A hybrid metaheuristic with learning for a real supply chain scheduling problem

Christian Pérez [a], Laura Climent [b], Giancarlo Nicoló [a], Alejandro Arbelaez [b], Miguel A. Salido [a],*

[a] *Instituto de Automática e Informática Industrial, Universitat Politècnica de València, Valencia, Spain*
[b] *Departamento de Ingenieria Informática, Universidad Autonoma de Madrid, Madrid, Spain*

ARTICLE INFO

ABSTRACT

In recent decades, research on supply chain management (SCM) has enabled companies to improve their environmental, social, and economic performance.

This paper presents an industrial application of logistics that can be classified as an inventory-route problem. The problem consists of assigning orders to the available warehouses. The orders are composed of items that must be loaded within a week. The warehouses provide an inventory of the number of items available for each day of the week, so the objective is to minimize the total transportation costs and the costs of producing extra stock to satisfy the weekly demand. To solve this problem a formal mathematical model is proposed. Then a hybrid approach that involves two metaheuristics: a greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA) is proposed. Additionally, a meta-learning tuning method is incorporated into our hybridized approach, which yields better results but with a longer computation time. Thus, the trade-off of using it is analyzed.

An extensive evaluation was carried out over realistic instances provided by an industrial partner. The proposed technique was evaluated and compared with several complete and incomplete solvers from the state of the art (CP Optimizer, Yuck, OR-Tools, etc.). The results showed that our hybrid metaheuristic outperformed the behavior of these well-known solvers, mainly in large-scale instances (2000 orders per week). This hybrid algorithm provides the company with a powerful tool to solve its supply chain management problem, delivering significant economic benefits every week.

1. Introduction

In the early stages of supply chain management (SCM), obtaining an optimized solution for the production of stock and its distribution in warehouses is essential. A subsequent stage of the SCM tries to find the optimized set of routes for a fleet of vehicles, satisfying the demands of a set of customers. This problem is called the vehicle routing problem (VRP) (Schiffer et al., 2019). In addition, inventory management (IM) is responsible for ordering, storing, using, and selling a company's inventory, including the management of raw materials, components, and finished goods, as well as the storage and processing of these items (Zhang et al., 2021). Optimization techniques have made possible integrated approaches among related sub-problems that compose the SCM. This has enabled it to combine several problems, such as VRP and IM (Mara et al., 2021).

Integrating the SCM's three sub-problems mentioned above (location–allocation, routing, and inventory management) gives a holistic view of the system. It considers the supply chain as an integrated problem (Rafie-Majd et al., 2018). Considering that the warehouse location is a strategic problem to determine the number and location of warehouses (Baumol and Wolfe, 1958); inventory management is a tactical problem to determine the number of orders and safety stock (Silver, 1981); route planning is an operational problem to determine the number and succession of deliveries, recently, more and more research is considering a combination of the above decisions, including the location-inventory problem (LIP) (Wang et al., 2020), the location-route problem (LRP) (Nikzamir and Baradaran, 2020), and the inventory-route problem (IRP) (Coelho et al., 2014). Simultaneous optimization of routing and inventory decisions will significantly reduce costs and thus improve customer service. The inventory routing

problem (IRP) integrates the operations of manufacturing, storing, and holding inventories and their transportation. It is an important and well-studied problem in the supply chain literature (Malladi and Sowlati, 2018; Cui et al., 2023). It is well known that the problems that make up the SCM belong to the NP-hard problems. Therefore, the combination of these problems will give way to a composite NP-hard problem as well (Malladi and Sowlati, 2018).

In this paper, a real-case industrial problem is addressed, based on inventory management, and composed of essential operations in logistics systems that influence the sustainability and performance of the supply chain logistics of a real company that serves several types of fruits to a well-known supermarket.

The proposed problem falls within the scope of the IRP. The optimization mainly focuses on transport costs and additional stock production costs. The problem has around two thousand orders that are shipped on a specific date within one week. Each order is loaded and delivered with a single vehicle and on a single route so that each vehicle can only deliver a single order each day of the week. The defined warehouses can produce the additional stock needed to supply the demand for orders when the current stock is not enough. On the other hand, they also serve as a depot for the vehicles delivering orders and storing the stock. The warehouses already have a predefined optimal location, and each order has a finite set of available warehouses, from two to fourteen, where the order can be loaded. An optimal solution to the problem is the assignment of a warehouse to each order so that the cost of taking the vehicle to the warehouse and delivering it to the customer, and the cost of producing the required items (if needed) for the order is minimized. To optimize the solution, the company applies a multi-layer greedy algorithm that was considered the base algorithm to develop our hybrid algorithm. Each layer aims to obtain solutions based on the optimization of an objective function. Deeper layers are assigned to more general functions so that deeper layers have more specific objectives. This allows the algorithm to optimize essential aspects of the company.

This paper proposed the hybridization of two well-known metaheuristics for solving and optimizing the above real supply chain problem. The hybridization of metaheuristics (Alorf, 2023) is often very successful because, at the beginning of its execution, it obtains a broad view of the search space and then applies domain-dependent operations (Gherbi et al., 2019). Thus, taking into account the topology of the problem and the progress made by the company, this paper proposes the design and development of a greedy randomized adaptive search procedure (GRASP) and a genetic algorithm (GA). The structure of the proposed hybridization method is based on raw input data to the GRASP algorithm. Primarily, this algorithm sorts the orders concerning problem characteristics by creating a ranked candidate list. At each iteration of GRASP, the objective function of n number of candidates is obtained to evaluate which will be the next assigned order. Once the GRASP execution is completed, the generated solutions are used as the initial population of the GA. Finally, during a determined number of iterations and by means of mutation and crossover methods, the genetic algorithm will search for optimized solutions according to the proposed objective function: minimizing the total transportation costs and the costs of producing extra stock.

The evaluation section presents a deep analysis of the proposed algorithm with real large-scale instances. Randomized instances of the data sets, as well as the implementation of the mathematical model in Minizinc are available at https://github.com/GPS-UPV/SCSP. Firstly, the proposed algorithm was compared against well-known solvers from the state of the art, including the constraint programming solver: IBM ILOG CP Optimizer. The results show that the proposed hybrid algorithm outperformed the state-of-the-art solvers for large-scale real instances. The difference in total costs obtained is quite significant (especially in the large-scale instances) and therefore, by using the hybrid algorithm, the company was able to drastically decrease their logistics cost saving a consistent amount on a weekly basis.

Furthermore, this paper includes an analysis of how the tuning of the parameters and meta-learning can improve the efficiency of the proposed approach. The experiments performed on a data set provided by the industrial partner (data sets ranging from 10 to 2000 orders) show the scalability of the proposed approach for large-scale instances. Several different cutoff times were used for analyzing the improvement that parameter tuning and Machine Learning bring to the convergence of the algorithms.

The paper is structured as follows. The state-of-the-art is presented in Section 2. The problem description and its formulation are shown in Section 3. The proposed algorithms are described in Section 4. The evaluation of the proposed algorithms compared with the baseline solvers is performed in Section 5. Finally, Section 6 summarizes the conclusions and future work.

2. Literature review

This section first describes previous works on IRP, focusing on those works that use GRASP or GA. Then, some literature review that applies the hybridization of metaheuristics on IRPs is described.

2.1. Inventory routing problems, GRASP and GA

The inventory routing problem (IRP) represents a set of problems in which one or more suppliers (warehouses), prepare and deliver orders to geographically dispersed customers. This set of problems provides integrated logistics solutions, taking into account the most important aspects of inventory management, vehicle routing, and delivery planning (Coelho et al., 2014).

In Archetti et al. (2007), deterministic IRPs were introduced, which generated stock replenishment policies to avoid stock-out in warehouses. The authors used a Branch and Bound algorithm to solve IRP with a maximum-level replenishment policy, in which the amount of stock a warehouse can handle in a specific period is limited (Archetti et al., 2012).

Furthermore, non-deterministic IRPs have been discussed in the literature. These problems are composed of changing or incomplete data, making it complex to obtain optimal solutions. They can have some variability in travel times, vehicle loading, and stock replenishment, so models are analyzed using fuzzy methods, robust optimization, and dynamic programming (Arab et al., 2020).

The most popular algorithm for solving IRPs is the Genetic Algorithms (GA) (Avella et al., 2018). GA is a non-deterministic optimization technique inspired by the process of natural selection. GA search is used to explore the problem domain and, together with a population of individuals configured with the characteristics of the problem, seeks to represent a set of potential solutions in the search space (Ho et al., 2008). For instance, Moin et al. (2011) uses GA to improve solutions for a multi-period IRP. They present a multi-objective solution that provides a new way of defining mutation and crossover operators. More recent works focused on developing an adaptive GA, applying different techniques during different phases of the algorithm to suit better the problem addressed (Mahjoob et al., 2022). In their approach, they use the fitness value of parents and offspring to create adaptive operators. Mutation and crossover rates are adjusted based on the fitness value returned. In our case, the same technique is used, where the alpha value that will prune the objective function is modified by the results obtained in the previous epoch.

The GRASP method is a well-known metaheuristic framework for sizeable combinatorial optimization problems. This technique is an iterative metaheuristic consisting of two main phases: the construction phase, where greedy random solutions are obtained and can be prepossessed before the next phase is performed. The second phase, local search, iterates the random solutions to modify the warehouses assigned to the orders, seeking to obtain better solutions in each iteration. The local search ends when the list of solutions has been

wholly traversed (Festa et al., 2018). For example, in Jaikishan and Patil (2019), the authors design a GRASP algorithm to solve an IRP. The problem handled has a set of orders, and each of the orders has a release and expiration date. These dates affect the costs of stock storage and have an additional cost associated with them if the dates are not met. On the other hand, they use a α-value to discriminate the solutions in a subset centered on random solutions ($\alpha = 1$) or greedy solutions ($\alpha = 0$), adapting the Restricted Candidate List (RCL) to the needs of the problem as in Feo and Resende (1995). This paper proposes a modification of the α-value function to weight the different costs in the objective function, avoiding greedy solutions and stacking in local optima.

