

Multi-criteria Intermodal Vehicle Routing Problem for Maintenance in Manufacturing Systems

Jarosław RUDY 

Wrocław University of Science and Technology, Department of Control and Quantum Computing, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Received: 04 March 2026

Accepted: 18 May 2026

Abstract

In this paper, a multi-objective optimization problem of delivering and picking up crews for maintenance and servicing of machines in production systems is considered. The problem is formulated as an intermodal variant of the Vehicle Routing Problem with Pickup and Delivery. Two optimization objectives of total distance traveled by all vehicles and total weighted tardiness of all servicing jobs are considered. A solution in the form of a modified Delimiters Representation paired with a disjunctive graph is proposed. The methods for checking solution feasibility and determining the objective function value are discussed and proven as theorems. Three solution methods for the considered problem are proposed: two metaheuristic methods Tabu Search and Genetic Algorithm and a greedy heuristic serving as a reference. 9765 problem instances of up to 40 servicing jobs are prepared as a benchmark and computer experiments are performed on that instances to test the effectiveness of the proposed solution methods. The results show that Tabu Search vastly outperforms the Genetic Algorithm and the greedy heuristic, providing better Pareto front approximations measured through the Hypervolume Indicator. The research also shows that combining both metaheuristics slightly improves the result. Experiments are also conducted to show the influence of problem parameters on the resulting solution.

Keywords

multi-objective optimization, vehicle routing, metaheuristics, pickup and delivery, time windows, intermodal transport, hypervolume indicator, tabu search, genetic algorithm.

Introduction

Modern manufacturing systems are often composed of a variety of specialized equipment. This can include machines of different sizes, robots, vehicles (like Automated Guided Vehicles) among others. Equipment like that can be large, advanced and expensive, which results in the need for servicing (Ogunfowora & Najjaran, 2023). Moreover, the emergence of Industry 4.0 has brought new possibilities such as maintenance prediction (Achouch et al., 2022; Zonta et al., 2020), leading to increased attention being directed to problems such as maintenance optimization (Pinciroli et al., 2023).

There can be many causes for servicing jobs, such as scheduled maintenance or emergency (e.g. a machine fault requiring repairs). Regardless of the cause, servicing is done by human crews of varying size with varying tools. In small manufacturing system such maintenance crews can be kept on-premises and reach the maintenance site without the need for additional planning. However, for larger systems crews often require to be transported to the maintenance site and picked up after the servicing jobs is done. This is especially true if the manufacturing system spans multiple locations or if the servicing crews are working for an external maintenance company (outsourcing).

In this paper, a problem of scheduling a delivery and pickup of crews for servicing jobs in manufacturing systems is considered. The problem is modeled as a variant of Vehicle Routing Problem (VRP). Since the maintenance results in downtime of machines and production, the maintenance has to be scheduled ahead if time in a form of a time window.

Corresponding author: Jarosław Rudy – Wrocław University of Science and Technology, Department of Control and Quantum Computing, Wrocław University of Science and Technology 11/17 Zygmunta Janiszewskiego Street, 50-372 Wrocław, Poland, e-mail: jaroslaw.rudy@pwr.edu.pl

© 2026 The Author(s). This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>)

In optimization problems time windows are typically one of two types. Hard time windows imply that the activity cannot start before the window starts and must complete before the window ends (Jolfaei et al., 2024). This type of time window translates into problem constraint (*i.e.* violating it leads to infeasibility). On the other hand, soft time windows allow for starting or completing the activity outside of the time window, but incur a penalty (*i.e.* it affects the resulting objective function value) (Jie et al., 2022).

The problem considered in this paper does not fit the above two types. Prior to the opening of the window the machine to be maintained might be still working or being prepared for maintenance (*e.g.* being powered down or cooling off). As such, the servicing cannot start before the time window. On the other hand, maintenance can require more time than the time window allows or the crew might arrive late. In both cases, the machine cannot resume operation until the maintenance is completed. If this causes the maintenance to continue after the time window closes, the production schedule will be affected, possibly incurring a loss due to not keeping the requested deadline. This is referred to as the servicing job tardiness. The result is that in this problem mixed hard-soft windows apply: starting the servicing before the time window is infeasible, while completing it after the time window incurs an objective function penalty.

From the perspective of the maintenance company, the common objective is to minimize the total distance traveled by the vehicles that dispatch the crews, as this minimize the operational costs (*e.g.* fuel usage). On the other hand, the company wants to avoid tardy jobs, as such jobs are detrimental either directly (*e.g.* contract penalties costs) or indirectly (*e.g.* reputation loss). Moreover, different customers might be assigned different weights (which reflect job priorities or customer importance). Since the second objective is not easily translated into monetary cost, we treat those objectives as separate. This leads to a bi-objective optimization, where both total traveled distance and total weighted tardiness are minimized. Note that the paper uses the concept of Pareto-efficiency, where the objective function value is a pair of numbers and the aim is to obtain the Pareto front (or approximate it).

Importantly, there is no restriction requiring that the crew needs to be picked up by the same vehicle that delivered it. Thus, the transport considered in this paper is intermodal. The overall problem will be referred to as the Intermodal Vehicle Routing Problem for Maintenance or IVRP-M for short.

Finally, note that while the problem is geared towards maintenance in production systems, it can easily be extended to maintenance in other areas such as off-

shore wind farms (Si et al., 2025) or even scheduling of at-home medical visits (Szwarc et al., 2024).

The main contributions of the considered paper can be summarized as shown below:

1. The IVRP-M problem with hard-soft windows is formulated.
2. A solution representation using a modified Delimiters Representation and a disjunctive graph is proposed.
3. Feasibility and objective function determination are discussed and proven.
4. Three solution methods for IVRP-M are proposed.
5. A large set of benchmark instances are proposed.
6. Computer experiments are conducted to verify the effectiveness of the proposed methods.

The remainder of this paper is structured as follows. Section 2 provides a literature review. In Section 3, the IVRP-M problem is formulated. Section 4 proposes a solution representation and discusses its properties. In Section 5 the solution methods are described. In Section 6 the results of the numerical experiments are shown. Section 7 contains a brief discussion. Finally, Section 8 provides the conclusions and future work directions.

Literature review

This section provides an overview of the related literature, focusing on maintenance in production systems and VRP problem for maintenance or similar pickup/delivery applications.

We start with maintenance in manufacturing systems. A joint maintenance and routing problem for preventive maintenance of machines with non-deterministic failures was considered in López-Santana et al. (2016). The solution method first assigned time windows to minimize the expected maintenance costs and then assigned technicians to minimize chance of failure and maintenance waiting time. A similar system for multi-period geographically distributed machines was proposed in López-Santana et al. (2023). The problem was decomposed into master and auxiliary problems to manage the balance in the maintenance cost, routing cost, and machine failure probabilities. A solution method for the dynamic maintenance VRP with time windows to minimize costs of preventive and corrective machine maintenance was considered in López-Ayala et al. (2021). The method had 98% reduced computation time compared with a reference method, while yielding 5% performance gap.

Next, maintenance scheduling taking into account the interactions with the production process was considered in Yang et al. (2008). The proposed Genetic

Algorithm (GA) using predicted machine failure probabilities was benchmarked against traditional strategies and offered superior performance. In paper [Kubzin & Strusevich \(2006\)](#) a two-machine scheduling problem with maintenance times depending on maintenance start time was considered. The authors proved polynomial and pseudo-polynomial complexity for two problem variants. Next, a problem of dynamic maintenance scheduling under uncertainty was considered in [Ruiz-Rodríguez et al. \(2024\)](#). The authors compares the effectiveness of GA, Reinforcement Learning (RL) and dispatch rules under different uncertainty conditions. A multi-objective Artificial Bee Colony method for flexible job shop scheduling with preventive maintenance was proposed in [Zhang et al. \(2024\)](#), obtaining verified state-of-the-art results. A Flexible Job Shop with joint scheduling and maintenance assignment was considered in [Wocker et al. \(2024\)](#) using real data from automotive industry and obtained makespans similar to the case without any maintenance scheduling. Finally, a review of maintenance planning with RL was shown in [Ogunfowora & Najjaran \(2023\)](#).

We will now move onto papers considering maintenance outside of manufacturing systems. [Radzki et al. \(2025a\)](#) considered an intermodal VRP with time windows for servicing teams while prioritizing customer coverage rather than minimizing costs. Similarly, [Radzki et al. \(2025b\)](#) focused on assessing the feasibility condition for routing and scheduling of servicing crews, followed up by validation experiments. Next, [Rastegar et al. \(2024\)](#) considered scheduling and routing problem for elevator maintenance, proposing a Mixed-Integer Programming formulations and a Variable Neighborhood Search. Similarly, [Jitkongchuen et al. \(2023\)](#) considered a routing problem for transmission stations, reducing the maintenance costs by 82% and the operation time by 52%. [Biteus & Lindgren \(2017\)](#) considered predictive random forest for maintenance planning of trucks at Scania to maximize fleet utilization and minimize costs.

Problem formulation

In this section, we define the IVRP-M problem, starting with a small motivating example before proceeding with more formal description. The summary of the notation used throughout the paper is shown in Table 1.

