Journal of Global Optimization
https://doi.org/10.1007/s10898-020-00910-2

GRASP with Variable Neighborhood Descent for the online order batching problem

Sergio Gil-Borrás[1] · Eduardo G. Pardo[2] · Antonio Alonso-Ayuso[2] · Abraham Duarte[2]

Received: 24 May 2019 / Accepted: 16 April 2020

Abstract
The Online Order Batching Problem (OOBP) is a variant of the well-known Order Batching Problem (OBP). As in the OBP, the goal of this problem is to collect all the orders that arrive at a warehouse, following an order batching picking policy, while minimizing a particular objective function. Therefore, orders are grouped in batches, of a maximum predefined capacity, before being collected. Each batch is assigned to a single picker, who collects all the orders within the batch in a single route. Unlike the OBP, this variant presents the peculiarity that the orders considered in each instance are not fully available in the warehouse at the beginning of the day, but they can arrive at the system once the picking process has already begun. Then, batches have to be dynamically updated and, as a consequence, routes must too. In this paper, the maximum turnover time (maximum time that an order remains in the warehouse) and the maximum completion time (total collecting time of all orders received in the warehouse) are minimized. To that aim, we propose an algorithm based in the combination of a Greedy Randomized Adaptive Search Procedure and a Variable Neighborhood Descent. The best variant of our method has been tested over a large set of instances and it has been favorably compared with the best previous approach in the state of the art.

Keywords Warehouse management · Online order batching problem · Order batching · Turnover time · Heuristics

1 Introduction

The picking of items in a warehouse, as a part of the supply chain management, follows a picking policy which determines how the picking of items is performed in the warehouse. It is possible to find different picking policies in the literature such as: single-order picking,

This research was partially funded by the projects: MTM2015-63710-P, RTI2018-094269-B-I00, TIN2015-65460-C2-2-P and PGC2018-095322-B-C22 from Ministerio de Ciencia, Innovación y Universidades (Spain); by Comunidad de Madrid and European Regional Development Fund, Grant Ref. P2018/TCS-4566; and by Programa Propio de I+D+i de la Universidad Politécnica de Madrid (Programa 466A).

✉ Eduardo G. Pardo
eduardo.pardo@urjc.es

Extended author information available on the last page of the article

Published online: 11 May 2020

batching and sort-after-picking, single-order picking with zoning, batching with zoning, among others. Order Batching can be considered as a family of picking policies which are based on grouping the orders received in the warehouse into batches, prior to start the picking process. Once the batches have been conformed, all the items within the orders of the same batch are picked together in the same picking route. There are many optimization problems related to the process of picking items in a warehouse when the picking policy is order batching.

Within this family of problems, the most classical version is usually referred to as Order Batching Problem (OBP) [44], which consists of minimizing the total time needed to collect a group of orders received in a warehouse. This version has raised a relevant interest in the scientific community. The OBP has been proved to be $\mathcal{NP}$-hard for general instances [12]. Nonetheless, it is solvable in polynomial time if each batch does not contain more than two orders [12]. Unfortunately, real warehouse instances does not usually fall into this category. Consequently, it has been heuristically approached in the last years by using both, heuristics and metaheuristics. The First-Come First-Served (FCFS) strategy might be the first heuristic approach implemented in warehouses to assign orders to batches. This strategy has been widely used due to its simplicity. Other important heuristic methods are *seed methods* [13,22,32] and *saving methods* [40]. In [4] it is possible to find a survey of those methods where the authors proposed a classification.

More recently, it is also possible to find metaheuristic-based approaches in the literature. The first metaheuristic algorithm applied to the problem was described in [23], where a Genetic Algorithm is introduced. Later, in [1], an algorithm based on the Variable Neighborhood Search methodology is proposed; specifically, the authors considered several neighborhoods in a Variable Neighborhood Descent (VND) scheme. Henn et al. [19] proposed an Iterated Local Search and a Rank-Based Ant System algorithms and in [21] Henn improved previous results by introducing two additional algorithms: Tabu Search and Attribute-Based Hill Climber. In [30], Oncan proposed an Iterated Local Search algorithm with a Tabu Thresholding method as the local search procedure. As far as we know, the latest proposal and the current state of the art for the problem, was made in [26] where the authors proposed a multi-start algorithm based on the Variable Neighborhood Search (VNS) methodology. In each iteration of this algorithm it starts by generating a different initial solution, which is improved using a Basic VNS. Then a post-optimization strategy based on General VNS is applied.

The OBP has been extended with the inclusion of different constraints or alternative objective functions. The two more outstanding ones are: the Order Batching and Sequencing Problem (OBSP) which introduces a constraint related to the due date of each order (see for instance [20,25]). Also, the Min–Max Order Batching Problem (Min–Max OBP) consists of looking for a balance in the workload of a group of pickers, when considering multiple pickers to collect the batches. We refer the reader to [11,28] to review the state of the art of this problem.

All the previous approaches, within the Order Batching family, can be considered as static variants. However, the development of online e-commerce and the reduction of delivery times guaranteed by sellers make this *static* approach of the OBP very restrictive and unrealistic. In real contexts, orders are entering continuously in the system and it is necessary to modify the initial batch allocation, to be able to attend the new orders within the window of time committed to the client. This dynamic version of the problem is known as the Online Order Batching Problem (OOBP). Therefore, the list of orders arrived to the warehouse is updated online and the algorithms have to create batches and routes without having the complete information of all the orders that need to be collected in a working day, nor the arrival time

of future orders not yet in the warehouse. Note that a fundamental difference of this problem compared to its static version is that the method of creating batches and routes is working the whole day. In the static version, the system provides an initial solution that is valid for the entire working day. In case that the order list is updated, the system must resolve a new problem from the beginning with the current set of orders. In the dynamic approach presented in this paper, the system is working continuously and every time a new order enters in the system, it is incorporated into the list of orders and considered immediately for obtaining efficient solutions.

As it is the case of the OBP, in the online version it is also possible to find different variants of the problem. As far as we know, the first contribution related to the OOBP can be traced batch to 1997, when a variant of the of the OOBP with multiple pickers was tackled in [44]. In this paper, the objective function consists of minimizing the turnover time. The authors proposed a simple FCFS method as a batching strategy and a Traversal (S-Shape) routing algorithm to determine the route for the pickers. Later, in 2007 [45] the OOBP was studied for multiple-block warehouses. In this case, the objective function tackled minimizes the average customer order throughput time, which considers the time dedicated to three of the main activities involved in the process of the orders (batching, picking, and sorting). Furthermore, the authors of this paper studied the influence of several factors of the problem in the determination of the optimal solution, such as the batch size or the allocation of the workers. Also, they compared different strategies for sorting the items within the same batch, such as Sort-While-Pick and the Pick-And-Sort. In 2011, Rubrico et al. [41] tackled the Online Rescheduling Problem with multiple pickers. They proposed two heuristic methods based on the Steepest Descent Insertion strategy, and on the Multistage Rescheduling strategy. Both combined with the S-Shape routing algorithm. In this case, the authors minimized the maximum total travel distance traversed by each picker. A combination of the study of the influence of multiple blocks and multiple pickers is tackled in [2]. The authors, studied the time window (Fixed Time Window vs Variable Time Window) which determine whether to wait for new orders or to try to sort again the orders into batches. This time, S-Shape and Largest Gap routing algorithms were compared, and the objective function is to minimize the maximum total travel distance traversed by each picker. Other recent variants of the problem tackles the OOBP integrated with the scheduling of the delivery, trying to minimize the service time of an order [47]. Later, in [46] the same problem was tackled, but considering multiple pickers. This time, the workload of the pickers is also studied. In [48], new constraints are introduced in the same problem. This time the authors considered multiple delivery zones and a capacity limit in the vehicle used to make the delivery.

The variant of the OOBP studied in this paper was first tackled in [18]. In this case the authors considered a single-block warehouse and only one picker. To tackle the problem, they proposed a heuristic algorithm based on Iterated Local Search (ILS) to minimize the maximum completion time of the customer orders, which arrive to the system within a certain time period. However, other objective functions, such as the maximum and the average time that an order remains in the system, were also reported. The proposed method was compared with a classical FCFS algorithm and with a Clarke and Wright method. This variant of the problem was also tackled later in [34], where the authors proposed a variant of the well-known Estimation of Distribution Algorithm (EDA) to tackle the problem. This time, the authors reported the average distance traversed by the picker as objective function. However, they did not compare their proposal with the latest algorithm in the state of the art for the problem [18], but with a previous version from the same authors based on Tabu Search methodology proposed in [21] and originally designed for the static version of the problem. Additionally,

the proposed EDA was not able to improve the results by the Tabu Search. Again, these authors used the S-Shape method as a routing strategy.