To solve this complex IRP, an algorithm that combines the GRASP and GA metaheuristics is proposed. By hybridizing these methods, the aim is to obtain near-optimal solutions that effectively minimize the costs associated with assigning orders to warehouses. This paper addresses a multi-vehicle, multi-product, and single-echelon Inventory Routing Problem (IRP), wherein warehouses have dual functions of stock storage and stock manufacturing when needed while being a vehicle deposit hub for the vehicles that transport the stock on specific weekdays.

It should be noted that a single-echelon IRP is characterized by optimizing the appropriate level of inventory for an individual unit within the supply chain network. However, due to the multi-vehicle and multi-product nature of the problem, the identification of an optimal route for the entire distribution network is required. Therefore, the proposed hybrid system aims to optimize the set of global variables by making the problem related to a multi-level supply chain problem, where the main objectives are to minimize transportation and inventory costs and satisfy client demand.

2.2. Hybridization approaches for IRPs

For years, there has been a rise in the hybridization of different algorithms. These works combine several algorithmic ideas that can belong to different branches of artificial intelligence, operations research, and computer science in general. Metaheuristics can be combined with many types of techniques and methods. Evolutionary algorithms, tabu search, simulated annealing, iterated local search, and optimization with ant colonies are commonly hybridized algorithms in the literature (Pellerin et al., 2020).

The interest in hybrid metaheuristics has risen considerably in the field of optimization and the best results found for many real-life optimization problems are often obtained by hybrid algorithms (Talbi, 2013). Furthermore, hybrid algorithms can provide more effective solutions to complex sustainable supply chain management problems (Abualigah et al., 2023). This type of hybridization is usually very successful because, at the beginning of their execution, they obtain a broad view of the search space and then apply problem-dependent operations (Gherbi et al., 2019). This fact has motivated the hybridization of the two metaheuristics presented in this paper.

Multilevel hybridization is based on combining different types of algorithms, either sequential, refining the results of one algorithm with another, or parallel, in which the algorithms obtain solutions simultaneously and feedback to each other (Gu et al., 2020). In the case of IRPs, the most common hybridization is sequential, which means that the first metaheuristic generates a set of initial solutions or modifies internal operators to adapt them to the specific problem. Then, this is used by the second metaheuristic.

For example, in Wu et al. (2021), a two-stage sequential hybridization is performed. First, a GA obtains a set of solutions by minimizing the time windows in the warehouses and the fuel consumption of the vehicles. Secondly, they focus on reducing the replenishment of distribution centers and retailers in each period using the gradient descent algorithm. The solutions obtained by this approach show a significant cost improvement over using GA or simulated annealing.

Another hybrid algorithm for solving the IRP is presented in Alvarez et al. (2020). The authors develop a hybrid system consisting of two parts. First, a set of feasible solutions is obtained through a construction heuristic to feed the main algorithm. The main algorithm applies the nearest neighbor heuristic to the problem to determine the optimal value of the continuous variables of the problem, allowing it to reduce the search tree and focus the problem on the most deterministic variables. Then a perturbation heuristic is applied to randomize the problem variables, such as the route, the number of customers, or the items delivered by each route, maintaining the values in each iteration of the problem that improve the solutions. Finally, the solution obtained is optimized using mathematical models that exchange routes or customers between the different solutions.

A sequential hybridization is shown in Oudouar and Zaoui (2022), where an ant colony (AC) algorithm is used to obtain the number of routes for collecting each supplier's stock and transporting it to an assembly plant in each period. Then, they apply a heuristic that evaluates inventory costs and changes routes to minimize this cost.

This paper uses a hybridization between a GRASP algorithm and a GA. The GRASP algorithm is used as a generator of initial solutions since obtaining these initial solutions improves the convergence of the GA algorithm and the results. These solutions are used as input data in a GA to improve them during iterations. The GA focuses on its crossover and mutation functions will help to obtain better solutions by extending the search tree.

3. Problem description and model formulation

This section describes the industry problem addressed in this paper. In addition, a mathematical model is formulated. The problem consists of assigning each week's orders to the available warehouses. Each order must be assigned to a single warehouse. The orders specify a certain number of items needed and the associated day of the week to be loaded. The warehouses provide an inventory of the number of items available each day of the week. The objective function of the problem is to minimize the total transportation costs and the costs of producing extra stock.

First, the constants of the problem are described and subsequently, the formal mathematical model is presented.

3.1. Constants

The list of all the constants is presented as:

$O = \{0, \ldots, |O|\}$: set of orders, indexed by $o \in O$

$I = \{0, \ldots, |I|\}$: set of items, indexed by $i \in I$

$W = \{0, \ldots, |W|\}$: set of warehouses, indexed by $w \in W$

$AW_o = \{aw_{o1}, \ldots, aw_{ow}\}$, $AW_o \subseteq W$: aw_{ow}

is the wth available warehouse for the order o

$tc_{ow} \in \mathbb{N}$: travel cost for shipping the order o

from the available warehouse $w \in AW_o$

$D_o = \{d_{o1}, \ldots, d_{oi}\}$: d_{oi} is the demand of the ith item of order o

$wd_o = \{1, \ldots, 7\}$: weekday of order o picking

$\Delta Q_w = \{\Delta q_{w1}(1), \ldots, \Delta q_{wi}(t)\}$: $\Delta q_{wi}(t)$ is the increment of units of item i in the warehouse w in the weekday t, where $t \in \{1, \ldots, 7\}$

p_i: price to manufacture one unit of item $i \in I$

The model is composed of a set of orders (O), a set of warehouses (W), and a set of items (I) that can be served in the warehouses. Each order ($o \in O$) can only be assigned to a subset of available warehouses (AW_o) with a transport cost of tc_{ow}. These warehouses are the only ones that can serve such orders because they are placed in the same area as the shop where the order was placed. In addition, each order has associated a set of a certain number of demanded items (D_o) to
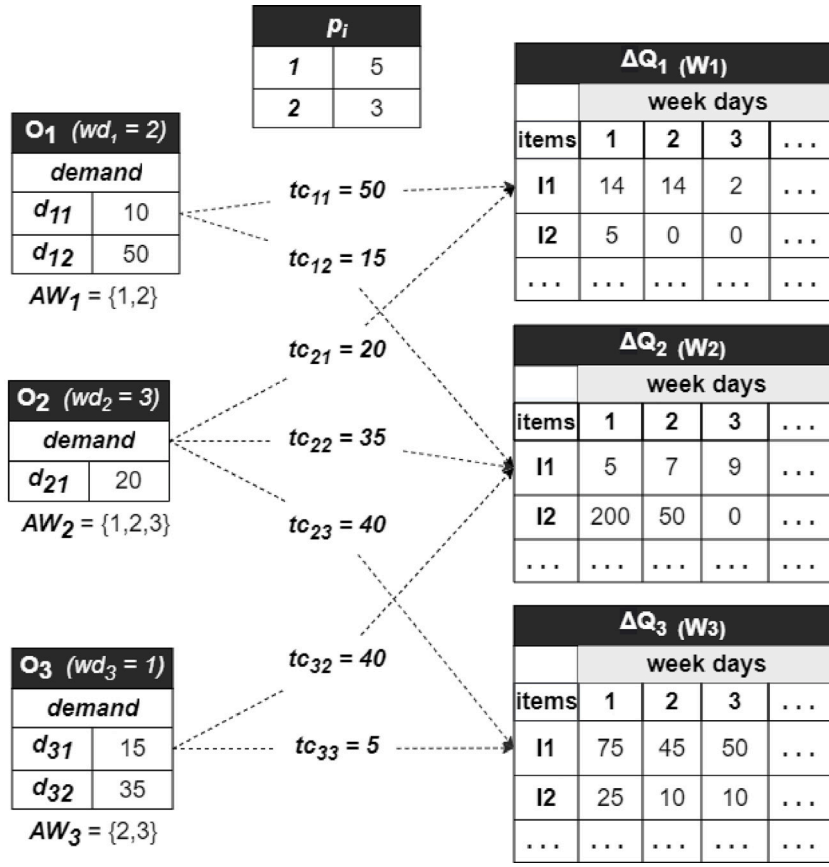
Fig. 1. Example of the input data of a toy instance.

be shipped on a specific weekday (wd_o). Furthermore, ΔQ_w represents the increment of the stock of each item in the warehouse w in each weekday.

For clarity purposes, Fig. 1 shows the input data of a toy instance. In the figure, there are three warehouses (see three tables on the right of the figure). The tables' rows represent the items served in the warehouse. The columns represent the days of the week. Then, each value in the table is the number of stock increments for each weekday. For example, in the warehouse W_3, there is an increment of 25 units of product I_2 on day 1, while on day 2, its increment of I_2 is 10 units. Note that ΔQ_w only represents the production of the stock (increment). However, the available stock in the warehouses depends not only on these increments but also on the orders that have already been allocated. Therefore, computing these current amounts represents a challenge. In Section 3.2, the variables for calculating these values are defined.

