the sum of the demands in a node over time periods t_1 to t_2 is greater than the inventory limit, then there must be at least one visit to this node in the interval. Constraints (15) state the same, but here the actual inventory at the node at the start of time period t_1 is taken into account instead. Rounding this up is not possible since the expression then becomes non-linear. The inequalities are also strengthened by adding the total demand for node i to the denominator.

3. Matheuristic

The formulation given in Section 2.1 is a valid and complete formulation for the IRP given that the set of routes, $\mathscr{R}$, includes every feasible route in the graph G. However, the number of feasible routes

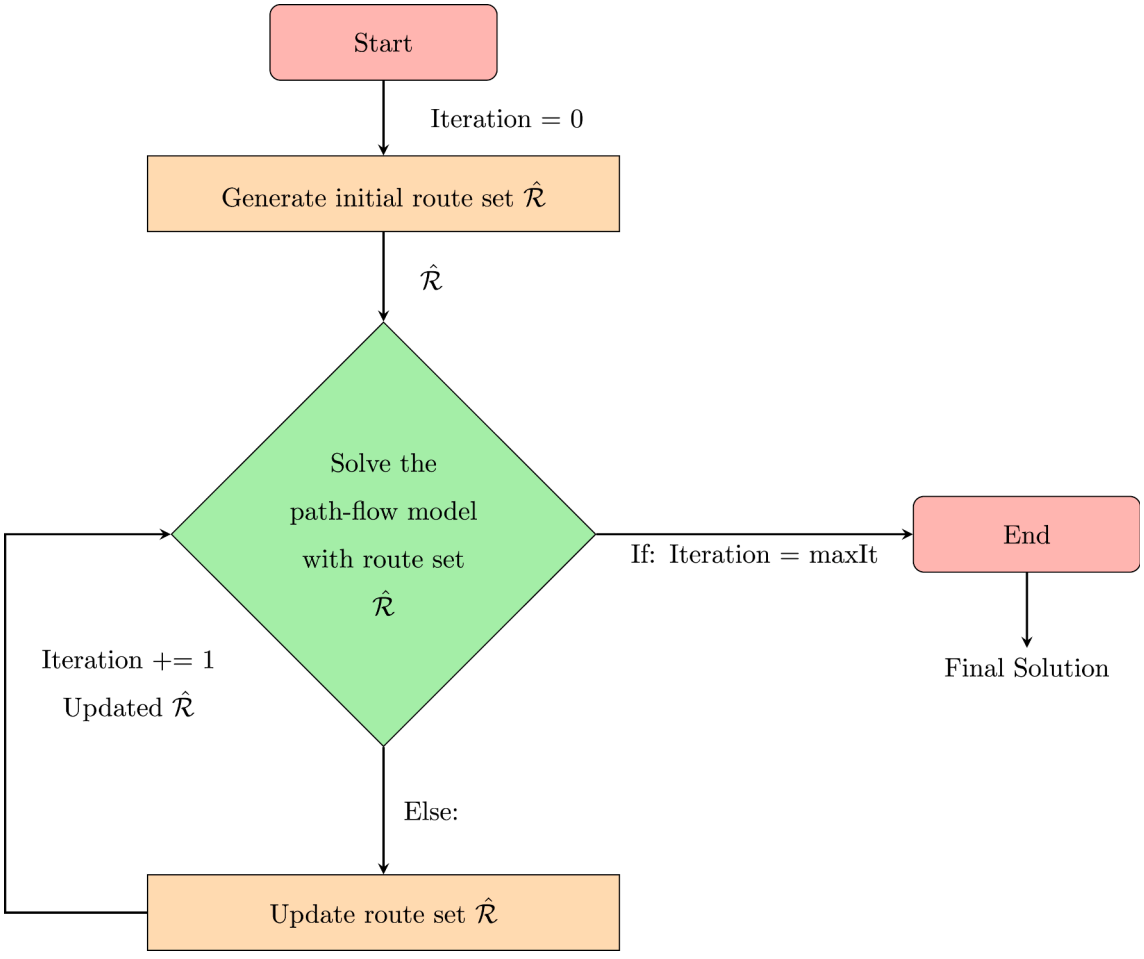


Fig. 1. The matheuristic first creates a set of routes using a giant tour and a split algorithm. These routes are used to solve a modified version of the path-flow model. The model is from there on solved iteratively where the routes of the current best solution are used to create a new set of routes.

grows exponentially with the number of customers and thus becomes so large that, even for small instances, it is not possible to generate all of them within a reasonable amount of time. However, the number of routes used in any feasible solution is bounded by $V{\cdot}T$. Thus, it is possible to obtain feasible (and optimal) solutions to the problem by replacing $\mathscr{R}$ with a small set of routes $\widehat{\mathscr{R}}$, given that $|\widehat{\mathscr{R}}|$ is in the same magnitude as $V{\cdot}T$.

The matheuristic presented in this paper exploits this fact by iteratively generating a set of promising routes and then solving the path-flow model presented in Section 2.1 using this subset. The outline of our heuristic solution method is illustrated in Fig. 1. The method starts by generating a giant tour, which is split into routes in different ways to generate an initial set of promising routes. Then, for a number of iterations, the method alternates between solving the path-flow model and a method that updates the set of routes based on the optimal solution to the path-flow model. In the following, we go through the details of each part of the matheuristic. Section 3.1 describes how the initial set of routes is generated. In Section 3.2, we explain how the set of routes is modified based on the previous solution to the path-flow model.

3.1. Generating an initial set of routes

The first step of the heuristic is to generate an initial set of promising routes. The details of this step are outlined in Algorithm 1. First, a *giant tour* (*GT*) with the minimal total distance on the graph $\mathscr{G} = (\mathscr{N}', \mathscr{A})$ is created by solving a travelling salesman problem using the function *solveTSP*$(\mathscr{N}', \mathscr{A})$. In the implementation, we obtain the giant tour by using the TSP-solver released by Helsgaun (2009), which is considered to be the fastest implementation of the Lin-Kernighan algorithm (Lin and Kernighan, 1973).

