Hindawi
Journal of Advanced Transportation
Volume 2024, Article ID 8753106, 15 pages
https://doi.org/10.1155/2024/8753106

*Research Article*

# Modification of the Clarke and Wright Algorithm with a Dynamic Savings Matrix

**Jan Fikejz**,[1] **Markéta Brázdová**,[2] **and Ľudmila Jánošíková**[3]

[1]*Faculty of Electrical Engineering and Informatics, University of Pardubice, Pardubice 532 10, Czech Republic*
[2]*Faculty of Transport Engineering, University of Pardubice, Pardubice 532 10, Czech Republic*
[3]*Faculty of Management Science and Informatics, University of Žilina, Žilina 010 26, Slovakia*

Correspondence should be addressed to Jan Fikejz; jan.fikejz@upce.cz

Received 1 May 2023; Revised 29 February 2024; Accepted 1 March 2024; Published 29 March 2024

Academic Editor: Tomio Miwa



The goods collection and delivery process often relates to distribution logistics problems. The task is to deliver goods from warehouses to customers under specific circumstances. Efforts to optimize the process are largely aimed at reducing overall costs of goods transportation. Among the prominent algorithms for solving the basic type of the delivery (or collection) problem, which includes a single depot and a homogeneous vehicle fleet, is the algorithm developed by Clarke and Wright in 1964. This algorithm minimizes transportation costs by maximizing the savings achieved through merging multiple routes into one. This paper primarily aims to solve the pickup and delivery problem where the goods must be delivered and empty packaging collected in a single process. The request of a customer can be routed from the depot or from another customer. Similarly, the destination of the request may be the depot or another customer. Unlike the original version of the Clarke and Wright algorithm, the initial routes are created to satisfy delivery orders, and therefore, the same customer can occur in multiple routes. Consequently, a situation may arise in which two routes containing one or more common vertices must be combined during the calculation. Furthermore, these vertices need not be the outermost vertices of the routes. This situation cannot be addressed by using the original version of the Clarke and Wright algorithm, and that is why we propose its modification. Merging routes through inner vertices means that the cost savings depend on the configurations of the routes, and therefore, they cannot be calculated a priori. Instead, the dynamic savings matrix must be used.

## 1. Introduction

The transportation of goods and people is an integral part of the everyday world, involving significant expenditures on vehicle fuels and maintenance each day. Clearly, this sector provides ample opportunities for cost optimization through advancements in technology, transportation network infrastructure, and sophisticated planning. In practical terms, the challenges to be addressed include the efficient transport of goods under specific conditions, aiming for minimal costs, the shortest possible time, or maximizing profits. The utilization of mathematical methods and optimization approaches proves convenient for addressing these complex problems.

Two types of problems are frequently encountered in goods logistics and distribution. The first one is the vehicle routing problem (VRP), where goods are transported from/to a depot, such as goods from warehouses to customers and packaging material from customers back to the warehouses. The other problem is the pickup and delivery problem (PDP), where goods are transported between customers. In distribution, multiple vehicle types are typically involved. The problems usually involve many other parameters, including time constraints (e.g., limited duration of a route, time windows for goods delivery to customers, mandatory driving breaks for rest, and maximum driving time for a vehicle), maximum route length, maximum number of customers serviced within a route, or maximum load dimensions.

The distribution problems are too complex to be tractable using a single method for their solution. Instead,

a suitable combination of methods and procedures must be applied. A heuristic approach is usually adopted when addressing complex problems. Many real-world problems have been successfully solved by operations research methods, allowing costs to be reduced by 5% to 20% [1], with the additional advantage of reducing greenhouse gas production, mainly $CO_2$.

This paper addresses a problem where the distribution processes encountered in both problems mentioned above occur; i.e., the goods are transported from/to a depot as well as among customers. Both good unloading and loading can occur at the customer's site. The goods loading/unloading costs and time demands are known as well. The objective is to identify vehicle routes that minimize the total goods delivery costs. A specific food distribution problem was chosen to validate the algorithm within a case study. The goods flow consists largely (~80%) of transports from the depot to various customers, mostly food sellers. The remaining fraction of the flow consists of goods transport between customers and collection from the customers. Figure 1 illustrates the flow of goods.

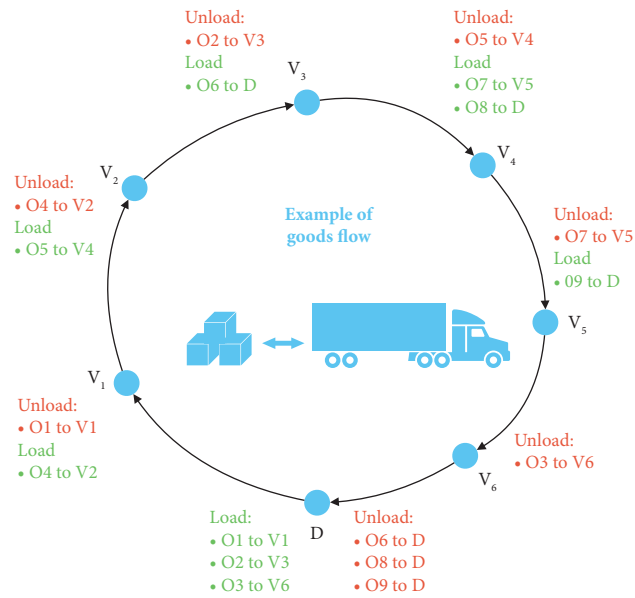


Figure 1: Example of goods flow.

## 2. Literature Review

We commence with the formal description of the classical VRP involving a single depot and a homogeneous vehicle fleet. The problem can be effectively represented using graph theory. The classical VRP is defined on a directed graph $G = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is the vertex set and $A = \{(i, j): i, j \in V, i \neq j\}$ is the arc set. Vertex 0 represents a depot where a set of identical vehicles, each with a capacity $Q$, is available. Each customer situated at vertex $i \in V\backslash\{0\}$ is associated with a nonnegative demand $q_i \leq Q$. A cost matrix $c_{ij}$ is defined on A. When the cost matrix is symmetric, i.e., $c_{ij} = c_{ji}$ for all $i, j \in V$, it is common practice to define the problem on an undirected graph $G = (V, E)$ where $E = \{[i, j]: i, j \in V, i < j\}$ is the edge set. The travel cost $c_{ij}$ may represent the length or travel time along the edge $[i, j]$. The objective is to determine a set of vehicle routes starting and ending at the depot, ensuring that each customer is visited by precisely one vehicle, the total demand of any route does not exceed $Q$, and the total routing cost is minimized [2].

The simplest VRP variant involves one depot and one vehicle type with a known capacity. The aim is to identify vehicle routes meeting these constraints while minimizing the goods delivery. In the simplest case, the cost is directly proportional to the distance travelled (sometimes it is immediately this distance). However, real-world situations often necessitate the application of more complex VRP variants involving multiple vehicle types and/or multiple depots. An overview of the various problem modifications and methods to address them can be found in the monograph [3].

Numerous heuristic approaches exist to tackle general VRPs, aiming to find a suboptimal problem solution. The methods not only differ in their selected approaches but also in the specific class of problems for which they were devised.

Pioneering work was conducted by Dantzig and Ramser in 1959, focusing on the identification of fuel tank truck routes between a large-volume terminal and service stations. They achieved a near-optimal solution using a method based on a linear programming formulation [4]. A well-known and frequently used heuristic algorithm for solving vehicle routing problems was developed by Clarke and Wright in 1964 [5]. This algorithm is designed to solve the classical VRP as defined previously. It is a constructive heuristic that gradually merges simple tours into more complex ones by applying a savings criterion, referring to the distance savings achieved by combining two routes into one. The advantages of the algorithm include its simplicity, short computing time, and easy implementation.

