Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

General variable neighborhood search for electric vehicle routing problem with time-dependent speeds and soft time windows

Luka Matijević[a]*

[a]*Mathematical Institute of the Serbian Academy of Sciences and Arts , Kneza Mihaila 36/III, Belgrade, 11001, Serbia*

C H R O N I C L E

Article history:
Received December 28 2022
Accepted February 8 2023
Available online
February, 8 2023

Keywords:
Green Vehicle Routing Problem
Alternative Fuel Vehicles
Metaheuristics
MILP
Green logistics

A B S T R A C T

With the growing environmental concerns and the rising number of electric vehicles, researchers and companies are paying more and more attention to green logistics. This paper studies the Electric Vehicle Routing Problem with time-dependent speeds and soft time windows. The purpose is to minimize the total distance travelled, while penalizing early or late arrivals at the customers' locations. For this purpose, we formulated the Mixed Integer Linear Program (MILP) and developed a General Variable Neighborhood Search (GVNS) metaheuristic, an efficient way to tackle this problem. To prove the efficiency of our approach, we tested the GVNS against the Adaptive Large Neighborhood Search (ALNS) algorithm and our MILP model, using a set of available benchmark instances. After an extensive experimental evaluation, we concluded that GVNS can find better quality solutions than other methods considered in this research or the same quality solution in less time.

1. Introduction

With the rising concern about the environmental impact of *greenhouse gases* (**GHG**) emissions, many countries committed to reducing their emission levels, either by imposing higher taxes on emissions or by subsidizing energy sources that are considered to be environmentally friendly. CO_2 is the most common GHG, attributing around 81% of all GHG in the EU (Agency, 2020). As approximately 26% of all GHG emissions come from the transportation sector (Agency, 2020), it is important to consider different tactics for reducing those emissions. For this reason, environmentally friendly logistics has received much attention in recent years.

The vehicle routing problem is a well-studied problem, with several books published on the topic Golden et al. (2008); Toth and Vigo (2014, 2002). It can be seen as a generalization of the Traveling Salesmen Problem with *N* salesmen. Many different variations of the VRP emerged over time, each characterized by a specific set of constraints and objective functions. The particular features characterizing a VRP variant are also called *attributes*. The most common attributes include capacities of vehicles, time windows, homogeneity or heterogeneity of a fleet of vehicles, asymmetry of the distances between nodes, multiple depots, multiple compartments, split deliveries, time dependency, and many others.

The *Environmentally friendly Vehicle Routing Problem* (**EF-VRP**) appeared quite recently, in response to the growing concerns about CO_2 emissions and climate change. It is an extension of the classic VRP which considers the environmental impact of routing a fleet of vehicles. EF-VRP is very often referred to as the Green Vehicle Routing problem, but this term is ambiguous, as it also designates one specific version of EF-VRP. EF-VRP can be divided into three different categories.

* Corresponding author
E-mail: luka@mi.sanu.ac.rs (L. Matijević)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2023 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.2.001

Pollution Routing Problem (**PRP**) was introduced by Bektaş and Laporte (2011). It considers the fleet of conventional *internal combustion vehicles* (**ICV**) but differs from the classical VRP in terms of the objective function, as PRP aims at minimizing the total GHG emissions of each route. When approximating emissions associated with each route, several factors can be taken into consideration, including the distance between customers, vehicle speeds, fuel consumption, load, road gradient, etc. This type of problem most often optimizes several distinct objectives at once, thus balancing the economic and environmental factors. *Green Vehicle Routing Problem* (**GVRP**) seeks to minimize the fuel consumption of a fleet of vehicles. The fleet can consist of ICVs, *alternative fuel vehicles* (**AFV**), or both. AFVs denote any vehicle that uses any other type of engine other than the petroleum-based internal combustion engine, including electric vehicles, hybrid electric vehicles, hydrogen vehicles, and others. The problem was first formulated by Erdoğan and Miller-Hooks (2012), who introduced the term GVRP. The emissions concerns are implicitly dealt with, as the assumption is that by simply using AFVs for the delivery process we eliminate the GHG emissions associated with routing. This is, of course, a simplified view as we do not take into consideration the emissions related to the production of AFVs nor the emissions resulting from generating electrical energy, which is still primarily comes from burning fossil fuels IEA (2019), but these problems are out of the scope of the VRP. A special case of GVRP that deals with the routing of electric vehicles is referred to as the *Electric Vehicle Routing Problem* (**EVRP**). The third type of problem that is sometimes classified as EF-VRP is the *VRP in reverse logistics*. This type of problem does not necessarily address emissions. More often, it deals with other environmentally beneficial aspects of VRP, such as the waste collection or recycling of used materials.

In this paper, we will look at a version of GVRP that uses electric vehicles with soft time windows and time-dependent speeds (**TD-EVRP-STW**). This problem has a real-world application as it addresses the needs of delivery companies with a fleet of electric vehicles. More particularly, last-mile delivery in urban neighborhoods is examined in this case. Considering the fact that vehicle speed is greatly influenced by the time of the day in heavily populated areas (rush hours, congestions, construction work, etc.), the time-dependent speeds have to be included in the model in order to correctly calculate the time necessary for the delivery of goods. Moreover, customer satisfaction is often correlated with the ability of the company to deliver purchased goods in a preferred time window. That being said, some small deviations from the preferred delivery time can be tolerated, therefore, soft time windows are preferable to hard time windows. For these reasons, we believe that the proposed formulation of the problem is relevant to the needs of the industry. To the best of our knowledge, this problem has not yet been researched in the literature. The contributions of this paper are multiple:

- We identified and formulated a version of EVRP that was not previously considered in the literature and has a clear application in the industry.
- We presented the reader with an extensive literature review on the topic of EVRP.
- We proposed a *Mixed Integer Linear Program* (**MILP**) formulation for TD-EVRP-STW and developed a *General Variable Neighborhood Search* metaheuristic for finding a good quality solution in a reasonable amount of time.
- We proposed a procedure for determining the departure times of each vehicle from visited customers and recharging stations in an efficient way.
- We analyzed the neighborhood structures used in our local search method and presented our findings so that other authors could have some insight into the quality of the aforementioned neighborhoods and could more easily decide which one to include in their own methods.

This paper is organized as follows. In Section 2, we described the problem considered in this article. Section 3 provides the reader with an extensive literature review on the topic. In Section 4, we propose our own mathematical model for TD-EVRP-STW, and in Section 5, we describe our proposed metaheuristic for finding a good quality solution to this problem. The experimental evaluation of our method is presented in Section 6, together with the analysis of used neighborhood structures and parameters. Finally, we presented the concluding remarks and potential areas for future research in Section 7.

2. Problem definition

Let us consider a distribution company with a fleet of electric vehicles at its disposal. Each day, the company receives a certain number of requests from its customers and needs to fulfill those requests in the most efficient manner. All the vehicles are the same in terms of capacity, consumption rate, and base speed. Each customer selects a certain amount of goods and a time interval in which the delivery would be preferable. The company then dispatches its vehicles, making sure that the total amount of goods loaded into the truck does not exceed its capacity. As electric vehicles have a limited range, they might need to visit a refueling station at some point while serving customers. Recharging electric vehicles is not as quick as refueling an ICV, so the dispatcher has to take that aspect into account when planning the arrival times of each vehicle at customers' locations. If a customer is visited outside of its preferred time window, it is more likely to be unsatisfied with the company's service and could potentially take business elsewhere in the future. Arriving late at the customer's location can obviously impact the satisfaction with the service, but arriving early can also be problematic as the customer may not be ready to receive the delivery. In some of the studies, this is mended by making the vehicle wait until the start of the time window. In our opinion, this approach is inadequate, because waiting at a certain location can result in a much higher penalty for being late at other customers' locations. For this reason, we allow vehicles to arrive early at the location with a certain penalty. In real-world situations, each refueling station has a limitation on how many vehicles it can serve at any given time and could have several different recharging technologies, allowing users to recharge their vehicles at a different rate, but also at different prices. In this paper, we ignored this factor and assumed that each refueling station has unlimited vehicle capacity and the same recharging rate. Additionally, in real-world scenarios, energy consumption is influenced by numerous factors, such as distance, road gradient, the weight of the vehicle, etc. For more information on different consumption models please refer to Abousleiman and Rawashdeh (2015); Chen et al. (2020). These models are most often nonlinear. Despite that, for this research, we assumed that consumption is a linear function of distance. We find the justification for this simplification in the fact that the primary goal of this research is to devise a reliable metaheuristic method that can find a good enough solution in a reasonable amount of time. Introducing the nonlinear element to our model would greatly increase the complexity of the mathematical model while having little to no effect on metaheuristic methods. Therefore, we concluded that demonstrating the advantage of our metaheuristic approach over an exact solver is a good indication of their relative performance, even when nonlinearity is introduced to the problem. For the same reason, we assumed that the charging function is a linear function of battery level, which is not the case in practice Montoya et al. (2017). Both these nonlinear elements can be easily added to our proposed metaheuristic without any change to the algorithm. We also assume that each time a vehicle visits a recharging station, its battery is recharged to its full capacity. The entire time interval in which the customers should be served is divided into several smaller intervals, to model the different road conditions and rush hours. Each interval is associated with an average speed by which a vehicle can travel during that time interval. These time-dependent speeds have to be taken into consideration, as they directly influence the ability of a vehicle to arrive at a customer's location during the preferred time window. In Figure 1, we display a generic version of EVRP. The depot is represented with a triangle, customers with circles, and refueling stations with rhombi. Each vehicle starts its journey from the depot with 100% of the battery levels. Each time a vehicle serves some customer its battery level gets lower until it's forced to visit one of the recharging stations and replenish the battery.

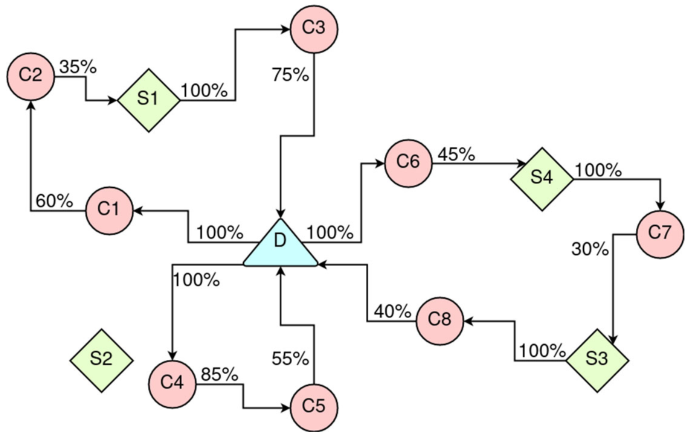

Fig. 1. A generic example of EVRP

3. Literature review

In this paper, we are primarily interested in *Electric Vehicle Routing Problem* (**EVRP**). In recent years, EVPR received much attention, as many articles have been published on the topic, considering various attributes and solution approaches to this problem.