To ground the formal model, consider a simple case where three factories have requested a service of their machines. Each service takes a specified amount of time and has time window planned for it. Starting the servicing before the window opens is impossible

Table 1
Notation overview

Symbol	Meaning
Indexes	
i, j	Indexes for jobs, crews and general use
k, l	Indexes for locations
r, s	Indexes for vehicles
Problem parameters	
J, n	Set of jobs, number of jobs
L	Set of locations
V, v	Set of vehicles, number of vehicles
p_i, w_i	Servicing time and priority of job i
b_i, e_i	Window starting and ending time of job i
$c_{k,l}$	Cost (time or distance) of traveling from location k to location l
D	Vehicle capacity
Solution	
$\pi, \pi(i)$	Delimiters Representation, i -th element π
G_π	Disjunctive graph for solution π
A_i, R_i	Crew arrival and retrieval time for job i
S_i, C_i	Start and completion time of job i
$D(\pi)$	Total distance traveled for solution π
$T(\pi)$	Total weighted tardiness for solution π

(as machine is in use and unavailable for servicing). Ending the servicing after the window closes is possible, but it interferes with the planned factory schedule and incurs a penalty (e.g. loss of reputation for the servicing company). To service the machines, three servicing crews (one per location) are deployed from the central hub of the servicing company via two available vehicles, each capable of holding up to two crews at the time. The challenge is to plan the routes of the two vehicles such that each location has one crew delivered for servicing and picked up afterwards, while minimizing two objectives: (1) the total distance traveled by both vehicles and (3) the total servicing time spend after window closing time for all three servicing location.

We will now move onto defining problem inputs. There is a set $J = \{1, 2, \dots, n\}$ of n servicing jobs. Each job $i \in J$ is defined with four parameters: (1) servicing (processing) time $p_i > 0$, (2) priority (weight) $w_i > 0$, (3) window start (beginning) time $b_i \geq 0$, and (4) window closing (ending) time $e_i > b_i$. Note that each job will be serviced by a separate crew. Thus, we also implicitly define n identical crews. However, since crews are indistinguishable, without the loss of generality we can assume crew i is assigned to job i .

Next, we define a set $L = \{0, 1, \dots, n\}$ of $n + 1$ locations. Location $k \in L \setminus \{0\}$ represents servicing job k and location 0 represents the depot. For each pair $k, l \in L$ of locations we define the cost of traveling from k to l as $c_{k,l}$. We assume that $c_{k,l} = 0$ when $k = l$ and $c_{k,l} > 0$ otherwise. Note that we do not assume that the traveling costs form a metric space *i.e.* we do not require that $c_{k,l} = c_{l,k}$ or $c_{k,l} \leq c_{k,u} + c_{u,l}$. This is to model real-life transport (*e.g.* one-way roads). Also, note that $c_{k,l}$ models both travel time and travel distance.

Finally, by $V = \{1, 2, \dots, v\}$ we denote the set of v identical vehicles, each with capacity $D > 0$. We also assume the following relationship between the values of n , v and D :

$$n \geq v \geq \left(\frac{n}{D}\right). \quad (1)$$

An example instance for $n = 3$, $v = 2$, and $D = 2$ is shown in Tab. 2. This instance matches the motivation example shown earlier.

Table 2
Example instance for $n = 3$, $v = 2$, and $D = 2$

i	0	1	2	3
p_i	—	3	5	3
w_i	—	4	8	6
b_i	—	6	4	2
e_i	—	11	10	7
$c_{0,i}$	0	2	1	4
$c_{1,i}$	1	0	4	2
$c_{2,i}$	4	2	0	6
$c_{3,i}$	3	6	5	0

The objective is to assign jobs to vehicles and to schedule a travel route for each non-empty vehicle as to service all jobs while minimizing the multi-objective function (described later in this section) under the following assumptions:

- All crews start and end at the depot.
- A servicing job $i \in J$ cannot start before crew i has been delivered to location i .
- A servicing job $i \in J$ cannot start before its window start time $b_i \geq 0$.
- A pickup of crew $i \in J$ cannot start before the completion of servicing job i .
- Each vehicle starts and ends its tour at the depot.
- A vehicle cannot visit the depot mid-route.
- At any given time, each vehicle $r \in V$ is either idle or is traveling between a pair of locations.
- Each vehicle $r \in V$ can carry at most D crews at a time.
- Delivery and pickup times are negligible.
- Servicing jobs and vehicle travel cannot be interrupted.

The two minimization objectives are as follows. Let π be a solution to the IVRP-M problem. First objective is the total distance traveled by all vehicles:

$$D(\pi) = \sum_{r=1}^v D_r(\pi), \quad (2)$$

where $D_r(\pi)$ is the distance traveled by vehicle r . This objective is meant to reduce the operational costs of managing the vehicle fleet (*e.g.* fuel costs). The second objective is the total weighted tardiness of all servicing jobs:

$$T(\pi) = \sum_{i=1}^n T_i(\pi). \quad (3)$$

The value $T_i(\pi)$ is the tardiness of job i , defined as:

$$T_i(\pi) = \max\{0, C_i - e_i\}, \quad (4)$$

where $C_i(\pi)$ is the completion time of job i in solution π . Thus, the tardiness of job i is zero, if i has completed no later than its window closing time (completion on time). Otherwise, the tardiness of i is the difference between its completion time and its window closing time (late completion). This objective is meant to reduce penalties (*e.g.* loss of reputation) incurred by forcing the customers to stay offline longer.

Before moving on, we will discuss two properties that directly follow from problem constraints described above. First, it is possible for a crew that was delivered to the location of its servicing job by one vehicle to be then picked up afterwards by a different vehicle. This means that the transport in the problem is intermodal. Second, the problem assumes mixed hard-soft time window. Indeed, a servicing job *cannot* start before its time window starts, thus the window start time is hard. On the other hand, a servicing job *can* complete after its time window ends, but it incurs an objective function penalty, thus the window end time is soft.

Solution representation

In this section we will propose a solution representation for the IVRP-M problem and the methods to compute the objective function value and test the feasibility of the solution.

It should be noted that the IVRP-M problem, similarly to many other optimization problems, can be formulated using mathematical programming models like Linear Programming or Mixed-Integer Linear Programming, with the non-auxiliary decision variables serving as the solution. However, we have knowingly

decided to not use this method, as it is useful mostly with exact solution methods, which we will not be using. Instead, we will use a solution representation based on Delimiters Representation paired with a disjunctive graph.

Delimiters Representation

Delimiters Representation¹ (DR) is a common representation method for VRP problems and their variants (see for example Gokalp & Ugur, 2020). In the case of IVRP-M, DR is a sequence $\pi = (\pi(1), \pi(2), \dots)$, where each element is a location. Each element $\pi(j)$ falls into one of the following three categories:

1. $\pi(j) = i$, such that $i \in J$.
2. $\pi(j) = -i$, such that $i \in J$.
3. $\pi(j) = 0$.

Value $-i$ represents delivery of a crew at the location of job i (vehicle load decreases). Similarly, value i indicates a pickup of crew at a location of job i (vehicle load increases). Finally, zero indicates a separation between vehicles. It is important to note that $\pi(j) = k$ and $\pi(j+1) = l$ means that some vehicle r travels from location $|k|$ to location $|l|$. If $k = 0$, then r is leaving the depot and if $l = 0$, then r is going to the depot. Value r indicates the current vehicle. At first $r = 1$ and increases by one with each zero in DR, except for the first zero.

For example, consider the following DR

$$\pi = (0, -1, -2, 3, 0, -3, 2, 1, 0) \quad (5)$$

Here we have two vehicles. At first $r = 1$. Vehicle r starts at the depot. Subsequence $(0, -1)$ means that the vehicle travel from the depot to location 1 and delivers a crew there. Then $(-1, -2)$ indicates that r travel from location 1 to 2 for another delivery. Next, r travels to location 3, where it performs a crew pickup. Finally, $(3, 0)$ indicates that r returns to the depot. Since this is a second zero, we have now $r = 2$. The second vehicle leaves the depot, traveling through locations 3, 2 and 1 and returns to the depot, performing one delivery and two pickups in this order. This interpretation of π will be important in the proofs later in this section.

Note that in this example both vehicles first did deliveries and then pickups, similarly to VRP with backhaul variants. However, it is not a requirement: a vehicle may do deliveries-only, pickups-only, interleave pickups with deliveries or even start a route with a pickup.

¹Sometimes referred to as Giant Tour Representations in some publications.

Next, we will introduce the concept of a *well-formed* DR. A DR π is well-formed if and only if π satisfies all of the following conditions:

1. π has exactly one value i for each $i \in J$.
2. π has exactly one value $-i$ for each $-i \in J$.
3. π starts and ends with a zero.
4. π has exactly $v + 1$ zeroes.
5. π has exactly $2n + v + 1$ elements.

Thus, a well-formed DR always contains one delivery and one pickup for each job and does not contain any other symbols besides deliveries, pickups and zeroes. We can now see that example π from (5) is well-formed. In fact, it contains 3 jobs and 2 vehicles, corresponding to the problem instance from Tab. 2 and the motivation example.