Since its development in 1964, the Clarke and Wright algorithm has undergone many improvements and modifications for specific problems, such as the multidepot problem [6] or scenarios where vehicles do not return to the depot [7]. Various modifications to the original saving criterion have been proposed in the literature [8]. Gaskell [9] and Yellow [10] parameterized the formula to enhance the algorithm's exploration ability. Paessens [11] introduced another term to allow merging customers with very different distances from the depot, while Altınel and Öncan [12] enhanced the criterion by incorporating customers' demands in addition to distance. Other modifications involve combining the algorithm with Monte Carlo simulation [13, 14]. Plenty of practical applications of the Clarke and Wright algorithm have been reported in the literature. Sarmah et al. [15] successfully applied the algorithm to the solid municipal waste collection problem for 15 municipality districts of Bilaspur (India). Jiang et al. [16] focused on optimizing the vehicle routing for baggage pickup in airport terminals. Noteworthy results were obtained when comparing the Clarke and Wright algorithm with other methods. Destyanto et al. [17], for instance, compared nearest

neighbour heuristics with the Clarke and Wright algorithm in the medicinal drug distribution area, demonstrating that the Clarke and Wright algorithm is the most effective tool for tackling this distribution problem.

Metaheuristics represent a distinct approach to solving VRPs, functioning as heuristic methods that form a superstructure above other heuristics types. Well-known metaheuristics include genetic algorithms, tabu search, simulated annealing, and ant colony optimization. These methods emulate natural phenomena, such as animal behaviour or genetic processes.

Metaheuristics, either independently or in conjunction with the Clarke and Wright algorithm, have been applied to solve problems across various application areas. For instance, Escobar et al. [18] utilized a hybrid approach to the VRP; here, they initially found the problem solution using the Clarke and Wright algorithm and then optimized it using tabu search metaheuristics. Addressing the mail delivery problem, Huang et al. [19] developed two integer models for the delivery service. Their heuristic algorithm is a combination of the Clarke and Wright algorithm with a tabu search metaheuristic. Zhai et al. [20] focused on combining the Clarke and Wright algorithm with ant colony optimization. Berghida and Boukra [21] and Pan et al. [22] also employed a combination of metaheuristics to address the VRP.

Heuristics and metaheuristics prove to be practical approaches for addressing real-world pickup and delivery problems [23]. A common technique involves creating an initial set of routes using an insertion heuristic. Customers are gradually inserted into the routes based on a time-spatial proximity criterion. Subsequently, either local search or a metaheuristic is applied to improve the solution. A simpler variant of the problem, featuring one depot but simultaneous chilled food pickup and delivery, was addressed by Ji et al. [24] using their own heuristics based on a genetic algorithm. Chun-Hua et al. [25] also developed a genetic algorithm to handle simultaneous pickups and deliveries, considering time windows and setting penalties for delayed delivery. Chen and Fang [26] approached the simultaneous pickup and delivery problem using a two-layer discrete particle swarm optimization, with the first phase dedicated to vehicle assignment and the second phase focused on identifying optimal routes. Yu and Lin [27] took a different approach, successfully employing simulated annealing metaheuristics for the PDP. Sathyanarayanan et al. [28] used a modified genetic algorithm to solve the combined problem with multiple depots and stochastic vehicle routing.

While the literature overview illustrates the frequent use of the Clarke and Wright algorithm and its combination with metaheuristics for routing problems with various parameters and constraints, to our knowledge, the Clarke and Wright algorithm has not been applied to solve the PDP. In this paper, we aim to fill this gap by proposing a new heuristic method based on the Clarke and Wright algorithm. This choice is motivated by several factors. The most important is the algorithm's computation speed. Metaheuristics may provide a better solution, but the computation takes a long time. When faced with a problem that occurs in operational management, we need to find a solution quickly even at the expense of its lower quality. Another advantage of the Clarke and Wright algorithm is that it makes easy to add additional constraints to the problems, such as time windows for handling goods, a maximum route travel time, a maximum number of stops on the route, mandatory breaks for the driver's rest, and load temperature regime combinations. This algorithm can be relatively easily implemented and modified.

## 3. Materials and Methods

The problem to be solved is a single depot PDP with a heterogeneous vehicle fleet. Customers' requests are specified through delivery orders. The delivery order data include the dispatch and destination locations of the goods, along with the quantity of goods to be delivered. A customer, as well as a depot, may serve as the pickup point for one request and, simultaneously, as the delivery point for another request. Multiple orders can have the same pickup points and/or destinations. Both unloading and loading operations are conceivable at a customer's site. Moreover, the cost of loading and unloading operations at each pickup and delivery location is also given, affecting the total cost to be minimized. We propose a solution method for the basic version of the problem where time windows are not specified.

Although we will solve the problem heuristically, a mathematical programming formulation can be useful for a better understanding of the problem and its features. The formulation stems from the graph model of the problem [23]. Let us define a network in which nodes correspond to the depot, pickup points, and delivery points. Different nodes may represent the same geographical location. If the request is dispatched from the depot, an artificial node corresponding to the pickup operation is added to the network. Delivery to the depot is modelled in a similar way. Let $r$ be the number of requests. We denote the set of pickup nodes by $P = \{1, \ldots, r\}$ and the set of delivery nodes by $D = \{r + 1, \ldots, 2r\}$. Furthermore, we define $N = P \cup D$. Let $V = N \cup \{0\}$ be the set of all nodes inclusive the depot and $A = V \times V$ be the set of all feasible arcs. Request $i$ consists of transporting $q_i$ units from node $i$ to $r + i$. Consequently, let $l_i = q_i$ and $l_{r+i} = -q_i$. Let $K$ be a set of vehicles, $Q_k$ be the capacity of vehicle $k$, and $c_{ijk}$ be the travel cost between nodes $i$ and $j$ related to vehicle $k$.

The mathematical model involves two types of decision variables: binary variables $x_{ijk}$ that equal to 1 if vehicle $k$ travels through the arc $(i, j) \in A$ and 0 otherwise and variables $y_{ik}$ giving the load of vehicle $k$ after the service at node $i$ have been completed. The linear programming formulation is as follows:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{\mathrm{ijk}} x_{\mathrm{ijk}}, \tag{1}$$

subject to

$$\sum_{j\in N} x_{ijk} = \sum_{j\in N} x_{j,r+i,k}, \quad \text{for } k \in K, i \in P, \tag{2}$$

$$\sum_{j\in N} x_{ijk} = \sum_{j\in N} x_{j,r+i,k}, \quad \text{for } k \in K, i \in P, \tag{3}$$

$$\sum_{i\in V} x_{ijk} = \sum_{i\in V} x_{jik}, \quad \text{for } k \in K, j \in N, \tag{4}$$

$$y_{ik} + l_i \le y_{jk} + M\left(1 - x_{ijk}\right), \quad \text{for } y_{ik} + l_i \le y_{jk} + M\left(1 - x_{ijk}\right), \tag{5}$$

$$y_{ik} + l_i + M\left(1 - x_{ijk}\right) \ge y_{jk}, \quad \text{for } k \in K, i \in V, j \in V, i \ne j, \tag{6}$$

$$l_i \le y_{ik} \le Q_k, \quad \text{for } k \in K, i \in P, \tag{7}$$

$$0 \le y_{ik} \le Q_k - l_i, \quad \text{for } k \in K, i \in D, \tag{8}$$

$$y_{0k} = 0, \quad \text{for } k \in K, \tag{9}$$

$$x_{ijk} \in \{0, 1\}, \quad \text{for } k \in K, i \in V, j \in V, i \ne j. \tag{10}$$

The objective function (1) minimizes the total travel cost. Constraints (2) ensure that each pickup location is visited. Constraints (3) impose that the delivery location is visited if the pickup location is visited and that the visit is performed by the same vehicle. Constraints (4) are the flow conservation constraints. Constraints (5) and (6) adjust the vehicle load during a route and, simultaneously, prevent a route from being separated into subtours that do not contain a depot. Constraints (7) and (8) are the capacity intervals at pickup and delivery nodes. The initial vehicle load is imposed by (9). Finally, constraints (10) define binary variables.

*3.1. The Clarke and Wright Algorithm.* The Clarke and Wright algorithm operates on a graph where vertices correspond to individual customers, not the requests. The formal description of the graph was provided at the beginning of the Literature Review Section.