After the giant tour is created, it is split into segments, and routes are created by inserting copies of the supplier node at the start and end of each segment. To do this, we utilize the split algorithm proposed by Vidal (2016) for the capacitated vehicle routing problem (CVRP). The split algorithm aims to partition a giant tour solution into separate routes. It does this by solving a shortest path problem on an acyclic graph $\widehat{\mathscr{G}} = (\mathscr{N}, \widehat{\mathscr{A}})$ where $\widehat{\mathscr{A}}$ includes one arc (i,j) with cost $C_{ij}^{\widehat{\mathscr{A}}} = C_{0(i+1)} + \sum_{k=i+1,\dots,j-1} C_{k(k+1)} + C_{j0}$ for any feasible route visiting customers $i+1$ to j. Here, the node order follows the ordering of the giant tour where 1 represents the first node in the giant tour. Traditionally, a version of Bellman's equation has been used to solve the problem, but the aforementioned paper introduces a more efficient labeling algorithm with additional dominance rules. By using the open-source code released by the author, we are able to solve the splitting problem with a limited fleet size in $O(N{\cdot}V)$.

However, to utilize the split algorithm on the obtained giant tour, the giant tour must be turned into a sequence of nodes by choosing a starting node (i^{Start}). In addition, since, unlike the CVRP, the IRP does not have pre-determined demands at each customer, the algorithm must create an array d of customer demands. Thus, the function *Split*(GT, i^{Start}, d) returns the set of routes obtained by applying the split algorithm to a given giant tour, start node, and demand combination.

To create a diverse set of routes in the initial set of routes $\widehat{\mathscr{R}}$, the function *Split*(GT, i^{Start}, d) is called n times with different input combinations. In a given iteration, the algorithm first selects the start node using the *getStartNode*(j) function, which returns the j-th closest node to the supplier. Since the first node in the giant tour is the first node on the first route, it is likely that this node should lie close to the supplier, and the number j is thus varied between 1 and 5. To create customer demand $d[i]$ at node i in a given iteration, the algorithm multiplies the upper inventory limit U_i with a percentage P. This percentage is reduced for each iteration of the algorithm by a given value D. Thus, we get a varied set of routes as splits of the giant tour using large customer demands give short routes, and splits using small customer demands give longer routes.

Algorithm 1: Generating initial set of routes

1. initialize d – array of customer demands
2. initialize P – percentage
3. $\widehat{\mathscr{R}} = \emptyset$
4. GT = solveTSP$(\mathscr{N}', \mathscr{A})$
5. **for** n iterations **do**
6. $\quad i^{Start}$ = getStartNode$((n \bmod 5) + 1)$
7. $\quad$ **for** $i \in \mathscr{N}'$ **do**
8. $\quad\quad d[i] = P{\cdot}U_i$
9. $\quad$ **end for**
10. $\quad \widehat{\mathscr{R}} = \widehat{\mathscr{R}} \cup$ Split(GT, i^{Start}, d);
11. $\quad P = P - D$
12. **end for**
13. **return** $\widehat{\mathscr{R}}$

3.2. VRP heuristic and operators

Algorithm 2: Updating route set

1. input: $\lambda_{rt}^*, r \in \widehat{\mathscr{R}}, t \in \mathscr{T}$
2. input: $q_{it}^*, i \in \mathscr{N}', t \in \mathscr{T}$
3. $\widehat{\mathscr{R}} = \{r \in \mathscr{R} : \exists t \in \mathscr{T}, \lambda_{rt}^* = 1\}$
4. $\quad$ **for** $t \in \mathscr{T}$ **do**
5. $\quad \widehat{\mathscr{R}} = \widehat{\mathscr{R}} \bigcup$ solveVRP(q_{it}^*)
6. **end for**
7. $\widehat{\mathscr{N}} = \emptyset$
8. $\widehat{\mathscr{R}} = \widehat{\mathscr{R}} \bigcup$ RemoveNodes$(\widehat{\mathscr{R}}, \widehat{\mathscr{N}})$ (Algorithm 3)
9. $\widehat{\mathscr{R}} = \widehat{\mathscr{R}} \bigcup$ InsertNodes$(\widehat{\mathscr{R}}, \widehat{\mathscr{N}})$ (Algorithm 4)
10. **for** $r \in \widehat{\mathscr{R}}$ **do**
11. $\quad r$ = solveTSP(r)
12. **end for**
13. **return** $\widehat{\mathscr{R}}$

After a solution from the path-flow model is obtained, the route set is updated with the aim of improving the solution of the path-flow model in the next iteration. A pseudo-code outlining how the route set is updated is given in Algorithm 2. First, the algorithm solves a VRP heuristically for each time period (lines 4–6), where the optimal quantities from the last solution of the path-flow model, q_{it}^*, are used to define the quantities delivered to each customer. This gives (near) optimal routing for these quantities, which may be an improvement on the initial route set. The new routes are added to the set of routes, $\widehat{\mathscr{R}}$, which is added to the path-flow model in the next iteration together with the routes given from the last solution. In this paper, the genetic algorithm proposed by Vidal et al. (2012) has been used as the VRP-heuristic. The authors of this paper have used the open-source implementation which is described in Vidal (2020).

Next, nodes from each of these routes are removed using the method *RemoveNodes*$(\widehat{\mathscr{R}}, \widehat{\mathscr{N}})$ (line 8). The resulting routes are added to the set $\widehat{\mathscr{R}}$. In addition, every node that is removed from a route is added to the set $\widehat{\mathscr{N}}$. These nodes are then re-inserted into different routes in the method InsertNodes (line 9). The resulting routes from these insertions are included in the set $\widehat{\mathscr{R}}$. The two methods are described in detail in Algorithms 3 and 4.

Once this process is completed, all routes to be used in the path-flow model in the next iteration have been generated. However, as a final step, the TSP-heuristic, *solveTSP*(r), is used on each route r to (possibly) reduce the distance driven (line 10–12). This is done because even though a node is removed from or inserted into a route in the cheapest possible way there is no guarantee that the resulting route is optimal. Finally, the route set $\widehat{\mathscr{R}}$ is returned and the path-flow model can be re-solved. Since the new set of routes, $\widehat{\mathscr{R}}$, includes the optimal routes from the previous iteration, we know that this route set provides a solution that is at least as good as in the previous iteration. This is ensured by

warm starting the solver with the solution of the previous iteration.