Disjunctive graph

Disjunctive graphs are a technique often used with scheduling and similar optimization problems (see for example Błażewicz et al., 2000). Initially, a disjunctive graph represents a problem regardless of any specific solution. It is composed of two types of edges: (1) directed edges (arcs) that represent the inherent constraints of the problem and (2) undirected edges. When a specific solution is introduced, the undirected edges are either directed (turned into arcs) or removed to represent that specific solution. Here we will focus on the disjunctive graph for IVRP-M with a specific solution already in place. The disjunctive graph $G_\pi = (\Phi, \Psi)$ is composed of set of vertices Φ and set of arcs Ψ . Vertices have no weight and represent activities (job start etc.). Arcs are weighted. The longest weighted path to a given vertex will represent the time the activity takes place. For IVRP-M with n jobs, we have $4n + 2$ vertices i.e. $\Phi = \{1, 2, \dots, 4n + 2\}$. There are 5 vertex types:

- Vertices $4(i - 1) + 1$ for $i \in J$ are denoted A_i and represent the arrival of crew at the location of job i .
- Vertices $4(i - 1) + 2$ for $i \in J$ are denoted S_i and represent the start of job i .
- Vertices $4(i - 1) + 3$ for $i \in J$ are denoted C_i and represent the completion of job i .
- Vertices $4i$ for $i \in J$ are denoted R_i and represent the pickup (retrieval) of a crew from the location of job i .
- Vertex $4n + 1$ represents start of the process (time 0) and is denoted 0. There is one such vertex, but for the sake of clarity, it is shown twice in some figures. The arcs are divided into 5 categories:
 - Fixed arcs (A_i, S_i) for $i \in J$ with weight 0. Represent that job cannot be started before crew arrival.
 - Fixed arcs $(0, S_i)$ for $i \in J$ with weight b_i . Represent that job cannot be started before the start of its time window.

- Fixed arcs (S_i, C_i) for $i \in J$ with weight p_i . Represent that job needs to take uninterrupted time from start to completion.
- Fixed arcs (C_i, R_i) for $i \in J$ with weight 0. Represent that the crew cannot be picked up before job completion.
- Disjunctive arcs representing vehicle travel. They are constructed based on π and have weights based on the corresponding $c_{k,l}$.

An example of the disjunctive graph for IVRP-M with 3 jobs and 2 vehicles and DR from (5) is shown in Fig. 1. In this example the path $0 \rightarrow A_1 \rightarrow A_2 \rightarrow R_3$ corresponds to the route of vehicle 1, while the path $0 \rightarrow A_3 \rightarrow R_2 \rightarrow R_1$ corresponds to vehicle 1.

Finally, note that the routes in G_π do not include the return to the depot. The reason for that will be explained later in this section.

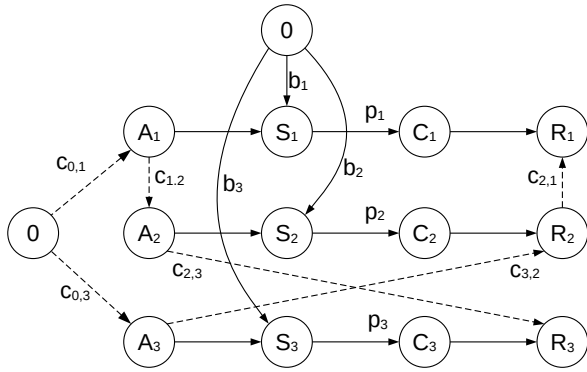


Fig. 1. Example disjunctive graph G_π for problem instance with $n = 3$ and $v = 2$

Objective function and feasibility

We will now discuss how to compute the objective function and check solution feasibility as well as the time complexity of those operations and formalize them using proofs.

We start with some definitions. Let π_r be a subsequence of π pertaining to vehicle $r \in V$. For the example π from (5) we have $\pi_1 = (-1, -2, 3)$ and $\pi_2 = (-3, 2, 1)$. Also, let $len(\pi_r)$ be the length of sequence π_r . Next, we define two functions:

$$init(\pi_r) = \sum_{i=1}^{len(\pi_r)} H(-\pi_r(i)), \quad (6)$$

$$load(\pi_r, i) = init(\pi_r) + \sum_{j=1}^i sgn(\pi_r(j)), \quad (7)$$

where $H(x)$ and $sgn(x)$ are Heaviside step function and sign function, respectively. Thus, $init(\pi_r)$ is the

number of crews vehicle r starts with when leaving the depot. In our example we have $init(\pi_1) = 2$ and $init(\pi_2) = 1$. On the other hand, $load(\pi_r, i)$ is the number of crews in vehicle r after it completes the first i deliveries/pickups in π_r .

Finally, by $D(\pi)$ and $T(\pi)$ we denote the values of the objectives for solution π i.e. the total distance traveled by all vehicles and the total weighted tardiness of all jobs, respectively.

We will now introduce a few Lemmas.

Lemma 1. If the solution π is feasible, then π is well-formed.

Proof. This follows from the well-formedness conditions 1–5 show in Section 4.1. Without conditions 1 or 2 either some crews would not be delivered/picked up or some non-existent crews would be delivered/picked up.

Without condition 3, the first vehicle would not start at the depot or the last vehicle would not end up at the depot. Next, for v vehicles we need $v - 1$ zeroes to delimit them, but due to condition 3, two more zeroes are needed. Thus, two more zeroes are added and condition 4 is necessary.

Thus, conditions 1–4 mean that π has at least $2n + v + 1$ elements. Condition 5 ensures it has no additional elements. ■

Lemma 2. For any feasible solution π the total distance traveled $D(\pi)$ is given by:

$$D(\pi) = \sum_{i=1}^{n+v} c_{|\pi(i)|, |\pi(i+1)|} \quad (8)$$

Proof. Due to interpretation of π , the pairs $(|\pi(0)|, |\pi(1)|)$, $(|\pi(1)|, |\pi(2)|)$, ... describe the subsequent travels (location pairs) done by the vehicles. To compute the value $D(\pi)$ we thus have to sum $c_{|\pi(i)|, |\pi(i+1)|}$ for all such pairs, which is done by formula (8). This assumes that π is well-formed, but this follows from Lemma 1 since π is feasible. ■

Lemma 3. If the solution π is feasible, then the disjunctive graph G_π constructed from π is acyclic.

Proof. The Lemma follows from the meaning of G_π , where arc (i, j) means to represent that event j cannot happen before even i completes. A cycle would thus indicate that some event i cannot happen before i completes, which is a contradiction. ■

It is important to note that Lemma 3 is the reason why G_π does not model the vehicles returning to the depot, as this would create a cycle. This issue can be solved by splitting the 0 vertex into “0-in” and “0-out” vertices, but it is not necessary, as such “incomplete” graph is sufficient as will be shown further in this section. Also, note that for feasible π the graph G_π is a Directed Acyclic Graph (DAG).

As an example of an infeasible solution consider the following π for which G_π is shown in Fig. 2.

$$\pi = (0, -1, -2, 4, -3, 0, -4, 2, 1, 3, 0). \quad (9)$$

For this DR the resulting graph contains a cycle:

$$(A_4, S_4, C_4, R_4, A_2, S_2, C_2, R_2, A_4). \quad (10)$$

Thus, times of delivery and pickup for job 2 are dependent on each other. Similar goes for job 4.

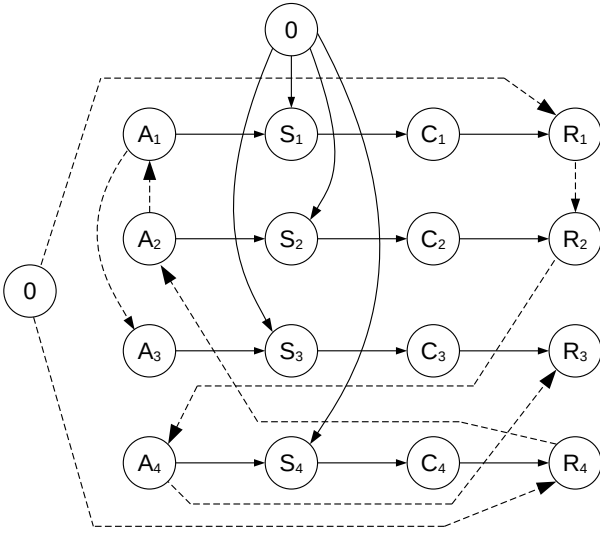


Fig. 2. Example graph for an infeasible solution for an instance with $n = 4$, $v = 2$ and $D = 3$

Lemma 4. For any feasible solution π the total weighted tardiness $T(\pi)$ can be computed using G_π .

Proof. Let $T_i(\pi)$ be the tardiness of job $i \in J$. If job i is completed before the end of its time the tardiness of i is zero. Otherwise, the tardiness of i is the difference between its completion and window end time. Thus:

$$T_i(\pi) = \max\{0, C_i - e_i\}. \quad (11)$$

Due to the structure of G_π , the value C_i is the length of the longest (weighted) path in G_π that ends at vertex C_i . Such a path always exists as long as G_π is a DAG, which is ensured by feasibility of π and Lemma 3. Value $T(\pi)$ is then computed as: Thus:

$$T(\pi) = \sum_{i=1}^n w_i T_i. \quad (12)$$

We can now move onto the main results of this section in the form of the following three Theorems. We will start with establishing the conditions sufficient for confirming the feasibility of π .