Felipe et al. (2014) considered EVRP with partial recharges and multiple recharging technologies, which allow the battery to replenish at different rates and costs. Recharging vehicles at higher speeds would result in lower recharging time but would have a higher price. Several local search methods were tested, including the *Simulated Annealing* (**SA**) (Kirkpatrick et al. (1983)) metaheuristic. Schneider et al. (2014) developed a *Variable Neighborhood Search* (**VNS**) and *Tabo Search* (**TS**) (Glover (1986)) hybrid for finding a solution to the EVRP with hard time windows and capacity constraints. Furthermore, a set of benchmark instances for EVRP is proposed. Bruglieri et al. (2015) considered a version of EVRP with hard time windows and partial battery recharge. The goal was to simultaneously minimize the number of vehicles, total travel time, total recharge time, and total waiting time. For this purpose, the authors used a metaheuristic based on VNS called *Variable Neighborhood Branching* (**VNB**), first proposed by Hansen et al. (2006). Goeke and Schneider (2015) examined a capacitated EVRP with hard time windows and a heterogeneous fleet, where at least a part of the fleet consists of electric vehicles. The objective was to minimize the total distance travelled by all vehicles, the total cost of vehicle propulsion and labor, and the cost of battery recharging. The *Adaptive Large Neighborhood Search* (**ALNS**) (Ropke and Pisinger (2006)) metaheuristic was used, with an acceptance criterion based on SA. EVRP with hard time windows was also explored in Hiermann et al. (2016). The authors considered a scenario in which a company does not own electric vehicles but leases them at a certain price. There are several types of vehicles at disposal, differing in capacity. The goal is to minimize the total travel cost as well as the acquisition cost of the vehicles. For this purpose, ALNS is used, coupled with the local search and a labeling procedure which helps to improve the solution. Another example of ALNS being used for EVRP is presented by Keskin and Çatay (2016). The goal was to minimize the total distance travelled, considering hard time windows and partial recharges. Yavuz and Çapar (2017) considered EVRP with a heterogeneous fleet consisting of both AFVs and ICVs. Vehicles do not only deliver goods to customers but provide certain services, therefore, the service time of a particular customer can be longer than in the standard version of the problem. This allows vehicles to be partially recharged while serving a customer, but visits to AFS still replenish the battery to its full capacity. A multi-objective variant of the problem is considered, with four distinct objectives: minimizing the total distance travelled by all vehicles, minimizing CO_2 emissions, minimizing the fuel cost, and minimizing the distance travelled by ICVs. For finding a Pareto-front for this problem, the VNS method was used. VNS is also used by Affi et al. (2018) for minimizing the total energy cost for a fleet of electric vehicles with capacity constraints. A basic version of EVRP was examined by Mavrovouniotis et al. (2018), who used *Ant Colony Optimization* (**ACO**) (Dorigo et al. (1996)) to minimize the total operation time of the fleet of electric vehicles. This approach was further improved in Mavrovouniotis et al. (2019), where the authors introduced parallelization to ACO. A similar approach to a basic EVRP was taken by Zhang et al. (2018a), who proposed using the *Ant Colony System* (**ACS**) to minimize the total distance travelled. The authors also presented the two-phase heuristic for tackling this problem, which consists of solving the TSP version of the problem by applying the *Nearest Neighbor Criteria* to it, after which the visits to the AFSs and depot are inserted into the solution, turning it into a solution to the initial problem. Zhang et al. (2018b) considered capacitated EVRP, with the goal of optimizing the total energy consumption. The consumption is estimated by taking into consideration vehicle weight, distance, speed, motor efficiency, etc. For this problem, the authors hybridized ACO with *Iterated Local Search* (**ILS**). Additionally, ALNS was also proposed for this problem but was outperformed by the ACO-ILS hybrid. The *multi-start local search* (**MSLS**) metaheuristic was applied to EVRP in Andelmin and Bartolini (2019). Hiermann et al. (2019) considered a fleet of vehicles consisting of ICVs, *hybrid electric vehicles* (**HEV**), and EVs. Furthermore, vehicles in each class can vary in terms of capacity, energy consumption, and battery capacity. Hard time windows constraint is also imposed on this problem, as well as partial recharge. The goal was to optimize the total cost, consisting of fixed costs like maintenance and acquisition of vehicles, and dynamic costs such as fuel consumption. The routes are obtained by using a *Genetic Algorithm* (**GA**), and the visits to the AFSs are inserted into a route by using dynamic programming. Macrina et al. (2019b) examined EVRP with partial recharges, hard time windows, and a heterogeneous fleet consisting of both ICVs and EVs. The goal was to minimize the total fuel and electricity cost. As a solution approach, the authors proposed a metaheuristic based on the *Large Neighborhood Search* (**LNS**) algorithm. The same problem was discussed in Macrina et al. (2019a), with ILS as a solution approach. SA was used for a capacitated EVRP by Normasari et al. (2019) to minimize the total distance travelled by all vehicles. Peng et al. (2019) minimized the total distance travelled by electric vehicles by using a *Memetic algorithm* (**MA**), with adaptive local search as a local search procedure. MA with a competition mechanism combined with VNS was proposed by Wang and Lu (2019) for minimizing the total distance travelled by all EVs. The algorithm starts by creating the initial population using the k-nearest neighbors algorithm, where each individual has a different, randomly selected starting point. VNS with SA acceptance criterion is used in each generation to intensify the search in promising areas. Zhang et al. (2019) considered EVRP with multiple depots, with the goal of minimizing the total distance travelled. The solution approach consists of assigning each customer to its nearest depot and then applying ACS to several single-depot EVRP instances. Li et al. (2020) examined the distribution sharing version of EVRP, where several distribution companies combine their assets in order to minimize the overall cost. For this purpose, the authors used the ACO algorithm. Ren et al. (2020) proposed bi-objective VNS for minimizing emissions and total delay time of a fleet consisting of both ICVs and EVs, where each customer is associated with a hard time window. The delay time refers to the observation that even if a customer is served inside a predetermined time window, the satisfaction with service is greater if the arrival time is closer to the beginning of the time window, as opposed to the end of the time window. Zhang et al. (2020b) presented two methods for minimizing CO_2 emissions of the multi-depot EVRP: *two-stage Ant Colony System* (**TSACS**) and *Partition-Based Algorithm* (**PBA**). Emissions are estimated simply by applying a conversion factor to the total distance travelled.

Even though soft time windows are a relatively common attribute for the classic version of VRP, there are not many articles in which this attribute is used with EVRP. One example of EVRP with soft time windows can be found in the conference paper by Zhang et al. (2018c). The authors considered EVRP in reverse logistics, with partial recharges, but only presented a mathematical model. It is said in the paper that TS was also implemented, but the authors do not provide any implementation details, nor a comparison with a MILP solver. EVRP with soft time windows was also examined in Urazel and Keskin (2021). The authors proposed two solution approaches: GA and a hybrid metaheuristic combining GA and SA, with the objective of minimizing the total distance travelled and penalties from missing time windows.

Similar to the soft time windows constraint, the time-dependent variant of EVRP was not extensively researched. A general review of time-dependent routing problems is presented by Gendreau et al. (2015), without explicitly considering EVs. There are several articles considering time-dependent variants for PRP, like Xiao and Konak (2015, 2017); Wang et al. (2019), which also consider soft time windows. To the best of our knowledge, only two articles applied the time-dependent speeds assumption to the EVRP and proposed some metaheuristic solution approach. Wang et al. (2020) investigated EVRP with hard time windows, time-dependent speeds, and path flexibility. The authors presented a MILP model for selecting the optimal path, where the objective was to minimize the energy consumption and the total distance, where relative influences of these two goals are controlled by their respective weights. Based on this model, the authors further developed a MILP model for the original problem with the objective of minimizing the total distance travelled. The General VNS method was applied to this problem with two neighborhood structures. Zhang et al. (2020a) analyzed the EVRP with time-dependent speeds, hard time windows, partial battery recharge, and congestion tolls. The objective was to minimize route duration costs, recharging costs, and congestion tolls. The authors presented a MILP model, and offered ALNS as a more practical solution method.

4. Mathematical model

TD-EVRP-STW can be formulated as follows. Let V be the set of customers to be visited, F is the set of refueling stations and D represents the depot. Considering the fact that refueling stations can be visited multiple times during the delivery, we create a set of dummy nodes F' by replicating nodes from F several times. With this in mind, we can create a directed complete graph $G = (N, A)$, where $N = D \cup V \cup F' \cup D$ is a set of vertices corresponding to the depot, customers and refueling stations, and $A = \{(i,j) | i,j \in N, i \neq j\}$ is the set of arcs connecting these vertices. The depot node is duplicated to differentiate between the origin and destination of each route. The weights associated with each arc correspond with the distance between the nodes connected by that arc. The entire period in which the delivery should take place is divided into several time intervals, each represented by a starting time, ending time, and the average speed of the vehicles. By introducing time intervals we can model more realistic situations, especially in the case of the last-mile delivery, where the average speed of vehicles may vary throughout the day depending on rush hours, congestion, or other reasons. The time needed to traverse each arc is determined dynamically, by taking into account the starting time and time intervals in which the arc traversing takes place. Each customer is associated with a demand and a time window, i.e. the preferred time in which the customer should be served. Arriving early or late at a customer's location results in a penalty, as it affects the satisfaction of a customer. All of the vehicles are homogeneous, which means that they have the same capacity, consumption rate, and speed, given a time interval. The time that a vehicle spends at each node is given for each customer as a service time $s_i, i \in V$, while the service time at the refueling stations depends on the remaining battery level and the refueling rate. Partial recharges are not considered, i.e. the battery is recharged fully each time a vehicle visits a recharging station. The mixed-integer linear program (**MILP**) formulation presented in this paper is inspired by (Schneider et al., 2014) and (Zhang et al., 2020a). Furthermore, we adopt the same naming convention as in the aforementioned publication. The complete list of symbols used in our MILP formulation is presented in Table 1. together with the meaning behind each of the symbols.