Algorithm 3: RemoveNodes

1. input: $\widehat{\mathscr{R}}, \widehat{\mathscr{N}}$
2. $\mathscr{R}^{new} = \emptyset$
3. **for** $r \in \widehat{\mathscr{R}}$ **do**
4. **for** $k \in \mathscr{K} \setminus \{\text{RandomRemoval}\}$ **do**
5. $r' = N^k(r)$
6. $\widehat{\mathscr{N}} = \widehat{\mathscr{N}} \bigcup (\widehat{\mathscr{N}}_r \setminus \widehat{\mathscr{N}}_{r'})$
7. $\mathscr{R}^{new} = \mathscr{R}^{new} \bigcup \{r'\}$
8. **end for**
9. **for** $cnt = 1, \ldots, Y$ **do**
10. $k =$ Random Removal
11. $r' = N^k(r)$
12. $\widehat{\mathscr{N}} = \widehat{\mathscr{N}} \bigcup (\widehat{\mathscr{N}}_r \setminus \widehat{\mathscr{N}}_{r'})$
13. $\mathscr{R}^{new} = \mathscr{R}^{new} \bigcup \{r'\}$
14. **end for**
15. **end for**
16. **return** $\mathscr{R}^{new}$

The method RemoveNodes is described in Algorithm 3 and consists of using a set, $\mathscr{K}$, of operators on each route in the set $\widehat{\mathscr{R}}$. These operators, represented by the functions $N^k(r)$, remove one node from a route in different ways before returning the resulting route. The set of operators $\mathscr{K}$ consists of *Cheapest Removal*, *Least Served Removal*, *Most Served Removal* and *Random Removal*. All operators have time complexity $O(n)$, except Random Removal that has $O(1)$.

– *Cheapest removal*: Removes the node from the route that gives the largest reduction in route cost, if removed.
– *Least served removal*: The node in the route that received the smallest quantity of goods in the last solution of the path-flow model is removed from the route.
– *Most served removal*: The node in the route that received the largest quantity of goods in the last solution of the path-flow model is removed from the route.
– *Random removal*: A randomly selected node is removed from the route.

An element of stochasticity is added to the algorithm in the form of *Random Removal* so that it is highly unlikely that two consecutive iterations of the path-flow model optimize over the same set of routes $\widehat{\mathscr{R}}$. Unlike the other operators, Random Removal is run several times. The number of times *Random Removal* is used per route is a parameter denoted Y.

All nodes removed from a route are added to the set $\widehat{\mathscr{N}}$ (line 6) and $\widehat{\mathscr{N}}_r$ is the subset of nodes visited on route r. The routes generated using the removal operators are added to the set $\mathscr{R}^{new}$ which thereafter is included in the set $\widehat{\mathscr{R}}$.

Algorithm 4: InsertNodes

1. input: $\widehat{\mathscr{R}}, \widehat{\mathscr{N}}$
2. $\mathscr{R}^{new2} = \emptyset$
3. $\mathscr{R}^1, \mathscr{R}^2 \subseteq \widehat{\mathscr{R}}$
4. **for** $n \in \widehat{\mathscr{N}}$ **do**
5. $\mathscr{R}^{new2} = \mathscr{R}^{new2} \bigcup \{I(\mathscr{R}^1, n)\} \bigcup \{I(\mathscr{R}^2, n)\}$
6. **end for**
7. **for** $r \in \mathscr{R}^1$ **do**
8. $\mathscr{R}^{new2} = \mathscr{R}^{new2} \bigcup \{C(r)\}$
9. **end for**
10. **return** $\mathscr{R}^{new2}$

The method InsertNodes is described in Algorithm 4 and consists of inserting the nodes in $\widehat{\mathscr{N}}$ back into different routes and hence creating new routes for the path-flow model. Here, the increase in route cost when inserting a node into a route decides which route it is inserted into. This is described as function $I(\mathscr{R}^*, n)$, which receives a set of routes, $\mathscr{R}^*$, and a node n. The function calculates the cheapest position to insert the node into each route. The node is inserted into the route with the lowest marginal cost and the resulting route is returned by the function. $I(\mathscr{R}^*, n)$ is run twice - once for the routes in the current optimal solution of the path-flow model and the routes generated by the VRP-heuristic (described by the set $\mathscr{R}^1$), and once for the routes generated by RemoveNodes (described by the set $\mathscr{R}^2$). By partitioning the route set $\widehat{\mathscr{R}}$ into two subsets, we ensure to create routes that are longer than our original ones and that nodes are inserted into routes that have been shortened.

InsertNodes also includes an operator named *Closest Insertion* which is described as function $C(r)$ in the algorithm. Closest Insertion takes a route and finds the closest neighbor to each of its nodes. All the closest neighbors are added to the route given that they are not already included. The resulting route is added to R^{new2} which thereafter is included in the set $\widehat{\mathscr{R}}$.

4. Computational study

To evaluate the proposed matheuristic, we have tested it on known benchmark instances for the IRP found in the literature. Section 4.1 introduces the benchmark instances used. In Section 4.2, implementation issues and parameter testing is discussed. The computational results are presented in Section 4.3, while the effect of the improvement phase is investigated in Section 4.4.