The algorithm starts with the initial solution, consisting of elementary routes. An elementary route, in the form of depot $\longrightarrow$ serviced vertex $\longrightarrow$ depot, is created for each serviced vertex. This initial solution is then modified, and routes are merged to achieve the maximum savings in the total cost. The savings, denoted as $\lambda_{ij}$, indicate how much the cost is reduced by combining two routes into one (Figure 2), as expressed by the equation:

$$\lambda_{ij} = c_{i0} + c_{0j} - c_{ij}. \tag{11}$$

Since the savings remain constant throughout the algorithm, they can be computed a priori. The savings are recorded in a matrix of dimension $(n+1) \times (n+1)$. In each iteration, the maximum element in the savings matrix must be; it determines the routes that can be joined, provided that the problem constraints are respected. The basic constraint relates to vehicle capacity: two routes can be merged into one only if the vehicle used can carry all the goods delivered via the two routes. After each step of the algorithm, more complex routes visiting two or more vertices may emerge. The routes can only be joined through their outermost vertices, the first after the depot or the last one before the depot. Connecting additional routes to vertices within the existing routes is not possible.

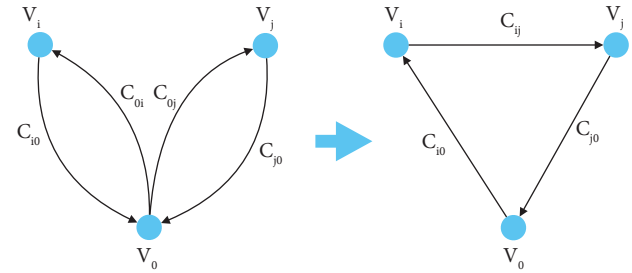


Figure 2: Basic principle of the Clarke and Wright algorithm.

*3.2. Modifications of the Clarke and Wright Algorithm.* We modify the Clarke and Wright algorithm in the following ways:

(1) The orders can be directed to/from any vertex

(2) A customer may be visited by multiple vehicles, meaning one vertex may occur in multiple routes

(3) Routes can be merged at any common vertex

(4) The savings resulting from joining routes depend on the configuration of the routes, necessitating changes during the algorithm run and repeated calculation, leading to a dynamic savings matrix

The algorithm is adapted to situations where the routes obtained by the initial solution can lead to the same vertices. In contrast to the basic Clarke and Wright algorithm variant, the initial solution is based on the delivery orders rather than the customers. Each route in the initial solution satisfies one order. The goods for one order can be managed by one vehicle (this need not be tested). However, multiple orders can be directed to the same network vertex. Thus, a situation can occur where two routes containing one or more identical vertices must be joined during the computation. This is a problem that cannot be solved using the original Clarke and Wright algorithm and its numerous modifications encountered in literature [8–12].

The calculation of savings is a next modification of the algorithm. In the original version of the Clarke and Wright algorithm, the savings are calculated in the initialization phase based on graph edge weighting and are independent of the configuration of the routes being combined. In the problem addressed here, savings are affected by multiple factors. They are calculated from the costs of the routes being combined, including a fixed cost component and a variable cost

component. These savings depend not only on the distance but also on the number of units of goods to be loaded/unloaded and other factors, such as the cost of having the vehicle driven to the ramp or the cost of the vehicle idle time during the loading/unloading operations. Furthermore, the savings accruing from route joining depend on the order of the vertices on the resulting route. Hence, the savings accruing from the potential combination of the newly created route with the existing routes must be calculated after each route combination performed. This affects the savings matrix in such a way that the savings for the newly created route are calculated and stored in a new row and column, while the rows and columns for the routes from which the new route was formed are eliminated from the matrix. In this way, the saving matrix becomes a dynamic matrix.

*3.3. Initial Solution.* Each order specifies the pickup vertex (dispatch site) and the delivery vertex (destination). The dispatch site can be the depot (typically) or any other vertex, and the same applies to the destination. In addition, the amount of goods to be delivered is defined. The orders are categorized into three groups. The initial solution is constructed with respect to these groups:

(1) Group 1 includes delivery orders with the depot as the dispatch site. The initial route for the order is depot ⟶ delivery vertex ⟶ depot.

(2) Group 2 encompasses pickup orders where the depot is the destination point. The initial route is depot ⟶ pickup vertex ⟶ depot.

(3) Group 3 involves orders where the depot is neither the dispatch site nor the destination site. Instead, the goods are transported between two nondepot vertices. The initial route is depot ⟶ pickup vertex ⟶ delivery vertex ⟶ depot.

*3.4. Saving Matrix.* As in the Clarke and Wright algorithm, the saving matrix must be set up first. The saving matrix has dimensions of the number of orders × number of orders, and it is not symmetric. Each matrix row and column represents one route, with every route originating and terminating at the depot. The matrix elements include 2 items: the savings and the potentially newly created route. The initial saving matrix is shown in Table 1.

The following symbols will be used to describe a route:

(i) $m$ is the number of vertices within a route

(ii) $v_{tk}$ is the $k$-th vertex on the route t

(iii) $v_{t1}$ is the first vertex on the route $t$ representing the depot, which is the vehicle's starting point

(iv) $v_{tm}$ is the last vertex on the route $t$ representing the depot, which is the vehicle's finish point

*3.5. Saving Calculation.* The value of the savings $\lambda_{ij}$ is obtained from the costs of the joined routes $i$ and $j$, as shown in the equation:

$$\lambda_{ij}\,(\text{savings}) = \text{cost\_of\_route}_i + \text{cost\_of\_route}_j - \text{cost\_of\_route}_{\text{new}}. \quad (12)$$

*3.6. Cost of a Route.* The cost of a route comprises four items: the cost of vehicle travel along the route, the cost of servicing the vertices on the route, the waiting cost, and the vehicle deployment cost. The total time of the route must also be determined for the total cost calculation. Therefore, the servicing times in the vertices and the time of travel between the vertices must also be determined.

Cost factors are the same for each route:

(i) cost_v is the vehicle deployment cost (price)

(ii) cost_t is the travel cost (price/km)

(iii) cost_w is the waiting cost (hourly price)

*3.7. Distance Matrix.* This matrix includes the distances $d_{ij}$ (lengths of the shortest paths) between the vertices $i, j \in V$. The first distance matrix row and column represent the depot.

*3.8. Travel Time Matrix.* This matrix includes the travel times $t_{ij}$ between the vertices $i, j \in V$. The first travel time matrix row and column represent the depot.

*3.9. Vertex Parameters.* The loading/unloading cost in vertex $k$ on route $t$ is as follows:

(i) cost_fix ($v_{tk}$) is the fixed cost (e.g., transport document inspection fee)

(ii) cost_var ($v_{tk}$) is the variable cost per transported goods unit

The loading/unloading time in vertex $k$ on route $t$ is as follows:

(i) t_fix ($v_{tk}$) is the fixed time (e.g., driving the vehicle in for handling)

(ii) t_var ($v_{tk}$) is the variable time per transported goods unit

The requirements in vertex $k$ on route $t$ are as follows:

(i) C_n ($v_{tk}$) is the number of units of goods loaded in vertex $v_{tk}$ (the sum of loaded units of goods from all orders on route $t$ for which the vertex is the dispatch point)

(ii) C_v ($v_{tk}$) is the number of units of goods unloaded in vertex $v_{tk}$ (the sum of unloaded units of goods from all orders on route $t$, for which the vertex is the destination)

*3.10. Service times in the Vertices on the Route and Servicing Cost.* The service time in $v_{tk}$ consists of the fixed and variable parts, where the latter is dependent on the number of loaded and unloaded units of goods equation:

Table 1: Initial savings matrix—first iteration of savings calculation.