Fig. 1 also shows three orders (O_o) represented with matrices, showing the demand and the weekday in which the order has to be loaded (see tables on the left of the figure). Note that the orders have a list of available warehouses (AW_o). The transport costs (tc_{ow}) are shown with dashed lines. The central table shows the price of extra manufacturing for each unit of the items offered (p_i).

3.2. The formal mathematical model

This section first presents the formal mathematical model of the problem. Then, an optimized solution for the example of Fig. 1 is explained. The mathematical model is represented as:

$$min \sum_{w \in |W|} \left(\sum_{o \in |O|} x_{ow} * tc_{ow} + \sum_{t \in 1,\dots,7} pc(w,t) \right) \quad (1)$$

$$s.t. \quad \sum_{w \in AW_o} x_{ow} = 1, \forall o \in O \quad (2)$$

$$s_{wi}(t+1) = s_{wi}(t) + \Delta q_{wi}(t+1) - \sum_{o \in |O|, x_{ow}=1, wd_o=t+1} d_{oi}$$

$$\forall i \in I, \forall w \in W, \forall t \in 0,\dots,6 \quad (3)$$

$$pc(w,t) = \sum_{i \in |I|} pci(w,t,i) \quad (4)$$

where:

$$x_{ow} = \begin{cases} 1, & \text{if order } o \text{ is assigned to its } w\text{th available warehouse } (aw_{ow}) \\ & \forall o \in O, \forall w \in AW_o, \\ 0, & \text{otherwise} \end{cases}$$

$$(5)$$

$$pci(w,t,i) = \begin{cases} |s_{wi}(t)| * p_i, & s_{wi}(t) < 0 \\ 0, & otherwise \end{cases} \quad (6)$$

The objective function of the problem (Eq. (1)) minimizes the total sum of the costs of shipping orders (tc_{ow}) for all the orders (O) and the extra production of stock ($pc(w,t)$, see Eq. (4)) for all the days of the week ($t \in 1,\dots,7$) and for all the warehouses (W) involved in the problem. Note that the transportation costs are only summed when an order has been assigned to a warehouse ($x_{ow} = 1$).

Eq. (2) ensures that each order must be only assigned to one of its available warehouses. Note that all the orders must be assigned. This is achieved by summing all the boolean variables x_{ow} associated with each order $o \in O$ and ensuring that the result is equal to one.
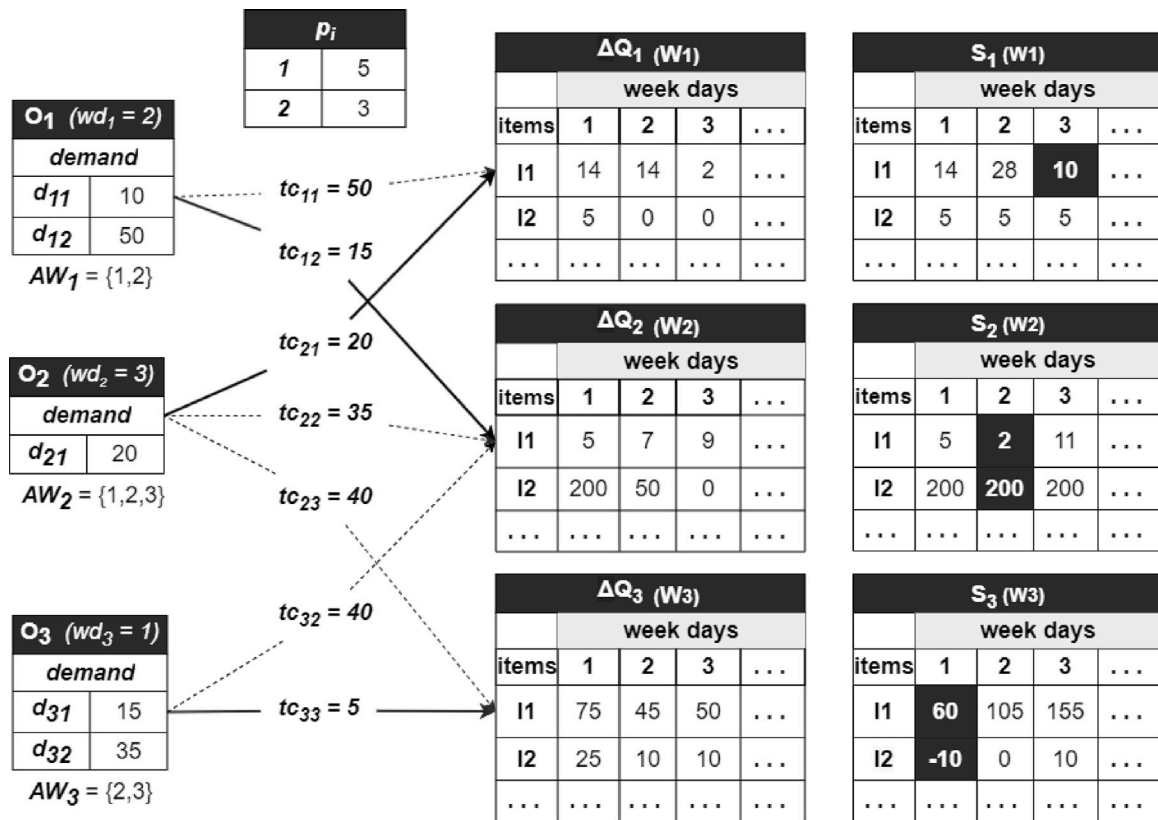
Fig. 2. An optimized solution and updated stocks for the toy instance of Fig. 1.

Furthermore, a set of variables $S_w = \{s_{w1}(1), \ldots, s_{wi}(t+1)\}$ for keeping track of the current amount of stock in the warehouses is defined. They are necessary for being able to correctly assign orders. Then, a new matrix of variables that has the same size as ΔQ_w is created. Recall that ΔQ_w contains all the daily increments of the stock. Eq. (3) computes $s_{wi}(t+1)$ based on the current stock of the previous day $s_{wi}(t)$ plus the increment units of the same day ($\Delta q_{wi}(t+1)$) minus the sum of the shipped units of all the orders of this weekday (see the sum of d_{oi} in the equation). If the result is negative, it means that there is a lack of units of the item i. This issue is solved by extra producing these units at a certain cost.

For computing the extra production cost of stock in a warehouse w for a specific weekday t, the variables $pc(w,t)$ are defined (see Eq. (4)). These variables are expressed as the sum of the extra production cost of each item produced in the warehouse $pci(w,t,i)$ (see Eq. (6)). They are computed as the absolute value of extra production units necessary $s_{wi}(t)$ (only if $s_{wi}(t)$ is a negative value) multiplied by the production price per unit of item i (p_i). Note that if $s_{wi}(t)$ is positive it means that there is more stock than demanded units. In this case, there is no extra production needed and therefore the extra production cost is zero.

As previously mentioned, the solution to the problem consists of assigning the orders to available warehouses. Thus, the boolean variables x_{ow} (see Eq. (5)) are defined, which take the value one if the order o is assigned to its available warehouse w and zero, otherwise.

Fig. 2 shows the optimized solution of the toy instance of Fig. 1. The orders assigned to a warehouse are shown with solid black arrows. Note that if it does not entail an extra production cost ($pc(w,t)$, see Eq. (4)), the optimal assignment for an order o is the warehouse with lower transportation costs (tc). For example, order O_1 can be assigned to the warehouses W_1 with a transportation cost of 50 and to W_2 with a transportation cost of 15, in both options, the warehouses can assume the stock supply without extra producing more units of items. Therefore, the optimal assignment is to W_1.

In this toy instance, there are only three orders and each one is assigned to a different warehouse. However, in real applications, there are thousands of orders and therefore typically the warehouses serve many orders on the same weekday. Thus, keeping updated the current stocks (tables S) each weekday is mandatory. Eq. (3) computes such values by taking into account the increments and decrements of stock on the previous weekdays.

For instance, in W_1 for item I_1 on day 3, the total number of units is the sum of the remaining ones on day 2 (28 units) plus the increment on day 3 (2 units) minus the demand of such item from O_2 (20 units), which is equal to 10 units. As previously mentioned, there might be current stock variables that have a negative value because these units must be extra-produced. For example, when order O_3 is assigned to W_3 on weekday 1, $s_{32}(1) = -10$ (i.e. $25 - 35 = -10$). Note that the total cost of making this delivery is $5 + (10 * 3) = 35$ (the unit production cost of I_2 is 3 (p_2)). This option is less costly than assigning O_3 to W_2, which implies a total cost of $40 + (10 * 5) = 90$.

Fig. 2 shows the optimized solution to the toy instance proposed in Fig. 1. Its objective function value is $(15 + 0) + (20 + 0) + (5 + 30) = 70$, where each parenthesis corresponds to each warehouse, being the first number the travel cost of the shipping (tc) and the last one, the extra production cost of stock (pc), where only is applied in the third warehouse.

4. Hybridization of GRASP and GA

This section describes the approach presented in this paper that hybridizes GRASP and GA for solving the industrial problem described in Section 3.

Fig. 3 shows the flowchart of the proposed hybrid algorithm. First, the GRASP algorithm sorts all orders according to a set of previously studied criteria. By iterating over a subset of values from this list, a local search method is applied to find the best possible assignment.
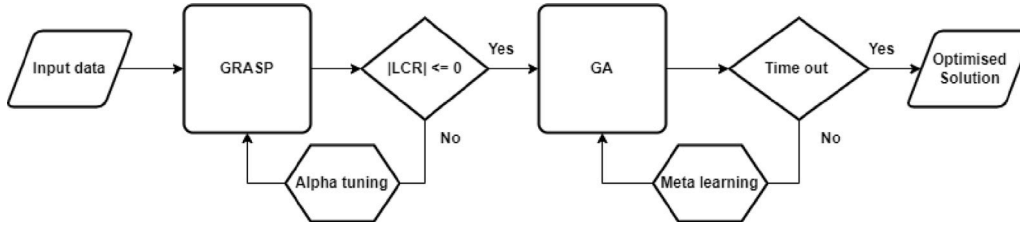
Fig. 3. Flowchart of the hybridization of GRASP and GA.