Theorem 1. Let π be a solution to IVRP-M. If the following conditions are met, then π is feasible:

1. π is well-formed.
2. G_π is a DAG.
3. $init(\pi_r) < D$ for all $r \in V$.
4. $load(\pi_r, i) < D$ for all $r \in V$ and $i \in \{1, 2, \dots, len(\pi_r)\}$.

Proof. Solution π is feasible if it meets all of the assumptions A–J from Section 3.

Assumption A is met since a well-formed DR ensures all vehicles start at the depot. Assumptions B–D are met due to the structure of G_π and it being acyclic (Lemma 3). Assumptions E and F are met due to well-formedness of π . Assumption G follows from the structure of G_π as the vehicle never stops, except for when it has to wait during pickup. Assumption H is met due to $init(\pi_r) < D$ and $load(\pi_r, i) < D$, meaning that vehicles are never overloaded, including when starting from the depot. Assumption I is met because vertices A_i , R_i and arcs (A_i, S_i) and (C_i, R_i) have weight of 0. Finally, assumption J is met by the structure of G_π . ■

Note that this means that the solution from (9) and Fig. 2 is infeasible for another reason: after picking the servicing crew from job 4, vehicle 2 carries 4 crews, exceeding its assumed capacity of 3.

Theorem 2. For any solution π , the feasibility of π can be verified in time $O(n)$.

Proof: First, from (1) it follows that $v \in O(n)$ and thus the length of π is $n + v + 1 \in O(n)$. Second, checking the conditions 1–4 from Theorem 1 is sufficient to ensure the feasibility of π .

Checking for well-formedness of π takes time $O(n)$ as it consists of counting the number of specific element types, the length of the sequence and checking its first and last element.

Checking if G_π is a DAG can be done in time $O(n)$ as well, since the existence of a cycle can be detected by the Kahn's algorithm.

Values $init(\pi_r) < D$ for all $r \in V$ are trivially determined by scanning π once in $O(n)$.

Finally, the value $load(\pi_r, i)$ can be computed recursively:

$$load(\pi_r, i) = load(\pi_r, i - 1) + \text{sgn}(\pi_r(i)), \quad (13)$$

$$load(\pi_r, 0) = init(\pi_r). \quad (14)$$

Thus, $load(\pi_r, i)$ can be computed in time $O(n)$. ■

Theorem 3. For any feasible solution π , the objective function value $(D(\pi), T(\pi))$ can be determined in time $O(n)$.

Proof: Once again we will use the fact that the length of π is $O(n)$. Since π is feasible, we can use formula (8) from Lemma 2 to compute $D(\pi)$ in time $O(n)$. To compute $T(\pi)$ we first need to construct the disjunctive graph G_π and find the longest paths to all vertices C_i . Since the number of vertices and arcs in G_π is $O(n)$, G_π can be constructed from π in time $O(n)$. Since G_π is a DAG (Lemma 3), the Kahn's algorithm can be used to find (any) topological order in G_π and topological sorting can be used to find all longest paths in time $O(n)$. Since π is feasible, we can use Lemma 4 and formulas (11)–(12) to compute $T(\pi)$ in time $O(n)$. Thus, the final time complexity of determining the objective function value is $O(n)$. ■

An example of a complete schedule for the instance from Tab. 2, the motivation example and DR from (5) is shown in Fig. 3. The distances traveled by vehicles are 12 and 15 respectively for total $D(\pi) = 27$. Jobs 1 and 3 are completed within their time windows and thus have tardiness equal 0. Job 2 on the other hand has tardiness of 1. The total weighted tardiness for this solution is $T(\pi) = 0+1\cdot 8+0 = 8$. The bi-objective function value in this case is thus $(D(\pi), T(\pi)) = (27, 8)$.

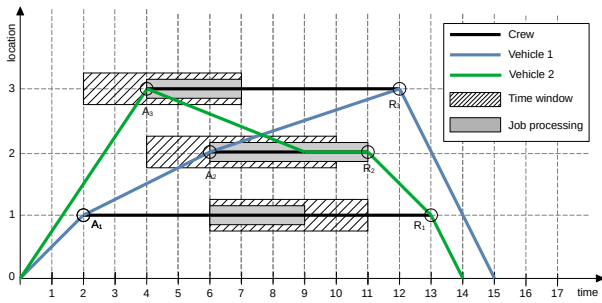


Fig. 3. Gantt schedule for problem instance shown in Tab. 2 and solution from (5)

Let us also note that for job 1 the window is not yet open by the time the crew is delivered. The reverse happens for jobs 2 and 3, as the jobs has to wait for the delivery of the crew, despite the window being started already. For job 2 the vehicle arrives to retrieve the crew before the job is completed and thus has to wait. The reverse happens for jobs 1 and 3 – the crews have already completed the job and have to wait to be picked up by the vehicle.

As a final theoretical result, we consider the feasibility of the IVRP-M problem as a whole (as opposed to feasibility of a single solution π). Due to its simplicity, we state it as a Proposition.

Proposition 1. Let I be an instance of the IVRP-M problem. If $vD \geq n$ then a feasible solution π exists and can be constructed in time $O(n + v)$.

Proof: If $vD \geq n$ then there is enough vehicles to board all servicing crews. We construct π such that crews 1 through D are placed in vehicle 1, crews $D + 1$ through $2D$ are placed in vehicle 2 and so on. For a given vehicle we first deliver all crews and then pick them all up. For $n = 5$ and $D = 2$ the result is:

$$\pi = (0, -1, -2, 1, 2, 0, -3, -4, 3, 4, 0, -5, 5, 0). \quad (15)$$

This π is created in time $O(n + v)$. The resulting graph G_π is acyclic, because we always cover “leftmost” vertices A_i before “rightmost” vertices D_i . We thus move left-to-right and no cycle can exist. The other 3 feasibility conditions from Theorem 1 are trivially met. ■

Solution methods

The solution space for the IVRP-M problem is large. For example, for 5 jobs and 3 vehicles, there are nearly 500 million possible DRs (including infeasible ones). Compare this with a regular VRP with 5 customers and 3 vehicles, where there are only around 5000 DRs. Moreover, due to multi-objective nature a straightforward approach to a Mixed-Integer Linear Programming (MILP) is difficult, as it requires running multiple MILP for subproblems with one objective fixed and the final optimality relies on the density of the resulting subproblem “grid”. Similarly, Branch-and-bound (BnB) approach requires sophisticated approach to lower and upper bounds calculation to be viable for anything but the smallest instances. As such exact methods require an approach that is beyond the scope of this paper.

Since this makes exact solution methods impractical for real-life problem instances, we focus on inexact approaches described below with a heuristic method used as a baseline. Note that the results of Theorems 1 through 3 are used to implement the procedures for checking solution feasibility and determining the objective function values which are used with the solution methods.

The first method is a deterministic greedy heuristic with a predictable $O(n^2(n + v))$ running time. While viable as a baseline, it is expected to converge fast to a local optimum. The second method is a deterministic Tabu Search (TS) metaheuristic that is able to avoid local optima due to its short-term memory. The third method is the Genetic Algorithm (GA), which is able to consider multiple solutions at once, but suffers from both its stochastic nature and challenges in constructing feasible offspring. The conceptual differences between algorithms are shown in Fig. 4.

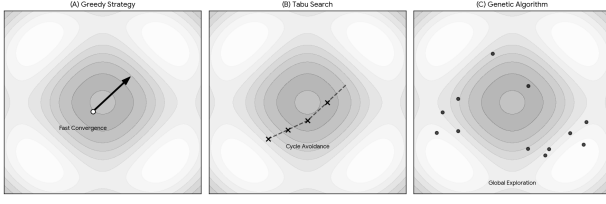


Fig. 4. Conceptual comparison of the three optimization strategies: (A) Greedy method algorithm focusing on immediate exploitation, (B) Tabu Search using memory to escape local optima; (C) Genetic Algorithm maintaining a diverse population for global exploration.

Greedy heuristic

Greedy algorithms are often simple and deterministic, while providing good quality for many problems. In our case the greedy method serves three purposes: (1) a standalone algorithm, (2) a reference algorithm, (3) initial solution for TS and GA methods. The greedy method is similar to the NEH algorithm known from scheduling (Kalczyński & Kamburowski, 2009) and its pseudocode is shown in Algorithm 1.

Algorithm 1. Greedy heuristic pseudocode

Input: problem instance.