| | Route 1 | Route 2 | ... | Route number of orders |
|---|---|---|---|---|
| Route 1 | — | $\lambda_{12}$ (savings)<br>$\lambda_{12}$ (route) | | $\lambda_{1\text{number of orders}}$ (savings)<br>$\lambda_{1\text{number of orders}}$ (route) |
| Route 2 | $\lambda_{21}$ (savings)<br>$\lambda_{21}$ (route) | — | | $\lambda_{2\text{number of orders}}$ (savings)<br>$\lambda_{2\text{number of orders}}$ (route) |
| ... | | | — | |
| Route $N$ ($N$ = number of orders) | $\lambda_{\text{number of orders1}}$ (savings)<br>$\lambda_{\text{number of orders1}}$ (route) | | | — |

$$\text{service\_time}(v_{tk}) = t\_\text{fix}(v_{tk}) + t\_\text{var}(v_{tk}) \times (C\_n(v_{tk}) + C\_v(v_{tk})). \quad (13)$$

The service cost in the route vertex $v_{tk}$ also consists of a fixed component, independent of the number of loaded/unloaded units (e.g., the cost of having the vehicle driven to the ramp), and a variable component that depends on the number loaded/unloaded units, such as the cost of the vehicle idle time during the loading/unloading operations. These costs are directly proportional to the vertex servicing time equation:

$$\text{service\_cost}(v_{tk}) = \text{service\_time}(v_{tk}) \times \text{cost\_}w + \text{cost\_fix}(v_{tk}) + (C\_n(v_{tk}) + C\_v(v_{tk}) \times \text{cost\_var}(v_{tk})). \quad (14)$$

*3.11. Cost of Travel between Two Vertices.* The cost of travel between vertices $v_{t,k-1}$ and $v_{tk}$ depends on the distance travelled equation:

$$\text{cost}(v_{t,k-1}, v_{tk}) = d(v_{t,k-1}, v_{tk}) \times \text{cost\_}t. \quad (15)$$

*3.12. Route Implementation Cost.* The route implementation cost is expressed by the equation:

$$\text{implementation\_cost} = \text{cost}(v_{t1}) + \text{cost}(v_{t1}, v_{t2}) + \text{cost}(v_{t2}) + \text{cost}(v_{t2}, v_{t3}) + \ldots + \text{cost}(v_{t,m-1}) + \text{cost}(v_{t,m-1}, v_{tm}) + \text{cost}(v_{tm}). \quad (16)$$

*3.13. Route Implementation Time.* The route implementation time includes only the net vehicle travel time and all vertex servicing times. The latter includes the vehicle idle time during the loading or unloading operation. The route implementation time does not include other components, such as waiting time for a free ramp, mandatory drivers' breaks for rest, and waiting time for a relevant time window, in situations where time windows are included in the problem being solved and the vehicle arrived at the vertex earlier. This is expressed by the equation:

$$\text{implementation\_time} = \text{service\_time}(v_{t1}) + t(v_{t1}, v_{t2}) + \text{service\_time}(v_{t2}) + \ldots + \text{service\_time}(v_{t,m-1}) + t(v_{t,m-1}, v_{tm}) + \text{service\_time}(v_{tm}). \quad (17)$$

*3.14. Total Route Time.* The total route time is the complete route duration, including waiting for the free ramp, mandatory drivers' rest breaks, and time of waiting for a free time window where this is relevant. Each of those constitutes a subproblem that has to be solved and used to determine the real route start and end times, which are input data in the computation equation:

$$\text{total\_route\_time} = \text{route\_end\_time} - \text{route\_start\_time}. \quad (18)$$

*3.15. Cost of Waiting for the Vehicle.* The previous items are used to determine the total vehicle waiting time, from which the total cost of waiting for the vehicle is derived. The vehicle waiting time is the following difference equation:

$$\text{waiting\_time} = \text{total\_route\_time} - \text{implementation\_time}. \quad (19)$$

The waiting cost is expressed by the equation:

$$\text{waiting\_cost} = \text{waiting\_time} \times \text{cost\_}w. \quad (20)$$

*3.16. Route Cost Calculation.* The total route cost is calculated as the sum of the travel cost, customer servicing cost, waiting cost, and vehicle deployment cost, as expressed by equation (21). The last-mentioned item can include vehicle renting cost.

The total cost of a vehicle is

$$\text{route\_cost} = \text{implementation\_cost} + \text{waiting\_cost} + \text{cost\_}v. \quad (21)$$

*3.17. Route Joining.* The following situations are examined when joining two routes into one

Variant 0: The routes are joined through the outermost vertices, have no common vertex except the depot, and do not overlap. For example, routes *D-1-2-D* and *D-3-4-D* create route *D-1-2-3-4-D* (Figure 3).

Variant 1: The last vertex (before the depot) of one route coincides with the first vertex (after the depot) of the other route, and this common vertex serves to join the routes together. For example, routes *D-1-2-3-D* and *D-3-4-D* create route *D-1-2-3-4-D* (Figure 4).

The difference between variants 0 and 1 is that in variant 0, the routes have no common vertex (except the depot), while in variant 1, they share one vertex (except the depot).

Variant 2: The last two vertices (before the depot) of the one route coincide with the two first vertices (after the depot) of the other route. For example, routes *D-1-2-3-D* and *D-2-3-4-D* create route *D-1-2-3-4-D* (Figure 5).

Variant 3: The last three vertices of route 1 coincide with the first three vertices of route 2 (in the same order). For example, routes *D-1-2-3-4-D* and *D-2-3-4-5-D* create route *D-1-2-3-4-5-D* (Figure 6).

Variant 4: Route $j$ contains only such vertices as also present on route $i$ in the same direction. For example, routes *D-1-2-3-4-5D* and *D-2-3-4-D* create route *D-1-2-3-4-5-D* (Figure 7).

The procedure in variants 1, 2, and 3 can be further generalized to multiple common vertices. In the future, we could consider modifying the algorithm in such a way that the number of common vertices to be examined for both routes will be the algorithm's parameter. One can expect that such exploration would improve the solution quality but increase the computing time. In our experiments, we did not investigate the sensitivity of the algorithm to this parameter.

The procedure of selecting the routes to be joined follows the principle of the Clarke and Wright algorithm. The maximum positive element is identified in the saving matrix. This element determines the routes that can (while meeting the constraints of the problem) be combined into one (in the following description of the algorithm, we consider the only constraint regarding the vehicle capacity). If the routes cannot be joined, the saving matrix element is not filled out and will not be considered for the selection of the maximum savings.

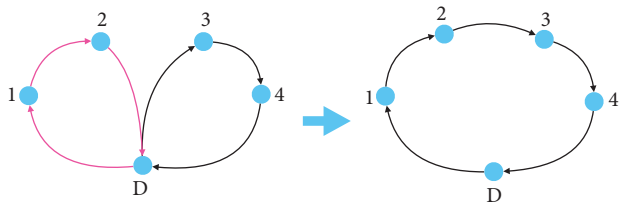


Figure 3: Example of variant 0.

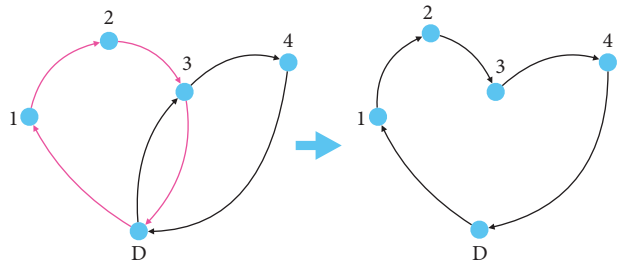


Figure 4: Example of variant 1.

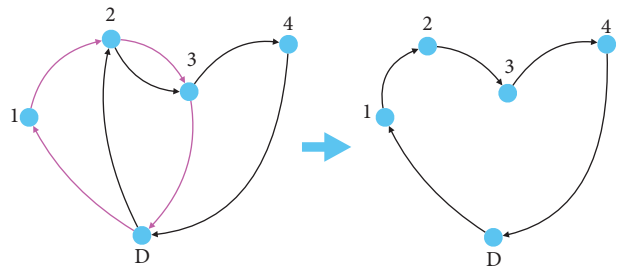


Figure 5: Example of variant 2.