This process is repeated N times to obtain good solutions that will represent the initial population of the GA. Secondly, the GA applies crossover, mutation, and selection operations to the set of solutions generated by the GRASP. It will allow the population to evolve toward better solutions as the algorithm iterates. During the iterations of the algorithm, solutions that have a fitness value (Eq. (1)) worse than the best solution found are discarded. For both algorithms, GRASP and GA, it is developed the learning functions that tune their meta-parameters over the metaheuristic iterations. In the GRASP algorithm, the function *Alpha tuning* is proposed to adapt the objective function to the current state of the problem. A new method called *meta-learning* is implemented in the GA, in which a set of tests is executed to evaluate the changes in the algorithm's metadata and to select the best combination for the following iterations.

4.1. GRASP

One of the most well-known Multi Random Start Local Search (MRSLS) is the greedy adaptive search procedure (GRASP), which was introduced by Feo and Resende (Feo and Resende, 1989). Each GRASP iteration consists of iteratively constructing a solution in a greedy way and then applying a local search procedure to find a local optimum.

At the beginning of the GRASP (Greedy Randomized Adaptive Search) algorithm, a solution is obtained by randomly assigning available warehouses to orders. Note that as an MRSLS, for each iteration of Algorithm 1, a random solution is generated (see line 3). The orders for this solution are classified according to their restrictive character. This type of sorting prioritizes the most restrictive orders by considering two parameters: the number of available warehouses and the number of items associated with each order. Once the orders are sorted, a Restricted Candidate List (RCL) is created, and then, in the iterative part of GRASP (from line 5 to 21), the local search is performed. In this part, chunks of the RCL of size $n = 3$ are iterated. This size is chosen because, after testing with different sizes, it is observed that the allocation is dominated by sorting. In each of these chunks, the allocations of all the available warehouses for each order are evaluated. Based on all these evaluations, the objective function for each order is minimized, and the solution is modified with the newly allocated warehouse. This modified order is removed from the Restricted Candidate List (RCL) and repeated until the list is empty. At the end of each iteration, a tuning function is performed in which the α value is modified (according to some parameters discussed below), and the quantities of products in the warehouses affected by the allocation are updated.

Finally, when the RCL list is empty and the costs have been calculated, the solution is returned. This solution consists of tuples with two values: the order id and the assigned warehouse id.

4.1.1. RCL ordering:
The RCL list is sorted to search for the optimized solution in a more flexible way. Before the list is sorted, a preprocess automatically assign each order where only one warehouse is available, avoiding unnecessary calculations. The list is ordered according to the number of warehouses available for each order and the demand to be delivered for each order. To sort the list, Eq. (7) is used to normalize the number of warehouses and items associated with the order. This equation returns

---

Algorithm 1 GRASP

1: *input:* Alpha value α, RCL list chunk size n
2: *output:* Optimized solution x
3: $x \longleftarrow$ **RandomSolution**(O, AW)
4: $RCL \longleftarrow$ **RCLordering**(x, AW, D)
5: **while** $|RCL| > 0$ **do:**
6:    $RCLIndexs \longleftarrow [\quad]$
7:    $RCLCosts \longleftarrow [\quad]$
8:    $j \longleftarrow 0$
9:    **while** $j < n$ **do:**
10:       $index_{ow}, cost_{ow} \longleftarrow$ **LocalSearch**(α, RCL_j)
11:       $RCLIndexs \longleftarrow RCLIndexs \cup index_{ow}$
12:       $RCLCosts \longleftarrow RCLCosts \cup cost_{ow}$
13:       $j \longleftarrow j + 1$
14:    **end while**
15:    $bestIndex \longleftarrow min(RCLCosts)$
16:    $bestWarehouse \longleftarrow RCLIndex[bestIndex]$
17:    $x[bestIndex] \longleftarrow bestWarehouse$
18:    $RCL \longleftarrow RCL \setminus RCLIndex[bestIndex]$
19:    $\alpha \longleftarrow$ **AlphaTuning**$(\alpha, |RCL|, bestWarehouse, costBest)$
20:    $S \longleftarrow$ **StockUpdate**$(bestWarehouse, bestIndex)$
21: **end while**

---

a value between 0 and 2, where 0 is the least restrictive value to assign the order to a warehouse and 2 is the most restrictive. This range is obtained because each summation is normalized between 0 and 1. The first summation measures the availability of warehouses compared to all possible warehouse options, while the second one evaluates the demand load of the products relative to other orders. As a result of this normalization and the combination of these two factors, the equation produces an overall score that reflects the degree of constraint in the allocation process, effectively guiding the decision-making process.

$$\frac{|AW_o|}{|W|} + \frac{\sum_{i \in I} \frac{D_{oi}}{max(D_o)}}{|D_o|} \quad \forall o \in RCL \tag{7}$$

4.1.2. Local search
Once the RCL list is sorted, a local search method (see Algorithm 2) is executed in order to obtain the best possible warehouse for each order.

This method gives the objective function (line 15) of each available warehouse for an order. The associated costs between the trip and the assigned warehouse are added to obtain this value. The transport cost (tc_{ow}) is a fixed value given by the problem instance between an order o and a warehouse w. The stock cost ($stockCost$) is a calculated value for each item requested in the order (lines 9–13). This cost is only increased if the quantity of the item requested by the order is greater than the current stock in the evaluated warehouse. In this case, the quantity required by the order (line 10) is multiplied by the item production cost (P_i) to meet the demand. In Perez et al. (2020), it is observed that the transport cost is higher than the storage cost in the first iterations and vice versa in the last iterations. Therefore, once all the costs have been

Algorithm 2 Local Search

1: *input*: Alpha value α, Order id o
2: *output*: Id of warehouse assigned $warehouseId$
3: $t \longleftarrow wd_o$
4: $bestCost \longleftarrow 0$
5: $warehouseId \longleftarrow 0$
6: **For** w in AW_o:
7:    $stockCosts \longleftarrow [\quad]$
8:    **For** i in I:
9:      **if** $|d_{oi} - S_{wit}| < 0$ **then**:
10:        $stockCosts \longleftarrow stockCosts \cup P_i * |d_{oi} - S_{wit}|$
11:      **else**:
12:        $stockCosts \longleftarrow stockCosts \cup 0$
13:      **end if**
14:    **end for**
15:    $ObjFunction \longleftarrow \alpha * tc_{ow} + (1 - \alpha) * \sum stockCosts$
16:    **if** $bestCost == 0$ **or** $ObjFunction < bestCost$ **then**:
17:      $warehouseId \longleftarrow w$
18:      $bestCost \longleftarrow ObjFunction$
19:    **end if**
20: **end for**

---

calculated, an α variable is used to adjust the values of the objective function.

4.1.3. Alpha tuning:

With the best warehouses for each of the orders in the RCL chunk list, a tuning method is used to update the α value, adapting it to the current situation of the instance. This method allows making the objective function dynamic by adjusting the different costs.

Algorithm 3 Alpha tuning

1: *input*: Alpha value α, RCL size n, Order id o, Warehouse id w, Assignment cost $cost_w$
2: *output*: Alpha value α
3: $F_\alpha \longleftarrow (1 - \alpha)/n$
4: **if** $cost_w - tc_{ow} < 0$ **then**:
5:    $\alpha \longleftarrow \alpha + F_\alpha$
6: **else**:
7:    $\alpha \longleftarrow \alpha - F_\alpha$
8: **end if**

As discussed in previous sections, the transport cost is weighted by the variable α, and the stock cost is weighted by $1 - \alpha$. This smooths out greedy behavior during the middle and final iterations. The Algorithm 3 returns the remainder of the current alpha value, taking into account that $\alpha \in [0, \dots, 1]$. This value is divided by the number of unassigned orders in RCL to obtain a factor (line 3) by which to increase or decrease the variable α. The increase or decrease of the variable α depends on the stock cost of the last assigned order. For instance, if the order was assigned to a warehouse where the stock was less than the demand, the value of α will be increased by the previously calculated factor; otherwise, it will be decreased. Considering that the assigned warehouse is the one that has lower costs than the other warehouses, it can be assumed that the other warehouses have less stock than the demand. This indicates that, generally, the warehouses reduce their stocks, and the following costs associated with the allocations tend to increase.

4.1.4. Stock update:

After evaluating the costs of all orders in the local search method, the warehouse w is obtained for the order o with the lowest objective function value. To keep the stock values of the warehouses up to date,

the Algorithm 4 is used. This algorithm subtracts, for all items i, the demand (d_{oi}) used by the order assigned to the solution. Depending on the day the stock is consumed ($t == 0$ *or* $t > 0$), the stock of the previous days will have to be considered. In the line 6, it can be seen that, as it is the first day on which the stock is counted, only the demand is subtracted from the initial stock (Δq_{wit}) where $t == 0$. Otherwise, the current stock of the previous day (S_{wit-1}) has to be considered. The quantity needed for the current allocation ($\Delta q_{wit} - d_{oi}$) has to be subtracted from it, as can be seen in the line 8. In this way, the stock is updated after assigning a warehouse to an order. This algorithm allows us to consider the current stock in each GRASP iteration to obtain actual objective function values.