It is well documented that order picking operations are one of the most important and costly processes in a warehouse [3,7]. Moreover, if the two decisions concerning the order picking (i.e., batching and routing) are simultaneously considered, the associated benefits can be substantially increased. According to [6], it is possible to reduce the travel time up to 35%, simply by properly designing the routes of the order pickers. Therefore, a key element that strongly affects the performance of these algorithms is the sequence used by each picker to retrieve the items in each batch. This problem classifies as a Steiner Traveling Salesman Problem (see [5]) embedded in a special kind metric space with properties that can be exploited to develop powerful heuristics. Ratliff and Rosenthal [37] proposed a polynomial optimal procedure based on dynamic programming for routing in a rectangular warehouse. The procedure is computationally efficient for warehouses with no cross aisles; however, the efficiency decreases when the warehouse has cross aisles and, in some cases, the proposed routes seems to be *illogical* to the pickers who, as a result, deviate their routes from the specified ones [12]. Alternatively, different routing heuristics have been proposed in the literature (see [15,35,39]). Petersen in [35] carried out a number of numerical experiments to compare six routing methods, concluding that *composite* [35,36] and *largest gap* were the best options among the studied methods. Recent approaches [26] empirically found that the *Combined* method, originally proposed in [38] was the strategy which performed better in single-block rectangular-shaped warehouses.

In this paper, we propose a novel algorithm to tackle the online order batching problem. Particularly, our proposal is based on two well-known metaheuristics: Greedy Randomized Adaptive Search Procedure (GRASP) [10] and Variable Neighborhood Descent [29]. The former is used as a general framework to build efficient starting points for the VND, which is in charge of improving the solution provided by GRASP. The VND used here is a classical basic implementation of the VNS framework. The proposed method outperforms previous attempts in the state of the art.

The rest of the paper is organized as follows: in Sect. 2 we present the variant of the OOBP problem tackled in detail. In Sect. 3 we introduce the different algorithms proposed for solving the problem. Section 4 presents the main computational results obtained, when the algorithms are applied to different warehouse layouts. Finally, the main conclusions of the work and the outline of future research plans are collected in Sect. 5.

2 Problem definition

In this paper we tackle the online order batching problem with a single picker in a one-block warehouse. As it was described in Sect. 1, this problem consists of collecting all the orders made by customers which arrive to a warehouse, minimizing a predefined objective function.

An order is a list of items demanded by a customer which are stored in the warehouse. Orders are grouped in batches of a predefined maximum capacity, before being collected. All the orders in the same batch are collected together, by the same picker, in a single route. In this sense, an order can not be split into more than one batch.

An important issue, in the variant tackled in this paper, is that the orders that have to be managed in a working day, are not fully available at the beginning of the day, but they arrive to the warehouse while pickers are working. This is why the problem is considered online.

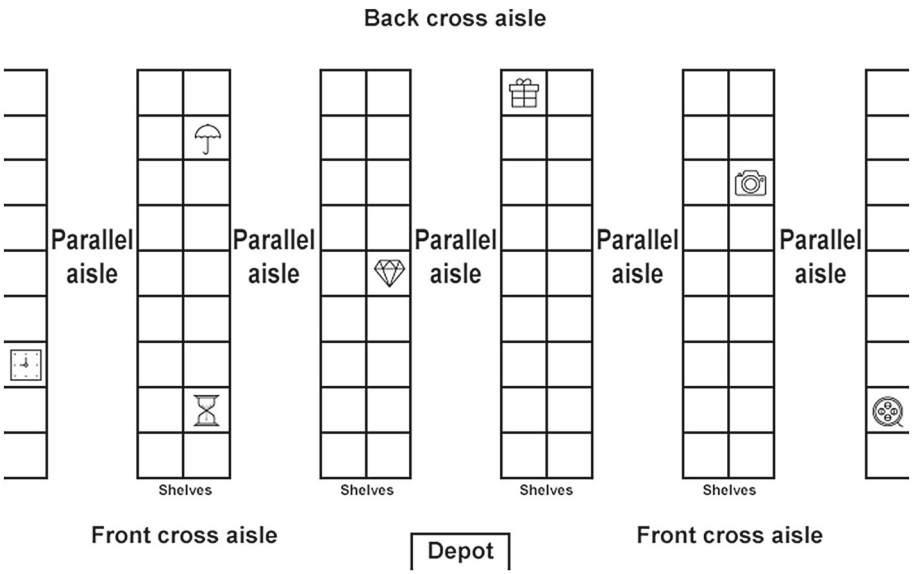


Fig. 1 Warehouse layout

The OOBP studied here considers only warehouses with a rectangular layout. This is, the warehouse is composed of two cross aisles (one at the front, and one at the back) and a variable number of parallel aisles. An example of this warehouse layout is depicted in Fig. 1. In this example, the warehouse has five parallel aisles formed by shelves at each side of the aisle, to store items. Particularly, each parallel aisle consist of 18 picking positions considering the shelves at both sides of the aisle.

Pickers start and finish their routes in a specific place of the warehouse, called depot. This depot is the place where the collected items must be handed once they have been retrieved. The depot is always placed in the front cross aisle, either in the middle of the aisle or in the leftmost corner.

The OOBP consist of three different tasks: batching, selecting, and routing. The batching task consists of grouping the orders already available in the system into batches. Then, once the batches are conformed, another algorithm has to select which, among all the conformed batches, is going to be collected next. Finally, a route to collect the orders within the selected batch need to be built. A picker will then start the retrieving process of all items in the batch, following the route previously built. When the picker finishes the collection of items and delivers them into the depot, a new batch and route is assigned to the picker to carry on his/her work.

In this paper we consider the minimization of two different objective functions related to the OOBP: minimizing the maximum completion time of all batches and minimizing the maximum turnover time of any order. These objective functions have been previously reported in the literature, however, they are considered separately (i.e., they can not be considered in a multi-objective optimization problem, since they are not necessary in conflict).

For a better understanding of these two objective functions, in Fig. 2 we show the life cycle of an order o_i through the timeline. Notice that, since this problem is considered online, the timeline is not bounded. This means that orders are arriving to the system continuously (i.e., 24 h a day/7 days a week) and we just observe what happens in the system in a particular

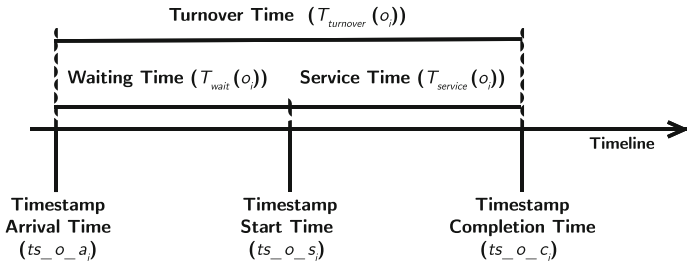


Fig. 2 Life cycle of an order o_i through the timeline

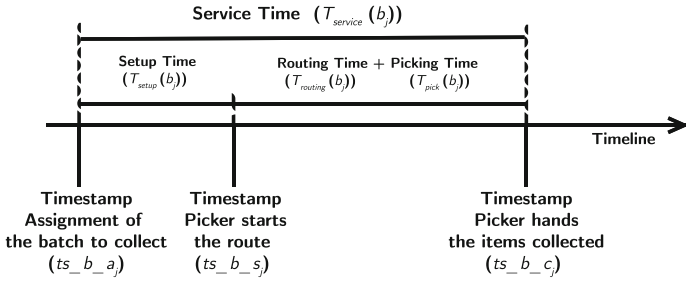


Fig. 3 Life cycle of a batch b_j through the timeline

chunk of time, in order to be able to observe the behavior of an algorithm and to compare it with other algorithms.

In this timeline we have highlighted three important timestamps (ts) in the life cycle of any order o_i: the arrival time ($ts\_o\_a_i$), the starting time ($ts\_o\_s_i$), and the completion time ($ts\_o\_c_i$). The arrival time represents the moment in the time when the order arrives to the warehouse. The starting time represents the moment in the time when the picker starts the route to collect the batch that contains the order. Finally, the completion time represents the moment in the time when the batch, that contains the order, is fully processed and the items in the orders collected are handed in the depot. With those three moments in the time at hand, it is also possible to define three periods of time in the life cycle of the order: the waiting time, $T_{wait}(o_i)$, the service time, also known as makespan, $T_{service}(o_i)$, and the turnover time, $T_{turnover}(o_i)$. The waiting time represents the time that an order remains in the system before being collected. It can be calculated as follows: $T_{wait}(o_i) = ts\_o\_s_i - ts\_o\_a_i$. The service time is the time needed to collect an order once the picker has started the route. It can be calculated as follows: $T_{service}(o_i) = ts\_o\_c_i - ts\_o\_s_i$. Finally, the turnover time is the time that an order remains in the system, either waiting, or being collected. It can be calculated as follows: $T_{turnover}(o_i) = ts\_o\_c_i - ts\_o\_a_i$.

It is important to highlight that, in the context of the OOBP, orders are not collected individually, but they are collected in batches (i.e., a group of orders that are collected together in a single route). Therefore, in Fig. 3 we represent a general schema of the life cycle of a batch through the timeline.