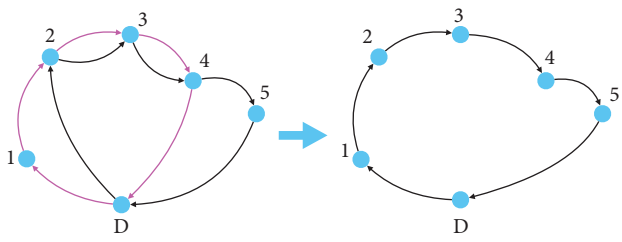


Figure 6: Example of variant 3.

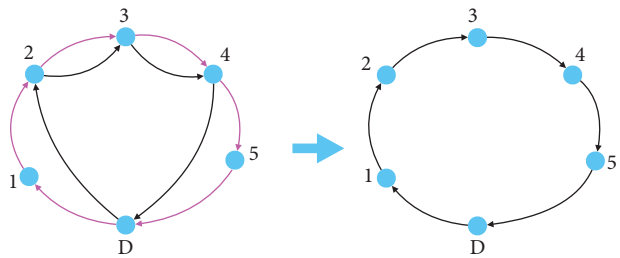


Figure 7: Example of variant 4.

The modified Clark and Wright algorithm operates with a dynamic saving matrix. In contrast to the classical Clark and Wright algorithm, which calculates savings solely from

distances and considers savings only between outermost vertices of the routes, the modified algorithm computes savings from costs that depend on various factors, including the configuration of the resulting route and the quantity of goods loaded and unloaded along the route. This distinction leads to variable savings throughout the calculation. When two routes are combined, the quantity of goods must be recalculated, resulting in changes to the costs of other routes that could potentially be combined in subsequent iterations. A new route is created and included in the matrix to replace the merged routes. Thus, after each of these operations, the saving matrix must be updated. The two rows and columns pertaining to the joined routes must be removed from the matrix, and simultaneously, a row and a column corresponding to the new route are added. This dynamic adjustment in matrix dimension occurs during the execution of the algorithm. The algorithm terminates when no positive savings are present in the saving matrix.

*3.18. Algorithm Implementation.* In this section we describe the implementation of the algorithm. The notation used is given in Table 2 and the functions are summarized in Table 3.

The basic concept of the modified approach is described by Algorithm 1.

Algorithm 2 creates the initial routes $R$ from the set of all orders $O$ by using the functions test_vehicle_capacity_on route and create_route.

Algorithm 3 calculates the saving matrix. The input parameters include the vehicle $t_x$ and the set of all routes $R$. The algorithm uses the function update_save_matrix, which is described in detail by using a pseudocode in Algorithm 4.

Algorithm 5 uses a loop (which runs until no positive savings are present in savings matrix $A$) in which the next route with the currently highest savings $s_{\text{cost}}$ is selected and the route $s_{\text{route}}$ is added to the set of all routes $R$. The initial routes $r_{s_{\text{index\_}i}}$ and $r_{s_{\text{index\_}j}}$ (now joined into the new one) are removed from the set of routes and from matrix $A$ by the function remove_row_and_column $(\downarrow\uparrow A, \downarrow i, \downarrow j)$.

Algorithm 4 handles the process of joining the routes $r_i$ and $r_j$ into a new potential route $r_{\text{new}}$. The procedure examines the specific variants where the end vertices of $r_i$ may coincide with the starting vertices of $r_j$. The following variants are tested:

(1) is_Last_And_First_Vertex_Same_Only

(2) are_2_Last_And_2_First_Vertex_Same

(3) are_3_Last_And_3_First_Vertex_Same

Furthermore, the function is_$R$2_Sub Set_$R$1_In_Direction is used to examine whether the vertices of route $r_j$ make up a subset of route $r_i$ in the same direction. If coinciding vertices in the right position according to the above conditions are identified, those vertices on the new route $r_{\text{new}}$ are joined together by using the function joined_vertex. If the new route $r_{\text{new}}$ meets the vehicle capacity constraint (test_vehicle_capacity_on route), the new savings (new_cost) are calculated by the function get_route_cost and saved in the saving matrix $A$.

## 4. Use of the Modified Clarke and Wright Algorithm in a Case Study

*4.1. Case Study.* The case study aimed to assess the effectiveness of the modified Clarke and Wright algorithm in the context of distribution logistics, particularly in the intelligent management of goods transport flows. The successful implementation of this algorithm has the potential to enhance efficiency, improve planning dynamics, optimize routing, and consequently lead to better utilization of vehicles transport capacity. This, in turn, could result in reduced $CO_2$ emissions, improved overall operating economy, and increased competitiveness.

For the case study, collaboration was stablished with MD Logistika a.s., a company specializing in the pickup and delivery of foods in the Czech Republic and transboundary trade. The company handles schedules for numerous vehicles daily, managing 500–700 delivery orders from various clients. Operational planning at MD Logistika involves addressing multiple constraints, including vehicle loading capacities, time windows for loading/unloading, drivers' rest breaks, ramp assignment, and the limitations on travel duration and distances for both route components and entire routes. Challenges such as a shortage of qualified staff for expert planning and the absence of comprehensive planning software that considers all constraints are prominent.

Currently, MD Logistika planners do not use any decision support tool and manually construct distribution plans. The primary aim of the study was to contribute to the innovation of the planning process. The goal was to develop a rapid algorithm capable of solving a relaxed version of the problem in a matter of seconds. Upon validation of its results by decision-makers, the algorithm could be refined, adapted to more complex problems with practical constraints, and potentially implemented in planning software. In the initial phase, the case study focused on transport capacity-related constraints, with time windows not being specified.

*4.2. Input Data.* Input data analysis is a crucial step in preparing for computational experiments. In the case of MD Logistika, the analysis focused on various aspects of their business data, including fleet information and customer details.

*4.3. Fleet Information.* The fleet used for the computation experiments was taken from the real operation at MD Logistika, consisting of approximately 200 trucks. The majority (85%) of vehicles in the fleet have a capacity of 33 transport units (pallets), while the remaining vehicles (15%) have capacities between 10 and 21 transport units. The primary parameters considered for each vehicle included the transport capacity and price per travelled kilometre.

*4.4. Customer Information.* Customer information was also derived from real data. Customers are primarily food product sellers. Key information includes their fixed costs

Table 2: Explanatory notes.

| Symbol | Meaning |
|---|---|
| $V = \{v_i \mid i = 1, \ldots,\}$ | Set of all vertices |
| $O = \{o_i \mid i = 1, \ldots,\}$ | Set of all delivery orders |
| $r_i = \{v_{i1}, v_{i2}, \ldots, v_{\text{im}0}\}$, | Ordered sequence of $i$-th route vertices |
| $R = \{r_i \mid i = 1, \ldots,\}$ | Set of all routes |
| $T = \{t_i \mid i = 1, \ldots,\}$ | Set of all vehicles |
| $s = (s_{\text{cost}}, s_{\text{route}}, s_{\text{index\_}i}, s_{\text{index\_}j})$ | Savings are defined by means of four parameters: $s_{\text{cost}}$, which is the cost savings of route $s_{\text{route}}$;$s_{\text{route}}$, which is the route consisting of vertices, and $s_{\text{index\_}i}$, $s_{\text{index\_}j}$, which are the coordinates in matrix $A$ |
| $A$ | Saving matrix |

Table 3: Functions.