4.1. Benchmark instances and solution methods

Two sets of benchmark instances have been used. The first set of instances are those created by Archetti et al. (2007) for the IRP with a single vehicle. It consists of 100 instances with three time periods ranging from 5 to 50 customers, where one-half of the instances has high inventory costs while the other half has low. For the rest of this paper, we call this subset of instances *SV-S3* (single-vehicle - small 3). There are also 60 instances with six time periods ranging from 5 to 30 customers. Again, one half of them has high inventory costs while the other half has low. These are called *SV-S6*. The instances were modified for the multi-vehicle version by Adulyasak et al. (2014) and Coelho et al. (2012a) by dividing the original vehicle capacity by the number of vehicles available and rounding to the nearest integer. Up to five vehicles are considered. These are called *MV-S3* and *MV-S6*, where *M* stands for multiple. The second set of benchmark instances was created by Archetti et al. (2012). The instances are larger and more challenging, consisting of 100 instances with 50 customers, 100 instances with 100 customers and 100 instances with 200 customers. They are ranging from one to five vehicles and one half of them has high inventory costs, while the other half has low. We call the single-vehicle version *SV-L6* and the multi-vehicle version *MV-L6* where *L* stands for large. All the instances have six time periods.

To evaluate the proposed matheuristic, our results are compared with the results presented in other papers that have used the same instances. Some of these methods are exact and some are heuristics. Most of the exact methods have focused on solving the smaller instances, while the newer heuristics have also focused on finding good solutions to the larger ones. As mentioned previously, our main focus has been on improving the solution quality of the larger instances. However, the heuristic is also tested on the smaller ones to evaluate its performance on instances with known optima. The methods and results presented in the literature span several years and have all used different CPUs and software. Hence, making a fair comparison of their time consumption is hard. In Table 1, we summarize the other solution methods and their most important features. A complete overview of which instances each paper has solved can be found in Table 2.

Table 1
Benchmark solution methods. We present the solution approach, running platform, number of threads and standard MIP solver. Note: Sol: Solution approach, E: Exact, H: Heuristic/Metaheuristic, M: Matheuristic, Def: Default.

Reference	Name	Sol	CPU	#Threads	Solver
Archetti et al. (2007)	A-BC	E	Pentium IV 2.8 GHz	Def	Cplex 9.0
Coelho and Laporte (2013)	CL-BC	E	Xeon 2.66 GHz	6	Cplex 12.3
Desaulniers et al. (2016)	D-BPC	E	Core i7-2600 3.4 GHz	1	Cplex 12.2
Avella et al. (2018)	AV-BC	E	Core i7-2620, 2.70 GHz	1	Xpress 7.6
Guimarães et al. (2020)	G-BC	E	Xeon E5-2630 v2 2.60 GHz	6	Gurobi 8.1
Manousakis et al. (2020)	M-BC	E	Intel Core i7-7700 CPU 3.60 GHz	8	Gurobi 8.1
Coelho et al. (2012b)	CL-H	H	Intel T7700, 2.4 GHz	Def	–
Adulyasak et al. (2014)	AB-H	H	2.10 GHz Duo CPU PC	Def	Cplex 12.3
Archetti et al. (2012)	AR-H1	M	Intel Dual Core 1.86 GHz	Def	Cplex 10.1
Archetti et al. (2017)	AR-H2	M	Xeon W3680, 3.33 GHz	8	Cplex 12.5
Chitsaz et al. (2019)	C-H	M	Xeon X5650 2.67 GHz	1	Cplex 12.6
Alvarez et al., 2020	AL-H	M	Xeon X5650 2.67 GHz	1	Cplex 12.8
Diniz et al. (2020)	D-H	M	Intel Core i7-8700 K 3.7 GHz	1	LEMON library
This paper	V-H	M	Xeon Gold 6144 3.5 GHz	1	Gurobi 9.0

4.2. Algorithmic implementation and parameters

The parameter values used in this computational study are a result of preliminary testing, where different parameter configurations of the matheuristic were tested and compared. The final parameter values selected are presented in Table 3. There are big differences in the number of customers, vehicles, and time periods between the various instances and different parameter settings are thus more suitable for different sets of instances. However, to avoid overfitting of the parameters to the benchmark instances, we have chosen to keep the parameters, except for the time limit, the same for all sets of instances.

The number of iterations, n, of the split algorithm is set as a result of the initial value of P which is multiplied by U_i, and the percentage, D, which P is decreased by in each iteration. The preliminary testing showed that having P larger than 80% usually results in the split algorithm splitting the giant tour into routes visiting a single customer while having P less than 40% gives a single route visiting all customers. Hence, having P outside of this range rarely results in additional routes. Decreasing P by 2.5% per iteration is, in most cases, sufficient to alter the routes between each iteration, and having P starting at 80% and ending at 40% gives $n = 16$.

Sometimes, commercial solvers can spend a large amount of time to close the duality gap of a MIP without improving the primal solution. Since the path-flow model is solved with a small set of routes, the dual bound is not meaningful, only the primal solution is. Therefore, a maximum time limit on the time spent solving the path-flow formulation in each iteration of the matheuristic is set. The parameter, *1. Time limit*, refers to the time limit put on the first run of the path-flow model, while *Time limit* refers to the consecutive runs as described in Section 3. The number of iterations and the number of times the Random Removal operator is used in RemoveNodes are both set based on the preliminary testing.

Moreover, there are stochastic elements in the matheuristic, and hence two different runs on the same instance will not necessarily lead to the same solution. As a consequence, every instance has been run ten times in our computational study.

4.3. Computational results

In this section, the results of the proposed matheuristic are compared with the results of the benchmark solution methods. First, a comparison between the results obtained by our matheuristic and the best-known solutions for the smaller benchmark instances is made in Section 4.3.1. Since the majority of these have been solved to optimality, a comparison can give valuable insight into how close to optimality the obtained solutions are. In Section 4.3.2, we present the results for the larger instances to see how good the proposed matheuristic is compared with existing heuristics and exact methods. The detailed computational results, and solutions, can be found at http://axiomresearchproject.com/publications/.

Table 3
Parameter values

Parameter	Value
Split algorithm: Start Percentage, P	80%
Split algorithm: Decr. Percentage, D	2.5%
Split algorithm: Iterations, n	16
Path Flow: 1. Time Limit	2 s · (number of customers)
Path Flow: Time Limit	1.5 s · (number of customers)
Path Flow: Iterations, maxIt	5
RemoveNode: number of Random Removal, Y	3

Table 2
Number of instances solved by each solution method. Note: V: number of vehicles.