In this timeline we have highlighted three important timestamps in the life cycle of any batch b_j: the assignment of a batch to the picker ($ts\_b\_a_j$), the starting time when the picker initiates the route to collect the items in the orders assigned to the batch ($ts\_b\_s_j$), and the completion time of the batch ($ts\_b\_c_j$), when the picker hands the items collected in the route, into the depot. With those three moments in the time at hand, it is also possible to

Table 1 Parameters and variables for the OBP

Parameters		
n	$\rightarrow$	Number of customer orders available at the system in a given timestamp
m	$\rightarrow$	Upper bound of the number of batches (a straightforward value is $m = n$)
$v_{routing}$	$\rightarrow$	Routing velocity: number of length units that the picker can traverse in the warehouse per unit of time
v_{pick}	$\rightarrow$	Number of items that the picker can search and pick per time unit
w_i	$\rightarrow$	Number of items of order o_i for $1 \leq i \leq n$
W	$\rightarrow$	Maximun number of articles that can be included in a batch (device capacity)
Variables		
x_{ji}	$\rightarrow$	$\begin{cases} 1, & \text{if order } o_i \text{ is assigned to batch } b_j, \\ 0, & \text{otherwise} \end{cases}$

define different periods of time in the life cycle of any batch: the service time, $T_{service}(b_j)$, the setup time (T_{setup}), the routing time $T_{routing}(b_j)$, and the picking time, $T_{pick}(b_j)$. The setup time represents the time that the system and the picker need to prepare the picking cart, to receive and analyze the list of assigned orders (the batch) and to perform any other administrative task before the route starts. It is usually considered constant for any batch, and it is defined in the problem instance. Once the picker is ready to departure, the routing and pick times are the times needed by the pick to traverse the aisles, looking for the items in the orders assigned, and to extract (search and pick) any item from the shelves, respectively. The sum of the setup, routing and pick times, is known as the service time. Notice that the service time is the time available for the algorithms to compose a new disposition of batches, considering only the orders that have arrived to the warehouse in a previous moment in the time that have not been collected yet. Therefore, the service time can be calculated as follows:

$$T_{service}(b_j) = T_{routing}(b_j) + T_{pick}(b_j) + T_{setup}, \quad \forall j \in \{1, \ldots, m\}.$$

Next, we formally define the OBP based in the formulation presented in [18]. Notice that the OOBP is equal to the OBP if we consider a particular instant in the time (i.e., it only takes in consideration the orders already in the system, presupposing that no more orders will arrive later). First, in Table 1, we introduce the parameters and variables needed to define the problem correctly. Then, the objective functions and constraints which define the OBP variants tackled in this paper are presented.

The routing time is determined by routing algorithm. For the sake of simplicity, we consider a function $\mathrm{d}(b_j)$ that receives the orders in the batch b_j and returns the traveled distance to collect those orders. This function depends on the routing strategy that will be presented in Sect. 3.3. Therefore, considering a velocity $v_{routing}$, the routing time is defined as follows:

$$T_{routing}(b_j) = \frac{\mathrm{d}(b_j)}{v_{routing}}, \quad \forall j \in \{1, \ldots, m\}.$$

Considering that w_i is the number of items in the order o_i assigned to b_j, and v_{pick} is the number of items that the picker is able to search and pick per unit of time, the picking time

for a batch b_j can by defined as follows:

$$T_{pick}(b_j) = \sum_{i=1}^{n} \frac{w_i x_{ji}}{v_{pick}}, \quad \forall j \in \{1, \ldots, m\}.$$

Finally, T_{setup}, the setup time, is considered a parameter and it is specified in the particular instance.

The first objective is to minimize the maximum completion time of any order and it is given by:

$$\min \max_{j \in \{1,\ldots,m\}} \big(ts\_b\_s_j + T_{service}(b_j)\big). \tag{1}$$

It is worth mentioning that this objective is determined by the moment in the time in which the picker delivers the last batch.

The second objective function considered in this paper is to minimize the maximum turnover time of the received orders. The turnover time of an order o_i, $T_{turnover}(o_i)$, can be calculated as follows:

$$T_{turnover}(o_i) = \sum_{j=1}^{m} \big(ts\_b\_s_j + T_{service}(b_j)\big) x_{ji} - ts\_o\_a_i, \quad \forall i \in \{1, \ldots, n\}.$$

And then, the second objective function can be expressed as:

$$\min \max_{i \in \{1,\ldots,n\}} T_{turnover}(o_i). \tag{2}$$

Note that the value of this objective function is determined by the turnover time of the order that remains longer in the system.

The set of feasible solutions, in both cases, is given by the following constraints:

– Constraints in (3) guarantee that each order is assigned only to one batch:

$$\sum_{j=1}^{m} x_{ji} = 1, \quad \forall i \in \{1, \ldots, n\}. \tag{3}$$

– Constraints in (4) guarantee that the maximum capacity of each batch is not exceeded:

$$\sum_{i=1}^{n} w_i x_{ji} \leq W, \quad \forall j \in \{1, \ldots, m\}. \tag{4}$$

– Constraints in (5) guarantee that the batch b_j starts to be collected, once the batch b_{j-1} has been collected:

$$ts\_b\_s_j \geq ts\_b\_s_{j-1} + T_{service}(b_{j-1}), \quad \forall j \in \{2, \ldots, m\}. \tag{5}$$

– Constraints in (6) guarantee that the route for collecting a batch b_j can not start before the timestamps (moments in the time) when the orders o_i assigned to that batch have arrived to the system:

$$ts\_b\_s_j \geq ts\_o\_a_i x_{ji}, \quad \forall i \in \{1, \ldots, n\}, \text{ and } \forall j \in \{1, \ldots, m\}. \tag{6}$$

– Constraints in (7) and (8) state the non negativity of $ts\_b\_s_j$ and $ts\_o\_s_i$, respectively:

$$ts\_b\_s_j \geq 0, \quad \forall j \in \{1, \ldots, m\}. \tag{7}$$

$$ts\_o\_s_i \geq 0, \quad \forall i \in \{1, \ldots, n\}. \tag{8}$$

Finally, constraints in (9) state that variables x_{ji} are binary:

$$x_{ji} \in \{0, 1\}, \quad \forall\, j \in \{1, \ldots, m\} \text{ and } \forall\, i \in \{1, \ldots, n\}. \tag{9}$$

Despite of the fact that this model, proposed in [18], represents the OBP instead of the OOBP, it helps to understand the objective of the OOBP. Additionally, this formulation is a non-linear mixed integer programming model that can not be used to solve real instances using a solver. This partially supports the suitability of the use of heuristic algorithms in this context, as we propose in this paper.

It is important to remark that, in the case of the turnover time, the comparison of the value of the objective function of two solutions presents some additional difficulties. Let us introduce the concept of slot of time as the time between two consecutive departures of the picker. Notice that the construction of a solution in the context of the OOBP in a particular moment in the time consist of: i) grouping the orders available in the warehouse into batches, and ii) determining the sequence in which those batches should be collected. Then, this partial solution is evaluated and compared with other comparable partial solutions (i.e., those computed in the same slot of time). When the picker is ready for a new departure, the first batch sorted in the sequence of the best solution found during the slot of time is assigned to the picker. The process is then repeated.

Any order newly arrived to the warehouse must wait until the partial solution under construction is completed. Then, it can be included in the construction process of the next partial solution (no matters if the slot of time has not finished). Additionally, once all the orders have been assigned to a batch, the batch containing the oldest non-collected order (i.e., the first order arrived to the warehouse among the not collected ones) is selected to be assigned to the picker once the next slot of time starts.

3 Algorithms

In this section we describe our algorithmic proposal to tackle the OOBP. First, in Sect. 3.1 we describe the method in charge of the simulation of the general processes that happen in the warehouse. These methods include: considering the orders provided by a dispatcher; performing the batching of the considered orders; choosing the next batch to be collected; determining the route to collect the batch; and determining the departure moment of the picker. Once the general schema is at hand, in Sect. 3.2 we present the algorithms proposed to tackle the batching task. This section is divided into Sect. 3.2.1 where we introduce the Greedy Randomized Adaptive Search Procedure [10] as the constructive method, and Sect. 3.2.2 where we present the Variable Neighborhood Descent [29] used as a local search within the GRASP. Finally, in Sect. 3.3, we detail the routing strategy used.

3.1 General schema

The objective functions of the OOBP studied in this paper consists of either minimizing the total time elapsed in collecting all the orders arrived to a warehouse in a predefined time horizon, or alternatively, minimizing the turnover time. Due to the online nature of this problem, in order to compute these objective functions, it is necessary to have a general algorithm able to manage all the processes involved. We call this algorithm, the orchestration method and it is presented in Algorithm 1.

The orchestration algorithm receives two input parameters: the time horizon for the reception of orders ($maxTime$) and the list of pending orders at the beginning of the process ($listOrders$). Notice that this list of orders, in an online system, might refer to as the orders arrived at night time, or the orders pending to be collected from the previous working day. However, we consider that all the work from the previous day is already done.