| Subprogram/function | Meaning |
|---|---|
| elementary_route ($\downarrow t_x, \downarrow$O) | This function creates the initial routes $R$ from the set of all orders $O$. It uses the functions test_vehicle_capacity_on route and create_route |
| save_matrixte ($\downarrow t_x, \downarrow R$) | This function creates the saving matrix. It uses the function update_save_matrix |
| reducete ($\downarrow\uparrow A, \downarrow t_x$) | In the loop (until no positive savings are present in matrix A), the function selects the next route with the largest savings $s_{\text{cost}}$ and adds route $s_{\text{route}}$ to the set of all routes $R$. The initial routes $r_{s_{\text{index\_}i}}$ and $r_{s_{\text{index\_}j}}$ (now joined into one) are removed from the set of routes and from savings matrix $A$ by using the function remove_row_and_column ($\downarrow\uparrow A, \downarrow i, \downarrow j$) |
| create_routete ($\downarrow v_1, \downarrow v_2, \downarrow v_3$)<br>create_routete ($\downarrow v_1, \downarrow v_2, \downarrow v_3, \downarrow v_4$) | This function creates the initial route $r_i$. The input parameters are the vertices of the new route. A variant of this function is selected based on information whether the operation is delivery, pickup, or goods transport between customers |
| create_savete ($\downarrow s_{\text{cost}}, \downarrow s_{\text{route}}, \downarrow s_i, \downarrow s_j$) | This function creates savings $s$. The input parameters are the route savings, the route consisting of vertices and coordinates $s_i, s_j$ in matrix $A$ |
| test_vehicle_capacity_on routete ($\downarrow r_i, \downarrow t_x$) | If the capacity of vehicle $t_x$ on route $r_i$ was not exceeded in any of the vertices (where goods are loaded/unloaded), this function returns true; otherwise, it returns false |
| update_save_matrixte ($\downarrow\uparrow A, \downarrow i, \downarrow j, \downarrow t_x$) | This function joins routes $r_i$ and $r_j$ into a new potential route $r_{\text{new}}$ and consecutively evaluates the specific variants where the end vertices of $r_i$ and starting vertices of $r_j$ may coincide:<br>(1) is_Last_And_First_Vertex_Same_Only<br>(2) are_2_Last_And_2_First_Vertex_Same<br>(3) are_3_Last_And_3_First_Vertex_Same<br>Furthermore, the procedure examines if the vertices of route $r_j$ constitute a subset of route $r_i$ in the same direction:<br>(4). is_$R$2_SubSet_$R$1_In_Direction<br>If identical vertices are identified in the right position according to the above conditions, then those vertices are combined together for the new route $r_{\text{new}}$ by means of the function joined_vertex. If the vehicle capacity is met on the new route $r$ (test_vehicle_capacity_on route), the function get_route_cost is used to calculate the new savings new_cost, which are then saved in the savings matrix $A$ |
| find_ max _save ($\downarrow A, \downarrow\lvert A\rvert$) | This function returns the highest savings $s$ from matrix $A$ |
| joined_vertext $\downarrow (v_1, \downarrow v_2)$ | This function returns the new vertex obtained by joining vertices $v_1$ and $v_2$ |
| is_Last_And_First_Vertex_Same_Only ($\downarrow r_i, \downarrow r_j$) | This function returns true if $v_{i,\lvert r_i\rvert-1} = v_{j,2}$<br>i.e., if the last serviced vertex of route $r_i$ (before returning to the depot) coincides with the first serviced vertex on route $r_j$ (after the depot) |
| are_2_Last_And_2_First_Vertex_Same ($\downarrow r_i, \downarrow r_j$) | This function returns true if $v_{i,\lvert r_i\rvert-2} = v_{j,2} \wedge v_{i,\lvert r_i\rvert-1} = v_{j,3}$ |
| are_3_Last_And_3_First_Vertex_Same ($\downarrow r_i, \downarrow r_j$) | This function returns true if $v_{i,\lvert r_i\rvert-3} = v_{j,2} \wedge v_{i,\lvert r_i\rvert-2} = v_{j,3} \wedge v_{i,\lvert r_i\rvert-1} = v_{j,4}$ |
| is_$R$2_Sub Set_$R$1_In_Direction ($\downarrow r_i, \downarrow r_j$)) | This function returns true if the vertices of route $r_j$ are subsets of route $r_i$ in the same direction |
| get_route_cost ($\downarrow r_i$) | This function returns the cost of route $r_i$ |
| remove_row_and_column ($\downarrow\uparrow A, \downarrow i, \downarrow j$) | This function removes the $i$-th row and $j$-th column and the $j$-th row and i-th column from matrix $A$ |
| add_row_and_column ($\downarrow\uparrow A$) | This function adds a new row and a new column to matrix $A$ |

```
(1) for each t_x ∈ T: x ∈ {1, ..., |T|} do
(2)     R = elementary_route (↓t_x, ↓O)
(3)     A = save_matrix (↓t_x,↓R)
(4)     reduce (↓↑A, ↓t_x)
(5) end
```

Algorithm 1: Basic concept of the algorithm.

```
(1) function elementary_route(↓t_x, ↓O) do
(2)     R = ∅
(3)     for each o_i ∈ O: i ∈ {1, ..., |O|} do
(4)         switch(o_i)
(5)             case: delivery_type
(6)                 r_i = create_route (↓v_depot, ↓v_unload_order, ↓v_depot)
(7)             case: colection_type
(8)                 r_i = create_route (↓v_depot, ↓v_load_order, ↓v_depot)
(9)             case: inter_type
(10)                r_i = create_route (↓v_depot, ↓v_load_order, ↓v_unload_order, ↓v_depot)
(11)        end
(12)        if(test_vehicle_capacity_on route(↓r_i, ↓t_x))
(13)        R = R ∪ r_i
(14)        end
(15)    end
(16)    return R
(17) end
```

Algorithm 2: Calculation of the elementary routes.

```
(1) function save_matrix(↓t_x, ↓R) do
(2)     A = |R| × |R|
(3)     for each i ∈ {1, ..., |R|} do
(4)         for each j ∈ {1, ..., |R|} do
(5)             if(i ≠ j)
(6)                 update_save_matrix (↓↑A, ↓i, ↓j, ↓t_x)
(7)             end
(8)         end
(9)     end
(10)    return A
(11) end
```

Algorithm 3: Calculation of the saving matrix.

```
(1) function update_save_matrix (↓↑A, ↓i, ↓j, ↓t_x) do
(2)     r_new = create_route()
(3)     if(is_Last_And_First_Vertex_Same_Only (↓r_i, ↓r_j))
(4)         for each k ∈ {1, ..., |r_i| − 2} do
(5)             r_new = r_new ∪ v_i,k
(6)         end
(7)         r_new = r_new ∪ joined_vertex (↓v_i,|r_i|−1, ↓v_j,2)
(8)         for each k∈ {3, ..., |r_j|} do
(9)             r_new = r_new ∪ v_j,k
(10)        end
```

Algorithm 4: Continued.

```
(11)  else
(12)    if(are_2_Last_And_2_First_Vertex_Same (↓r_i, ↓r_j))
(13)      for each k ∈ {1, ..., |r_i| − 3} do
(14)        r_new = r_new ∪ v_{i,k}
(15)      end
(16)      r_new = r_new ∪ joined_vertex (↓v_{i,|r_i|−2}, ↓v_{j,2})
(17)      r_new = r_new ∪ joined_vertex (↓v_{i,|r_i|−1}, ↓v_{j,3})
(18)      for each k ∈ {4, ..., |r_j|} do
(19)        r_new = r_new ∪ v_{j,k}
(20)      end
(21)    else
(22)      if(are_3_Last_And_3_First_Vertex_Same (↓r_i, ↓r_j))
(23)        for each k ∈ {1, ..., |r_i| − 4} do
(24)          r_new = r_new ∪ v_{i,k}
(25)        end
(26)        r_new = r_new ∪ joined_vertex (↓v_{i,|r_i|−3}, ↓v_{j,2})
(27)        r_new = r_new ∪ joined_vertex (↓v_{i,|r_i|−2}, ↓v_{j,3})
(28)        r_new = r_new ∪ joined_vertex (↓v_{i,|r_i|−1}, ↓v_{j,4})
(29)        for each k∈ {5, ..., |r_j|} do
(30)          r_new = r_new ∪ v_{j,k}
(31)        end
(32)      else
(33)        if(is_R2_SubSet_R1_In_Direction (↓r_i, ↓r_j))
(34)          for each k ∈ {1, ..., |r_i|} do
(35)            for each l ∈ {1, ..., |r_j|} do
(36)              if (v_{i,k} = v_{j,l})
(37)                r_new = r_new ∪ joined_vertex (↓v_{j,k}, ↓v_{j,l})
(38)              else
(39)                r_new = r_new ∪ v_{i,k}
(40)              end
(41)            end
(42)          end
(43)        else
(44)          for each k ∈ {1, ..., |r_i| − 1} do
(45)            r_new = r_new ∪ v_{i,k}
(46)          end
(47)          for each k ∈ {2, ..., |r_j|} do
(48)            r_new = r_new ∪ v_{j,k}
(49)          end
(50)        end
(51)      end
(52)    end
(53)  end
(54)  if(test_vehicle_capacity_on_route (↓r_new, ↓t_x))
(55)    cost = get_route_cost (↓r_new)
(56)    new_cost = get_route_cost(↓r_i) + get_route_cost(↓r_j) − cost
(57)    A[i, j] = create_save(↓new_cost, ↓r_new, ↓i, ↓j)
(58)  else
(59)    A[i, j] = null
(60)  end
(61) end
```

Algorithm 4: Updating the saving matrix.

and variable costs related to goods loading/unloading. Additional details encompass customer identification and position information (GPS). The distance matrix and matrix of travel times between customers were obtained along with the input data.