Output: Pareto front approximation.

```

1  jobs ← (); front ← (); π ← emptyDR(v)
2  for i in 1 to n do
3      jobs.add(-i, i)
4  sortByDeadlines(jobs)
5  foreach job in jobs do
6      if lastJob then:
7          for p in 2 to π.length()-1 do
8              π.insert(job, p)
9              if π.feasible() then
10                 front.update(π)
11             end
12             π.remove(p)
13     else
14         cand ← ()
15         for p in 2 to π.length()-1 do
16             π.insert(job, p)
17             if π.feasible() then
18                 cand.add(π)
19             end
20             π.remove(p)
21         end
22         π.insert(job, topsis(cand))
23     end
24 end
25 return front
    
```

First, in lines 1–4 we prepare DR filled with $v + 1$ zeroes (empty vehicles), empty Pareto front and all jobs (both deliveries and pickups) sorted in descending order of their window end times e_i . Next, for each job (line 5) we proceed depending on whether it is the last job to be inserted or not. For the last job (lines 6–12) we tentatively insert the job into each position in π that is feasible (lines 7–9) and remove the job afterwards (line 12). Since each resulting tentative π is complete, we use it to try to update the current Pareto front approximation (line 10). Solution π is added to the front only if no solution in the front dominates it. All solutions currently in the front that are dominated by π are also removed from the front. A solution π_1 dominates solution π_2 if and only if π_1 is not worse than π_2 on any objective and is better on at least one. Thus, the front is made from non-dominated solution.

For other jobs (lines 13–22) we proceed similarly, except for each tentative feasible solution π we add it to the list of candidates (line 18). After all candidates are determined, we have to choose the final position for that job (line 22). In single-objective optimization it would be straightforward, but not for multiple objectives. The use of the domination relation is also not ideal, as neither solution might dominate the other. To solve this we use the well-known TOPSIS method (Hwang & Yoon, 1981), which ranks the solutions.

Tabu Search

TS is a deterministic local search method that has been successfully applied to many optimization problems including VRP (Yue et al., 2024) and scheduling (Idzikowski et al., 2021). The pseudocode of our implementation is shown in Algorithm 2.

In line 1 we obtain the initial front from the greedy algorithm (Algorithm 1) and choose the best (TOPSIS-wise) of that front as our initial solution π . This also means that TS cannot obtain worse results than the greedy method. In line 2 tabu list is prepared as a matrix of size $v + 1 \times v + 1$, initially filled with zeroes and iteration counter is set to 0.

Afterwards TS proceeds iteratively until time limit is reached (line 3). In each iteration we consider a neighborhood $N(\pi)$ of the current solution π . To this end, we define a move $swap(\pi, i, j)$, which swaps elements i and j in solution π . Next, $N(\pi)$ is set of solutions created from π by swapping any elements of π except the first and last:

$$N(\pi) = \{swap(\pi, i, j) : 1 < i < j \leq n + v\} \quad (16)$$

Note that $swap(\pi, i, j) = swap(\pi, j, i)$, thus the condition $j > i$ prevents duplicating neighbors. The basic neighborhood has $O((n + v)^2)$ solutions.

The neighborhood is evaluated in lines 5–13. In line 8 neighbors that are infeasible or forbidden (tabu) are ignored. Similarly, *neighborhoodWidth* parameter is used so only given percentage of neighbors is checked (this is based on random numbers, but with fixed seed, keeping the algorithm deterministic). Tabu list element $[i, j]$ keeps the number of iteration at which move $swap(\pi, i, j)$ becomes available, thus allowing for tabu check in time $O(1)$ in line 8.

In lines 14–18 the best candidate (TOPSIS-wise) is chosen as the next solution. The tabu list is updated based on the *cadence* parameter. The Pareto front and iteration number are updated as well.

Algorithm 2. Tabu Search pseudocode

Input: problem instance I, parameters
(*timeLimit*, *neighborhoodWidth*, *cadence*).

Output: Pareto front approximation.

```

1  front ← greedy(I);  $\pi \leftarrow \text{topsis}(\text{front})$ 
2  tabu ← prepareTabuList(); it ← 0
3  while timeLimit not reached do
4    cand ← ()
5    for i in 2 to  $\pi.\text{length}()-1$  do
6      for j in i+1 to  $\pi.\text{length}()-1$  do
7         $\pi.\text{swap}(i, j)$ 
8        if  $\text{rand}(100) \leq \text{neighborhoodWidth}$  and
            $\pi.\text{feasible}()$  and  $\text{it} \leq \text{tabu}[i, j]$ 
9          cand.add( $\pi$ )
10         end
11          $\pi.\text{swap}(i, j)$ 
12       end
13     end
14      $i, j \leftarrow \text{topsis}(\text{cand})$ 
15      $\pi.\text{swap}(i, j)$ 
16      $\text{tabu}[i, j] \leftarrow \text{it} + \text{cadence}$ 
17     front.update( $\pi$ )
18     it ← it+1
19   end
20   return front

```

Genetic Algorithm

GA is a popular population-based metaheuristic that employs a process similar to biological evolution. GA was applied to optimization problems like VRP (Idzikowski et al., 2024). The pseudocode of GA is shown in Algorithm 3.

The algorithm first creates an empty Pareto front and an initial population of size defined by parameter *populationSize* (lines 1–5). Parameter *initialType* determines the type of solution in the initial population with 3 options: (1) random feasible solutions, (2) random solutions obtained through greedy method and

then diversified by applying $0.2n$ random moves (the same as for the TS method), (3) the same as before, but one set of solutions from the greedy is kept without the diversification. The main loop of GA (lines 6–33) continues till a defined *timeLimit* is reached.

Algorithm 3. Genetic Algorithm pseudocode

Input: problem instance I, parameters
(*timeLimit*, *populationSize*, *crossoverProb*,
mutationProb, *elitism*, *initialType*).

Output: Pareto front approximation.

```

1  pop ← (); front ← ()
2  thres ← round(elitism * populationSize)
3  for i in 1 to populationSize:
4    pop.add(createSpecimen(initialType))
5  end
6  while timeLimit not reached do
7    par ← (); chld ← ()
8    ranked ← rankByTOPSIS(pop); pop ← ()
9    for i in 1 to 2*populationSize:
10     par.add(tournament(ranked))
11   end
12   for i in 1 to populationSize:
13     if rand() < crossoverProb:
14       chld.add(OX(par[2i], par[2i+1]))
15       chld.add(OX(par[2i+1], par[2i]))
16     else:
17       chld.add(par[2i], par[2i+1])
18   end
19   end
20   for i in 1 to populationSize:
21     if rand() < mutationProb:
22       chld[i].mutateByRandomSwap()
23   end
24   end
25   rankedChld ← rankByTOPSIS(chld)
26   for i in 1 to populationSize:
27     if i ≤ thres:
28       pop.add(ranked[i])
29     else
30       pop.add(rankedChld[i-thres])
31   end
32   end
33   end
34   return front

```

In the loop we prepare empty lists of parents and children (line 7). Next, we perform parent selection (lines 8–11) by ranking solutions with TOPSIS and then selecting *populationSize* pairs of parents through a tournament selection — $\text{round}(\sqrt{n})$ solutions are chosen randomly and the best-ranked becomes the parent.

The population of children via crossover is performed in lines 12–19. The crossover happens for each parent pair with probability *crossoverProb*. If crossover is not applied, the parents become the offspring. If the crossover is applied, it is performed similarly to the well-known OX crossover operator. A random section of π from the first parent is copied to the first offspring. Then the not-yet inserted jobs are inserted to the offspring in the same order as they appear in the second parent. Feasibility is checked after each insertion. If the solution is infeasible, we try to insert the job into an empty vehicle. If the solution is still infeasible or there are no remaining vehicles, we act as if the crossover did not happen.

Mutation (lines 20–24) is implemented by performing a random TS move with probability *mutationProb*.

Finally, after each generation the new population needs to be constructed (lines 25–32). First, the best (TOPSIS-wise) *elitism* percentage of parents are chosen for the new population. The remaining population is created from the best children.

Note that each time a new solution is obtained (during crossover, mutation or initial population preparation) it is used to update the Pareto front.

Computer experiments

In this section we provide the results of computer experiments, starting with assumptions about problem instances, equipment and software and calibration of the solution methods. The code, and instances and results for the main research are available on a Github repository ([Rudy, 2026](#)).

Experimental setup

All algorithms were implemented in C++ 11.4.0. All experiments were performed on a DGX machine with AMD EPYC 7742 3.2 GHz processor with 64 physical (128 logical) cores and 503 GiB of RAM running under Ubuntu Linux 22.04.5 LTS (kernel version 5.15). Since GA is a probabilistic method, the experiments with it were conducted multiple times (5 in preliminary research and 11 otherwise) for each instance and the median run was reported. Moreover, for multi-objective optimization it was observed [Rudy & Zelazny \(2015\)](#) that the Pareto front created from a sum of two metaheuristic can be significantly better than the fronts for the individual algorithms. We have applied this to create algorithm denoted GATS.

Regarding problem instances, we prepared a method for generating random instances. Let $x \sim U(a, b)$ denote that x is a realization of random variable from

uniform distribution from a (inclusive) to b (inclusive). We assume $b_i \sim U(0, 240)$, $e_i \sim U(b_i + 60, b_i + 180)$ *i.e.* window length of 60 to 180, $w_i \sim U(1, 10)$, $d_i \sim U(30, 150)$. Finally, $p_i \sim U(1, \lfloor (e_i - b_i)R + 0.5 \rfloor)$, where R is a fraction of window length. For example, for window length 60 and $R = 0.7$ we have $p_i \sim U(1, 42)$.