Name			A-BC	CL-BC	D-BPC	AV-BC	G-BC	M-BC	CL-H	AB-H	AR-H1	AR-H2	C-H	AL-H	D-H	V-H
Set	V	Size	E	E	E	E	E	E	H	H	M	M	M	M	M	M
SV-S3	1	100	100	100	–	–	100	–	100	–	100	–	100	100	100	100
SV-S6	1	60	60	60	–	–	60	–	60	–	60	–	60	60	60	60
MV-S3	2	100	–	100	100	10	100	100	–	100	–	100	100	100	100	100
	3	100	–	100	100	10	100	100	–	100	–	100	100	100	100	100
	4	100	–	100	100	10	100	100	–	–	–	100	100	100	100	100
	5	100	–	100	100	10	100	100	–	–	–	100	100	100	100	100
MV-S6	2	60	–	60	60	40	60	60	–	50	–	60	60	60	60	60
	3	60	–	60	60	40	60	60	–	50	–	60	60	60	60	60
	4	60	–	60	60	40	60	60	–	–	–	60	60	60	60	60
	5	58	–	58	58	38	58	58	–	–	–	58	58	58	58	58
SV-L6	1	60	–	60	–	–	60	60	–	–		–	60	–	–	60
MV-L6	2	60	–	40	–	–	60	40	–	–	–	60	60	–	–	60
	3	60	–	40	–	–	60	40	–	–	–	60	60	–	–	60
	4	60	–	–	–	–	60	40	–	–	–	60	60	–	–	60
	5	60	–	–	–	–	60	40	–	–	–	60	60	–	–	60

4.3.1. Small benchmark instances

The results of running the proposed matheuristic (denoted V-H) 10 times on each of the 798 small test instances are compared with existing results from the literature. All the computational results are summarized in Tables 4–6. In Table 4, the average gap of each solution method is presented. The gap of a solution method on an instance is calculated as the method's objective value minus the best known objective value (not considering the objective value of V-H) divided by the best known objective value. The average per instance set is then calculated. For V-H, two values are given. *V-H Best* is the average gap of the best solution obtained over all instances in each set. For *V-H Average*, the average gap over the ten runs has been calculated for each instance, and then the average over these for each instance set is reported.

It is clear that the most recent exact methods, G-BC and M-BC, outperform all the heuristics in terms of solution gap. This is natural since 642 of the 798 small test instances have been solved to optimality. In fact, the only subsets of instances where the majority of instances still have not been solved to proven optimality are *MV-S6* with four and five vehicles. *V-H Best* and *V-H Average* have the smallest solution gaps for these instances among all heuristics. Overall, the results of V-H are competitive compared with the other heuristics and *V-H Best* has an average gap of 0.99% over all instances. D-H is the only heuristic that has a better average gap with 0.37% over all instances. However, there is a significant performance difference between the three and six time period instances for V-H. *V-H Best* has an average gap of 0.40% for the multi-vehicle instances with six time periods which is lower than all the other heuristics.

In Table 5, the number of best-known solutions found by each solution method is presented. *V-H Best* has the same meaning as above, but its column has an extra number written within parentheses. This shows the number of best-known solutions that have a strictly lower objective value than the previously best-known solution, i.e. the number of new best-known solutions found. *V-H Best* finds more best-known solutions than any of the other heuristics, except *D-H*, with 306 solutions, of which 14 are new solutions. The column *V-H Worst* shows the number of instances where all the ten runs of V-H have found the best-known solution. With *215* best-known solutions, it is still competitive compared with the other heuristics. As in Table 4, V-H performs significantly better on the six time period instances, compared with those that have three time periods. For the six time period instances, *V-H Best* finds the best-known solution for 59 out of 180 instances for the three, four and five vehicle instances. This is significantly higher compared with the other heuristics.

Table 6 presents the average computing time for each solution method in seconds on each set of instances. *V-H Average* represents the average computing time over the ten runs, while *V-H Max* is the average of the runs with the maximal computing times for each instances. The results show that V-H's computing time is similar to C-H and D-H, and it is substantially shorter than the other solution methods. The best exact solution method on the MV-S6 instances, M-BC, spends on average 4,210 s with an average gap of 0.02%, while V-H spends, on average, 138 s with an average gap of 0.75%. Thus, the results indicate that V-H produces close to optimal solutions within a small fraction of the computing time used by exact methods.

4.3.2. Large benchmark instances

The results of running the proposed matheuristic (denoted V-H) ten times on each of the 300 large test instances are compared with existing results from the literature. The results are summarized in Tables 7–9. In Table 7, average gaps are presented. Here, *V-H Best* and *V-H Average* have the same meaning as in Section 4.3.1. Both G-BC and M-BC as well as AR-H1 produce lower gaps for the single-vehicle instances. However, the results for *V-H Best*, an average gap of 1.65%, are in fact quite good taken into account that in this subset of instances all 50 customer instances are solved to optimality and most 100 customers instances have a duality gap of less than one percent for its best-known solution. On the multi-vehicle instances, V-H reports lower average gaps than all the other solution methods except for two vehicles. *V-H Best* has an average gap of −0.54% for *MV-L6*, while M-BC, which has the second-lowest average gaps, has an average gap of 0.28%. However, M-BC has not been tested on the 200 customers instances which are the largest and most challenging ones. In fact, the authors themselves state that they are not solved, because their exact solution method provides no insightful results within 2 h on these instances. C-H, which has been tested on all instances, has in comparison a gap of 1.24%.

In Table 8, the number of best-known solutions for each solution method is presented. *V-H Best* and *V-H Worst* have the same meaning as in Section 4.3.1 and the extra number shows the number of strictly improving best-known solutions. *V-H Best* has the best-known solution for 179 out of 240 multi-vehicle instances and naturally outperforms the

Table 4
Average gaps for the different solution methods. The best average gap for each subset of instances is highlighted.