The method runs while there are orders pending to be collected or the maximum allowed time has not been reached (step 4). Then, it checks if there are new orders arrived to the system (step 5) and updates the list of pending orders. Once this list has been updated, the batching algorithm is run (step 6) and the new solution is compared to the best found solution until the moment. Then, if there is a picker available and the solution contains batches not collected yet (step 8), the most suitable batch from the best solution ($bestSolution$) is chosen and the routing algorithm (step 10) will construct a route to collect the batch. Then, the picker will collect all the orders within the selected batch (step 11). Finally, the list of pending orders is updated, by removing the orders collected (step 12). Notice that the algorithm do not await until the picker comes back from its route but it is continuously running in order to have the best possible solution available as soon as the picker becomes available again.

Algorithm 1 Orchestration method

1: **Procedure** Orchestration($maxTime, listOrders$)
2: $pendingOrders \leftarrow listOrders$
3: $bestSolution \leftarrow \emptyset$
4: **while** (`getTime()` $< maxTime$) || ($pendingOrders \neq \emptyset$) **do**
5: $pendingOrders \leftarrow pendingOrders \cup$ `getNewOrders()`
6: $solution \leftarrow$ `batchingAlgorithm`($pendingOrders$)
7: `update`($bestSolution, solution$)
8: **if** `isPickerAvailable()` **then**
9: $batch \leftarrow$ `selectBatchAlgorithm`($bestSolution$)
10: $route \leftarrow$ `routingAlgorithm`($batch$)
11: `collect`($batch, route$)
12: `remove`($pendingOrders, batch$)
13: **end if**
14: **end while**

There are three remarkable methods within the orchestration procedure presented in Algorithm 1. The batching algorithm (`batchingAlgorithm`), which conforms the batches to be collected, is described in Sect. 3.2. The routing algorithm (`routingAlgorithm`), which determines the route that a picker must follow to collect a batch, is described in Sect. 3.3. Finally, the selection algorithm (`selectBatchAlgorithm`), which determines the next batch of the solution to be assigned to an available picker. In this case, we do not dedicate a whole section to the algorithm, since it follows a very simple heuristic. Particularly, this method selects the batch which contains the oldest order in the system. This method is commonly named as FIRST in the related literature [18].

3.2 Batching algorithm

As it was previously mentioned, the batching is one of the key procedures in the context of the OOBP. It consists of grouping all the orders received in a warehouse in a set of batches of a maximum predefined size with the aim of minimizing a particular objective function (typically the distance needed to collect all the orders).

In this paper we propose a batching algorithm based on the combination of a GRASP procedure (presented in Sect. 3.2.1 and used as a constructive method) and a VND (presented in Sect. 3.2.2 and used as a local search within the GRASP).

3.2.1 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure was introduced in [10] as a multi-start method to find high-quality solutions to hard optimization problems. Each iteration is composed of two steps: (1) construction and (2) improvement. The general schema of GRASP is presented in Algorithm 2. The construction phase combines the greediness of a particular greedy function and the randomization of some decisions during the construction. The constructive procedure within GRASP (step 4) proposed in this paper is explained next. On the other hand, the improvement phase (step 5), usually based on a local search method, is in this case based on a metaheuristic procedure. Particularly, we have replaced the local search with a VND method, explained in detail in Sect. 3.2.2. Therefore, in each iteration, the GRASP method constructs an efficient solution, and this solution is further improved with a VND procedure. This GRASP schema is repeated until the method runs out of time, or the maximum number of iterations is reached.

Algorithm 2 Greedy Randomized Adaptive Search Procedure

1: **Procedure** GRASP($maxTime$)
2: $bestSolution \leftarrow \emptyset$
3: **repeat**
4: $\quad solution' \leftarrow$ `Constructive()`
5: $\quad solution'' \leftarrow$ `LocalSearch(`$solution'$`)`
6: $\quad bestSolution \leftarrow$ `Update(`$bestSolution, solution''$`)`
7: **until** `getTime()` $> maxTime$
8: **return** $bestSolution$

The GRASP constructive method proposed in this paper is presented in Algorithm 3 and it is based on two basic principles: a greedy function that selects a group of candidate items to be added to the solution in the next iteration, and a particular randomization of the decisions made by that function. The method starts from an empty solution (step 2) and in each iteration it adds a new order to the solution. All available orders are initially inserted in the so called Candidate List (CL) (step 3). We propose the use of a greedy function (`f`) based on the weight of the orders in the CL in such a way that heaviest order is considered first (ties are broken at random). Therefore, the GRASP constructive sorts all the orders, already in the system but not yet assigned to a batch, in a descending way with respect to its weight. Then, a threshold th is calculated (step 6) based on the maximum ($\arg\max$ `f`(CL)) and minimum ($\arg\min$ `f`(CL)) weight of any order in the CL, and a random value $\alpha \in U[0, 1]$ (step 4). This threshold is used to determine the percentage of the best candidates to be included in a new list, called Restricted Candidate List (RCL) (step 7). Finally, an order is chosen at random from this RCL (step 8), and it is included in the partial solution being constructed in this iteration. The chosen order is inserted in the first batch with enough available space (step 9) and it is removed from the CL (step 10).

Notice that once a new order is added to the solution, it is necessary to determine the batch where it will be allocated. In the first iteration, a new empty batch is created. Then, in the following iterations, the algorithm tries to insert the selected order in this batch and

if it does not fit in the batch, then another empty batch is created to allocate this order, and so on. Notice that the sequence of batches used to try the insertion of the order is the same sequence in which the batches are created.

Classical stopping criteria for the GRASP are related to a particular number of iterations or a predefined time horizon. However, in this paper, we have run the GRASP method as long as there is not an idle picker. This means that each time that a new batch is assigned to a picker, the GRASP procedure is run again, but this time it does not consider any order which is being collected (i.e., it is in the assigned batch) or it has been collected before.

It is worth mentioning that due to the online context of this problem, the CL is being updated every time that the GRASP procedure starts again, with the latest orders arrived to the system.

Algorithm 3 Constructive procedure

1: **Procedure** Constructive($listOrders$)
2: $solution \leftarrow \emptyset$
3: $CL \leftarrow listOrders$
4: $\alpha \leftarrow$ `getRandomValue()`
5: **while** $CL \neq \emptyset$ **do**
6: $\quad th \leftarrow \arg\max\ \mathtt{f}(CL) - \alpha(\arg\max\ \mathtt{f}(CL) - \arg\min\ \mathtt{f}(CL))$
7: $\quad RCL \leftarrow$ `buildRestrictedCandidateList`(th, CL)
8: $\quad order \leftarrow$ `randomOrderSelection`(RLC)
9: $\quad$ `insertOrder`$(solution, order)$
10: $\quad CL \leftarrow CL \setminus \{order\}$
11: **end while**
12: **return** $solution$

3.2.2 Variable Neighborhood Descent

Variable Neighborhood Search is a metaheuristic proposed by Mladenović and Hansen in 1997 as a general method to solve hard optimization problems. The authors introduced the idea of changing the neighborhood structure within the search in order to reach different local optima. There are many variants of VNS. Some of the best-known are: Reduced VNS (RVNS), Variable Neighborhood Descent (VND), Basic VNS (BVNS), General VNS (GVNS), Skewed VNS (SVNS), and Variable Neighborhood Decomposition Search (VNDS) [16,17,29]. However, more recent approaches have appeared in the last few years, such as: Variable Formulation Search (VFS) [33], Parallel Variable Neighborhood Search (PVNS) [9,28], or Multi-Objective Variable Neighborhood Search (MO-VNS) [8].

In this paper, we propose the use of a VND procedure as a local search within the GRASP methodology. VND was proposed in the context of Variable Neighborhood Search in [29], as a general strategy to systematically explore a group of neighborhoods. The obtained result of a VND procedure is a local optimum with respect to all the neighborhood structures considered. The final order of the studied neighborhoods determines the performance of the method. Typically, neighborhood structures are sorted from the smallest and fastest to explore, to the largest one and slowliest to explore. However, this rule must be empirically tested when considering a particular problem and the associated neighborhood structures.

We use a standard and basic implementation of VND, which can be considered as the classical VND method. In Algorithm 4 we present the pseudocode of the VND procedure proposed in this paper. Particularly, this algorithm receives an initial solution as starting point

and it considers three different neighborhood structures (named in the pseudocode as N_1, N_2, and N_3) explored by a local search procedure. This local search procedure follows a first improvement strategy. The particular neighborhoods proposed are detailed ahead. The method will initially explore the first neighborhood (N_1) in step 7 and then it will determine if an improvement has been made (step 13) or not. If a neighborhood is not able to improve the current solution, then the method will jump to the next available neighborhood (step 17) until the maximum number of neighborhoods is reached (step 19). When the exploration of a neighborhood improves the current solution, the best solution found is updated and the method starts again from the first neighborhood (step 15).