The above are common parameters for all instances. Specific instances are obtained by providing specific values n , vf , D , R and the seed for the random number generator. We assumed:

- $n \in \{10, 15, 20, \dots, 30, 35, 40\}$,
- $vf, R \in \{0.2, 0.3, 0.4, \dots, 0.7, 0.8\}$,
- $D \in \{2, 3, 4, 5\}$.

Value vf is factor for computing v using the formula $v = \lfloor n \cdot vf + 0.5 \rfloor$. This yields 1372 parameter combinations. For each combination we attempted to generate 10 instances. However, each time we checked if the greedy algorithm can find a feasible solution for that instance. If yes, the instance was added to the benchmark dataset and was discarded otherwise. This way 9765 instances in total were generated for use in the primary experiments.

Measure of quality

The measure of quality for comparing algorithms is established as follows. First, assume problem instance I . Each of the 4 algorithms is run for this instance, obtaining 4 Pareto front approximations. Then for each front we calculate a Hypervolume Indicator (HVI). HVI is calculated as an area of a figure bound by solutions from the front and a nadir point Z . For two objectives, that figure is a 2D shape. Point Z is usually defined by the worst value of each objective among *all* compared fronts multiplied by some factor. A common value is 1.2, which is also true in the case of this paper. An example of HVI for two Pareto front approximations is shown in Fig. 5. The blue front has bigger area and thus higher HVI.

For instance I we thus obtain four HVI values denoted $HVI_{\text{greedy}}(I)$, $HVI_{\text{TS}}(I)$, $HVI_{\text{GA}}(I)$ and $HVI_{\text{GATS}}(I)$ respectively. We will now define relative measures by using greedy algorithms as a reference *i.e.* for algorithm $A \in \{\text{TS}, \text{GA}, \text{GATS}\}$ and problem instance I we define $RHVI_A(I)$ as:

$$RHVI_A(I) = \frac{HVI_A(I)}{HVI_{\text{greedy}}(I)}. \quad (17)$$

This value is defined for a single instance I , but we will report aggregated values of $RHVI_A(I)$ as an average over multiple instances. As said earlier, the median run of GA is used for $HVI_{\text{GA}}(I)$.

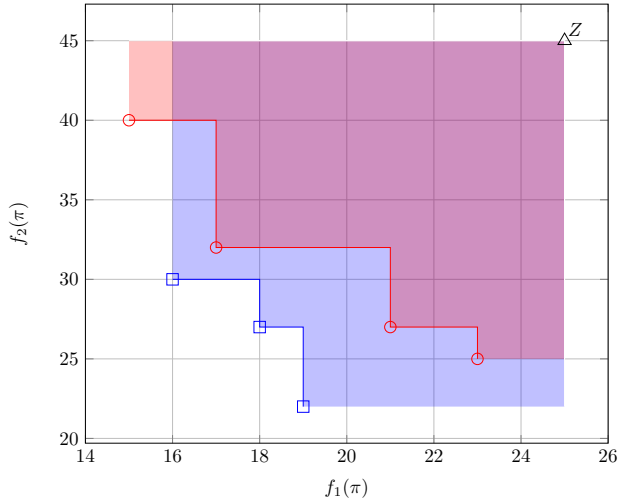


Fig. 5. Visualization of HVI calculation for two fronts with 4 (red) and 3 (blue) solutions for two objectives

Preliminary research

A separate smaller instance dataset was used to fine-tune the parameters for TS and GA methods. For TS we tested various neighborhood widths and three types of cadence: linear in the size of π (i.e. $f(n+v)$), square root in the size of π (i.e. $f\sqrt{n+v}$) and constant (f), each with several different factors f considered. For GA we tested different population sizes, three methods of creating the initial population (as described in Section 5.3), and several different values for crossover and mutation probabilities as well as elitism size. The parameter values chosen for the primary research are shown in Table 3.

Table 3
Algorithm parameters

Parameter	Value
Tabu Search	
<i>timeLimit</i>	$[0.5n + 0.5]$
<i>neighborhoodWidth</i>	0.10
<i>cadence</i>	$[2(n+v+2) + 0.5]$
Genetic Algorithm	
<i>timeLimit</i>	$[0.5n + 0.5]$
<i>populationSize</i>	50
<i>initialType</i>	greedy plus worsened greedy
<i>crossoverProb</i>	1.00
<i>mutationProb</i>	0.15
<i>elitism</i>	0.21

Main results

The results for the main experiments for TS, GA and GATS algorithms are shown in Tab. 4, with each row corresponding to aggregated $RHVI_A(I)$ for instance of a given size. Additional rows represent minimum, average and maximum $RHVI_A(I)$ over all instances as well as standard deviation. Since GA and GATS are stochastic methods, we also included 95% confidence intervals ($\alpha = 0.05$) for those methods, calculated as pooled standard deviation.

Table 4
Aggregated $RHVI$ for tested algorithms

n	TS	GA	GATS
10	1.759	2.232 ± 0.076	2.689 ± 0.047
15	1.877	1.946 ± 0.067	2.828 ± 0.048
20	1.934	1.837 ± 0.059	2.882 ± 0.044
25	1.852	1.722 ± 0.051	2.759 ± 0.041
30	1.768	1.691 ± 0.047	2.622 ± 0.039
35	1.736	1.681 ± 0.035	2.559 ± 0.029
40	1.723	1.650 ± 0.042	2.513 ± 0.039
Min	1.000	1.000	1.000
All	1.807	1.823 ± 0.055	2.694 ± 0.041
Max	16.037	15.667	16.072
St. dev.	1.149	1.292	1.719

First, we observe that the effectiveness of TS (its advantage over the greedy method) remains mostly stable across all number of jobs, while GA gets progressively worse. GA appears to obtain better HVIs for instances up to 15 jobs, but for larger instances TS starts to outperform it. Regardless, both algorithms on average obtain 1.7 to 2.2 times higher HVIs than the greedy method, implying that on average each objective has 30 to 50% better value.

Even more interestingly, the GATS algorithm consistently outperforms both TS and GA, obtaining nearly 50% higher HVIs than both algorithms. GATS also gets slightly lower results as n increases, but it still remains ahead of the other methods. An illustrative examples of the front approximations for two instances for TS and GA algorithms are shown in Figs. 6 and 7. These figures showcase the general trend, as TS usually obtains smaller total distance, but higher total weighted tardiness. Moreover, fronts found by TS usually have at most 3 solutions and often have just one. On the other hand, fronts obtained by GA usually have at least 5 solutions. This matches the characteristics of both methods (local search for TS and population-

based for GA), explains the effectiveness of GATS and showcases how useful combining metaheuristics can be for multi-objective optimization problems.

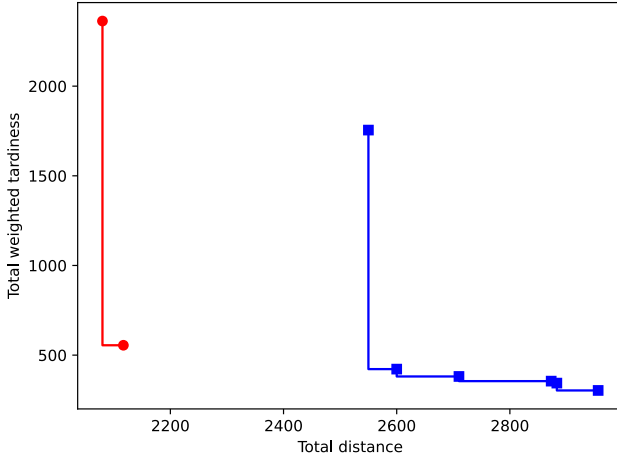


Fig. 6. The example TS (red) and GA (blue) fronts for $n = 20$, $v = 10$, $D = 5$, $R = 0.8$

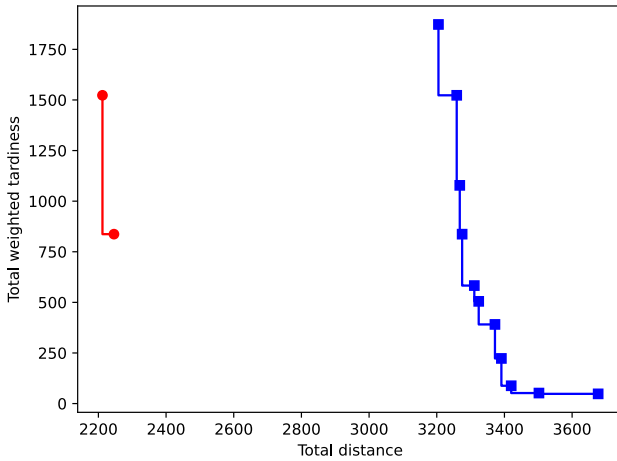


Fig. 7. The example of TS (red) and GA (blue) front for $n = 25$, $v = 10$, $D = 5$, $R = 0.7$

Next, from the last four rows of Table 4 we observe there are cases where neither algorithm improves the greedy method (minimum row), although this is expected for some small instances. The maximum improvement for all algorithm is at impressive 15-16 times better than the greedy method. Standard deviation shows that TS is more stable than GA, while GATS is least stable by far.