Algorithm 4 Stock update

1: *input*: Id of assigned warehouse w, Id of order o
2: *output*: Current amount of stock in the warehouses S_{wit}
3: $t \longleftarrow wd_o$
4: **For** i in I:
5:    **if** $t == 0$ **then**:
6:      $S_{wit} \longleftarrow min(\Delta q_{wit} - d_{oi}, 0)$
7:    **else**:
8:      $S_{wit} \longleftarrow min(S_{wit-1} + \Delta q_{wit} - d_{oi}, 0)$
9:    **end if**
10: **end for**

4.2. Genetic algorithm (GA)

This section describes the Genetic Algorithm (GA) proposed in this paper, including the crossover, mutation, and selection methods. Then, a learning system for obtaining the best parameters configuration is described.

GAs are commonly used to generate solutions according to an optimization criterion using operators inspired by biology. In a GA, a population of candidate solutions is evolved toward better solutions. Each of these solutions is represented as a chromosome, which is composed of a vector in which each position is an order defined in the problem, and the value of this position or gene is its assigned warehouse (see Fig. 4 for an example).

Algorithm 5 shows the main structure of the GA proposed in this paper. The inputs of this algorithm are the *alpha* parameter, the mutation ratio, the crossover ratio, the number of offspring in the crossover, and the number of survivors in each crossover.

First, the initial population and the fitness value are obtained. Over time, the different techniques are iterated to allow us to extend the search tree and obtain an optimized solution.

Algorithm 5 Genetic Algorithm

1: *input*: Initial population DS, Alpha value α, Mutation rate m, Crossover rate c, Crossover descendants number d, Selection survivors number s and Number of iterations in meta-learning *iters*
2: *output*: Best solution x
3: **while** *!timeout* **do**:
4:    $MP \longleftarrow [\alpha, m, c, d, s]$
5:    $\alpha, m, c, d, s \longleftarrow$ **meta-learning**(*iters*, MP, DS)
6:    $DS \longleftarrow$ **crossover**(c, d, DS)
7:    $DS \longleftarrow$ **mutation**(m, DS)
8:    $fitness \longleftarrow$ **getFitness**(α, DS)
9:    $DS \longleftarrow$ **selection**($s, fitness, DS$)
10:    **if** $x < min(fitness)$ **then**:
11:      $x \longleftarrow DS(min(fitness))$
12:    **end if**
13: **end while**

O_i	0	1	2	3	4	5	...	n
AW_O	7	13	2	9	4	3	...	N

Warehouse 7 of Order 0 Warehouse 2 of Order 2 Warehouse 4 of Order 4 Warehouse i of Order j

Fig. 4. Codification of a chromosome.

The 50% of the initial population is generated by the GRASP algorithm (line 1). The other 50% of the initial population is generated, from the previous solutions, by applying a randomization method. This method obtains a random amount of orders in each solution (between 30% and 50%) and changes the assigned warehouses to a different warehouse from the list of available ones. This method allows us to extend the search tree, generating various possibilities in the crossover function. Then, the GRASP solutions and the modified solutions are passed to the GA as input.

4.2.1. Meta-learning:

Meta-learning is the study of the metadata of learning algorithms to improve learning. Metadata is the set of algorithm parameters that provide information about the problem and its state. For our GA, the metadata are the crossover rates, mutation rates, and the number of crossover offspring. The goal is to improve the variability of the population by expanding the search space and to escape from local optima.

This method results in tests with different configurations of the above parameters. This set of tests is obtained using the Generalized Subset Designs (GSD) method explained in Surowiec et al. (2017). The GSD method efficiently explores parameter configurations in experimental tests. It involves selecting a subset of variables from a larger set and creating an orthogonal matrix representing different parameter combinations. Each configuration is evaluated using GA to assess its performance. GSD helps to identify significant factors, optimize solutions, and reduce the number of tests required, making it valuable in various domains, including engineering and artificial intelligence.

Algorithm 6 meta-learning

1: *input*: Number of iterations *iters*, Meta-learning parameters MP and Population DS
2: *output*: Best parameters δ
3: $ortogonal \longleftarrow$ **getOrtogonalArray**(MP_m, MP_c, MP_d)
4: $sol \longleftarrow [\quad]$
5: **For** o **in** $ortogonal$:
6: $\quad sol_o \longleftarrow [\quad]$
7: $\quad$ **For** i **in** $iters$:
8: $\quad\quad sol_o \longleftarrow sol_o \cup$ **GeneticAlgorithm**($DS, o_\alpha, o_m, o_c, MP_d, MP_s$)
9: $\quad$ **end for**
10: $\quad sol \longleftarrow sol \cup min(sol_o)$
11: **end for**
12: $\delta \longleftarrow$ **getBestParam**($sol, ortogonal$)

As shown in the Algorithm 6, this function receives the number of iterations, a set of the algorithm's parameters to improve and actual population. It then generates the orthogonal matrix (line 3) from a hypercube containing all possible combinations of the parameters. The hypercube is transformed into the original space of the parameters by assigning the values to a Latin square to obtain the set of tests designed. The function evaluates each configuration by running several iterations of the GA (line 8), obtaining the normalized standard deviation of the

objective function (Eq. (1)). The normalization is performed between 1 and -1, where 1 is the most significant standard deviation, and −1 is the smallest. To obtain the best configuration, the normalized deviations are summed where a given value of a parameter is used in the configuration. Then, the values of each parameter are ordered, and the values closest to 0 for each parameter are taken to form the best configuration of the tests. The sum of the deviations closest to 0 is obtained not to open the search space too much (the maximum deviation) but to avoid local optima (the minimum deviation).

4.2.2. Crossover:

Once the parameters have been obtained, the next objective is to cross the different current chromosomes to generate a new offspring population. To perform this crossover, random chromosome pairs are taken from the entire phenotype, and in the case of an odd phenotype size, the last chromosome that does not have a pair is reserved for the next iteration.

As shown in Fig. 5, each of the chromosome pairs obtained is fractionated according to the parameter c. This parameter indicates the size of the fraction of the first chromosome passed on to its first offspring (see the dark gray color in the figure). The first offspring's genes are made up of the $(1 - c)$ fraction of the second chromosome. The second offspring is composed of the fractions of both chromosomes that were not selected for the first offspring (see the light gray color in the figure).

4.2.3. Mutation:

To avoid local optima, GA performs a mutation process (see line 7 of Algorithm 5). This process consists of randomly changing the assigned warehouses in a set of orders in a solution.

The mutation is performed on both "parents", as the offspring are newly found solutions, and their fitness value has not been reached and could deteriorate in the mutation process. The variable m is the percentage of orders to mutate in parent solutions. This method only considers orders with one available warehouse ($|AW_o| > 1$), to avoid losing a possible warehouse reassignment in an order. The value of the variable m is obtained by the function meta-learning and is between 3% and 5%.

Fig. 6 shows the mutation performed on the first chromosome. In the upper vector, the current solution is shown with the identifier of the warehouses and the number of warehouses available for each order. In the first position, it can be seen that the first order with only one available warehouse does not enter the mutation-selection. Then, the orders to be modified are chosen, and a random selection is made among the available warehouses to modify the solution with the new ones.

4.2.4. Selection:

Once all the chromosomes have gone through the crossover and mutation process, the value of the objective function is obtained. This value ranks the chromosomes and their offspring against each other to decide which will be selected for the next iteration. The number of chromosomes selected for the next iteration is obtained from the s parameter of the algorithm input.
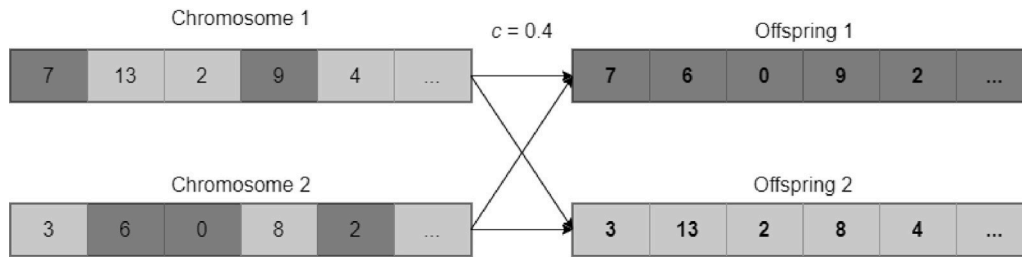
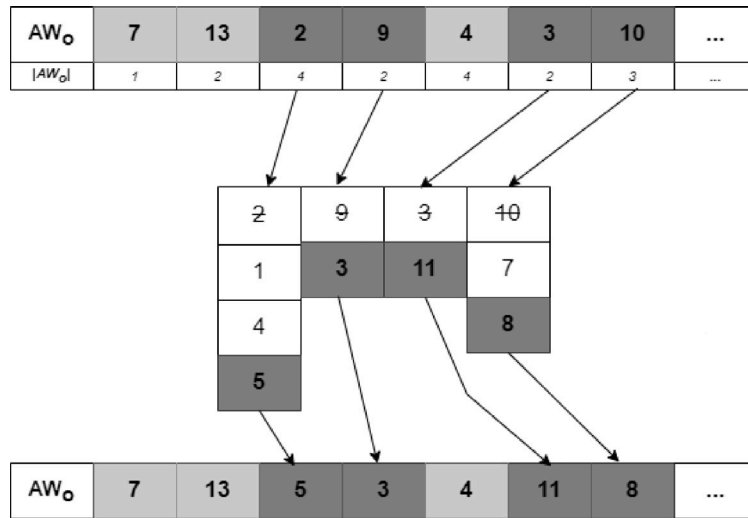Fig. 5. Random selection to compose the *n* offspring.
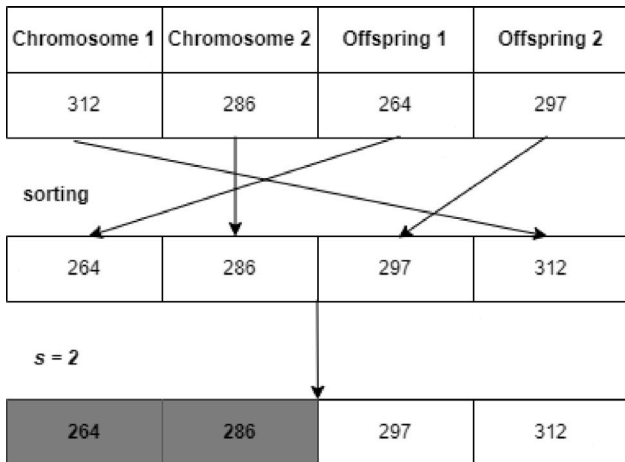

Fig. 6. Example of mutation of a chromosome.