Algorithm 4 Variable Neighborhood Descent

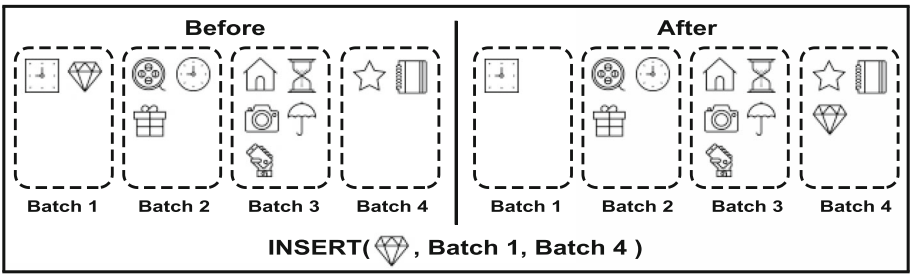
```
1: Procedure VND(solution)
2: k ← 1
3: k_max ← 3
4: bestSolution ← solution
5: repeat
6:    if k == 1 then
7:       solution' ← LocalSearch(bestSolution, N_1)
8:    else if k == 2 then
9:       solution' ← LocalSearch(bestSolution, N_2)
10:   else if k == 3 then
11:      solution' ← LocalSearch(bestSolution, N_3)
12:   end if
13:   if evaluate(solution') < evaluate(bestSolution) then
14:      bestSolution ← solution'
15:      k = 1
16:   else
17:      k = k + 1
18:   end if
19: until k > k_max
20: return bestSolution
```

We propose in this paper three different neighborhood structures to tackle the OOBP. These neighborhood structures are named `Insert`, `Swap1`, and `Swap2` respectively, and they are graphically shown in Fig. 4.

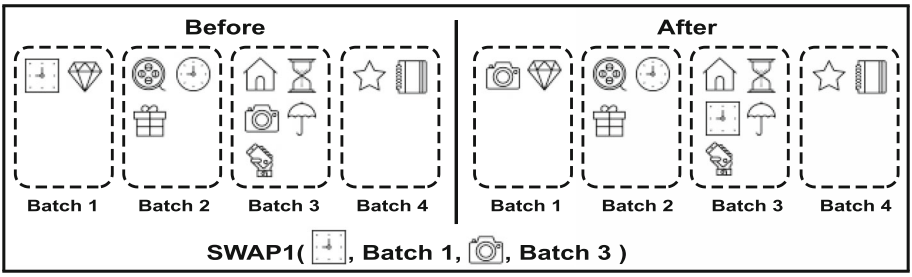
The `Insert` neighborhood, represented by an example in Fig. 4a, considers all possible solutions reached by the insertion of any order in the solution into all the available batches. In the example depicted in Fig. 4a it is represented the insertion of a diamond (originally allocated in Batch 1) into Batch 4. We depict the configuration of the batches before and after the `Insert` operation.

The `Swap1` neighborhood, represented by an example in Fig. 4b, considers all possible solutions reached by the interchange of any pair of orders in a different batch in the solution, into all the available batches. In the example depicted in Fig. 4b it is represented the exchange of a clock (originally allocated in Batch 1) with a photo camera (originally allocated in Batch 4). We depict the configuration of the batches before and after the `Swap1` operation.

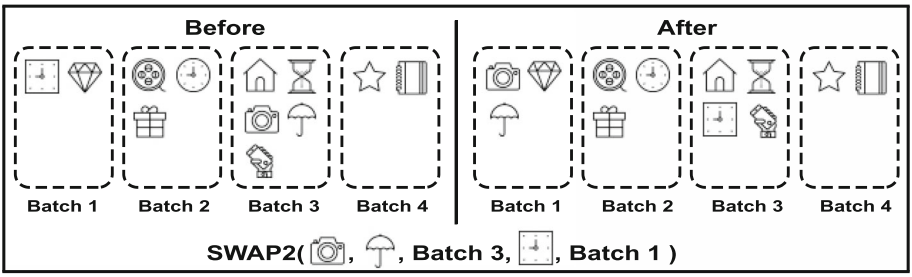
Finally, the `Swap2` neighborhood, represented by an example in Fig. 4c, considers all possible solutions reached by the exchange of every pair of two orders within the same batch, with any single order in any other batch. In the example depicted in Fig. 4c it is represented the exchange of a photo camera and an umbrella (originally allocated in Batch 3) with a clock (originally allocated in Batch 1). We depict the configuration of the batches before and after the `Swap2` operation.



a Insert 1x0.



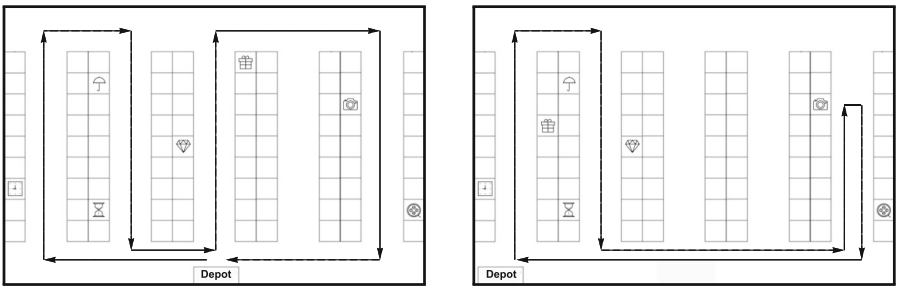
b Swap1 1x1.



c Swap2 2x1.

Fig. 4 Neighborhood structures

Notice that it is mandatory that the resulting batches do not violate the maximum capacity restriction on any batch. Otherwise the operation is considered unfeasible and, in this case the obtained solution after a move is not considered as part of the studied neighborhood.



a Warehouse layout with the depot placed in the center of the front cross aisle.

b Warehouse layout with the depot placed in the left corner of the front cross aisle.

Fig. 5 Route examples calculated with the S-Shape strategy

3.3 Routing algorithm

The routing algorithm determines the route that a picker must follow in order to collect all the items within the orders of the same batch. This route always starts and ends in the same point, the depot. The depot is placed in the front cross aisle, either in the center or in the left corner of the aisle.

In the literature there are a variety of algorithms to solve the routing problem in a warehouse. This problem can be considered a variant of the well-known Traveling Salesman Problem. There are heuristic, metaheuristic, and exact algorithms for solving the problem. In our case, we use the S-Shape heuristic method introduced in [5,14] that is widely used in the literature of the OBP and OOBP due to its simple implementation and fast performance. Additionally, the obtained routes are easily understandable by the pickers.

Given a batch, the S-Shape method identifies those parallel aisles where there are items to collect (at least one item). Then, each of those aisles are completely traversed from one cross aisle (either the front or the back cross aisle) to the opposite cross aisle. The first parallel aisle to be traversed is the leftmost aisle that contains an item to be collected. Then, the picker will enter only in the parallel aisles that contain at least one item that need to be collected. This process is repeated until the last aisle with items is traversed. Then, the picker returns to the depot using the front cross aisle. Notice that if the picker has to traverse an odd number of parallel aisles, the picker will enter in the last aisle from the frontal cross aisle. In this case, the picker will travel up to the most distant item and then he/she will perform an U-turn returning to the front cross aisle.

In Fig. 5 we show an example of two different routes designed using the S-Shape algorithm through a rectangular warehouse. Particularly, in Fig. 5a it is shown an example of a warehouse layout with the depot placed in the center of the front cross aisle. In this case, the route determined by the S-Shape algorithm traverses 4 different aisles before coming back to the depot. In Fig. 5b it is shown a different example of a warehouse layout where the depot is placed in the left corner of the front cross aisle. In this case, the picker must traverse an odd number of parallel aisles. Then, the two first parallel aisles are fully traversed and, in the last one, the picker performs an U-turn when it reaches the last item to collect.

4 Computational results

In this section we present the computational results obtained with the algorithms proposed in Sect. 3. First, in Sect. 4.1 we present the instances used in the experimentation. Then, we describe in Sect. 4.2 the configuration of the dispatcher of orders which determines the moment in the time of the arrival of each order to the system. In Sect. 4.3 we perform a set of preliminary experiments to adjust the parameters of the proposed methods and, additionally, to show the merit of the different parts of the final algorithm. Finally, in Sect. 4.4 we compare our best variant with the current state of the art of the OOBP for the tackled version of the problem.

The experiments were run an Intel (R) Core (TM) 2 Quad CPU Q6600 2.4 Ghz machine, with 4 GB DDR2 RAM memory. The operating system used was Ubuntu 18.04.1 64 bit LTS, and all the codes were developed in Java 8.

4.1 Instances

In order to test the algorithms proposed in this paper, we have selected two sets of instances derived from the previous literature in the state of the art of the OOBP. Notice, that an instance for the OOBP is formed by a group of parameters derived from: the warehouse characteristics, the list of orders, and the arrival scheduler. The warehouse is also defined by a certain group of parameters such as: the number of aisles, the length of each aisle, the sorting policy, etc. The list of orders indicates the products that must be collected to satisfy the demand by the customers. Finally, the arrival scheduler determines the moment in the time when a particular order arrives to the warehouse.

All the instances within the selected data sets present common characteristics, as far as the warehouse shape is concerned: rectangular shape, one block with two cross aisles (one at the front, one at the back) and a variable number of parallel aisles. This warehouse distribution is the same previously presented in Fig 1. However, despite of the fact that all instances have the same warehouse structure we consider different warehouse types, which differ in other parameters such as: the number of parallel aisles, the length of the aisles, the width of the aisles, the number of picking positions per aisle, etc. For each warehouse type, additionally, we study different lists of orders. These lists vary in the number of orders and each order varies in the number and composition of items.