Set	*V*	# Instances	A-BC	CL-BC	D-BPC	AV-BC	G-BC	M-BC	CL-H	AB-H	AR-H1	AR-H2	C-H	AL-H	D-H	V-H Best	V-H Average
SV-S3	1	100	**0**	**0**	–	–	**0**	–	0.44	–	**0**	–	1.93	1.03	0.02	0.16	0.16
SV-S6	1	60	**0**	**0**	–	–	**0**	–	0.49	–	0.14	–	1.09	2.68	0.15	0.77	1.15
MV-S3	2	100	–	0.01	12.32	1.13	**0**	0.01	–	8.48	–	0.14	1.79	2.58	0.04	1.17	1.32
	3	100	–	0.77	7.92	2.90	**0.03**	0.05	–	9.14	–	0.38	1.30	2.70	0.11	2.07	2.38
	4	100	–	3.27	6.73	7.29	0.30	**0.11**	–	–	–	1.07	1.58	2.84	0.51	1.61	2.03
	5	100	–	5.86	5.92	11.24	0.85	**0.06**	–	–	–	1.71	2.08	2.89	0.65	1.47	1.85
MV-S3: Average Gap		400	–	2.48	8.22	5.64	0.30	0.06	–	8.81	–	0.83	1.69	2.75	0.33	1.58	1.90
MV-S6	2	60	–	1.43	33.11	1.38	**0**	0.03	–	4.05	–	0.35	3.98	3,01	0.12	0.44	0.73
	3	60	–	1.84	36.80	3.46	0.17	**0.04**	–	3.71	–	1.74	4.76	2.68	0.47	0.43	0.81
	4	60	–	3.75	36.41	4.91	0.63	**0.02**	–	–	–	3.24	5.61	2.59	0.85	0.41	0.75
	5	58	–	4.89	34.39	6.53	1.05	**0**	–	–	–	4.00	6.80	2.52	1.19	0.31	0.70
MV-S6: Average Gap		238	–	2.98	35.18	4.07	0.46	0.02	–	3.88	–	2.33	5.28	2.70	0.66	0.40	0.75
Total: Average Gap		798	0	2.03	18.33	4.32	0.29	0.04	0.46	7.20	0.05	1.38	2.74	2.52	0.37	0.99	1.28

Table 5
The number of best-known solutions for the different solution methods. The solution method with the highest number for each subset of instances is highlighted.

Set	*V*	# Instances	A-BC	CL-BC	D-BPC	AV-BC	G-BC	M-BC	CL-H	AB-H	AR-H1	AR-H2	C-H	AL-H	D-H	V-H Best	V-H Worst
SV-S3	1	100	**100**	**100**	–	–	**100**	–	49	–	98	–	23	57	96	84 (0)	84 (0)
SV-S6	1	60	**60**	**60**	–	–	**60**	–	22	–	24	–	7	8	44	25 (0)	11 (0)
MV-S3	2	100	–	98	66	7	**100**	99	–	11	–	59	19	33	87	56 (0)	40 (0)
	3	100	–	75	71	2	**86**	**86**	–	10	–	54	23	23	71	32 (0)	19 (0)
	4	100	–	44	69	0	71	**75**	–	–	–	32	17	14	43	18(1)	8 (0)
	5	100	–	38	80	0	58	**84**	–	–	–	23	20	22	36	16 (0)	11 (0)
MV-S6	2	60	–	40	18	15	**59**	46	–	0	–	14	0	1	31	16 (0)	10 (0)
	3	60	–	22	15	0	41	**43**	–	2	–	14	0	2	13	24 (5)	11 (1)
	4	60	–	13	17	0	20	**50**	–	–	–	4	0	1	5	20 (5)	14 (1)
	5	58	–	8	14	0	16	**54**	–	–	–	6	0	2	4	15 (3)	7 (1)
Total small instances		798	160	498	350	24	611	537	71	23	122	206	109	163	430	306 (14)	215(3)

Table 6
The average solution time for each solution method. The solution times of other methods are directly adopted from the corresponding papers. Note: N/A = not available.

Set	*V*	# Instances	A-BC	CL-BC	D-BPC	AV-BC	G-BC	M-BC	CL-H	AB-H	AR-H1	AR-H2	C-H	AL-H	D-H	V-H Average	V-H Max
SV-S3	1	100	436	10	–	–	28	–	522	–	336	–	47	N/A	32	2	2
SV-S6	1	60	880	33	–	–	114	–	458	–	664	–	50	N/A	54	9	11
MV-S3	2	100	–	703	3,418	2,350	456	368	–	N/A	–	936	62	N/A	64	16	16
	3	100	–	2,556	3,226	4,285	1,791	2,611	–	N/A	–	1,207	65	N/A	105	17	18
	4	100	–	4,547	2,877	5,228	3,385	3,738	–	–	–	615	69	N/A	150	20	21
	5	100	–	5,052	2,395	5,400	4,135	3,735	–	–	–	694	65	N/A	202	25	26
MV-S6	2	60	–	2,610	5,292	4,114	1,616	2,658	–	N/A	–	1,797	80	N/A	136	53	60
	3	60	–	5,496	5,622	5,050	4,294	4,549	–	N/A	–	2,214	76	N/A	227	141	156
	4	60	–	6,034	5,503	5,124	5,693	4,703	–	–	–	1,108	81	N/A	325	178	180
	5	58	–	6,254	5,896	5,190	6,148	4,930	–	–	–	1,294	65	N/A	414	181	182

other methods. It also finds new best-known solutions for seven single-vehicle instances with 200 customers.

Table 9 gives the average computing time for each solution method for the set of larger instances. Unsurprisingly, the exact solution methods spent significantly more time than the heuristics, since they focus both on finding primal and dual bounds for each instance. However, for the larger instances, also all heuristics proposed in the literature spend, on average, more than one hour (3600 s) on each instance. C-H and AR-H2 are the two other heuristics that have been tested on the full set of larger instances. AR-H2 used a CPU with approximately the same clock speed as V-H, however, they used eight cores versus the one core used by V-H. C-H also used one core, but their clock speed is lower. However, V-H only spent a small fraction of the computing time C-H spent on the larger instances. For some of the largest instances, C-H spent almost 25,000 s, while V-H never spent more than 2,300 s. Hence, it is unlikely that the improved computing times are only a result of the

Table 7
Average gaps for the different solution methods. The best average gap for each subset of instances is highlighted.