Table 1
Variable and parameter definitions

Symbol	Meaning
$0, N+1$	Depot nodes
F	The set of recharging stations
F'	Dummy nodes representing the set of replicated nodes from F
F'_0	Recharging nodes F'including the depot node 0, $F' \cup \{0\}$
V	Set of customers $V = \{1, \ldots, N\}$
V_0	Set of customers including the depot, $V_0 = V \cup \{0\}$
V'	Set of customers and recharging stations, $V' = V \cup F'$
V'_0	Set of customers, recharging stations and depot 0, $V'_0 = V' \cup \{0\}$
V'_{N+1}	Set of customers, recharging stations and depot $\{N+1\}$, $V'_{N+1} = V' \cup \{N+1\}$
$V'_{0,N+1}$	Set of customers, recharging stations and depot instances $\{0\}$ and $\{N+1\}$, $V'_{0,N+1} = V' \cup \{N+1\} \cup \{0\}$
K	Set of time intervals
ψ_k	The beginning of the time interval $k \in K$
υ_k	The end of the time interval $k \in K$
L_k	The vehicle speed during the time interval $k \in K$
c_1, c_2	Coefficients for the objective function
d_{ij}	The distance between nodes i and j
t_{ijk}	The travel time between nodes i and j during the time interval k
C	Vehicle capacity
g	Recharging rate
h	Vehicle consumption rate
Q	Battery capacity
q_i	Demand of node i, 0 if $i \notin V$
s_i	Service time of node i, 0 if $i \notin V$
e_i	The start of the time window of node i
l_i	The end of the time window of node i
α	Penalty coefficient for arriving early at some node
β	Penalty coefficient for arriving late at some node
M	Large enough constant (upper bound)
x_{ij}	Decision variable indicating that the arc (i,j) is travelled
z_{ijk}	Decision variable indicating that the arc (i,j) is travelled during the time interval k
τ_i	Decision variable specifying the arrival time at node i
ϕ_i	Decision variable specifying the departure time from node i
u_i	Decision variable specifying the remaining cargo on arrival at node i
y_i	Decision variable specifying the remaining battery level on arrival node i
φ_{ijk}	Decision variable specifying the distance travelled from node i to node j during the time interval k
μ_i	Decision variable indicating the penalty for arriving early at node i
η_i	Decision variable indicating the penalty for arriving late at node i
p_i	Decision variable indicating the total penalty at node i

$$min\left(c_1 \sum_{i\in V_0'} \sum_{j\in V'_{N+1}, i\neq j} d_{ij}\ x_{ij} + \sum_{i\in V} p_i\right) \tag{1}$$

subject to

$$\sum_{j\in V'_{N+1}, i\neq j} x_{ij} = 1, \qquad \forall i \in V \tag{2}$$

$$\sum_{j\in V'_{N+1}, i\neq j} x_{ij} \leq 1, \qquad \forall i \in F' \tag{3}$$

$$\sum_{j\in V'_{N+1}, i\neq j} x_{ji} - \sum_{j\in V'_{0}, i\neq j} x_{ij} = 0, \qquad \forall j \in V' \tag{4}$$

$$\tau_i + \sum_{k\in K} t_{ijk} + s_i - l_0(1-x_{ij}) \leq \tau_j, \qquad \forall i \in V_0, \qquad \forall j \in V'_{N+1}, \qquad i \neq j \tag{5}$$

$$\tau_i + \sum_{k\in K} t_{ijk} + g(Q-y_i) - (l_0 + gQ)(1-x_{ij}) \leq \tau_j, \qquad \forall i \in F', \qquad \forall j \in V'_{N+1}, i \neq j \tag{6}$$

$$\mu_i = \max(0, e_i - \tau_i), \qquad \forall i \in V \tag{7}$$

$$\eta_i = \max(0, \tau_i - l_i), \qquad \forall i \in V \tag{8}$$

$$p_i = \alpha\mu_i + \beta\eta_i, \qquad \forall i \in V \tag{9}$$

$$0 \leq u_j \leq u_i - q_i x_{ij} + C(1-x_{ij}), \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad i \neq j \tag{10}$$

$$0 \leq u_0 \leq C \tag{11}$$

$$0 \leq y_j \leq y_i - (h \cdot d_{ij})x_{ij} + Q(1-x_{ij}), \qquad \forall i \in V, \qquad \forall j \in V'_{N+1}, \qquad i \neq j \tag{12}$$

$$0 \leq y_j \leq Q - (h \cdot d_{ij})x_{ij}, \qquad \forall i \in F'_0, \qquad \forall j \in V'_{N+1}, \qquad i \neq j \tag{13}$$

$$d_{ij} \cdot x_{ij} = \sum_{k\in K} \varphi_{ijk}, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad i \neq j \tag{14}$$

$$\phi_i \geq \tau_i + s_i, \qquad \forall i \in V'_{0,N+1} \tag{15}$$

$$\phi_i \geq \tau_i + g(Q-y_i), \qquad \forall i \in F' \tag{16}$$

$$x_{ij} - z_{ijk} \geq 0, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{17}$$

$$\sum_{k\in K} z_{ijk} \geq x_{ij}, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1} \tag{18}$$

$$\varphi_{ijk} - d_{ij} \cdot z_{ijk} \leq 0, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{19}$$

$$z_{ijk} - M \cdot \varphi_{ijk} \leq 0, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{20}$$

$$t_{ijk} = \frac{\varphi_{ijk}}{L_k}, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{21}$$

$$t_{ijk} \leq \upsilon_k - \psi_k, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{22}$$

$$\phi_i + t_{ijk} \leq \upsilon_k + \upsilon_{last}(1-z_{ijk}), \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{23}$$

$$\psi_k + t_{ijk} \leq \tau_j + \upsilon_{last}(1-z_{ijk}), \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{24}$$

$$\phi_i + \sum_{k\in K} t_{ijk} \leq \tau_j + \upsilon_{last}(1-x_{ij}), \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1} \tag{25}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in V'_0, \qquad \forall j \in V'_{N+1} \tag{26}$$

$$z_{ijk} \in \{0,1\}, \quad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{27}$$

$$\tau_i, \phi_i, u_i, y_i, \mu_i, \eta_i, p_i \geq 0, \qquad \forall i \in V'_{0,N+1} \tag{28}$$

$$\varphi_{ijk} \geq 0, \qquad \forall i \in V'_0, \qquad \forall j \in V'_{N+1}, \qquad \forall k \in K \tag{29}$$

The objective function consists of two parts: minimizing the total distance travelled by all vehicles and minimizing the penalty for missing time windows for customers (1). The relative influence of these two parts is controlled by constants c_1 and c_2. Constraints (2) ensure that each customer has to be visited exactly once, and constraints (3) define that each dummy node representing a refueling station can be visited at most once. Constraints (4) ensure the connectivity of each route, by specifying that each node except the depot must have the same number of incoming and arcs. Constraints (5) define the arrival times at each node, thus ensuring the time feasibility of arcs leaving depot and customers, and constraints (6) do the same for arcs leaving refueling stations. Constraints (7), (8), and (9) calculate the penalty for missing a time window at each node. Considering the fact that the penalty is minimized as a part of the objective function, the *max* function can easily be represented as a set of linear constraints (30), without including any new variables.

$$y = \max(x_1, x_2, \dots, x_n) \leftrightarrow y \geq x_i, \forall i = 1, \dots, n \tag{30}$$

Please keep in mind that this transformation works only if y is minimized in the objective function. For a more generic linearization technique, please refer to Asghari et al. (2022). Constraints (10) and (11) ensure that there is enough cargo to fulfill the demand of each customer. Constraints (12) and (13) guarantee that the battery level will not fall below zero during the delivery. Constraints (14) state that the distance travelled in all periods must be equal to the distance between nodes i and j, if the arc (i, j) is travelled. Constraints (15) define the departure time from customers and depot, based on the service time and arrival time. Similarly, constraints (16) define the departure time from refueling stations. Constraints (17) and (18) make a connection between variables x and z, by specifying that if arc (i, j) is travelled, there needs to be at least one period associated with it, otherwise, there cannot be any period associated with that arc. Constraints (19) specify that the distance travelled at any period may not be greater than the total distance of arc (i, j), if that arc is travelled. Constraints (20) ensure that if the arc is travelled during a period k, the distance crossed during that period should be greater than zero. Constraints (21) determine the time travelled during each period, constraints (22) forbid the time travelled to be greater than the length of the period, and constraints (23) limit the time travelled to the remaining time in that period, considering the departure time of the vehicle. These constraints also ensure that the FIFO property is satisfied, as vehicles automatically change speeds as soon as a new period begins. This approach addresses the issues with the step function presented in Malandraki and Daskin (1992) and criticized in Ichoua et al. (2003), which assumes that a vehicle will travel the whole ark (i, j) at a speed associated with the period in which the vehicle departed from node i. Constraints (24) ensure that the actual time in which the traveling is completed for some period (i.e. sum of beginning time for a given period and traveling time in that period) does not exceed the arrival time at the next node. Constraints (25) define that if arc (i, j) is travelled, the sum of departing time at node i and the total traveling time during all periods should not exceed the arrival time at node j. Finally, constraints (26)-(29) define the nature of decision variables.

5. General Variable Neighbourhood Search

Variable Neighborhood Search (**VNS**) is a single solution metaheuristic, based on local search and systematical change of neighborhood structures. It was first proposed by Mladenović and Hansen (1997), and since then it has been successfully applied to numerous problems Kovač et al. (2018); Yazdani et al. (2010); Matijević (2022). In its basic form, VNS consists of three main steps: *shaking*, *local search*, and *neighborhood change*. In the shaking step, the algorithm chooses a random solution from neighborhood N_k of the incumbent solution. The purpose of this step is to diversify the search and allow the algorithm to escape local optima. The solution generated in the shaking step is passed to the local search procedure, which tries to improve it by assessing its neighbors, using one or more neighborhood structures. After the local search, the obtained solution is evaluated in the Neighborhood change step, and if it is better than the incumbent solution, it is accepted as the new best solution, so the search continues from the smallest neighborhood, otherwise, the search moves on to the next neighborhood. There are many variations of the VNS method. The simplest one is the *Reduced VNS*, which consists only of shaking and neighborhood change steps. *Skewed VNS* differs from the basic VNS in the neighborhood change step, as it allows a little bit worse solutions to be accepted, as long as they are relatively distant from the incumbent solution. Similarly, *Plateau VNS* allows solutions with the same quality as the incumbent solution to be accepted. This is done in order to prevent the algorithm from getting stuck in some local optimum. A deterministic version of VNS called *Variable Neighborhood Descent* (**VND**) does not include the shaking step but sequentially searches all neighborhoods. It uses the first-improvement approach, i.e. as soon as the algorithm finds an improvement in some neighborhood, it goes back to the first neighborhood, and the search continues. Nonetheless, the neighborhoods themself can be explored by using either the first-improvement or the best-improvement approach. The variation of VNS that uses VND as its local search procedure is called *General Variable Neighborhood Search* (**GVNS**).

5.1. Solution representation

The solution S to our TD-EVRP-STW is represented as an array of doubly linked lists, each list representing a vehicle $(S_i, i = 1, \dots, N)$. Each element in a doubly linked list represents one customer, described with an ordered quadruple $S_{ij} = (S_{ij}^{index}, S_{ij}^{arrival}, S_{ij}^{departure}, S_{ij}^{type}), i = 1, \dots, N,\ j = 1, \dots, K_i$. S_{ij}^{index} contains an index corresponding with a customer, depot, or recharging station, $S_{ij}^{arrival}$ and $S_{ij}^{departure}$ represent arrival and departure times for the node S_{ij} respectively, and S_{ij}^{type} defines the type of a node, i.e. it contains the information on whether the node represents a depot, customer, or a recharging station. The sequence of nodes in each list defines the ordering in which they are traversed in a route. The general example of our solution structure is presented in Figure 2.