The first data set used in the experimentation was introduced in [1] and it was originally defined for the static version of the OBP. The summary of the characteristics of this dataset is reported in Table 2. This dataset includes four different warehouse configurations (denoted as W1, W2, W3, and W4). For each warehouse configuration there are lists of orders whose size ranges from 50 to 250 orders. From the total number of instances originally proposed by the authors (2400), a selection of instances was made in [26] in order to establish a reduced dataset. In this paper, the authors found that using the whole dataset did not provide significant differences when compared to using only a reduced selection of instances. In this paper, we have used 64 instances from the reduced data set in order to perform our experiments.

The second data set used in our experiments was originally proposed in [18]. This data set is composed of 1600 instances. Again, a selection of instances was made in [26] in order to have a reduced data set which can be handled in a reasonable amount of time. In this case, we have also selected 64 diverse instances for our experiments. In Table 3 we present the main characteristics associated to this warehouse (W5). This time, the size of the lists of orders ranges from 40 to 100 orders.

Table 2 Warehouse characteristics [1]

	W1	W2	W3	W4
Storage policy	Random/ABC			
Depot position	Center/corner			
Order size	U(1, 7)	U(2, 10)	U(5, 25)	U(1, 36)
Item weight	1	1	1	U(1, 3)
Order picker capacity (weight)	12	24	150	80
Number of parallel aisles	4	10	25	12
Number of items per aisle	2×30	2×20	2×25	2×16
Total number of items	240	400	1250	384
Parallel aisle length (m)	50	10	50	80
Centre distance between two aisles (m)	4.3	2.4	5	15

Table 3 Warehouse characteristics [18]

	W5
Storage policy	Random/ABC
Depot position	Center
Order size	U(5, 25)
Item weight	1
Order picker capacity (weight)	30/45/60/75
Number of parallel aisles	10
Number of items per aisle	2×45
Total number of items	900
Parallel aisle length (m)	45
Centre distance between two aisles (m)	5

Notice that the two datasets selected have been widely used in the context of the OBP [4,24–28,42,43] and the OOBP [34,47,49]. Also, it is important to notice that the experiments in the context of the OOBP are performed in real time. This means that the time horizon of the arrival of orders is the minimum time that the algorithm needs to be run. Therefore, in the context of the OOBP it is not possible to consider a very large number of instances nor very long time horizons for the arrival of orders. In Table 4 we describe all the important parameters used in our experimentation. Among others, we include the parameters related to the arrival of orders to the warehouse. This configuration is common for the 2 data sets considered. In brief, the time horizon for the arrival of orders has been set to 4 h and the statistical distribution of the time instants of the arrivals is determined by an exponential distribution.

When considering the turnover time (maximum time that an order remains in the system) as objective function, it is necessary to set a moment in the time as a reference time. This allows the algorithm to calculate the amount of awaiting time of a particular order in the system with respect to that moment in the time. Otherwise, since the algorithm is continuously exploring new solutions, it is not possible to compare a solution with other one obtained in a previous moment, since the reference point in the time varies. This reference time is fixed and then updated each time that a picker receives a new batch to collect.

Table 4 Configuration of the parameters in the experiments

Arrival period	4 h
Dispatcher distribution	Exponential
Velocity of the picker (LU/min)	48
Time pick by item (items/min)	6
Setup time (min)	3
Time window fit (s)	400
Initial order in queue	0
Random seed	50
Routing strategy	S-shape
Start picking strategy	As soon as picker is available
Batch selection strategy	Batch with the oldest order

Table 5 Lambda values

#Orders	40	60	80	100	150	200	250
λ	0.167	0.250	0.333	0.417	0.625	0.833	1.042

4.2 Order dispatcher

The arrival of orders to the warehouse in the context of the OOBP is distributed through a particular time horizon. In this sense it is necessary to configure a simulation environment, which provides the orders to the system prior to calculate the batch configuration and later the picking route. The time horizon of the arrival of orders to the warehouse is set to 4 h.

In order to simulate the arrival time for each order, we follow a Poisson point process. Since the time horizon is set to 4 h ($t = 4$). The number of events in the interval of length t is a Poisson random variable $X(t)$ with mean $E[X(t)] = \lambda * t$. The λ value is selected depending on the number of orders considered in the experiment. In this case, the λ values chosen for our experiments are compiled in Table 5.

4.3 Preliminary experiments

In order to determine the best configuration of our algorithm and, also to test the contribution of each part of the algorithm, we have performed a set of preliminary experiments. To that aim, we have selected a subgroup of instances from the total datasets. Particularly, we have selected 16 out of 128 instances to perform the preliminary experiments. These instances have been selected in order to include in the reduced dataset the most diverse instances.

4.3.1 Determination of the best GRASP parameters

The GRASP constructive algorithm introduced in Sect. 3 has two main parameters: (i) the value of the α parameter, and (ii) the number of constructions. As far as the value of α is concerned, it determines the voracity in the selection of orders to be included in the Restricted Candidate List. In this paper, we use a greedy criterion based on the weight of the orders (the heaviest order is considered first).

Table 6 Performance of GRASP with different α values

	0.0	0.2	0.4	0.6	0.8	1.0	Random
Dev. (%)	0.84	0.64	0.59	0.53	0.62	1.03	0.42
#Best	3	3	1	3	1	1	4

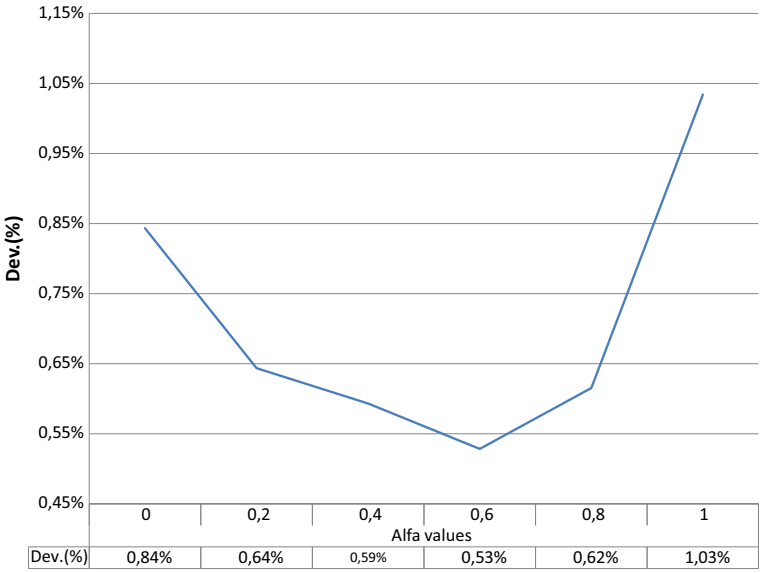


Fig. 6 Representation of the average deviation obtained with different α values

In the first preliminary experiment we test different values of α in order to determine the most suitable value. In Table 6 we report the average deviation with respect to the best value found in the experiment, and the number of best solutions found for different values of the parameter α. Notice, that when $\alpha = 1.0$ the inclusion of orders in the Restricted Candidate List is fully random. On the other hand, when $\alpha = 0.0$, the inclusion of orders in the Restricted Candidate List is fully greedy. The deviation obtained with the different values of α is (Alfa) represented in Fig. 6. As it is shown in the figure, the evolution of the average deviation to the best value found decreases until $\alpha = 0.6$ and increases from this point. However, the differences among the different values of α studied are very small. Therefore, we have also included in the Table 6 the results obtained when α takes a different and random value in each GRASP iteration. As it is possible to see in the table, the best solutions found in the experiment are obtained when α is set to a random value. Notice that this experiment has been performed by running the GRASP constructive algorithm in isolation for a time limit of 60 s.

Next, we test the influence of the second parameter related to GRASP (i.e., number of constructions). In this case, we are not interested in fixing the number of constructions of the algorithm, since it will depend on the available time between the routes of the picker. However, it is interesting to review the influence of the constructive procedure in the quality of the initial solution.

In Table 7 we report the evolution in the deviation of the GRASP constructive method with respect to the best value of the experiment, when considering from 10 to 10,000,000 of

Table 7 Evolution of deviation when increasing the number of GRASP constructions

	10	100	1,000	10,000	100,000	1,000,000	10,000,000
Dev. (%)	4.66	3.22	2.60	1.82	1.07	0.56	0.00
#Best	0	0	0	0	0	1	15
CPUt (s)	0.01	0.02	0.12	0.92	9.11	90.35	900.96

Table 8 Different neighborhood orders within the VND

	$\{N_3, N_1, N_2\}$	$\{N_3, N_2, N_1\}$	$\{N_3, N_2, N_1\}$	$\{N_2, N_1, N_3\}$
Dev. (%)	0.67	1.11	0.93	1.06
#Best	4	3	4	6
CPUt (s)	14.59	8.98	10.60	12.77

constructions. Particularly, we find big differences in the improvement of the deviation from 10 to 100 constructions, but this improvement is reduced when we increase the number of constructions. However, the time increases considerably. In order to relate this values with the real execution of the algorithm it is important to notice that an average picking route might last from 250 to 1200 s.