Fig. 7. Example of selection in a family.

For example, Fig. 5 shows how both chromosomes in the crossover function produce two offspring. Then the fitness value of each of the four solutions is obtained and ordered from lowest to highest as shown in Fig. 7 to obtain the set *s* from the list finally.

5. Evaluation

In this section, an empirical evaluation of the proposed hybrid system is conducted by analyzing its behavior in terms of efficiency, solution quality, and scalability. All the experiments were performed on an Intel 2.20 GHz i7-8 gen CPU with 32Gb of RAM. The evaluation

was carried out on a set of instances from a statistical study of actual data obtained from the industrial partner. The dataset contains a set of orders placed by customers that need to be fulfilled in the following week (seven-day time frame), including the vehicles available to deliver the orders (between one and fourteen vehicles per order), the delivery cost, and the production price of each item. Randomized instances of the data sets, as well as the implementation of the mathematical model in Minizinc, can be found in this repository: https://github.com/GPS-UPV/SCSP.

The evaluation is divided into two comprehensive assessments. In the first part of the evaluation, the solutions were compared with the state-of-the-art complete and incomplete solvers. Specifically, two constraint programming (CP) solvers and two local search solvers. Then, the solver that obtained the best results was used as a baseline for comparing it with the proposed approaches. The purpose of this evaluation was to analyze the scalability and performance of the approaches, starting from small sizes of instances up to actual large-scale instances (approximately 2000 orders).

In the second part of the evaluation, the focus was on analyzing the efficiency of the proposed methods for the actual large-scale instances. The aim was to analyze how the quality of the solution improves over the computation time and which metaheuristic was more efficient based on the instance size and time cut-off. Notably, the CP solver was not used for this evaluation as it did not scale for these instances' sizes. The emphasis then shifts to comparing the proposed approaches with a focus on how the quality of the solutions was affected by the meta-learning.

5.1. Evaluating the performance and scalability

In this section, the mathematical model of the problem (see Section 3) was evaluated with several solvers. The sizes of the instances

Table 1
Total costs divided by the number of orders per week for some state-of-the-art solvers, in absolute values.

Size	Minizinc			IBM ILOG
	GECODE 6.3.0	YUCK 2023	OR-TOOLS 9.6	CP Optimizer
10	**112,4**	**112,4**	**112,4**	**112,4**
20	337,6	18248,0	252,8	**126,4**
50	7709,2	86878,0	710,0	**142,0**
100	–	489307,8	1562,7	**156,3**
200	–	–	3166,5	**158,4**
500	–	–	8324,9	**173,3**
700	–	–	14669,0	**261,4**
1000	–	–	22893,4	**285,9**
1500	–	–	44259,1	**481,0**
2000	–	–	81806,8	**2185,4**

indicated the number of orders per week and they ranged from 10 to 2000. The tables show the average results of the total costs divided by the number of orders (per week). It was fixed as cut-off computation time (in seconds) as the size of the instances (a.k.a. number of orders per week).

Table 1 evaluates the following state-of-the-art solvers: GECODE 6.3.0, YUCK 2023, OR-TOOLS 9.6 (all three implemented in Minizinc), and the IBM ILOG CP optimizer. It is worth noticing that YUCK is the best local search solver in the Minizinc competitions since 2020 and OR-TOOLS is the best overall solver since 2019. The results of the evaluation show that CP Optimizer significantly outperformed the other solvers. Note that Gecode and Yuck were unable to find a solution in the given time from sizes of 100 and 200, respectively. The bigger the instances, the more outstanding the improvement of the costs of the solutions of CP Optimizer over the other solvers. For this reason, CP Optimizer was selected as the baseline solver for comparing the approaches presented in this paper.

Tables 2 and 3 show the results of the performance and scalability of the following algorithms: CP Optimizer, as the baseline solver, and GRASP, GA, and its hybridization (denoted as HYBRID), where the first sub-column of our solvers represents the original method (e.g. GRASP) and the second sub-column represents the original method with meta-learning, denoted as +T (e.g. GRASP + T). Table 2 shows total costs divided by the number of orders per week, so it represents the average cost of each order. It can be observed that CP Optimizer had better behavior for small instances. However, for bigger instances, the hybrid approaches outperformed all solvers. It must be taken into account that the company is working with instances of 1000 and 2000 orders, so CP optimizer, and therefore the solvers evaluated in Table 1, were unable to obtain a competitive solution in the given time. The percentage of improvement of the proposed algorithm over the baseline CP OPTI-MIZER is shown in Table 3. it is calculated as $\%\Delta = $ (CP Optimizer – Solver)/ CP Optimizer)*100. It can be observed that the improvement for large-scale instances is significant, with a saving of 92% for 2000 orders.

From the analysis of the results, it can be noticed that GRASP had good results even with smaller size/timeouts, while GA was not so competitive. However, for large instances, both techniques were competitive concerning CP Optimizer. Nevertheless, the hybridization of the two approaches in our HYBRID approach provided superior performance in both small and large instances, outperforming both GRASP and GA.

The hybridization of GRASP and GA results in a competitive technique that takes advantage of the best features of both approaches. GRASP tends to restrict the initial search space due to its greedy tendency, favoring the discovery of solutions more likely to provide good performance. On the other hand, the GA opens up the possibility of escaping from local optima through the mutation operation, exploring different regions of the solution space. The hybridization of GRASP and GA leverages the strengths of both techniques, leading to an improved

search for optimal solutions. In addition, the meta-learning enhances the results, which translate into significant economic benefits for the company, saving thousands of euros per week. The hybrid approach contributes to the discovery of more cost-effective solutions, making it a valuable tool for the company's operations and profitability.

5.2. Evaluating the efficiency and solution quality

To analyze the efficiency and solution quality of the proposed methods, we focused on their performance over time in the three largest instance sizes. The results are shown in absolute and relative terms in Tables 4 and 5, respectively, where relative means how far the solution quality of the method is from the best-performing method for that instance size and timeout (e.g. in $Size = 1000$ and $Timeout = 10$ GRASP+T was the best method and GA+T was the worst with a 60.07% deterioration in performance).

Analyzing the absolute values from Table 4, it can be observed that the HYBRID approach produced the best results in almost all the evaluated cases. It confirms the hypothesis from the previous evaluation that the combination of both GRASP and GA methods would allow the hybrid system to perform similarly or better than the original method, which had the best performance in the analyzed instances. It can be observed that the hybrid system produces solutions of good quality in both small instances with low timeouts (i.e. maximum deterioration of 16.17% max against GRASP) and the best solution in any size with large timeouts.

An interesting follow-up to our previous evaluation is a deeper comparison between GRASP and GA. As observed before, and now confirmed with the Table 4, the size of the problem strongly influences the prediction of which method will perform better; in fact, for the smaller size of 1,000 orders, GRASP had better behavior in all timeouts, while for the largest instance of 2,000 orders, GA performed better than GRASP in all timeouts. This provides further confirmation of our insight into the exploration versus exploitation trade-off in Section 5.1. An interesting case study was the analysis of the instance of size 1500. In this case, GA outperformed GRASP until the 200 s timeout, after which GRASP outperformed GA. In fact, if the results without meta-learning are observed, the GA showed a better performance in this problem size, but the meta-learning tuning gave a better performance boost to the GRASP (around 10%) than to the GA (around 1%–2%).

In order to better analyze the effect of meta-learning on the proposed method, Table 6 shows the percentage distance from the method without meta-learning. Negative percentages are interpreted as meta-learning improving the minimization function and therefore generating a better solution. On the other hand, positive percentages mean that the meta-learning has increased the value of the minimization function and therefore worsened the solution. It can be observed that the meta-learning tuning always improved the solution quality for the highest timeouts of each size. However, for the GA-based methods (GA and HYBRID) the improvement was usually around 1% or less, while for GRASP the improvement was in double digits for the largest and medium sizes: 10.09% and 41.68% respectively. One hypothesis for this behavior is that the more time is allocated to the method, the more opportunities the meta-learning tuning has to explore the search space and thus escape from a local optimum.

Another interesting result is the fact that the method that received a larger performance boost from the meta-learning tuning was GRASP (double digits in the largest two sizes). Our insight is that the meta-learning tuning helps the GRASP to escape the early decision, which significantly restricts the search space. It should be noted that in size of 1000 orders, the meta-learning did not significantly improve the GRASP, however, it should be noted that the GRASP already produced a very good solution from the first timeout of 10 s and then improved the solution by less than 10% in the remaining 990 s, while in the same period, the HYBRID produced a solution that improved by more than 25%, reaching a better solution quality than the GRASP.