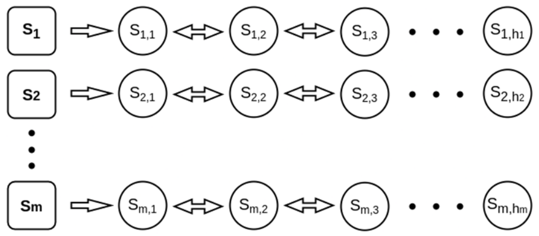

Fig. 2. Representation of a solution

5.2. Neighbourhood structures

Choosing the right neighborhoods for local search and shaking procedures can have a tremendous impact on the overall performance of an algorithm. In this paper, we identified ten different neighborhood structures, some of which were previously presented in Affi et al. (2018).

- $N_1(S, \varepsilon)$: Changes the departure time of a customer by ε, either by adding or subtracting ε to/from the current departure time.
- $N_2(S)$: Moves a customer to some other position on the same route.
- $N_3(S)$: Moves a customer from one route to another.
- $N_4(S)$: Swaps the positions of two customers inside the same route.
- $N_5(S)$: Swaps the positions of two customers who are on different routes.
- $N_6(S)$: Removes a refueling station from a route.
- $N_7(S)$: Moves a refueling station to some other position inside the same route.
- $N_8(S)$: Replaces a refueling station with another refueling station.
- $N_9(S, k)$: Moves k consecutive nodes to some other position inside the same route.
- $N_{10}(S, k)$: Moves k consecutive nodes from one route to another.

5.3. Feasibility

There are two possible ways in which the solution to our TD-EVRP-STW can be infeasible. First, the sum of demands of all customers in a route can exceed the capacity of a truck, and second, the battery can be depleted at some point, making it impossible to continue the delivery. Checking and fixing the feasibility of a solution that breaks the capacity constraint is relatively straightforward, as removing some customers from the route will result in a feasible solution. On the other hand, fixing the feasibility of a solution that breaks the battery level constraint can be a little more complicated. In Algorithm 1, we present our method for inserting refueling stations into the solution to make it feasible. In this algorithm, we go through all the nodes in each truck and adjust the current battery level based on the type of node, i.e. if the node represents a refueling station we set the battery level to full capacity (Line 8), otherwise, we subtract the energy spent from the battery (Line 10). If the current battery level is not enough to reach the next customer, we try to add a refueling station to mend this issue (Lines 12-25). In each step, we try to find the best possible station to insert into the current position (Line 14). If no station can be inserted (Line 15), i.e., the battery level is not high enough to reach any of the stations, we move to the previous node (Line 17) and try to insert a station at that position. If a station can be inserted, battery levels are adjusted, and the algorithm continues from the newly inserted station (Line 24).

Algorithm 1: The algorithm for fixing the feasibility of a solution

```
1:  procedure FIX_SOLUTION (Solution S)
2:      for ∀truck ∈ S do
3:          battery ← BATTERY_CAPACITY;
4:          for j = 1 to size(S[truck]) do
5:              spent ← calculate consumption between nodes j and (j − 1);
6:              if battery ≥ spent then
7:                  if node j is a refueling station then
8:                      battery ← BATTERY_CAPACITY;
9:                  else
10:                     battery ← battery − spent;
11:                 end if
12:             else
13:                 for k = j to 1 do
14:                     station ←Find the best station to add at position k;
15:                     if no station can be added then
16:                         battery ← battery + consumption (k, k − 1);
17:                         k ← k − 1;
18:                     else
19:                         Insert station at position k in S;
20:                         break;
21:                     end if
22:                 end for
23:                 battery ← recalculate battery level;
24:                 j ← k;
25:             end if
26:         end for
27:     end for
28:     return S
29: end procedure
```

5.4. Initial solution

VNS is a metaheuristic that requires an initial solution that serves as a starting point for the search. There are many different ways in which the initial solution can be generated. The most straightforward approach is to generate the initial solution by randomly assigning the customers to vehicles, and then fixing the created solution by adding trips to recharging stations as needed. Even though the stochastic nature of this approach is suitable for some algorithms (for example, Multi-start VND), generating a good initial solution raises the overall performance of VNS, therefore, we opted for a more sophisticated approach. In Algorithm 2 we present our greedy approach to generating an initial solution, inspired by the procedure presented in Zhang et al. (2020a). The solution is first initialized by adding two depot instances into each available vehicle, representing the origin and the final destination of each route (Line 2). Furthermore, we initialize the set of available customers that have not yet been assigned to any vehicle (Line 3). We then proceed by calculating the best position for each unassigned customer (Line 7). The customer whose addition to the solution impacts the objective function the least is inserted into the solution at the best-determined position (Line 12), and the inserted customer is removed for the set of unassigned customers (Line 13). The method continues until all the customers are assigned to some vehicle. In the end, the solution is checked for feasibility and the necessary trips to refueling stations are added to the solution (Line 15).

Algorithm 2: The algorithm for generating the initial solution

```
1:  procedure INIT_SOLUTION(Instance data $D$)
2:      $S \leftarrow$ initialize the solution by adding depot instances to each vehicle;
3:      $A \leftarrow$ set of all customers;
4:      while $A \neq \emptyset$ do
5:          $best\_customer \leftarrow -1$;
6:          for $\forall customer \in A$ do
7:              Calculate the best position of $customer$ in $S$;
8:              if $customer$ is better than $best\_customer$ then
9:                  $best\_customer \leftarrow customer$;
10:             end if
11:         end for
12:         Insert the $best\_customer$ into the best position in $S$;
13:         $A \leftarrow A \setminus \{best\_customer\}$;
14:     end while
15:     $S \leftarrow$ FIX_SOLUTION($S$);
16:     return $S$
17: end procedure
```

5.5. Determining a schedule

In order to minimize the penalty cost of a solution, we need to determine the arrival and departure times for each node. We present our method for determining the schedule in Algorithm 3. It is important to notice that this algorithm does not have to find the best possible scheduling, as it is further going to be optimized in the neighborhood N_1 during the local search. The basic idea behind our approach is simple. For each route, we traverse all the nodes and calculate their arrival and departure times. Calculating the arrival time of a node is straightforward, as it is the sum of the departure time of the previous node in a route and the travel time between those two nodes (Line 7). The exception is the depot node at the beginning of each route, whose arrival time is set to zero (Line 5). As for the departure time of each node, we choose the lowest departure time that allows the vehicle to arrive at the next node without creating any penalty for early arrival (Line 8). In other words, the departure time should be set in such a way that the vehicle arrives at the beginning of the time window of the next node or after it.

Algorithm 3: The algorithm for calculating a schedule for a given solution

```
1:  procedure CALC_SCHEDULE(Solution $S$)
2:      $N \leftarrow$ number of routes in $S$
3:      for $i = 1$ to $N$ do
4:          $M \leftarrow$ number of nodes in route $S_i$
5:          $S_{i0} \leftarrow 0$                                        //Depot instance
6:          for $j = 2$ to $M$ do
7:              $S_{ij}^{arrival} \leftarrow S_{i(j-1)}^{departure} + travel\_time(S_{i(j-1)}, S_{ij})$
8:              $S_{ij}^{departure} \leftarrow \max(S_{ij}^{arrival} + service\_time(S_{ij}), ready(S_{i(j+1)}) - travel\_time(S_{ij}, S_{i(j+1)}))$
9:          end for
10:     end for
11:     return $S$
12: end procedure
```

5.6. Shaking

In the classic version of the VNS algorithm, the shaking procedure selects a random neighbor at the distance k from the incumbent solution. Our version of the shaking procedure differs from the classic version in the sense that it selects a random neighbor at a distance of *at least k* from the incumbent solution. We made this change as it allowed us to find overall better results, as determined experimentally. Furthermore, after extensive testing, we selected two neighborhoods to use in our shaking procedure, which are selected stochastically in each iteration of the algorithm. We present our shaking procedure in Algorithm 4. In the main loop of the algorithm (Lines 4-12), we first randomly select a neighborhood structure (Line 5) and the size of that neighborhood (Line 6). The size is limited by the parameter l_{max}. We then transform the current solution using the selected neighborhood structure and size (Lines 9 and 11).

Algorithm 4: Shaking

```
1:  procedure SHAKE(Solution $S$, $k$, $l_{max}$)
2:      $i \leftarrow 0$
3:      $S' \leftarrow S$
4:      for $i$ = 1 to $k$ do
5:          $p \leftarrow$ random integer between 0 and 1
6:          $l \leftarrow$ random integer between 0 and $l_{max}$
7:          switch ($p$)
8:              case 0 :
9:                  $S' \leftarrow$ random neighbor from $N_9(S', l)$
10:             case 1 :
11:                 $S' \leftarrow$ random neighbor from $N_{10}(S', l)$
12:     end for
13:     return $S'$
14: end procedure
```

5.7. Variable neighbourhood descent

In Algorithm 5, we present our variable neighborhood descent method, using the first eight neighborhoods from Section 5.2. Nonetheless, not all neighborhoods contribute equally to the search, as we will see in Section 6.4. Therefore, some neighborhoods can (and should) be excluded. The ordering of neighborhoods can also impact the overall performance of the method, but this aspect was not further examined in this publication. Considering the fact that proposed neighborhood are rather complex, we utilized the *first improvement* strategy, i.e. as soon as the local search finds a better solution in some neighborhood, it returns the solution to VND. The parameter ε is used in neighborhood structure N_1. Even though ε is provided as an input parameter in our approach, this parameter can be determined dynamically using some reinforcement learning strategy.

Algorithm 5: Variable neighborhood descent

```
1:  procedure VND(Solution $S$, Set of neighborhoods $N$ , $\varepsilon$)
2:      for $i$ = 1 to 10 do
3:          $improved \leftarrow$ LS($N_i$ , $S$, $S'$ )          // LS is the local search in neighborhood $N_i$
4:          if $improved$ then
5:              $S \leftarrow S'$
6:              $i \leftarrow 1$
7:              continue
8:          end if
9:      end for
10:     return $S$
11: end procedure
```

5.8. General variable neighbourhood search

In Algorithm 6 we provide an overview of the proposed GVNS algorithm. The input parameters are the problem instance, a set of neighborhoods used in VND, the standard VNS parameters k_{min}, k_{step}, and k_{max} used to define the number of transformations in the shaking procedure, an additional parameter used for shaking denoted with l_{max}, and parameter epsilon used to define the increment in the neighborhood N_1. For the stopping criterion, we choose the maximum allowed CPU time. The algorithm starts by creating an initial solution, using the method presented in Algorithm 2 (Line 2). If the solution is infeasible, apply the shaking procedure to it until it becomes feasible (Lines 3-6). This step is not entirely necessary, as VND may be able to find a feasible solution during the local search, but this approach provided us with better performance. After the solution is initialized and the schedule is calculated by using Algorithm 3, we try to improve it by applying VND to it (Line 8). In the algorithms main loop, we apply the shaking procedure to the currently best-known solution, thus obtaining a new, perturbed solution. As the newly created solution may be infeasible, we repeat the shaking step until we acquire a feasible solution. The schedule is then recalculated, and VND is applied to the new solution. If the solution found by VND has a lower value of the objective function compared to the best-known solution, it is accepted as a new best-known solution (Line 21) and the parameter *k* is set to k_{min} (Line 22), otherwise, k is increased by the value of k_{step} (Line 24).