Set	*V*	# Instances	CL-BC	G-BC	M-BC	AR-H1	AR-H2	C-H	V-H Best	V-H Average
SV-L6	1	60	11.04	**0.23**	0.25	0.39	–	3.64	1.65	1.98
MV-L6	2	60	69.64	3.94	**0.11**	–	3.89	1.77	0.22	0.56
	3	60	115.61	10.11	0.04	–	4.48	1.36	**−0.77**	−0.46
	4	60	–	19.11	0.80	–	6.17	0.81	**−0.80**	−0.46
	5	60	–	25.42	0.18	–	6.35	1.00	**−0.79**	−0.37
MV-L6: Average Gap			92.63	14.65	0.28	–	5.22	1.24	−0.54	−0.18
All Instances: Average Gap			65.43	11.67	0.28	0.39	5.22	1.72	−0.10	0.25

Table 8
The number of best-known solutions for the different solution methods. The solution method with the highest number for each subset of instances is highlighted.

Set	*V*	# Instances	CL-BC	G-BC	M-BC	AR-H1	AR-H2	C-H	V-H Best	V-H Worst
SV-L6	1	60	20	**44**	15	8	0	0	7 (7)	4 (4)
MV-L6	2	60	0	11	20	0	0	2	**28 (28)**	18 (18)
	3	60	0	0	14	0	0	0	**47 (46)**	37 (37)
	4	60	0	0	6	0	0	3	**51 (51)**	32 (32)
	5	60	0	0	4	0	0	3	**53 (53)**	29 (29)
Total large instances		300	20	55	59	8	0	8	**186 (185)**	120 (120)

Table 9
The average solution time for each solution method. The solution times of other methods are directly adopted from the corresponding papers.

Set	V	# Instances	CL-BC	G-BC	M-BC	AR-H1	AR-H2	C-H	V-H Average	V-H Max
SV-L6	1	60	64,509	5,562	5,734	3630	–	6,668	105	122
MV-L6	2	60	86,400	7,200	7,200	–	4,066	6,657	351	390
	3	60	86,400	7,200	7,200	–	4,540	4,209	635	735
	4	60	–	7,200	7,200	–	4,257	5,132	953	1,096
	5	60	–	7,200	7,200	–	4,418	5,527	1,197	1,280

difference in hardware. Also looking at the exact methods that have solved the largest instances, it is clear that V-H still has the best performance. On the multi-vehicle instances, V-H only spends a small fraction of time compared with what the exact methods do and still obtain much better solutions. On the single-vehicle instances, where V-H does not obtain the best solutions, we notice that V-H spends 105 s on average while all the other solution methods spend over 3,600 s.

In summary, we see that on the larger instances, the heuristic proposed in this paper produces, on average, less costly solutions than previous heuristics suggested, and it does so by using significantly less computing time. The heuristic has found the best-known solution on 186 out of the 300 larger instances, and where 185 of them are an improvement of what is previously reported in the literature. The numbers are 179 out of 240 instances if we only consider the multi-vehicle instances.

4.4. Effect of improvement phase

As seen in Section 3, the proposed matheuristic consists of two parts. The first part involves creating an initial solution through the use of a giant tour. Then follows an iterative process where the solution is improved by a set of operators and a VRP-heuristic. The effect the different operators, and the VRP heuristic, has on the final objective value of *VH-Best* is summarized in Table 10. The table shows the average improvement in percentage between the initial objective value and the final objective value for each subset of instances. The percentage is calculated as the final objective value minus the initial objective value divided by the initial objective value.

From the numbers presented in Table 10, it is clear that the improvement phase contributes to reducing the objective value obtained from solving the path-flow model with only columns from the giant tour. However, each iteration of the matheuristic increases the computing time and there is a trade-off between performance and time spent. Further, increasing the number of iterations generally has a diminishing effect, and only marginally improves the solution. Hence, finding the right number of iterations is crucial and something we tested extensively in the preliminary testing.

Table 10
Overview of the objective value improvements due to operators

Set	V	Improvement [%]
SV-S3	1	−0.18
SV-S6	1	−3.29
MV-S3	2	−2.10
	3	−2.31
	4	−2.91
	5	−3.13
MV-S6	2	−3.04
	3	−2.45
	4	−1.74
	5	−1.55
SV-L6	1	−1.78
MV-L6	2	−3.36
	3	−3.76
	4	−3.71
	5	−3.45

Another interesting analysis is to study where the routes that contribute to an improved solution originate from. Most of the routes in the solution from an iteration are identical to the routes in the previous iteration's solution. However, as seen in Table 10, the objective value usually gets better from the first iteration to the last. We have analyzed all routes that have contributed to a better solution and tracked their origin. Table 11 shows the percentage that each operator (and the VRP heuristic) contributes to this set of improved routes.

From Table 11 we observe that the VRP heuristic is the second largest contributor of routes that improve the objective value. This is reasonable due to the nature of the heuristic. Especially in larger instances, the initial distribution of nodes between vehicles can be suboptimal in a time period given the stated demand, and the heuristic tries to find a better distribution. In addition, the operators which remove nodes produce more improving routes than the operators which add nodes. This indicates that the initial solution created from the routes from the giant tour consists of routes that are generally too long. This is reasonable since they consist of segments of the giant tour. In an optimal solution, the nodes from a segment might not be part of an optimal route, which is a weakness of our proposed heuristic. However, the operators that remove nodes are included in the matheuristic to take care of this. The random operators are the most used removal operators. This is an indication that it is often not obvious which nodes should be removed from a route.