Table 2
Total costs divided by the number of orders per week of the hybrid approach, in absolute values.

Size	IBM ILOG	GRASP		GA		HYBRID	
	CP Optimizer	GRASP	GRASP+T	GA	GA+T	HYBRID	HYBRID+T
10	**112,4**	**112,4**	**112,4**	308,6	308,2	121,8	116,2
20	**126,4**	**126,4**	**126,4**	309,8	309,3	126,4	126,4
50	**142,0**	144,7	144,7	309,6	308,6	142,0	142,0
100	**156,3**	157,7	157,7	308,4	308,3	**156,3**	**156,3**
200	158,4	167,1	167,1	305,8	305,5	**155,7**	**155,7**
500	173,3	175,7	175,7	261,3	256,5	161,1	**160,9**
700	261,4	187,8	187,6	240,7	236,1	172,7	**170,3**
1000	285,9	188,8	188,5	218,4	215,4	174,3	**174,2**
1500	481,0	228,7	205,6	213,6	212,1	174,2	**173,9**
2000	2185,4	936,3	546,0	214,0	212,6	171,7	**171,2**

Table 3
Percentaje of improvement with respect CP Optimizer (calculated as %Δ = (CP Optimizer − Solver)/CP Optimizer)*100.

Size	IBM ILOG	GRASP		GA		HYBRID	
	CP Optimizer	GRASP	GRASP+T	GA	GA+T	HYBRID	HYBRID+T
10	**112,4**	**0,0%**	**0,0%**	−174,5%	−174,1%	−8,3%	−3,3%
20	**126,4**	**0,0%**	**0,0%**	−145,1%	−144,7%	0,0%	0,0%
50	**142,0**	−1,9%	−1,9%	−118,0%	−117,3%	**0,0%**	**0,0%**
100	**156,3**	−0,9%	−0,9%	−97,3%	−97,2%	**0,0%**	**0,0%**
200	158,4	−5,5%	−5,5%	−93,1%	−93,0%	1,7%	**1,7%**
500	173,3	−1,4%	−1,4%	−50,8%	−48,0%	7,1%	**7,2%**
700	261,4	28,2%	28,3%	7,9%	9,7%	33,9%	**34,9%**
1000	285,9	34,0%	34,1%	23,6%	24,7%	39,0%	**39,1%**
1500	481,0	52,5%	57,2%	55,6%	55,9%	63,8%	**63,8%**
2000	2185,4	57,2%	75,0%	90,2%	90,3%	92,1%	**92,2%**

Table 4
Efficiency and solution quality analysis in largest size instances with different timeouts.

Size	Timeout	GRASP		GA		HYBRID	
		GRASP	GRASP +T	GA	GA +T	HYBRID	HYBRID +T
1.000	10	193.25	**193.08**	308.93	309.06	292.89	224.31
	50	193.17	**193.08**	308.11	308.38	215.30	217.63
	100	193.14	**192.98**	307.24	307.58	211.95	212.60
	200	188.76	**188.54**	304.24	304.66	212.62	213.17
	500	188.76	**188.54**	258.44	260.43	191.61	188.67
	700	188.76	188.54	236.01	236.30	178.86	**178.62**
	1000	188.76	188.54	218.36	215.41	174.29	**174.17**
1.500	10	513.28	513.31	309.72	308.57	213.35	**195.48**
	50	423.79	449.82	309.83	306.80	213.35	**196.40**
	100	422.46	430.18	309.92	306.36	**195.41**	196.15
	200	408.09	411.39	305.62	303.72	194.48	**189.53**
	500	228.70	205.63	260.21	254.21	193.77	**179.88**
	700	228.70	205.63	237.52	230.53	188.95	**173.92**
	1000	228.70	205.63	219.06	214.95	179.02	**173.89**
	1500	228.70	205.63	213.97	212.13	174.20	**173.89**
2.000	10	955.91	955.80	309.67	308.90	263.13	**262.97**
	50	955.91	955.80	309.01	307.74	235.28	**231.99**
	100	955.91	955.80	308.64	307.29	193.11	**192.71**
	200	936.25	944.71	304.77	301.97	189.45	**188.85**
	500	936.25	546.02	255.88	255.25	190.36	**189.06**
	700	936.25	546.02	233.37	232.80	188.08	**187.42**
	1000	936.25	546.02	217.55	216.34	180.62	**180.47**
	1500	936.25	546.02	213.97	212.56	173.11	**172.56**
	2000	936.25	546.02	213.97	212.56	171.70	**171.22**

In conclusion, meta-learning tuning can be considered a very valuable tool to improve the performance of every single method, especially when tackling real-size instances. Moreover, the current results provided an interesting reflection to pursue further experiments and analyses on how to improve the meta-learning tuning in order to increase the performance boost for GA-based methods (GA and HYBRID).

6. Conclusions

This paper deals with an industrial application of a logistic problem that has been formulated as a multi-vehicle, multi-product, and single-echelon in which the warehouses serve two different purposes: storing the stock and manufacturing the required stock to market. To solve this problem, a formal mathematical model has been proposed, and an algorithm that hybridizes the GRASP and GA metaheuristics has been developed, for obtaining near-optimal solutions that minimize the costs of assigning the orders to the warehouses, i.e. minimizing the total transportation costs and the costs of producing extra stock. In addition, a meta-learning tuning method has been developed and embedded into the hybrid system to improve its performance.

To evaluate the proposed approach, the formal mathematical formulation has been implemented using commercial software (IBM ILOG and MINIZINC) and it has been compared with the state-of-the-art complete and incomplete solvers and benchmarked against the real case data

Table 5

Relative distance from the best performing method calculated as $\%\Delta = (Alg - Alg^-)/Alg^-$.

Size	Timeout	GRASP		GA		HYBRID	
		GRASP	GRASP + T	GA	GA + T	HYBRID	HYBRID + T
1.000	10	0.09%	**0.00%**	60.00%	60.07%	51.69%	16.17%
	50	0.05%	**0.00%**	59.58%	59.72%	11.51%	12.71%
	100	0.08%	**0.00%**	59.21%	59.39%	9.83%	10.17%
	200	0.12%	**0.00%**	61.36%	61.59%	12.77%	13.06%
	500	0.12%	**0.00%**	37.07%	38.13%	1.63%	0.07%
	700	5.68%	5.55%	32.13%	32.29%	0.14%	**0.00%**
	1000	8.38%	8.25%	25.37%	23.68%	0.07%	**0.00%**
1.500	10	162.58%	162.59%	58.44%	57.86%	9.14%	**0.00%**
	50	115.78%	129.04%	57.76%	56.21%	8.63%	**0.00%**
	100	116.20%	120.15%	58.60%	56.78%	**0.00%**	0.38%
	200	115.31%	117.06%	61.25%	60.25%	2.61%	**0.00%**
	500	27.14%	14.31%	44.66%	41.32%	7.72%	**0.00%**
	700	31.50%	18.23%	36.57%	32.55%	8.64%	**0.00%**
	1000	31.52%	18.26%	25.98%	23.62%	2.95%	**0.00%**
	1500	31.52%	18.26%	23.05%	22.00%	0.18%	**0.00%**
2.000	10	263.50%	263.46%	17.76%	17.47%	0.06%	**0.00%**
	50	312.05%	312.00%	33.20%	32.65%	1.42%	**0.00%**
	100	396.04%	395.98%	60.16%	59.46%	0.21%	**0.00%**
	200	395.76%	400.24%	61.38%	59.90%	0.32%	**0.00%**
	500	395.22%	188.81%	35.35%	35.01%	0.69%	**0.00%**
	700	399.55%	191.34%	24.52%	24.21%	0.36%	**0.00%**
	1000	418.77%	202.55%	20.54%	19.87%	0.08%	**0.00%**
	1500	442.55%	216.42%	23.99%	23.18%	0.32%	**0.00%**
	2000	446.82%	218.91%	24.97%	24.15%	0.28%	**0.00%**

Table 6

Meta-learning relative improvement calculated as $\Delta ML = (Ag^{+T} - Ag^{-T})/Ag^{-T}$.

Size	Timeout	GRASP ΔML	GA ΔML	HYBRID ΔML
1.000	10	−0.09%	0.04%	−23.42%
	50	−0.05%	0.09%	1.08%
	100	−0.08%	0.11%	0.31%
	200	−0.12%	0.14%	0.26%
	500	−0.12%	0.77%	−1.54%
	700	−0.12%	0.12%	−0.14%
	1000	−0.12%	−1.35%	−0.07%
1.500	10	0.01%	−0.37%	−8.38%
	50	6.14%	−0.98%	−7.95%
	100	1.83%	−1.15%	0.38%
	200	0.81%	−0.62%	−2.54%
	500	−10.09%	−2.31%	−7.17%
	700	−10.09%	−2.94%	−7.96%
	1000	−10.09%	−1.88%	−2.87%
	1500	−10.09%	−0.86%	−0.18%
2.000	10	−0.01%	−0.25%	−0.06%
	50	−0.01%	−0.41%	−1.40%
	100	−0.01%	−0.44%	−0.21%
	200	0.90%	−0.92%	−0.32%
	500	−41.68%	−0.25%	−0.68%
	700	−41.68%	−0.24%	−0.35%
	1000	−41.68%	−0.56%	−0.08%
	1500	−41.68%	−0.66%	−0.32%
	2000	−41.68%	−0.66%	−0.28%

large-scale instances, that are close, in size, to instances provided by our industrial partner. These results confirm the scalability and efficiency of the hybrid metaheuristic with meta-learning tuning as a very competitive approach to solving real-case instances for the Inventory Route Problem.