Algorithm 6: General variable neighborhood search

```
1:  procedure GVNS(Instance data $D$, Set of neighborhoods $N$, $k_{min}$, $k_{step}$, $k_{max}$, $l_{max}$, $\varepsilon$)
2:      $S \leftarrow$ INIT_SOLUTION($D$)
3:      while $S$ is infeasible do
4:          $S \leftarrow$ SHAKE($S$, 1, $l_{max}$)
5:          $S \leftarrow$ FIX_SOLUTION($S$)
6:      end while
7:      $S \leftarrow$ CALC_SCHEDULE($S$)
8:      $S \leftarrow$ VND($S$, $N$ , $\varepsilon$)
9:      while the stopping criterion is not met do
10:         $k \leftarrow k_{min}$
11:         while $k \leq k_{max} \wedge$ the stopping criterion is not met do
12:             $S' \leftarrow$ SHAKE($S$, $k$, $l_{max}$)
13:             $S' \leftarrow$ FIX_SOLUTION($S'$)
14:             while $S'$ is infeasible do
15:                 $S' \leftarrow$ SHAKE($S'$, 1, $l_{max}$)
16:                 $S' \leftarrow$ FIX_SOLUTION($S'$)
17:             end while
18:             $S' \leftarrow$ CALC_SCHEDULE($S'$)
19:             $S'' \leftarrow$ VND($S'$, $N$ , $\varepsilon$)
20:             if $f(S'') < f(S)$ then
21:                 $S \leftarrow S''$
22:                 $k \leftarrow k_{min}$
23:             else
24:                 $k \leftarrow k + k_{step}$
25:             end if
26:         end while
27:     end while
28: end procedure
```

6. Experimental evaluation

In this section, we will evaluate the performance of our proposed method, comparing it to the exact solver and ALNS algorithm based on Zhang et al. (2020a). ALNS method that we implemented is quite similar to the one presented in the original article, utilizing the same insertion and removal operators, but it differs from it slightly as it uses Algorithm 2 for generating the initial solution and Algorithm 3 for determining the schedule of vehicles, as we found these procedures to be more suited for the soft windows constraint compared to the ones presented in the original publication. The experiments were conducted on a benchmark dataset presented in Schneider et al. (2014). The instances from this dataset were split into two groups, based on their size. A set of smaller instances with up to 15 customers contains 35 instances, and the set of instances with up to 100 customers consists of 56 instances.

6.1. Experimental setup

The experiments were conducted on a personal laptop with an *Intel i7-10750H* processor and 32GB of RAM, under Ubuntu 20.04 OS. We implemented both algorithms in the C++ programming language and compiled them with *GCC 9.4.0* compiler with the O3 optimization flag. To test our MILP formulation, we used the *CPLEX 20.1.0* commercial solver. In order to obtain the best possible results, we need to carefully tune the parameters of the algorithms. For the ALNS algorithm, we used the same values of parameters as in the original paper. For the full list of parameters and their values, please refer to Zhang et al. (2020a). The proposed GVNS has five parameters that need to be tuned before evaluation. First, there are k_{min}, k_{step}, and k_{max} parameters, which control the number of transformations in each iteration of the algorithm. Parameters k_{min} and k_{max} represent the minimal and maximal number of transformations to be conducted respectively, while k_{step} defines the increment in the number of transformations after each iteration in which the algorithm did not find a solution of better quality compared to the best-known solution. The l_{max} parameter represents the maximal size of the neighborhood used in each individual transformation. Lastly, the ε parameter designates the difference between the new and the old departure times in neighborhood N_1. In order to determine the best possible values for the aforementioned parameters, we used the *iRace*[1] package for the R programming language, with a budget of 2000 tests. In Table 2 we present the best-found values for each GVNS parameter, obtained by iRace. The first column shows the parameter in question, the second column displays the values that each parameter was allowed to assume during the testing, and in the last column, we provide the best-found values for each parameter. The parameters c_1 and c_2 that define the relative importance of distance and penalty in the objective function were both set to 1. For testing purposes, the entire interval in which the delivery takes place was divided into five equal subintervals,

[1] https://cran.r-project.org/web/packages/irace/index.html

each with its own speed multiplier. The speed multipliers were generated at random beforehand and were the same for each instance and each repetition. The exact multiplier values were set to 1, 0.8, 1.5, 1, and 1.2, in that order.

Table 2
The best-found values of each GVNS parameter

Parameter	Allowed values	The best value
k_{min}	$k_{min} \in \{1, 2, 3, 4\}$	$k_{min} = 1$
k_{step}	$k_{step} \in \{1, 2, 3\}$	$k_{step} = 3$
k_{max}	$k_{max} \in \{5, 7, 9, 11, 13, 15, 20\}$	$k_{max} = 7$
l_{max}	$l_{max} \in \{1, 2, 3, 4, 5\}$	$l_{max} = 3$
ε	$\varepsilon \in \{5, 10, 20, 30\}$	$\varepsilon = 5$

6.2. Case study I

In the first case study, we examined the performance of our approach compared to the CPLEX and ALNS. The stopping criterion for all three methods was the CPU time limit, which was set to 60 minutes for CPLEX, and 10 minutes for ALNS and GVNS. To demonstrate the stability of the obtained results, tests for ALNS and GVNS were repeated 30 times with different seed values for the random number generator. As the CPLEX searches for the optimal solution in a deterministic manner, we did not perform any repetitions. As explained in Section 4, the MILP model requires us to create dummy nodes representing refueling stations by replicating actual station nodes to allow multiple visits to refueling stations. The number of times we replicate the station nodes should be carefully determined, as setting this parameter too low could prevent us from finding certain good quality solutions by making them infeasible Froger et al. (2019), while setting it too high can substantially increase the model complexity. For our purposes, we created dummy vertices by multiplying station nodes by 3 for instances with up to 10 customers, and by 6 for instances with 15 customers. These numbers were determined experimentally and provided us with the best results. This particular problem does not affect metaheuristics as we can model multiple visits to refueling stations much more efficiently. This is a clear advantage compared to MILP solvers, as the complexity of the problem is much less influenced by the number of refueling stations.

In Table 3, we present the obtained results for each test instance, considering all three methods. The first three columns display the information about instances, such as name, number of customers, and number of refueling stations. The following two columns represent results obtained by CPLEX, the value of the objective function with a corresponding gap, and the time in which the CPLEX reported a solution. In columns 6-9, we present results obtained by the GVNS algorithm. More precisely, column six contains the average value of the objective function over 30 execution, together with the standard deviation in the parenthesis. Column seven shows the average time in which the best-found solution was reported, again with standard deviation. In columns eight and nine, we presented the value of the objective function for the best and the worst solution found during all 30 repetitions, together with an information on how many times GVNS obtained that particular solution. Columns 10-13 contain the same information as the previous four columns but for the ALNS algorithm. The bolded values designate solutions with the best-known value of the objective function.

As we can see from Table 3, GVNS found the same solution as CPLEX in 23 cases, a better solution in 10 cases, and was outperformed by CPLEX in 2 cases. In cases when GVNS found a better solution, the minimum difference in the objective function value was 0.6499%, the average difference was 7.2314%, and the maximal difference between GVNS and CPLEX solutions was 23.813%. On the other hand, in cases when CPLEX found a better solution than GVRP, the minimum, average, and maximum differences were 0.1286%, 0.5152%, and 0.9017%, respectively. These results clearly demonstrate the superiority of the proposed GVNS in terms of finding good quality solutions, even in a fraction of the time utilized by CPLEX.

As for the comparison between GVNS and ALNS, we can observe that both algorithms managed to find solutions of the same quality for most instances. In fact, there are only two instances in which the best-found solution differed, and in both cases, GVNS provided us with a better solution. Nonetheless, looking at the number of times each algorithm managed to find the best-known solution over 30 repetitions, we can see that GVNS was able to find the best solution more frequently. More precisely, out of 33 instances for which both algorithms obtained the same best-known solution, in 18 cases the count was the same (usually for instances with fewer customers), and in 15 cases the GVRP found the best-known solution more frequently. This is to be expected as both metaheuristics are able to find a good solution in a reasonable amount of time. Furthermore, comparing average values of the objective function shows that GVNS and ALNS performed the same in 18 cases, and GVNS outperformed ALNS in 17 cases, with the maximum difference in average objective function value being 3.72%. Using the *Wilcoxon signed-rank test* with correction, we calculated the $p\text{−}value = 0.00032$, so with the significance level of $\alpha = 0.05$, we conclude that there is a statistically significant difference between the average objective values of the two algorithms. Finally, we can observe that GVNS outperformed ALNS in terms of speed as well. For the sake of fairness, we only compared the average time for instances with the same average objective value. We can see that the difference in average CPU time needed for obtaining the best-known solution was up to 198.2935% (110.7236% on average) in favor of GVNS. With the $p\text{−}value = 0.0000076$ calculated by Wilcoxon signed-rank test with correction and $\alpha = 0.05$, we can conclude that there is a statistically significant difference between average CPU times of GVNS and ALNS.