4.3.2 VND neighborhoods

The VND algorithm proposed in Sect. 3 includes three different neighborhoods (see Sect. 3.2.2), named as N_1, N_2, and N_3. One of the key decisions with respect to the neighborhoods included in a VND is determining the order in which they are placed within the algorithm. Many researchers set the neighborhood order depending on it size (smaller neighborhoods come first since they are faster to traverse). However, we have considered to place in first position not only the smaller neighborhood (N_3) but also the second smallest (N_2). With this at hand, we have reported the performance of the possible different orders of the neighborhoods within the VND (see Table 8).

As we can see in the Table 8 the best combination of neighborhoods is N_3, N_1, and N_2, with an average deviation of 0.67%. However the combination with the largest number of Best values is N_2, N_1, and N_3. In this case, we have decided to use the combination with the smallest deviation (N_3, N_1, and N_2) for our final configuration.

4.3.3 Contribution of each neighborhood

A key parameter when designing a VND is checking that all the neighborhoods included in the algorithm contribute to the process of search. In case any neighborhood do not show improvements it should be removed and the saved time used for other tasks. To test this fact, we have reported the number of improvements made with each neighborhood when they are combined in the VND method. Taking in consideration the preliminary data set, we have provided a random solution as starting point for the VND. In Table 9 we report the number of improvements performed by the local search which traverses each neighborhood. As it is shown in the table, all the neighborhoods produce improvements in the search. Particularly,

Table 9 Improvements produced by each local search within the VND

	#improvements	%improvements
N_1	24	35.82
N_2	26	38.81
N_3	17	25.37

Table 10 Comparison of the VND with respect to the local search

	VND	LS-1	LS-2	LS-3
Dev. (%)	0.12	5.42	8.34	10.21
#Best	15	1	0	0
CPUt (s)	22.89	4.03	13.83	1.13

N_1 and N_2 are the neighborhoods which produce the largest number of improvements (24 and 26 respectively). However, the difference with respect to N_3 is not very remarkable.

4.3.4 Comparison between VND and local search procedures

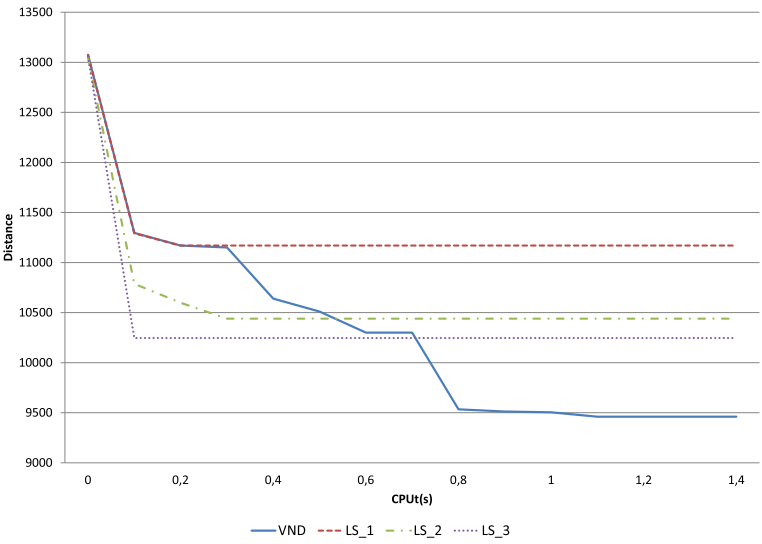
A local search procedure runs until no further improvements can be made in a neighborhood for a particular objective function, i.e., the obtained solution is a local optimum. On the other hand, the combination of different local search procedures within a VND must provide a local optimum with respect to all the neighborhood structures considered. Additionally, it is expected that the quality of the solution found by a VND procedure is better than the solution found by each local search in isolation.

In Table 10 we report the average deviation, the number of best solutions found, and the CPU time of four different algorithms. On one hand, we have run the VND procedure starting from a random solution, which was provided as an input parameter. This random solution was also provided to each of the local search procedures separately. In this case, it is possible to notice that the VND procedure is able to find a much better solution than each local search in isolation. However, as it was expected, the CPU time of the VND is larger than the CPU time of each local search independently.

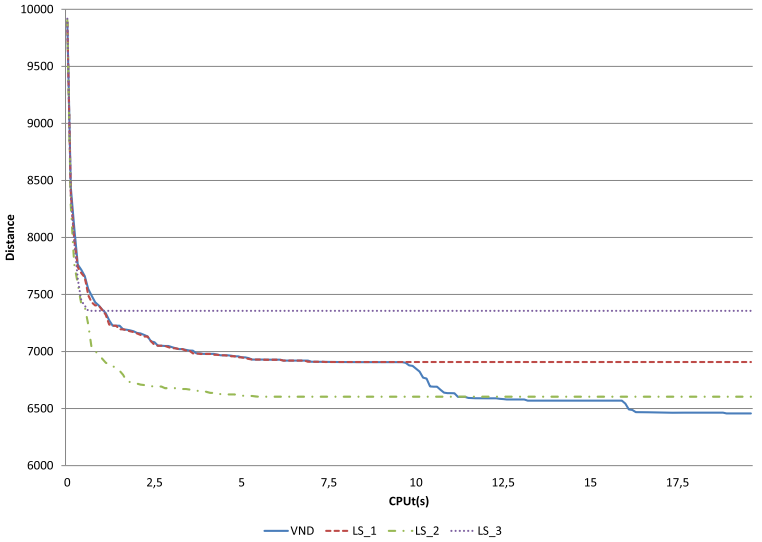
To complement the results presented in Table 10 we present the evolution in the objective function value over the time of each local search procedure compared to the VND method. In this experiment we have selected two representative instances from the preliminary dataset: one small (80 orders, represented in Fig. 7a) and one large (250 orders, represented in Fig. 7b). As it is shown in the figures, the performance of the VND is the best among all the methods compared, for the two instances considered. However the local search procedures converge to a local optimum faster than the VND.

4.3.5 Performance of the static exact approach in the state of the art

Our last preliminary experiment is devoted to illustrate the suitability of using heuristic approaches in the context of the OOBP, instead of using an exact approach. To that aim we have tested the performance of the mathematical formulation introduced in Sect. 2 over the preliminary data set. It is important to notice that the mathematical model proposed in the state of the art was designed for the static version of the problem (i.e., all the orders are available at the beginning of the process). Additionally, the model is incomplete, and it can



a Instance $W5\_abc2\_80\_46$ composed of 80 orders.



b Instance $W2\_250\_090$ composed of 250 orders.

Fig. 7 Evolution of the Local Search procedures and the VND method over two particular instances

not be directly executed in a solver, since the function "d" (which calculates the distance of a given solution) is not defined in the original paper where the model was proposed. In order to overtake this difficulty we have used the distance formulation proposed in [31].

We run the mathematical model for each instance on Gurobi 9.0.0rc2 (win64). Since the sizes of the instances considered are quite large for the model, we reduced the size of each instance by selecting a reduced number of orders (choosen at random) from each instance. We started in 5 orders per instance and increased the number of orders per instance one

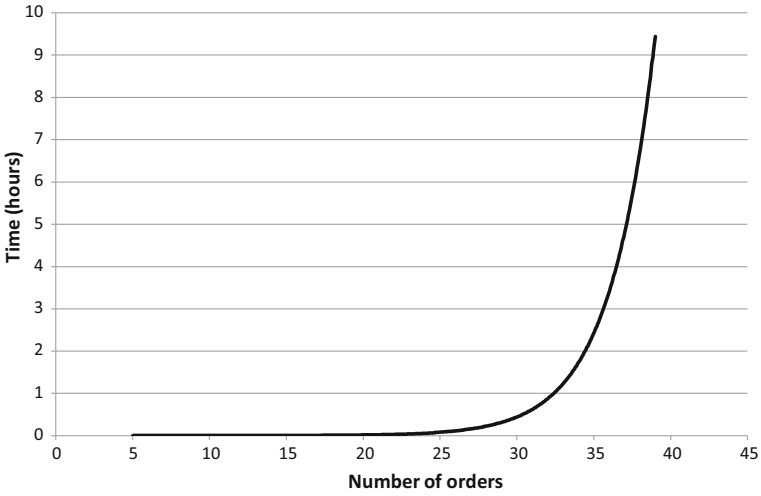


Fig. 8 Average performance of the mathematical formulation introduced in [18]

by one. As expected, the mathematical model was able to solve small-size instances to the optimal solution. However, when reaching the instance sizes considered in this paper (larger than 40 orders per instance) the time needed exceeded the time horizon considered.

In Fig. 8 we report the nonlinear regression obtained when considering the results provided by the solver, which shows the trend in the increase of time when the size of the instance also increases.