5. Concluding remarks

In this paper, we propose a heuristic algorithm that relies on creating a giant tour and a split algorithm to create promising routes that make up the route set for a path-flow formulation of the IRP. The path-flow model is re-solved for a number of iterations where the route set is updated between each iteration with the help of simple operators. Computational results show that the matheuristic provides high-quality solutions for larger benchmark instances of the IRP. Out of 240 larger multi-vehicle instances, the matheuristic finds the best-known solutions on 179 instances, where 178 of them have a strictly lower objective value than the previously best-known solution. It also finds a new best upper bound for seven single-vehicle instances. In addition, the matheuristic often only spends a small fraction of the time that existing solution methods from the literature use. Although it has not been the focus of this paper, the proposed matheuristic also performs very well on smaller instances. Its average solution gap is similar to other heuristics and it has the second-highest number of best-known solutions among all

Table 11
The percentage of the routes that improve the solution that comes from different operators and the VRP heuristic

Route	Percentage
Cheapest Removal	12.6
Least Served Removal	9.3
Most Served Removal	4.7
Random Removal	17.7
Closest Insertion	2.9
Cheapest Insertion	27.6
VRP-heuristic	25.2

reported heuristics.

CRediT authorship contribution statement

Simen T. Vadseth: Conceptualization, Methodology, Software, Investigation, Writing - original draft, Writing - review & editing, Visualization. **Henrik Andersson:** Conceptualization, Methodology, Supervision, Writing - review & editing. **Magnus Stålhane:** Conceptualization, Methodology, Supervision, Writing - original draft, Writing - review & editing, Project administration.

Acknowledgement

The authors thank Doctor Masoud Chitsaz for providing the updated results of Chitsaz et al. (2019) and for sharing the material he had gathered on the best-known solutions for the benchmark instances, and Professor Rafael Martinelli for sharing the detailed results of Diniz et al. (2020). Doctor Masoud Chitsaz and Professor Jean-Francois Cordeau also provided valuable feedback during the writing of this paper. We also thank two anonymous referees for their constructive comments and suggestions, which helped improve the quality of the paper. Lastly, we would like to thank the Norwegian Research Council for funding the research and our collaborators in the AXIOM project for their valuable feedback.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, athttps://doi.org/10.1016/j.cor.2021.105262.

References

Adulyasak, Y., Cordeau, J.-F., Jans, R., 2014. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. INFORMS Journal on Computing 26 (1), 103–120.
Alvarez, A., Cordeau, J.-F., Jans, R., Munari, P., Morabito, R., 2020. Formulations, branch-and-cut and a hybrid heuristic algorithm for an inventory routing problem with perishable products. European Journal of Operational Research 283 (2), 511–529.
Andersson, H., Hoff, A., Christiansen, M., Hasle, G., Løkketangen, A., 2010. Industrial aspects and literature survey: Combined inventory management and routing. Computers & Operations Research 37 (9), 1515–1536.
Archetti, C., Speranza, M.G., 2014. A survey on matheuristics for routing problems. EURO Journal on Computational Optimization 2, 223–246.
Archetti, C., Bertazzi, L., Laporte, G., Speranza, M.G., 2007. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. Transportation Science 41 (3), 382–391.
Archetti, C., Bertazzi, L., Hertz, A., Speranza, M.G., 2012. A hybrid heuristic for an inventory routing problem. INFORMS Journal on Computing 24 (1), 101–116.
Archetti, C., Boland, N., Grazia Speranza, M., 2017. A matheuristic for the multivehicle inventory routing problem. INFORMS Journal on Computing 29 (3), 377–387.
Avella, P., Boccia, M., Wolsey, L.A., 2018. Single-period cutting planes for inventory routing problems. Transportation Science 52 (3), 497–508.
Bell, W.J., Dalberto, L.M., Fisher, M.L., Greenfield, A.J., Jaikumar, R., Kedia, P., Mack, R. G., Prutzman, P.J., 1983. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. Interfaces 13 (6), 4–23.
Boschetti, M.A., Maniezzo, V., Roffilli, M., Röhler, A.B., 2009. Matheuristics: Optimization, simulation and control. International Workshop on Hybrid Metaheuristics 171–177.
Chitsaz, M., Cordeau, J.-F., Jans, R., 2019. A unified decomposition matheuristic for assembly, production, and inventory routing. INFORMS Journal on Computing 31 (1), 134–152.
Coelho, L.C., Laporte, G., 2013. The exact solution of several classes of inventory-routing problems. Computers & Operations Research 40 (2), 558–565.
Coelho, L.C., Laporte, G., 2014. Improved solutions for inventory-routing problems through valid inequalities and input ordering. International Journal of Production Economics 155, 391–397.
Coelho, L.C., Cordeau, J.-F., Laporte, G., 2012a. Consistency in multi-vehicle inventory-routing. Transportation Research Part C: Emerging Technologies 24, 270–287.
Coelho, L.C., Cordeau, J.-F., Laporte, G., 2012b. The inventory-routing problem with transshipment. Computers & Operations Research 39 (11), 2537–2548.
Coelho, L.C., Cordeau, J.-F., Laporte, G., 2014. Thirty years of inventory routing. Transportation Science 48 (1), 1–19.
Desaulniers, G., Rakke, J.G., Coelho, L.C., 2016. A branch-price-and-cut algorithm for the inventory-routing problem. Transportation Science 50 (3), 1060–1076.
Diniz, P., Martinelli, R., Poggi, M., 2020. An efficient matheuristic for the inventory routing problem. International Symposium on Combinatorial Optimization 273–285.
Guimarães, T.A., Schenekemberg, C.M., Coelho, L.C., Scarpin, C.T., Pécora Jr, J.E., 2020. Mechanisms for feasibility and improvement for inventory-routing problems. Tech. Rep. CIRRELT-2020-12, Université de Montréal, Canada.
Helsgaun, K., 2009. General k)opt submoves for the Lin-Kernighan TSP heuristic. Mathematical Programming Computation 1 (2–3), 119–163.
Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Operations Research 21 (2), 498–516.
Manousakis, E., Repoussis, P., Zachariadis, E., Tarantilis, C., 2020. Improved branch-and-cut for the inventory routing problem based on a two-commodity flow formulation. European Journal of Operational Research 290, 870–885.
Vidal, T., 2016. Split algorithm in O(n)) for the capacitated vehicle routing problem. Computers & Operations Research 69, 40–47.
Vidal, T., 2020. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. arXiv preprint arXiv:2012.10384.
Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W., 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research 60 (3), 611–624.