Table 3
The obtained results for smaller-sized instances

Instance			CPLEX		GVNS				ALNS			
Name	Cust.	Stat.	Obj.(gap %)	Time	Obj. Avg. (std)	Time Avg. (std)	Best (count)	Worst (count)	Obj. Avg. (std)	Time. Avg. (std)	Best (count)	Worst (count)
c101C5	5	3	**247.15 (0)**	14.7	**247.15 (0)**	0.2678 (0.2543)	247.15 (30)	247.15 (30)	**247.15 (0)**	0.8335 (0.3706)	247.15 (30)	247.15 (30)
c101C10	10	5	405.9973 (28.42)	3600	394.5648 (2.2345)	226.8014 (190.6104)	393.7633 (24)	401.13 (3)	395.1857 (2.7192)	304.8418 (181.0817)	393.7633 (16)	401.13 (5)
c103C5	5	2	**165.667 (0)**	6.28	**165.667 (0)**	0.002 (0.0005)	165.667 (30)	165.667 (30)	**165.667 (0)**	0.4668 (0.2785)	165.667 (30)	165.667 (30)
c103C15	15	5	411.1223 (39.18)	3600	371.9071 (0.7842)	102.3415 (156.6711)	371.7 (10)	374.792 (2)	372.2163 (1.1716)	131.6318 (160.1343)	371.7 (9)	374.792 (5)
c104C10	10	4	**273.931 (0.01)**	3600	274.3203 (0.8855)	4.7618 (7.6079)	273.931 (25)	276.267 (5)	274.9849 (3.2652)	12.2628 (8.2824)	273.931 (23)	291.531 (1)
c106C15	15	3	**275.1332 (0)**	425.11	**275.1332 (0)**	31.0991 (23.8411)	275.1332 (30)	275.1332 (30)	**275.1332 (0)**	76.3916 (38.5943)	275.1332 (30)	275.1332 (30)
c202C10	10	5	**243.203 (0.01)**	3600	**243.203 (0)**	9.4267 (12.8681)	243.203 (30)	243.203 (30)	243.6141 (1.2544)	14.4889 (12.8857)	243.203 (27)	247.314 (3)
c202C15	15	5	**376.7893 (9.01)**	3600	**376.7893 (0)**	71.363 (96.8459)	376.7893 (30)	376.7893 (30)	377.5313 (1.9241)	125.9959 (104.8385)	376.7893 (26)	382.3544 (4)
c205C10	10	3	**228.281 (0)**	15	**228.281 (0)**	0.0378 (0.0066)	228.281 (30)	228.281 (30)	**228.281 (0)**	0.5335 (0.2554)	228.281 (30)	228.281 (30)
c208C15	15	4	**300.549 (0)**	1487.95	**300.549 (0)**	69.3003 (83.7877)	300.549 (30)	300.549 (30)	300.6855 (0.7478)	95.3886 (85.6675)	300.549 (29)	304.645 (1)
c206C5	5	4	**236.579 (0)**	92.78	**236.579 (0)**	0.4292 (0.3674)	236.579 (30)	236.579 (30)	**236.579 (0)**	1.3827 (0.801)	236.579 (30)	236.579 (30)
c208C5	5	3	**236.579 (0)**	58.65	**236.579 (0)**	0.3533 (0.3406)	236.579 (30)	236.579 (30)	**236.579 (0)**	1.2281 (0.6301)	236.579 (30)	236.579 (30)
r102C10	10	4	**249.1893 (11.7)**	3600	**249.1893 (0)**	45.5037 (47.2898)	249.1893 (30)	249.1893 (30)	**249.1893 (0)**	128.7177 (65.7105)	249.1893 (30)	249.1893 (30)
r102C15	15	8	424.8538 (36.64)	3600	416.1563 (3.7612)	166.441 (71.0992)	413.9341 (21)	424.8538 (3)	419.1562 (8.2498)	270.225 (99.3421)	413.9341 (16)	453.4532 (1)
r103C10	10	3	**171.7077 (0)**	791.53	173.263 (0)	2.2252 (1.8091)	173.263 (30)	173.263 (30)	173.263 (0)	3.178 (2.0072)	173.263 (30)	173.263 (30)
r104C5	5	3	**136.69 (0)**	47.92	**136.69 (0)**	0.0023 (0.0004)	136.6897 (30)	136.69 (30)	**136.69 (0)**	0.0525 (0.0318)	136.6897 (30)	136.69 (30)
r105C5	5	3	**156.0821 (0)**	20.54	**156.283 (0)**	0.1269 (0.11)	156.283 (30)	156.283 (30)	**156.283 (0)**	1.3124 (0.5844)	156.283 (30)	156.283 (30)
r105C15	15	6	367.0149 (32.88)	3600	342.7246 (15.8141)	264.6581 (161.3221)	336.154 (22)	388.523 (3)	355.751 (21.613)	347.4743 (142.2992)	344.156 (23)	403.643 (1)
r202C5	5	3	**128.777 (0)**	5.14	**128.777 (0)**	0.1173 (0.0882)	128.777 (30)	128.777 (30)	**128.777 (0)**	0.6969 (0.2525)	128.777 (30)	128.777 (30)
r202C15	15	6	**345.2274 (19.84)**	3600	369.7575 (19.1934)	90.5568 (173.7279)	342.991 (3)	414.161 (3)	370.7734 (19.1621)	63.6811 (91.9829)	344.467 (3)	418.536 (1)
r203C5	5	4	**179.056 (0)**	80.03	**179.056 (0)**	0.6123 (0.6922)	179.056 (30)	179.056 (30)	**179.056 (0)**	1.5965 (0.9156)	179.056 (30)	179.056 (30)
r203C10	10	5	**218.213 (0)**	606.9	239.2702 (15.8805)	43.5722 (45.4715)	218.213 (7)	270.894 (3)	244.1671 (16.065)	53.4799 (44.3986)	218.213 (4)	270.894 (5)
r209C15	15	5	**293.2 (10.92)**	3600	295.1307 (5.0064)	133.3656 (169.178)	293.2 (26)	307.68 (4)	297.6247 (6.5201)	191.3249 (146.6217)	293.2 (20)	307.68 (8)
rc102C10	10	4	**423.4943 (8.71)**	3600	**423.4943 (0)**	94.8426 (173.0076)	423.4943 (30)	423.4943 (30)	**423.4943 (0)**	131.4661 (99.7477)	423.4943 (30)	423.4943 (30)
rc103C15	15	5	434.9762 (45.51)	3600	**402.341 (0)**	102.4502 (150.6796)	402.341 (30)	402.341 (30)	403.354 (3.8551)	127.8777 (151.8048)	402.341 (28)	417.536 (2)
rc105C5	5	4	**238.052 (0)**	1372.92	**238.052 (0)**	0.3568 (0.9968)	238.052 (30)	238.052 (30)	**238.052 (0)**	1.1336 (1.5678)	238.052 (30)	238.052 (30)
rc108C5	5	4	**253.931 (11.43)**	3600	**253.931 (0)**	6.5363 (7.9409)	253.931 (30)	253.931 (30)	**253.931 (0)**	11.1432 (8.633)	253.931 (30)	253.931 (30)
rc108C10	10	4	**357.799 (16.35)**	3600	364.059 (22.5867)	83.1012 (145.8542)	345.927 (18)	391.257 (12)	370.7887 (23.7882)	103.11 (147.3792)	345.927 (14)	401.543 (2)
rc108C15	15	5	413.9445 (26.23)	3600	**372.2392 (2.7791)**	217.0653 (176.2477)	370.4737 (19)	376.748 (8)	373.7327 (5.2261)	305.4246 (162.587)	370.4737 (15)	396.4561 (1)
rc201C10	10	4	**310.057 (0)**	982.29	311.1458 (2.4763)	147.9935 (190.0385)	310.057 (25)	316.59 (5)	312.7317 (4.2272)	169.1451 (185.1627)	310.057 (20)	324.046 (2)
rc202C15	15	5	**397.1997 (35.3)**	3600	399.0313 (12.9554)	217.623 (172.0473)	392.912 (10)	446.109 (2)	406.1117 (20.4693)	313.5418 (171.4261)	392.912 (8)	446.109 (6)
rc204C5	5	4	**176.394 (0)**	401.35	**176.394 (0)**	1.7956 (2.2517)	176.394 (30)	176.394 (30)	**176.394 (0)**	4.6005 (2.5321)	176.394 (30)	176.394 (30)
rc204C15	15	7	394.5283 (40.37)	3600	**313.4586 (10.9739)**	85.5175 (66.5296)	310.575 (28)	353.829 (2)	319.2553 (16.6798)	114.2233 (73.6352)	310.575 (23)	353.829 (5)
rc208C5	5	3	**167.983 (0)**	15.59	**167.983 (0)**	0.0064 (0.0014)	167.983 (30)	167.983 (30)	**167.983 (0)**	0.0605 (0.0284)	167.983 (30)	167.983 (30)
rc205C10	10	4	**325.9774 (0)**	1891.52	**325.9774 (0)**	10.7609 (8.401)	325.9774 (30)	325.9774 (30)	**325.9774 (0)**	22.3282 (12.7623)	325.9774 (30)	325.9774 (0)

6.3. Case study II

For the second case study, we compared the performance of ALNS and GVNS on a larger set of instances, each containing 100 customers and 21 refueling stations. CPLEX was not considered for this case study as it was unable to find a feasible solution in a reasonable amount of time. The stopping criterion for both algorithms was the limit of CPU time, which was set to 20 minutes. All the tests were repeated ten times. For this case study, we excluded neighborhood structure N_1 from the VND. The reason for this is that it contributes very little to the performance of the algorithm, as we are going to show in Section 6.4. In Table 4, we presented the results obtained by GVNS and ALNS on a set of larger-sized instances. For practical reasons, we only included the results for 22 instances. The results for all of the tested instances can be found at https://www.mi.sanu.ac.rs/~luka/resources/TDEVRPTW/ resultsLargeAll.pdf . The first column contains the name of the instance. The following four columns show the results obtained by the ALNS method, while the last four columns display results obtained by the GVNS. For each algorithm, we present the objective value, the number of trucks used in a solution, the total distance travelled by vehicles, and the total penalty generated by missing time windows. Each instance has three rows associated with it. The first row shows the average values over ten independent runs, with the standard deviation provided in parenthesis. The second and third rows correspond to the best-found and worst-found solutions, respectively. The bolded fields for each instance designate solutions with the lower objective function value. We first compared best-obtained solutions for both algorithms, observing that GVNS provided a better solution in 52 cases, and in 4 cases both algorithms managed to find a solution of the same quality. The minimal, average, and maximal differences in the solution quality were 0.1439%, 3.3175%, and 17.9404%, respectively. Using the Wilcoxon signed-rank test with the correction, we determined the *p−value* = 0.00000000036, so with the significance level of α = 0.05, we conclude that there is a statistically significant difference between algorithms in regard to the best-obtained solution. Comparing the average quality of the obtained solutions, we observed that GVNS outperformed ALNS in 55 cases and underperformed in just one instance. In situations where GVNS surpassed ALNS, the minimum, average, and maximum difference was 0.1092%, 3.0321%, and 28.9643%, respectively. The difference in average quality solution for the instance in which ALNS provided better results was 0.2877%. Moreover, we performed the Wilcoxon signed-rank test with the correction and obtained *p−value* = 0.0000000001, so we concluded with the significance level of α = 0.05 that GVNS and ALNS were statistically different in terms of average solution value.