Lastly, concerning the stability of the stochastic methods, we note that both GA and GATS are very robust, with 95% confidence intervals below 0.08 for all size groups. Thus, small number of trial is enough to obtain the reported results with good confidence. We also observe that the robustness generally improves

with instance size and that it is considerably higher for GATS than for GA. Since this result applies to all further research, we omit it in the following subsections.

Sensitivity analysis

As further research, we analyzed the previous results to establish how instance parameters v , D and R affect the quality of solutions. We will thus choose a parameter value, for example $R = 0.8$ and report results aggregated over all instances with that parameter value. However, this brings one change to how $RHVI_A(I)$ is calculated. Previously, each instance I had 4 fronts (greedy, TS, GA and GATS). Now, for the analysis of R we instead have 28 fronts (4 algorithms times 7 values of R). Since HVI values depend on the set of fronts, they have to be recalculated. Thus, the smallest HVI out of those 28 HVIs will be the normalization basis for $RHVI_A(I)$. This means some greedy algorithms can now obtain HVI higher than that smallest value and thus $RHVI_{greedy}(I)$ is now included in the results.

The results for the analysis of the R parameter are shown in Table 5. All algorithms experience loss in quality as R (job servicing time relative to the length of the window) increases. This loss is the largest for the greedy and GA methods and the smallest for GATS. However, the loss stays in range 20-30% despite R increasing four times. We conclude that the influence of this parameter on solution quality is moderate. Moreover, the algorithms now appear clearly ranked with GATS being the best, followed by TS, GA and finally the greedy method.

Table 5
Aggregated $RHVI$ for different values of R

R	Greedy	TS	GA	GATS
0.2	2.381	3.075	2.643	3.329
0.3	2.216	2.974	2.485	3.239
0.4	2.068	2.863	2.340	3.129
0.5	1.955	2.762	2.230	3.055
0.6	1.832	2.649	2.114	2.951
0.7	1.694	2.507	1.971	2.808
0.8	1.648	2.327	1.838	2.665
All	1.956	2.737	2.232	3.025
Loss	30.79%	24.33%	30.45%	19.95%

The final analysis concerns the parameters vf and D . Since those parameters are tied together, we analyzed them together with the results presented in Tab. 6. For $D = 2$ and $D = 4$ the increase in vf (the number of

vehicles relative to the number of jobs) does not seem to affect the quality of solutions. For the cases of $D = 3$ and $D = 5$ the increase is much more noticeable, but it mainly occurs for small values of vf and remains stable afterwards. All of this indicates that the algorithms are robust and are able to provide similar solutions even when relatively low number of vehicles is available.

In this analysis the relations between algorithms change a little, as GA seems to struggle to outperform the greedy method by more than 10%. Similarly, the solutions provided by TS are almost as good as those obtained by GATS. This indicates that the front provided by GATS is based mostly on TS with minor influence from GA. As a final remark, we observe that while TS and GA benefit from their cooperation, TS seems to be better overall. It managed to outperform GA consistently in both sensitivity analysis scenarios, while still providing comparable results during the main experiment.

Table 6
Aggregated *RHVI* for different pairs of D and vf

D	vf	Greedy	TS	GA	GATS
2	0.5	1.768	2.767	1.943	2.816
	0.6	1.771	2.792	2.009	2.868
	0.7	1.756	2.794	2.023	2.869
	0.8	1.747	2.803	2.025	2.867
3	0.3	2.003	2.475	2.182	2.700
	0.4	2.272	3.131	2.464	3.279
	0.5	2.243	3.222	2.479	3.355
	0.6	2.254	3.213	2.503	3.364
	0.7	2.237	3.246	2.499	3.359
	0.8	2.287	3.242	2.529	3.362
4	0.3	2.541	3.138	2.707	3.340
	0.4	2.588	3.264	2.816	3.493
	0.5	2.571	3.311	2.825	3.536
	0.6	2.564	3.294	2.800	3.518
	0.7	2.577	3.245	2.794	3.482
	0.8	2.554	3.272	2.785	3.449
5	0.2	1.541	2.431	1.705	2.951
	0.3	2.726	3.234	2.936	3.488
	0.4	2.755	3.235	3.020	3.553
	0.5	2.777	3.302	3.023	3.592
	0.6	2.747	3.253	3.000	3.559
	0.7	2.798	3.299	3.026	3.569
	0.8	2.764	3.269	2.990	3.514

Discussion and managerial insights

In this section we provide a brief discussion regarding the strengths, limitations and use of proposed methods in the decision-making process in practical applications to vehicle and crew routing in servicing, maintenance and similar real-life systems.

Observing the results of the experiments we notice an important difference between TS and GA methods. TS usually obtains better solutions (i.e. fronts closer to the true Pareto-optimal front) and is deterministic with good convergence rate. It is thus preferable for online systems where quick decisions are necessary. The fronts found by TS usually have smaller number of solutions, thus making the decision-making process easier. On the other hand, GA has been proven to be highly robust despite its stochastic nature, meaning that it can be run only once or twice and still provide good results with relatively high confidence. Moreover, while weaker on average, GA generally finds fronts with higher number of solutions. This can also be beneficial, as it provides the decision-makers with more options and is thus suitable for a wider range of scenarios and trade-offs. Finally, we observe that both methods tend to search different fragments of the solutions space. Thus, combining their results increases the stability and enriches the obtained Pareto fronts with a small increase in the computation time.

Another practical aspect is intermodality. Unlike many other features (like vehicle capacity and time windows) it is usually possible in practice to implement it instead of requiring picking up crews to be done by the same vehicle that delivered the crew. It is obvious that allowing intermodality will not worsen the results, but it is less obvious which particular scenarios benefit from it. One possible case is when the customers are isolated from each other, instead of being clustered in a few groups. In such a case a vehicle either has to wait for the service to be finished to pick up the crew (thus wasting vehicular resources) or has to risk moving away from the delivery location and having to cover a lot of distance for the pickup.

One possible way to solve this issue is to increase the number of vehicles or crews. However, the research conducted in this paper suggest that with good algorithms the results do not improve significantly when increasing the number of vehicles. This means the practitioners should be careful with increasing the vehicle fleet size, as it will always incur cost, but does not guarantee improvement in solution quality. The proposed algorithms can be used to assess whether a higher number of resources leads to any significant benefit. Moreover, while this paper discusses the min-

imal number of vehicles necessary for there to be at least one feasible solution, the result applies only to the IVRP-M. Separate studies are necessary in the case of more complex problem (see for example Radzki et al., 2025b).

Lastly, it should be noted that the proposed methods, while focused on optimization and feasibility, could be also used to estimate the answer to questions such as “is it possible to obtain solution where the objective function is not worse than X”. The objective function can also be modified to cut off or count values that are too large, helping practitioners with figuring out whether the obtained fuel costs and tardiness are acceptable in their business model.

Conclusions and future work

In this paper, an optimization problem of planning deliveries and pickups of crews for the purposes of the maintenance of machines in manufacturing systems was considered. The problem assumed hard-soft time windows, intermodal transport and two-objective objective function of minimizing the distance traveled by all vehicles and total weighted tardiness of all servicing jobs.

A problem formulation and solution representation based on the Delimiters Representation and a disjunctive graph was proposed. Problem properties regarding the objective function and solution feasibility were discussed and proven through a series of theorems.

Four solution methods were proposed (1) a reference greedy algorithm, (2) Tabu Search local search metaheuristic, (3) Genetic Algorithm population metaheuristic and (4) a mixture of Tabu Search and Genetic Algorithm, which worked by combining the Pareto front approximations obtained by both solution methods.

A benchmark dataset of 9765 random problem instances were prepared to verify the properties of the problem and the effectiveness of the algorithms. Tabu Search algorithm proved better than the Genetic Algorithm in most scenarios and both considerably outperformed the greedy method. Moreover, the combined Tabu Search-Genetic Algorithm technique proved to be the best, showcasing that both metaheuristics complement each other by exploring different parts of the Pareto set. Additional experiments were performed to test the sensitivity of the algorithms to scenarios like variable number of vehicles. The results showcased that the proposed algorithm are robust, providing good results even with low number of vehicles and relatively small time window length. Regarding future

research, we consider several method for improvement and extension. Firstly, a more sophisticated method of combining the Pareto fronts of metaheuristic could be established. Secondly, an exact algorithm such as Mixed-Integer Linear Programming or Branch-and-Bound method could be proposed for the considered problem. Finally, the benchmark dataset could be improved by taking real-life data into account.