Declaration of competing interest

Data availability

I have shared the link to my data/code in the paper.

Acknowledgments

sets from the industrial partner, with extensive study of large-scale instances.

The evaluation section shows that the solutions obtained from the hybrid system with meta-learning significantly improve the results obtained by the baseline solver (CP Optimizer), with an improvement in the total cost up to 92.2% for realistic large-scale instances of size 2000 (number of orders per week) and near-optimal solutions for small size instances.

This paper also evaluates the performance of meta-learning in the hybrid algorithm. The tuning method provided significant performance improvements over the single metaheuristics (up to −41.68% improvements) for large-size instances). Thus, the evaluation shows that the proposed approach significantly outperforms state-of-the-art solvers for

References

Abualigah, L., Hanandeh, E.S., Zitar, R.A., Thanh, C.L., Khatir, S., Gandomi, A.H., 2023. Revolutionizing sustainable supply chain management: A review of meta-heuristics. Eng. Appl. Artif. Intell. 126, 106839. http://dx.doi.org/10.1016/j.engappai.2023.106839, URL: https://www.sciencedirect.com/science/article/pii/S0952197623010230.

Alorf, A., 2023. A survey of recently developed metaheuristics and their comparative analysis. Eng. Appl. Artif. Intell. 117, 105622. http://dx.doi.org/10.1016/j.engappai.2022.105622.

Alvarez, A., Cordeau, J.F., Jans, R., Munari, P., Morabito, R., 2020. Formulations, branch-and-cut and a hybrid heuristic algorithm for an inventory routing problem with perishable products. European J. Oper. Res. 283 (2), 511–529. http://dx.doi.org/10.1016/j.ejor.2019.11.015.

Arab, R., Ghaderi, S., Tavakkoli-Moghaddam, R., 2020. Bi-objective inventory routing problem with backhauls under transportation risks: two meta-heuristics. Transp. Lett. 12 (2), 113–129. http://dx.doi.org/10.1080/19427867.2018.1533624.

Archetti, C., Bertazzi, L., Hertz, A., Speranza, M.G., 2012. A hybrid heuristic for an inventory routing problem. INFORMS J. Comput. 24 (1), 101–116. http://dx.doi.org/10.1287/ijoc.1100.0439.

Archetti, C., Bertazzi, L., Laporte, G., Speranza, M.G., 2007. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. Transp. Sci. 41 (3), 382–391. http://dx.doi.org/10.1287/trsc.1060.0188.

Avella, P., Boccia, M., Wolsey, L.A., 2018. Single-period cutting planes for inventory routing problems. Transp. Sci. 52 (3), 497–508. http://dx.doi.org/10.1287/trsc.2016.0729.

Baumol, W.J., Wolfe, P., 1958. A warehouse-location problem. Oper. Res. 6 (2), 252–263. http://dx.doi.org/10.1287/opre.6.2.252.

Coelho, L.C., Cordeau, J.F., Laporte, G., 2014. Thirty years of inventory routing. Transp. Sci. 48 (1), 1–19. http://dx.doi.org/10.1287/trsc.2013.0472.

Cui, Z., Long, D.Z., Qi, J., Zhang, L., 2023. The inventory routing problem under uncertainty. Oper. Res. 71 (1), 378–395. http://dx.doi.org/10.1287/opre.2022.2407.

Feo, T.A., Resende, M.G., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. 8 (2), 67–71.

Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. J. Global Optim. 6 (2), 109–133. http://dx.doi.org/10.1007/BF01096763, Publisher: Kluwer Academic Publishers.

Festa, P., Pastore, T., Ferone, D., Juan, A.A., Bayliss, C., 2018. Integrating biased-randomized grasp with monte carlo simulation for solving the vehicle routing problem with stochastic demands. In: 2018 Winter Simulation Conference. WSC, IEEE, Gothenburg, Sweden, pp. 2989–3000. http://dx.doi.org/10.1109/WSC.2018.8632348.

Gherbi, Y.A., Lakdja, F., Bouzeboudja, H., Gherbi, F.Z., 2019. Hybridization of two metaheuristics for solving the combined economic and emission dispatch problem. Neural Comput. Appl. 31 (12), 8547–8559. http://dx.doi.org/10.1007/s00521-019-04151-7.

Gu, T., Li, S., Kou, Z., Wu, X., 2020. Hybrid optimisation algorithm for solving the multi-level spare parts inventory optimisation problem. IET Collab. Intell. Manuf. 2 (1), 14–21. http://dx.doi.org/10.1049/iet-cim.2019.0058.

Ho, W., Ho, G.T., Ji, P., Lau, H.C., 2008. A hybrid genetic algorithm for the multi-depot vehicle routing problem. Eng. Appl. Artif. Intell. 21 (4), 548–557. http://dx.doi.org/10.1016/j.engappai.2007.06.001.

Jaikishan, T.S., Patil, R., 2019. A reactive GRASP heuristic algorithm for vehicle routing problem with release date and due date incurring inventory holding cost and tardiness cost. In: 2019 IEEE International Conference on Industrial Engineering and Engineering Management. IEEM, (ISSN: 2157-362X) pp. 1393–1397. http://dx.doi.org/10.1109/IEEM44572.2019.8978851.

Mahjoob, M., Fazeli, S.S., Milanlouei, S., Tavassoli, L.S., Mirmozaffari, M., 2022. A modified adaptive genetic algorithm for multi-product multi-period inventory routing problem. Sustain. Oper. Comput. 3, 1–9. http://dx.doi.org/10.1016/j.susoc.2021.08.002.

Malladi, K.T., Sowlati, T., 2018. Sustainability aspects in inventory routing problem: A review of new trends in the literature. J. Clean. Prod. 197, 804–814. http://dx.doi.org/10.1016/j.jclepro.2018.06.224.

Mara, S.T.W., Kuo, R., Asih, A.M.S., 2021. Location-routing problem: a classification of recent research. Int. Trans. Oper. Res. 28 (6), 2941–2983. http://dx.doi.org/10.1111/itor.12950.

Moin, N., Salhi, S., Aziz, N., 2011. An efficient hybrid genetic algorithm for the multi-product multi-period inventory routing problem. Int. J. Prod. Econ. 133 (1), 334–343. http://dx.doi.org/10.1016/j.ijpe.2010.06.012.

Nikzamir, M., Baradaran, V., 2020. A healthcare logistic network considering stochastic emission of contamination: Bi-objective model and solution algorithm. Transp. Res. E 142, 102060. http://dx.doi.org/10.1016/j.tre.2020.102060.

Oudouar, F., Zaoui, E.M., 2022. A novel hybrid heuristic based on ant colony algorithm for solving multi-product inventory routing problem. In: Saidi, R., El Bhiri, B., Maleh, Y., Mosallam, A., Essaaidi, M. (Eds.), Advanced Technologies for Humanity. In: Lecture Notes on Data Engineering and Communications Technologies, vol. 110, Springer International Publishing, Cham, pp. 519–529. http://dx.doi.org/10.1007/978-3-030-94188-8_46.

Pellerin, R., Perrier, N., Berthaut, F., 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. European J. Oper. Res. 280 (2), 395–416. http://dx.doi.org/10.1016/j.ejor.2019.01.063.

Perez, C., Salido, M.A., Gurrea, D., 2020. A metaheuristic search technique for solving the warehouse stock management problem and the routing problem in a real company. In: SGAI 2020: Artificial Intelligence XXXVII. Springer, pp. 187–201. http://dx.doi.org/10.1007/978-3-030-63799-6_15.

Rafie-Majd, Z., Pasandideh, S.H.R., Naderi, B., 2018. Modelling and solving the integrated inventory-location-routing problem in a multi-period and multi-perishable product supply chain with uncertainty: Lagrangian relaxation algorithm. Comput. Chem. Eng. 109, 9–22. http://dx.doi.org/10.1016/j.compchemeng.2017.10.013.

Schiffer, M., Schneider, M., Walther, G., Laporte, G., 2019. Vehicle routing and location routing with intermediate stops: A review. Transp. Sci. 53 (2), 319–343. http://dx.doi.org/10.1287/trsc.2018.0836.

Silver, E.A., 1981. Operations research in inventory management: A review and critique. Oper. Res. 29 (4), 628–645. http://dx.doi.org/10.1287/opre.29.4.628.

Surowiec, I., Vikström, L., Hector, G., Johansson, E., Vikström, C., Trygg, J., 2017. Generalized subset designs in analytical chemistry. Anal. Chem. 89 (12), 6491–6497. http://dx.doi.org/10.1021/acs.analchem.7b00506, Publisher: American Chemical Society.

Talbi, E.G., 2013. In: Talbi, E.G. (Ed.), Hybrid Metaheuristics. Springer Berlin Heidelberg, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-30671-6.

Wang, M., Wu, J., Kafa, N., Klibi, W., 2020. Carbon emission-compliance green location-inventory problem with demand and carbon price uncertainties. Transp. Res. E 142, 102038. http://dx.doi.org/10.1016/j.tre.2020.102038.

Wu, W., Zhou, W., Lin, Y., Xie, Y., Jin, W., 2021. A hybrid metaheuristic algorithm for location inventory routing problem with time windows and fuel consumption. Expert Syst. Appl. 166, 114034. http://dx.doi.org/10.1016/j.eswa.2020.114034.

Zhang, S., Huang, K., Yuan, Y., 2021. Spare parts inventory management: A literature review. Sustainability 13 (5), 2460. http://dx.doi.org/10.3390/su13052460.