Table 4
The obtained results for larger-sized instances

Instance	ALNS				GVNS			
	Objective value	Vehicles	Distance	Penalty	Objective value	Vehicles	Distance	Penalty
c101_21	2208.851 (736.804)	16.9 (1.524)	1537.326 (64.71)	671.526 (686.892)	**1650.005 (294.033)**	16.9 (1.37)	1446.274 (72.064)	203.73 (239.524)
	1481.1	15	1480.96	0.141602	**1374.14**	15	1374	0.141602
	3057.56	18	1613.82	1443.74	**2048.77**	18	1526.09	522.685
c102_21	1516.38 (122.82)	15.9 (0.876)	1465.95 (51.694)	50.433 (74.846)	**1299.874 (68.466)**	15.4 (0.516)	1290.616 (55.461)	9.258 (16.67)
	1421.2	15	1421.2	0	**1187.22**	15	1187.22	0
	1838.37	16	1590.27	248.105	**1421.19**	16	1372.05	49.1373
c103_21	1720.493 (628.596)	16.5 (0.85)	1484.17 (81.953)	236.323 (574.083)	**1541.492 (375.653)**	16.1 (0.876)	1405.434 (97.734)	136.058 (294.194)
	1366.7	16	1366.7	0	**1243.71**	14	1238.97	4.73721
	3484.82	16	1633.98	1850.85	**2561.5**	16	1619.95	941.55
c104_21	1224.964 (32.397)	14.3 (0.675)	1224.964 (32.397)	0(0)	**1200.821 (39.963)**	13.9 (0.994)	1200.521 (40.116)	0.3 (0.949)
	1150.25	14	1150.25	0	**1130.44**	12	1130.44	0
	1262.35	14	1262.35	0	**1242.34**	15	1242.34	0
c105_21	1532.231 (162.645)	16.7 (0.483)	1488.046 (60.186)	44.184 (135.743)	**1400.483 (223.821)**	15.2 (1.135)	1313.184 (91.429)	87.298 (180.895)
	1415.43	16	1415.4	0.026123	**1223.34**	14	1223.34	0
	1963.76	16	1533.3	430.457	**1789.51**	15	1359.05	430.457
c106_21	1469.229 (143.815)	15.8 (1.033)	1437.072 (90.812)	32.156 (73.281)	**1298.467 (71.95)**	15.1 (1.101)	1294.555 (73.825)	3.911 (6.297)
	1348.12	15	1348.12	0	**1231.74**	14	1231.74	0
	1829.71	18	1592.43	237.284	**1468.89**	18	1468.89	0
c107_21	1319.289 (61.044)	14.9 (0.876)	1319.056 (61.069)	0.232 (0.348)	**1246.548 (39.162)**	14.6 (0.516)	1245.015 (43.149)	1.534 (4.33)
	1265.14	15	1264.84	0.297058	**1143.76**	14	1129.91	13.8513
	1470.42	17	1470.42	0	**1281.82**	15	1281.53	0.297058

Table 4
The obtained results for larger-sized instances (Continued)

Instance	ALNS				GVNS			
	Objective value	**Vehicles**	**Distance**	**Penalty**	**Objective value**	**Vehicles**	**Distance**	**Penalty**
c108_21	1307.453 (42.379)	15.5 (0.972)	1306.742 (41.396)	0.711 (1.191)	**1258.402 (32.677)**	15.1 (0.568)	1258.205 (32.862)	0.197 (0.622)
	1252.46	14	1252.46	0	**1212.02**	15	1212.02	0
	1389.39	15	1386.21	3.17517	**1319.02**	16	1319.02	0
c109_21	1268.842 (58.657)	14.6 (0.699)	1263.632 (48.894)	5.211 (16.478)	**1219.18 (59.344)**	14.4 (0.699)	1213.969 (49.431)	5.211 (16.478)
	1181.24	15	1181.24	0	**1121.53**	13	1121.53	0
	1382.98	15	1330.88	52.1069	**1335.75**	15	1283.64	52.1069
c201_21	1271.64 (31.389)	14 (0)	1271.64 (31.389)	0 (0)	**1175.132 (27.082)**	13.6 (0.843)	1175.132 (27.082)	0 (0)
	1196.89	14	1196.89	0	**1121.74**	12	1121.74	0
	1292.61	14	1292.61	0	**1187.97**	14	1187.97	0
c202_21	1121.197 (28.388)	12.6 (0.699)	1121.197 (28.388)	0 (0)	**1086.505 (39.015)**	11.8 (1.033)	1086.505 (39.015)	0 (0)
	1062.38	12	1062.38	0	**1042.9**	10	1042.9	0
	1139.43	13	1139.43	0	**1139.43**	13	1139.43	0
c203_21	1112.963 (23.172)	12.3 (0.823)	1112.963 (23.172)	0 (0)	**1098.578 (31.623)**	11.9 (0.876)	1098.578 (31.623)	0 (0)
	1077.21	12	1077.21	0	**1036.96**	11	1036.96	0
	1134.68	13	1134.68	0	**1134.68**	13	1134.68	0
c204_21	1100.842 (16.01)	11.9 (0.876)	1100.842 (16.01)	0 (0)	**1086.385 (23.129)**	11.5 (0.707)	1086.385 (23.129)	0 (0)
	1065.19	11	1065.19	0	**1047.83**	11	1047.83	0
	1118.63	13	1118.63	0	**1118.63**	13	1118.63	0
c205_21	1095.445 (25.588)	12.5 (0.527)	1095.445 (25.588)	0 (0)	**1073.696 (31.827)**	12 (0.667)	1073.696 (31.827)	0 (0)
	1055.94	12	1055.94	0	**1024.28**	12	1024.28	0
	1116.72	13	1116.72	0	**1116.72**	13	1116.72	0
c206_21	1022.344 (0.055)	10 (0)	1022.344 (0.055)	0 (0)	**1021.228 (1.866)**	10 (0)	1021.228 (1.866)	0 (0)
	1022.24	10	1022.24	0	**1017.86**	10	1017.86	0
	1022.37	10	1022.37	0	**1022.37**	10	1022.37	0
c207_21	1041.199 (9.935)	10.8 (0.422)	1041.199 (9.935)	0 (0)	**1036.506 (14.596)**	10.7 (0.483)	1036.506 (14.596)	0 (0)
	1020.12	10	1020.12	0	**1004.18**	10	1004.18	0
	1046.07	11	1046.07	0	**1046.07**	11	1046.07	0
c208_21	1053.814 (20.642)	11.1 (0.738)	1053.814 (20.642)	0 (0)	**1049.379 (25.318)**	11.1 (0.738)	1049.379 (25.318)	0 (0)
	1022.81	10	1022.81	0	**1006.47**	10	1006.47	0
	1077.61	12	1077.61	0	**1077.61**	12	1077.61	0
r101_21	1892.42 (14.961)	19.9 (0.316)	1686.072 (12.654)	206.344 (10.719)	**1872.506 (27.849)**	19.9 (0.316)	1682.58 (22.528)	189.923 (25.032)
	1859.76	19	1652.75	207.011	**1828.89**	20	1643.85	185.041
	1901.18	20	1689.78	211.395	**1901.18**	20	1689.78	211.395
r102_21	1808.176 (50.381)	18.9 (0.316)	1651.975 (25.402)	156.2 (25.665)	**1777.659 (68.93)**	19 (0.471)	1640.069 (66.378)	137.587 (48.62)
	1671.53	18	1586.02	85.51	**1608.52**	18	1514.8	93.717
	1832.58	19	1664.66	167.915	**1832.58**	19	1664.66	167.915
r103_21	1501.224 (48.232)	16.9 (0.876)	1439.245 (36.443)	61.977 (18.068)	**1481.169 (57.683)**	16.7 (0.949)	1421.912 (43.104)	59.258 (21.4)
	1428.19	16	1388.39	39.7954	**1398.92**	16	1359.96	38.9554
	1556.37	18	1472.89	83.4774	**1556.37**	18	1472.89	83.4774
r104_21	1222.17 (15.452)	13.7 (0.483)	1199.449 (13.599)	22.72 (11.432)	**1220.57 (18.471)**	13.7 (0.483)	1198.065 (15.666)	22.504 (11.24)
	1183.26	13	1180.89	2.3715	**1172.74**	13	1170.15	2.58221
	1230.5	14	1203.43	27.0691	**1230.5**	14	1203.43	27.0691
r105_21	1840.121 (21.286)	19 (0)	1680.522 (18.17)	159.598 (10.5)	**1765.674 (56.647)**	18.9 (0.316)	1626.078 (28.465)	139.601 (30.432)
	1790.13	19	1631	159.127	**1609.11**	18	1549.59	59.5186
	1858.86	19	1687.65	171.21	**1788.44**	19	1633.03	155.417

6.4. Parameter analysis

In this section, we will explore the relative influence of each neighborhood structure and each GVNS parameter on the solution quality. For this purpose, we utilized a gradient boosting regressor, provided in *Extreme Gradient Boosting* (**XGBoost**) library for Python programming language[2] . In Fig. 3, we present the relative influence of each neighborhood structure considered in the local search step. Neighborhoods N_9 and N_{10} were not tested as they are used only in the shaking step. It is important to notice that the choice of neighborhood structures is not the only factor contributing to the overall performance of the VND procedure, as the ordering of selected structures can have a substantial impact on solution quality. Nonetheless, the ordering of neighborhood structures was not examined in this paper. As we can see from Fig. 3, the neighborhood structures that contributed the most to the overall performance of the algorithm were N_2 and N_3, both of which are customer-based neighborhoods. This was to be expected, especially for neighborhood N_3, which is the only neighborhood that can change the number of customers in a vehicle. Neighborhood structure N_4 is also a customer-based neighborhood that contributes a lot to the solution quality, but it has a lower relative influence as it can be simulated by using N_2. Removing a visit to a refueling station (N_6) and exchanging one recharging station for another (N_8) had a moderate influence on solution quality while moving the visit to the recharging station to some other position (N_7) had less influence, probably because it is more likely to result in infeasible solutions. Surprisingly, exchanging customers between two vehicles (N_5) had a somewhat low influence on the solution quality, but this can be due to the fact that it can be simulated using the N_3 neighborhood structure. Finally, neighborhood structure N_1 had virtually no contribution to the solution quality whatsoever. This is most likely because the scheduling procedure presented in Algorithm 3 is capable of finding good enough departure times, so there is rarely a need for further optimization. Moreover, this neighborhood structure can have a negative impact on the overall performance of the algorithm when applied to large instances, as it consumes much processing power without contributing to the solution quality. In Fig. 4, we present the relative importance of each of the five GVNS parameters used in our method. As evident, parameter k_{max} is the most important parameter, as it defines the number of transformations during the shaking step, so if not set properly together with the l_{max} parameter, the algorithm can get stuck in local optimum as some of the perspective solutions might be out of reach for the algorithm. Not surprisingly, the ε parameter had the least influence on the performance of GVNS, as it is used only to define the difference in departure times for neighborhood structure N_1, and as we previously concluded, neighborhood N_1 has the lowest impact on the performance of the algorithm.