*4.5. Delivery Orders.* Delivery orders represent requirements for goods transportation between nodes. The majority (80%) of the requests are from the depot to customers, while the remaining fraction (20%) includes goods transport between customers and the collection of items from customers to the

```
(1) function reduce(↓↑A, ↓t_x) do
(2)   if (|A| = 0)
(3)     return
(4)   s = find_ max _save(↓A, ↓|A|)
(5)   while(s_cost > 0) do
(6)     R = R ∪ s_route
(7)     R = R∖r_{s_index_i}
(8)     R = R∖r_{s_index_j}
(9)     remove_row_and_column(A, s_index_i, s_index_j)
(10)    add Row And Column(A)
(11)    for each i ∈ {1, ..., |A|} do
(12)      for each j ∈ {1, ..., |A|} do
(13)        if(i ≠ j)
(14)          update_save_matrix(↓↑A, ↓i, ↓j, ↓t_x)
(15)        end
(16)      end
(17)    end
(18)    s = find_ max _save(↓A, ↓|A|)
(19)  end
(20) end
```

Algorithm 5: Route reduction.

Table 4: Effectiveness of the optimization procedure.

| Scenario | Number of initial routes | Number of routes after optimization | % routes reduced | Computing time (s) |
|---|---|---|---|---|
| 1 | 633 | 69 | 89.1 | 9.3 |
| 2 | 677 | 72 | 89.4 | 10.2 |
| 3 | 659 | 76 | 88.6 | 9.1 |
| 4 | 652 | 62 | 90.6 | 9.6 |
| 5 | 659 | 73 | 89.0 | 9.9 |
| 6 | 634 | 71 | 88.8 | 9.8 |
| 7 | 631 | 62 | 90.2 | 9.6 |
| 8 | 622 | 67 | 89.3 | 10.2 |
| 9 | 622 | 60 | 90.5 | 10.1 |
| 10 | 651 | 62 | 90.6 | 9.9 |

depot. The goods are transported on pallets, with each pallet representing one transport unit. The data includes the number of transported pallets and the loading/unloading site.

## 5. Results and Discussion

In the case study collaboration with MD Logistika, ten scenarios were devised, each corresponding to a specific day of operation. These scenarios captured real-world daily requirements consisting of 600 to 700 delivery orders.

The algorithm processed each day consecutively with the goal to optimize the initial routes that were set up based on the delivery orders. The scenarios are presented in Table 4 as follows:

(1) Number of initial routes: This indicates the initial set of routes established based on the delivery orders for each day.

(2) Final number of routes: After the optimization process using the modified Clarke and Wright algorithm, the final number of routes is determined.

(3) Percentage of routes reduced: This metric reflects the percentage reduction in the number of routes achieved through the optimization process. It provides insights into the algorithm's effectiveness in consolidating routes.

(4) Total computing time: It is the time taken for the algorithm to complete the optimization process for each scenario.

These scenarios and their associated metrics help evaluate the algorithm's performance in real-world scenarios, providing valuable insights into route optimization, reduction efficiency, and computational speed.

The evaluation of the experiments in the case study focused on various optimization results, as outlined in Table 5. The key optimization metrics are as follows:

Table 5: Optimization results.

| Scenario | Total cost (CZK thsd) | Cost per pallet (CZK) | Total length (km) | Orders processed | Pallets delivered | Orders not processed | Pallets not delivered |
|---|---|---|---|---|---|---|---|
| 1 | 1,254 | 370.4 | 50,681 | 627 | 3,386 | 6 | 83 |
| 2 | 1,417 | 454.5 | 58,647 | 662 | 3,122 | 15 | 246 |
| 3 | 1,337 | 414.2 | 53,892 | 654 | 3,228 | 5 | 152 |
| 4 | 1,451 | 433.6 | 59,011 | 647 | 3,346 | 5 | 136 |
| 5 | 1,450 | 424.2 | 58,610 | 650 | 3,419 | 9 | 143 |
| 6 | 1,374 | 406.5 | 55,707 | 630 | 3,383 | 4 | 75 |
| 7 | 1,337 | 450.9 | 53,272 | 622 | 2,964 | 9 | 203 |
| 8 | 1,189 | 380.8 | 47,638 | 618 | 3,123 | 4 | 88 |
| 9 | 1,319 | 445.2 | 53,344 | 616 | 2,963 | 6 | 174 |
| 10 | 1,281 | 396.1 | 53,591 | 647 | 3,236 | 4 | 68 |
| Average | 1,341 | 417.6 | 54 439 | 637 | 3,217 | 6.7 | 137 |
| Std. deviation | 81 | 28 | 3 487 | 16 | 161 | 3.3 | 56 |

(1) Total cost is the overall cost incurred in the transportation process, considering factors such as travel costs, servicing costs, waiting costs, and vehicle deployment costs

(2) Cost per pallet is the cost associated with transporting each pallet, providing insights into the efficiency of cost allocation

(3) Total length is the length of optimized routes, representing the total distance travelled by the vehicles

(4) The number of processed orders is the count of delivery orders that were successfully processed and incorporated into the optimized routes

(5) The number of delivered pallets is the count of pallets that were successfully transported on the optimized routes

(6) The number of remaining orders is the count of orders that were not included in the optimized routes due to capacity constraints.

(7) The number of remaining pallets is the count of pallets that were not included in the optimized routes, indicating any unprocessed or pending items.

These metrics collectively provide a comprehensive overview of the algorithm's performance, including cost efficiency, order processing capability, and the overall impact on the goods transportation process. The ability to handle various aspects of the real-world logistics challenges is crucial for the algorithm's practical utility.

The above table demonstrates that in each of the scenarios, the algorithm was unable to plan all delivery orders, mainly due to the inadequate capacity of the fleet available for the relevant day. Goods dispatch planners confirmed that this situation is not uncommon in everyday practice. Since the proportion of orders that could not be processed is at the level of 1%, the company solves each problem individually by outsourcing the delivery. For the algorithm, this shortcoming can be eliminated by extending the input fleet. Furthermore, the mean cost of satisfying the orders is about CZK 1,341 thousand, (STD CZK 81 thousand), the mean summary length of all routes is 54.4 thousand km (STD about 3.5 thousand km), and the mean number of orders processed daily is 637 (STD 16).

Table 6: Statistics per route.

| Scenario | Average number of stops | Average length of a route (km) | Average duration of a route (minutes) |
|---|---|---|---|
| 1 | 3.54 | 286.33 | 383.68 |
| 2 | 3.36 | 297.7 | 385.28 |
| 3 | 3.50 | 288.19 | 381.77 |
| 4 | 3.37 | 307.35 | 397.19 |
| 5 | 3.42 | 308.47 | 397.67 |
| 6 | 3.48 | 307.77 | 390.49 |
| 7 | 3.38 | 289.52 | 380.12 |
| 8 | 3.70 | 285.26 | 387.82 |
| 9 | 3.67 | 317.52 | 413.29 |
| 10 | 3.77 | 318.99 | 414.25 |

As a result of a more detailed analysis of the scenarios, Table 6 includes the average number of stops on a route and the average length and duration of a route for each scenario.