References

- Achouch M., Dimitrova M., Ziane K., Sattarpanah Karganroudi S., Dhouib R., Ibrahim H., & Adda M. (2022). On predictive maintenance in industry 4.0: Overview, models, and challenges. *Applied sciences*, 12(16), 8081, DOI: [10.3390/app12168081](https://doi.org/10.3390/app12168081).
- Biteus J., & Lindgren T. (2017). Planning flexible maintenance for heavy trucks using machine learning models, constraint programming, and route optimization. *SAE International Journal of Materials and Manufacturing*, 10(3), 306–315, <https://jstor.org>.
- Błażewicz J., Pesch E., & Sterna M. (2000). The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2), 317–331, DOI: [10.1016/S0377-2217\(99\)00486-5](https://doi.org/10.1016/S0377-2217(99)00486-5).
- Gokalp O., & Ugur A. (2020). A multi-start ILS-RVND algorithm with adaptive solution acceptance for the CVRP. *Soft Computing*, 24(4), 2941–2953, DOI: [10.4271/2017-01-0237](https://doi.org/10.4271/2017-01-0237).
- Hwang C.L., & Yoon K. (1981). Multiple attribute decision making: methods and applications: a state-of-the-art survey, DOI: [10.1007/978-3-642-48318-9](https://doi.org/10.1007/978-3-642-48318-9).
- Idzikowski R., Rudy J., & Gnatowski A. (2021). Solving non-permutation flow shop scheduling problem with time couplings. *Applied Sciences*, 11(10), 4425, DOI: [10.3390/app11104425](https://doi.org/10.3390/app11104425).
- Idzikowski R., Rudy J., & Jaroszczuk M. (2024, June). Solving a vehicle routing problem for a real-life parcel locker-based delivery. In *International Conference on Dependability of Computer Systems* (pp. 69–79). Cham: Springer Nature Switzerland, DOI: [10.1007/978-3-031-61857-4_7](https://doi.org/10.1007/978-3-031-61857-4_7).
- Jie K.W., Liu S.Y., & Sun X.J. (2022). A hybrid algorithm for time-dependent vehicle routing problem with soft time windows and stochastic factors. *Engineering Applications of Artificial Intelligence*, 109, 104606, DOI: [10.1016/j.engappai.2021.104606](https://doi.org/10.1016/j.engappai.2021.104606).

- Jitkongchuen D., Pawapootanont P., & Thusaranon P. (2023, March). Applying a saving algorithm for vehicle routing problems to optimize the maintenance route of the transmission station. In *2023 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON)* (pp. 452–456). IEEE, DOI: [10.1109/ECTIDAMT-NCON57770.2023.10139534](https://doi.org/10.1109/ECTIDAMT-NCON57770.2023.10139534).
- Jolfaei A.A., & Alinaghian M. (2024). Multi-depot vehicle routing problem with roaming delivery locations considering hard time windows: Solved by a hybrid ELS-LNS algorithm. *Expert Systems with Applications*, 255, 124608, DOI: [10.1016/j.eswa.2024.124608](https://doi.org/10.1016/j.eswa.2024.124608).
- Kalczynski P.J., & Kamburowski J. (2009). An empirical analysis of the optimality rate of flow shop heuristics. *European Journal of Operational Research*, 198(1), 93–101, DOI: [10.1016/j.ejor.2008.08.021](https://doi.org/10.1016/j.ejor.2008.08.021).
- Kubzin M.A. and Strusevich V.A. (2006). Planning machine maintenance in two-machine shop scheduling. *Operations Research*, 54(4), 789–800, DOI: [10.1287/opre.1060.0301](https://doi.org/10.1287/opre.1060.0301).
- López-Ayala C.A., Jurado-Valbuena W., & López-Santana E.R. (2021). Proposal of a Dynamic Algorithm for the Maintenance and Vehicle Routing Problem with Time Windows. *Ingeniería*, 26(3), 436–449, DOI: [10.14483/23448393.18482](https://doi.org/10.14483/23448393.18482).
- López-Santana E., Akhavan-Tabatabaei R., Dieulle L., Labadie N., & Medaglia A.L. (2016). On the combined maintenance and routing optimization problem. *Reliability Engineering & System Safety*, 145, 199–214, DOI: [10.1016/j.res.2015.09.016](https://doi.org/10.1016/j.res.2015.09.016).
- López-Santana E., Méndez G., & Franco C. (2023). On the multi-period combined maintenance and routing optimisation problem. *International Journal of Production Research*, 61(23), 8265–8290, DOI: [10.1080/00207543.2023.2180301](https://doi.org/10.1080/00207543.2023.2180301).
- Ogunfowora O., & Najjaran H. (2023). Reinforcement and deep reinforcement learning-based solutions for machine maintenance planning, scheduling policies, and optimization. *Journal of Manufacturing Systems*, 70, 244–263, DOI: [10.1016/j.jmsy.2023.07.014](https://doi.org/10.1016/j.jmsy.2023.07.014).
- Pincirolì L., Baraldi P., & Zio E. (2023). Maintenance optimization in industry 4.0. *Reliability Engineering & System Safety*, 234, 109204, DOI: [10.1016/j.res.2023.109204](https://doi.org/10.1016/j.res.2023.109204).
- Radzki G., Bocewicz G., Nielsen I., Rudy J., & Banaszak Z. (2025). Multimodal processes driven distribution of customers' ordered services. *Journal of Industrial and Production Engineering*, 1–31, DOI: [10.1080/21681015.2025.2583199](https://doi.org/10.1080/21681015.2025.2583199).
- Radzki G., Bocewicz G., Rudy J., Idzikowski R., & Banaszak Z. (2025). Assessing Feasibility in Service Teams Transport Scheduling with Dedicated and Flexible Dispatch Approaches. *Applied Sciences*, 15(23), 12727, DOI: [10.3390/app152312727](https://doi.org/10.3390/app152312727).
- Rastegar M., Karimi H., & Vahdani H. (2024). Technicians scheduling and routing problem for elevators preventive maintenance. *Expert Systems with Applications*, 235, 121133, DOI: [10.1016/j.eswa.2023.121133](https://doi.org/10.1016/j.eswa.2023.121133).
- Rudy J., <https://github.com/jaroslav-rudy/Intermodal-Vehicle-Routing-Problem-for-Maintenance>, 2026 [accessed on 23.01.2026].
- Rudy J., & Żelazny D. (2015, March). GACO: a parallel evolutionary approach to multi-objective scheduling. In *International Conference on Evolutionary Multi-Criterion Optimization* (pp. 307–320). Cham: Springer International Publishing, DOI: [10.1007/978-3-319-15934-8_21](https://doi.org/10.1007/978-3-319-15934-8_21).
- Ruiz-Rodríguez M.L., Kubler S., Robert J., & Le Traon Y. (2024). Dynamic maintenance scheduling approach under uncertainty: Comparison between reinforcement learning, genetic algorithm simheuristic, dispatching rules. *Expert Systems with Applications*, 248, 123404, DOI: [10.1016/j.eswa.2024.123404](https://doi.org/10.1016/j.eswa.2024.123404).
- Si G., Xia T., Gebraeel N., Wang D., Pan E., & Xi L. (2025). Holistic opportunistic maintenance scheduling and routing for offshore wind farms. *Renewable and Sustainable Energy Reviews*, 207, 114991, DOI: [10.1016/j.rser.2024.114991](https://doi.org/10.1016/j.rser.2024.114991).
- Szwarc E., Wójcik R., Bocewicz G., & Banaszak Z. (2024, May). Optimizing ambulance routing for timely delivery and pick-up of nursing teams visiting patients undergoing home therapy. In *Conference on Automation* (pp. 24–38). Cham: Springer Nature Switzerland, DOI: [10.1007/978-3-031-78266-4_3](https://doi.org/10.1007/978-3-031-78266-4_3).
- Wocker M.M., Ostermeier F.F., Wanninger T., Zwinkau R., & Deuse J. (2024). Flexible job shop scheduling with preventive maintenance consideration. *Journal of Intelligent Manufacturing*, 35(4), 1517–1539, DOI: [10.1007/s10845-023-02114-3](https://doi.org/10.1007/s10845-023-02114-3).
- Yang Z., Djurdjanovic I., & Ni J. (2008). Maintenance scheduling in manufacturing systems based on predicted machine degradation. *Journal of intelligent manufacturing*, 19(1), 87–98, DOI: [10.1007/s10845-007-0047-3](https://doi.org/10.1007/s10845-007-0047-3).
- Yue B., Yang J., Ma J., Shi J., & Shangguan L. (2024). An improved sequential insertion algorithm and tabu search to vehicle routing problem with time windows. *RAIRO-Operations Research*, 58(2), 1979–1999, DOI: [10.1051/ro/2024069](https://doi.org/10.1051/ro/2024069).

- Zhang Z., Fu Y., Gao K., Pan Q., & Huang M. (2024). A learning-driven multi-objective cooperative artificial bee colony algorithm for distributed flexible job shop scheduling problems with preventive maintenance and transportation operations. *Computers & Industrial Engineering*, 196, 110484, DOI: [10.1016/j.cie.2024.110484](https://doi.org/10.1016/j.cie.2024.110484).
- Zonta T., Da Costa C.A., da Rosa Righi R., de Lima M.J., Da Trindade E.S., & Li G.P. (2020). Predictive maintenance in the Industry 4.0: A systematic literature review. *Computers & industrial engineering*, 150, 106889, DOI: [10.1016/j.cie.2020.106889](https://doi.org/10.1016/j.cie.2020.106889).