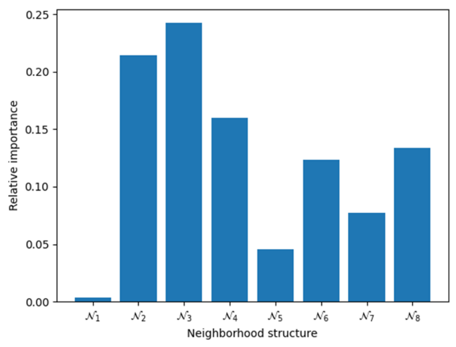

Fig. 3. Relative influences of each neighbourhood structure

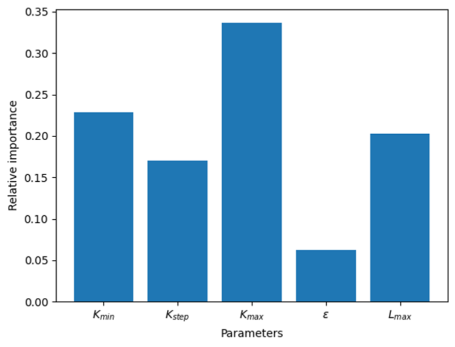

Fig. 4. Relative influences of each GVNS parameter

7. Conclusion

In this paper, we presented the electric vehicle routing problem with soft time windows and time-dependent speeds. The goal was to minimize the total distance travelled by all vehicles and to minimize the penalty cost for missing time windows. We proposed a MILP formulation, and a General Variable Neighborhood Search algorithm for finding solutions to this problem. The experimental evaluation was performed on a set of benchmark instances from the literature, which was split into two subsets based on their size. We compared our approach to the Adaptive Large Neighborhood Search algorithm with much success. Several real-world aspects of this problem were not considered, each presenting an opportunity for future research. For example, partial battery recharge is a common attribute for EVRP but was not considered in this paper. Furthermore, more realistic consumption models and objective functions can be developed, which would further increase the applicability of the problem. Finally, recharging stations differ from one another. In practice, each recharging station has a certain limit on how many vehicles it can serve at any moment, different recharging rates, and potentially different prices. These are all important aspects that can motivate further research in the area of EVRP.

Acknowledgements

This work was supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, Agreement No. 451-03-9/2021-14/200029 and by the Science Fund of the Republic of Serbia, Grant ”AI4TrustBC: Advanced

[2] https://xgboost.readthedocs.io

Artificial Intelligence Techniques for Analysis and Design of System Components Based on Trustworthy BlockChain Technology".

References

Abousleiman, R., & Rawashdeh, O. (2015). Energy consumption model of an electric vehicle. *2015 IEEE transportation electrification conference and expo (ITEC)*, (pp. 1–5).

Affi, M., Derbel, H., & Jarboui, B. (2018). Variable neighbourhood search algorithm for the green vehicle routing problem. *International Journal of Industrial Engineering Computations, 9*, 195–204.

Agency, E. E. (2020). Annual European Union greenhouse gas inventory 1990–2018 and inventory report 2020. *Annual European Union greenhouse gas inventory 1990–2018 and inventory report 2020*. Retrieved from https://www.eea.europa.eu/publications/european-union-greenhouse-gas-inventory-2020

Andelmin, J., & Bartolini, E. (2019). A multi-start local search heuristic for the green vehicle routing problem based on a multigraph reformulation. *Computers & Operations Research, 109*, 43–63.

Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-e-hashem, S. M., & Dulebenets, M. A. (2022). Transformation and Linearization Techniques in Optimization: A State-of-the-Art Survey. *Mathematics, 10*, 283.

Bektaş, T., & Laporte, G. (2011). The pollution-routing problem. *Transportation Research Part B: Methodological, 45*, 1232–1250.

Bruglieri, M., Pezzella, F., Pisacane, O., & Suraci, S. (2015). A variable neighbourhood search branching for the electric vehicle routing problem with time windows. *Electronic Notes in Discrete Mathematics, 47*, 221–228.

Chen, Y., Wu, G., Sun, R., Dubey, A., Laszka, A., & Pugliese, P. (2020). A review and outlook of energy consumption estimation models for electric vehicles. *arXiv preprint arXiv:2003.12873*.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26*, 29–41.

Erdoğan, S., & Miller-Hooks, E. (2012). A green vehicle routing problem. *Transportation research part E: logistics and transportation review, 48*, 100–114.

Felipe, Á., Ortuño, M. T., Righini, G., & Tirado, G. (2014). A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research Part E: Logistics and Transportation Review, 71*, 111–128.

Froger, A., Mendoza, J. E., Jabali, O., & Laporte, G. (2019). Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. *Computers & Operations Research, 104*, 256–294.

Gendreau, M., Ghiani, G., & Guerriero, E. (2015). Time-dependent routing problems: A review. *Computers & operations research, 64*, 189–197.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research, 13*, 533–549.

Goeke, D., & Schneider, M. (2015). Routing a mixed fleet of electric and conventional vehicles. *European Journal of Operational Research, 245*, 81–99.

Golden, B. L., Raghavan, S., Wasil, E. A., & others. (2008). *The vehicle routing problem: latest advances and new challenges* (Vol. 43). Springer.

Hansen, P., Mladenović, N., & Urošević, D. (2006). Variable neighbourhood search and local branching. *Computers & Operations Research, 33*, 3034–3045.

Hiermann, G., Hartl, R. F., Puchinger, J., & Vidal, T. (2019). Routing a mix of conventional, plug-in hybrid, and electric vehicles. *European Journal of Operational Research, 272*, 235–248.

Hiermann, G., Puchinger, J., Ropke, S., & Hartl, R. F. (2016). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research, 252*, 995–1018.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European journal of operational research, 144*, 379–396.

IEA. (2019). Global Energy & CO2 Status Report 2019. *Global Energy & CO2 Status Report 2019*. Retrieved from https://www.iea.org/reports/global-energy-co2-status-report-2019

Keskin, M., & Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. *Transportation research part C: emerging technologies, 65*, 111–127.

Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science, 220*, 671–680.

Kovač, N., Davidović, T., & Stanimirović, Z. (2018). Variable neighbourhood search methods for the dynamic minimum cost hybrid berth allocation problem. *Information Technology and Control, 47*, 471–488.

Li, Y., Lim, M. K., Tan, Y., Lee, Y., & Tseng, M.-L. (2020). Sharing economy to improve routing for urban logistics distribution using electric vehicles. *Resources, Conservation and Recycling, 153*, 104585.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., & Laporte, G. (2019a). The green mixed fleet vehicle routing problem with partial battery recharging and time windows. *Computers & Operations Research, 101*, 183–199.

Macrina, G., Laporte, G., Guerriero, F., & Di Puglia Pugliese, L. (2019b). An energy-efficient green-vehicle routing problem with mixed vehicle fleet, partial battery recharging and time windows. *European Journal of Operational Research, 276*, 971–982.

Malandraki, C., & Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science, 26*, 185–200.

Matijević, L. (2022). Variable Neighbourhood Search for Multi-label Feature Selection. *International Conference on Mathematical Optimization Theory and Operations Research*, (pp. 94–107).

Mavrovouniotis, M., Ellinas, G., & Polycarpou, M. (2018). Ant colony optimization for the electric vehicle routing problem. *2018 IEEE Symposium series on computational intelligence (SSCI)*, (pp. 1234–1241).

Mavrovouniotis, M., Li, C., Ellinas, G., & Polycarpou, M. (2019). Parallel ant colony optimization for the electric vehicle routing problem. *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, (pp. 1660–1667).

Mladenović, N., & Hansen, P. (1997). Variable neighbourhood search. *Computers & operations research, 24*, 1097–1100.

Montoya, A., Guéret, C., Mendoza, J. E., & Villegas, J. G. (2017). The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological, 103*, 87–110.

Normasari, N. M., Yu, V. F., Bachtiyar, C., & others. (2019). A simulated annealing heuristic for the capacitated green vehicle routing problem. *Mathematical Problems in Engineering, 2019*.

Peng, B., Zhang, Y., Gajpal, Y., & Chen, X. (2019). A memetic algorithm for the green vehicle routing problem. *Sustainability, 11*, 6055.

Ren, X., Huang, H., Feng, S., & Liang, G. (2020). An improved variable neighbourhood search for bi-objective mixed-energy fleet vehicle routing problem. *Journal of Cleaner Production, 275*, 124155.

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

Schneider, M., Stenger, A., & Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation science, 48*, 500–520.

Toth, P., & Vigo, D. (2002). *The vehicle routing problem.* SIAM.

Toth, P., & Vigo, D. (2014). *Vehicle routing: problems, methods, and applications.* SIAM.

Urazel, B., & Keskin, K. (2021). A Hybrid Solution Approach for Electric Vehicle Routing Problem with Soft Time-Windows. *El-Cezeri, 8*, 994–1006.

Wang, L., & Lu, J. (2019). A memetic algorithm with competition for the capacitated green vehicle routing problem. *IEEE/CAA Journal of Automatica Sinica, 6*, 516–526.

Wang, L., Gao, S., Wang, K., Li, T., Li, L., & Chen, Z. (2020). Time-dependent electric vehicle routing problem with time windows and path flexibility. *Journal of Advanced Transportation, 2020*.

Wang, Y., Assogba, K., Fan, J., Xu, M., Liu, Y., & Wang, H. (2019). Multi-depot green vehicle routing problem with shared transportation resource: Integration of time-dependent speed and piecewise penalty cost. *Journal of Cleaner Production, 232*, 12–29.

Xiao, Y., & Konak, A. (2015). A simulating annealing algorithm to solve the green vehicle routing & scheduling problem with hierarchical objectives and weighted tardiness. *Applied Soft Computing, 34*, 372–388.

Xiao, Y., & Konak, A. (2017). A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem. *Journal of Cleaner Production, 167*, 1450–1463.

Yavuz, M., & Çapar, I. (2017). Alternative-fuel vehicle adoption in service fleets: Impact evaluation through optimization modeling. *Transportation Science, 51*, 480–493.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighbourhood search algorithm. *Expert Systems with Applications, 37*, 678–687.

Zhang, R., Guo, J., & Wang, J. (2020a). A time-dependent electric vehicle routing problem with congestion tolls. *IEEE Transactions on Engineering Management*.

Zhang, S., Gajpal, Y., & Appadoo, S. S. (2018a). A meta-heuristic for capacitated green vehicle routing problem. *Annals of Operations Research, 269*, 753–771.

Zhang, S., Gajpal, Y., Appadoo, S. S., & Abdulkader, M. M. (2018b). Electric vehicle routing problem with recharging stations for minimizing energy consumption. *International Journal of Production Economics, 203*, 404–413.

Zhang, S., Zhang, W., Gajpal, Y., & Appadoo, S. S. (2019). Ant colony algorithm for routing alternate fuel vehicles in multi-depot vehicle routing problem. In *Decision science in action* (pp. 251–260). Springer.

Zhang, W., Gajpal, Y., Appadoo, S., Wei, Q., & others. (2020b). Multi-depot green vehicle routing problem to minimize carbon emissions. *Sustainability, 12*, 3500.

Zhang, X., Yao, J., Liao, Z., & Li, J. (2018c). The electric vehicle routing problem with soft time windows and recharging stations in the reverse logistics. *International Conference on Management Science and Engineering Management*, (pp. 171–182).