In the case study, the Clarke and Wright algorithm modification was tested on a selected subproblem, primarily focusing on the constraints posed by the limited transport capacity. Since our algorithm was not designed to account for all practical constraints, such as limitations associated with the use of the loading ramps, driver's breaks for rest, and time windows allocated to goods loading/unloading operations, it was not possible to directly compare the numerical results with real schedules.

The main conclusion resulting from empirical testing is that despite utilizing the dynamic saving matrix concept, one of the algorithm's favourable key properties is retained: the relatively short computing time. If it becomes evident that the new algorithm maintains low computing complexity even after the inclusion of all constraints compelled by real-world circumstances, it will be feasible to use it in everyday dispatch planning practice, and moreover, in real time, owing to its potential high speed. Furthermore, if the algorithm is to be integrated into the daily work of dispatchers, a comprehensive software tool must be set up to include not only the algorithm itself but also basic input/output control, preferably transformed into a textual and graphic format.

## 6. Conclusions

The objective of this paper was to outline a modification of the Clarke and Wright algorithm tailored for application in delivery order-oriented distribution logistics. The proposed modification was then put to the test within a case study utilizing real internal data from MD Logistika company. The algorithm was tested on a selected subproblem, with a specific focus on constraints related to transport capacity.

The expert assessment of the results by MD Logistika's operative planning professionals provided evidence that the outcomes are reasonable, especially concerning the total cost per route—a crucial parameter for efficient and competitive goods delivery. In essence, the accuracy of the modifications made to the Clarke and Wright algorithm has been affirmed. Consequently, additional real-world constraints can be gradually incorporated into the algorithm during the subsequent stages of research. As a result, the operational planning dispatchers may soon have access to an efficient software tool, aiding them in planning large-scale goods delivery with significantly reduced costs and staffing requirements compared to common industry practices.

## Data Availability

The datasets used and analysed during the current study are available from the corresponding author on reasonable request.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

The work was supported from the Operational Program for Entrepreneurship and Innovation for Competitiveness within the Project Development of Technology for Intelligent Management of Transport Flows of Goods with registration number CZ.01.1.02/0.0/0.0/17_107/0012371 and by the Slovak Research and Development Agency, grant number APVV-19-0441.

## References

[1] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia, 2002.

[2] G. Laporte, "What you should know about the vehicle routing problem," *Naval Research Logistics*, vol. 54, no. 8, pp. 811–819, 2007.

[3] P. Toth and D. Vigo, *Vehicle Routing: Problems, Methods, and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, 2014.

[4] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[5] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.

[6] F. A. Tillman, "The multiple terminal delivery problem with probabilistic demands," *Transportation Science*, vol. 3, pp. 192–204, 1969.

[7] S. Raff, B. Golden, A. Assad, and M. Ball, "Routing and scheduling of vehicles and crews: the state of the art," *Computers and Operations Research*, vol. 10, no. 2, pp. 63–211, 1983.

[8] T. Doyuran and B. Çatay, "A robust enhancement to the Clarke–Wright savings algorithm," *Journal of the Operational Research Society*, vol. 62, no. 1, pp. 223–231, 2011.

[9] T. J. Gaskell, "Bases for vehicle fleet scheduling," *Journal of the Operational Research Society*, vol. 18, no. 3, pp. 281–295, 1967.

[10] P. C. Yellow, "A computational modification to the savings method of vehicle scheduling," *Operational Research Quarterly*, vol. 21, no. 2, pp. 281–283, 1970.

[11] H. Paessens, "The savings algorithm for the vehicle routing problem," *European Journal of Operational Research*, vol. 34, no. 3, pp. 336–344, 1988.

[12] İ. K. Altınel and T. Öncan, "A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem," *Journal of the Operational Research Society*, vol. 56, no. 8, pp. 954–961, 2005.

[13] A. A. Juan, J. Faulin, R. Ruiz, B. Barrios, and S. Caballé, "The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem," *Applied Soft Computing*, vol. 10, no. 1, pp. 215–224, 2010.

[14] A. A. Juan, J. Faulin, J. Jorba, D. Riera, D. Masip, and B. Barrios, "On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics," *Journal of the Operational Research Society*, vol. 62, no. 6, pp. 1085–1097, 2011.

[15] S. P. Sarmah, R. Yadav, and P. Rathore, "Development of vehicle routing model in urban solid waste management system under periodic variation: a case study," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1961–1965, 2019.

[16] Y. Jiang, R. Yang, C. Zang et al., "Toward baggage-free airport terminals: a case study of London city airport," *Sustainability*, vol. 14, no. 1, p. 212, 2021.

[17] A. R. Destyanto, N. F. Fajar, D. G. Mandhasiya et al., "Improving service level and utilization of distribution using discrete event simulation by comparing three vehicle routing problem algorithm: a case study of drugs distribution company," *RECENT PROGRESS ON: MECHANICAL, INFRASTRUCTURE AND INDUSTRIAL ENGINEERING: Proceedings of International Symposium on Advances in Mechanical Engineering (ISAME): Quality in Research 2019*, vol. 2227, 2020.

[18] L. M. Escobar, A. M. David, J. W. Escobar, R. Linfati, and G. E. Mauricio, "A hybrid metaheuristic approach for the capacitated vehicle routing problem with container loading constraint," in *Proceedings of the 2015 International Conference on Industrial Engineering and Systems Management (IESM)*, pp. 1374–1382, IEEE, Seville, Spain, October 2015.

[19] Z. Huang, W. Huang, and F. Guo, "Integrated sustainable planning of self-pickup and door-to-door delivery service with multi-type stations," *Computers and Industrial Engineering*, vol. 135, pp. 412–425, 2019.

[20] L. Zhai, I. M. Ather, Z. Wang, and Q. Zheng, "Improved ant system algorithm and its application for vehicle routing problem," in *Proceedings of the 2016 2nd Workshop on Advanced Research and Technology in Industry Applications*, May 2016.

[21] M. Berghida and A. Boukra, "Resolution of a vehicle routing problem with simultaneous pickup and delivery," *International Journal of Applied Metaheuristic Computing*, vol. 6, no. 3, pp. 53–68, 2015.

[22] X. Y. Pan, J. Wu, Q. W. Zhang, D. Lai, and C. Zhang, "A hybrid algorithm for the scheduling of vehicles with simultaneous pickups and deliveries," *Applied Mechanics and Materials*, vol. 475-476, no. 476, pp. 733–736, 2013.

[23] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis, "VRP with pickup and delivery," in *The Vehicle Routing Problem*, P. Toth and D. Vigo, Eds., pp. 225–242, Society for Industrial and Applied Mathematics, Philadelphia, 2002.

[24] Y. Ji, H. Yang, and Y. Zhou, "Vehicle routing problem with simultaneous delivery and pickup for cold-chain logistics," in *Proceedings of the 2015 International Conference on Modeling, Simulation and Applied Mathematics*, Paris, France, August 2015.

[25] L. Chun-Hua, Z. Hong, and Z. Jian, "Vehicle routing problem with time windows and simultaneous pickups and deliveries," in *Proceedings of the 2009 16th International Conference on Industrial Engineering and Engineering Management*, pp. 685–689, IEEE, Beijing, China, October 2009.

[26] R. M. Chen and P. J. Fang, "Solving vehicle routing problem with simultaneous pickups and deliveries Based on a two-layer particle swarm optimization," in *Proceedings of the 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 212–216, IEEE, Toyama, Japan, July 2019.

[27] V. Yu and S. Y. Lin, "Solving the location-routing problem with simultaneous pickup and delivery by simulated annealing," *International Journal of Production Research*, vol. 54, no. 2, pp. 526–549, 2016.

[28] S. K. Sathyanarayanan, K. Suresh Joseph, and S. K. V. Jayakumar, "A hybrid population seeding technique based genetic algorithm for stochastic multiple depot vehicle routing problem," in *Proceedings of the 2015 International Conference on Computing and Communications Technologies (ICCCT)*, pp. 119–127, IEEE, Chennai, India, February 2015.