4.4 Final experiments

Once we have tested the different components of our algorithms, we have configured our best variant of the GRASP + VND method with the following parameters: the number of iterations of GRASP + VND is not predefined. The procedure is run as many times as possible between the departure of the picker to collect a batch, and the arrival of the picker to the depot after collecting all the items, plus the setup time. The α value in the GRASP constructive procedure has been set to random and it is reset in each iteration of the method. The three neighborhoods proposed in Sect. 3.2.2 have been included in the VND following the order: N_3, N_1, and N_2.

The best configuration of our algorithm GRASP + VND (GR+VND in the tables) is compared with the best method in the state of the art, the Iterated Local Search (ILS) proposed in [18], and denoted in the final experiments as ILS. This final comparison has been performed over the two datasets previously presented. We report the summary of the results in Tables 11 and 12.

In Table 11 we consider the minimization of the maximum completion time as objective function. This objective function was proposed in [18] and it returns the moment in the time when the last processed order is completed. In other words, this objective function equals the total time needed to collect all the orders arrived to the warehouse. The results for the other objective function considered in this paper are reported in Table 12. In this case, the comparison with the state of the art considers the minimization of the maximum turnover time of an order as objective function. This is the maximum time that an order stays in the

Table 11 Comparison with the state of the art considering the minimization of the maximum completion time as objective function

	Albareda		Henn		Total	
	GR + VND	ILS [18]	GR + VND	ILS [18]	GR + VND	ILS [18]
Dev. (%)	0.10	3.38	0.11	2.41	0.11	2.90
#Best	60	4	55	11	115	15

Table 12 Comparison with the state of the art considering the minimization of the maximum turnover time of an order as objective function

	Albareda		Henn		Total	
	GR + VND	ILS [18]	GR + VND	ILS [18]	GR + VND	ILS [18]
Dev. (%)	0.29	8.14	0.63	5.80	0.46	6.97
#Best	61	3	52	12	113	15

system (i.e. the difference between the arrival time to the warehouse and the time when it is completed).

Despite of the fact that we provide both objective functions, we consider that the most interesting one, attending to real scenarios for this problem, is the turnover time (see Table 12). In fact, our algorithms were only configured to minimize this objective function (not the maximum completion time). We only report the maximum completion time for the solutions found when minimizing the maximum turnover time. The main reason of including the maximum completion time is for the sake of clarity, since this objective function was the main objective function proposed in [18].

Considering the minimization of the maximum completion time reported in Table 11, we appreciate that the GRASP + VND is the best overall method, since it achieves a 0.11% of deviation with respect to the best values found in the experiment, while ILS achieves a 2.90% of deviation. Additionally, the number of best solutions found by GRASP + VND (115) is much larger than the number of best solutions found by ILS (15).

As far as the minimization of the maximum turnover time is concerned, we report the obtained values by each algorithm in Table 12. Again, GRASP + VND appears to be the best method. In this case, the obtained deviation (0.46%) is considerably better than the one by ILS (6.97%). The number of best found solutions by GRASP + VND (113) is also much larger than the number of best solutions found by ILS (15).

To complete the comparison of the results, we have carried out a test of mean difference of the results obtained by the two methods. Even when the hypothesis of normality is not verified, the size of the sample used (128 in total, 64 from Albareda and 64 from Henn) allows us to use the t-test for the difference of means in paired samples (note that both methods have been tested on the same instances). The result could not be more significant: with a p-value less than 10^{-10}, the null hypothesis $H_0 : \mu_{ILS} \leq \mu_{GV}$ can be rejected, where μ_{ILP} and μ_{GV} are the mean of the objective function (turnover or distance) by ILS and GRASP + VND, respectively. Therefore, it can be accepted that the results proposed by the GRASP + VND are statistically better than those provided by the ILS. We have performed these tests for the two sets of instances (Albareda and Henn) separately, and the results are similar, with p-values practically null in all cases.

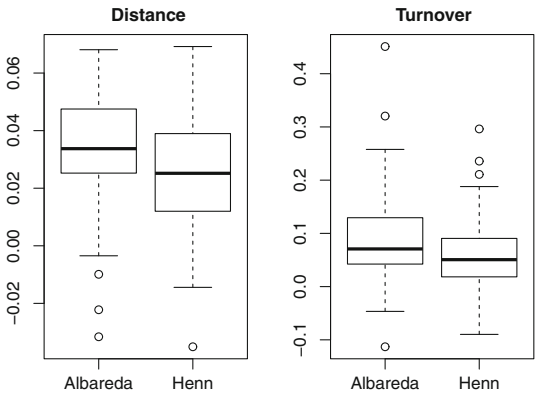


Fig. 9 Relative difference between objectives ILS and GRASP + VND

To conclude, we have analyzed the relative improvement of the GRASP + VND versus ILS. Figure 9 shows the distribution of the ratio $\frac{z_{ILP}-z_{GV}}{z_{GV}}$ for the two objectives and the two sets of instances, Albareda and Henn. The two boxplots on the right show the results for the turnover and the two on the left for the distance. For each of the objectives, the relative improvement is shown separately for Albareda and Henn instances. Note that negative values imply that ILS provides better results than GRASP + VND. From these boxplots, we can conclude again that in most cases GRASP + VND provides better solutions; in those few cases where the ILS solution is better, it does not exceed 2% improvement compared to almost 7% that GRASP + VND can improve the solution. It is interesting to note that GRASP + VND achieves lower improvements in Henn instances, which are precisely where the ILS was tested.

In order to facilitate future comparisons, we report the best values found in our experiments, for each of the considered instances. Particularly, in Table 13 in Appendix A, we report the results for the dataset introduced in [1], and the results for the data set introduced in [18] in Table 14 in Appendix B. All this information, together with the set of instances used in our experiments are available at http://grafo.etsii.urjc.es/optsicom/oobp/.

5 Conclusions

In this paper, we have tackled the online order batching problem with a single picker and in a single-block warehouse. This problem is a variant of the well-known family of problems commonly referred to as Order Batching Problem. It is based on the order batching picking strategy as an efficient way of retrieving the orders arrived in a warehouse, i.e., orders are grouped into batches before they are collected. Additionally, every batch is collected in a single route of a picker.

The OOBP presents the difficulty that all the orders that must be retrieved are not available at the beginning of the process, but they arrive to the warehouse at any time. For the sake of simplicity, we have considered a time horizon of 4 h for the arrival of orders, however, the picker might need longer times to collect all the orders arrived in that time horizon.

To achieve our goals we have proposed several heuristic algorithms which are combined within a GRASP and VND metaheuristics. Particularly, we propose a GRASP algorithm as a general framework to tackle the problem, and we use the VND metaheuristic as a local search procedure. The constructive phase of the GRASP is based on the weight of the orders as a greedy criterion (i.e., the heaviest, the first). The α parameter of GRASP has been

set to a random value on each iteration. The VND method proposed includes three different neighborhood structures that have been empirically sorted. Finally, we have used the S-Shape algorithm as the routing strategy to calculate the best route for collecting the orders grouped in each batch.

It is worth mentioning that in this paper we have considered two different objective functions: (i) minimize the maximum completion time; and (ii) minimize the maximum turnover time. The former, minimizes how long it takes to collect all the orders arrived to the system. This objective function is widely used in many papers. The latter minimizes the maximum time that an order remains in the system. As far as we understand the problem, this objective function represents the most realistic scenario for this variant of the OBP. However, we have reported and compared with previous proposals in the state of the art, the values obtained for both objective functions. The proposed algorithms improve the previous results in the state of the art in both cases, finding improvements of more than 3% and 6%, on average, for each objective function, respectively. The statistical tests performed corroborated in both cases that there are significant differences among the results found.

Our findings indicated that the running time for this problem is not one of the biggest issues, since the time horizon on a real scenario is very large. There are also many small factors that must be taken in consideration, such as the departure time of the picker, once the batches are ready to be collected. In a first approach, the sooner the better, however, awaiting might result in a final improvement, since there are more chances for the arrival of new orders to the system, that might be included in the next batch. Also, we found that the use of several neighborhoods is a key strategy, since it is not always easy to perform moves within the available space in the batches. We have detected that sorting the orders in a descending weight might help to obtain batches with less wasted space. Empirical results indicate that the fewer number of batches, the better.

It is important to highlight that when considering the total time needed to collect all the orders as objective function, the time of collecting a particular batch does not change, no matters the moment in the time when the route to collect this batch starts. However, when considering the turnover time as objective function, not only the batch configuration is important, but also the moment in the time when the picking route starts for each batch. This is an additional difficulty of this variant of the problem. In this sense, it is necessary to set a moment in the time as the hypothetical starting point, taken as a reference, to compare two different solutions.

As a final observation, the online version of the OBP makes possible many situations where a short number of orders are handled at the same time. Therefore, it could be an interesting opportunity for designing matheuristic algorithms in the future.

Appendix A: Results per instance (instance set Albareda [1])

Table 13 Results per instance for the dataset introduced in [1] over the two considered objective functions

Instance	Objective function		Instance	Objective function	
	T (s)	Turnover (s)		T (s)	Turnover (s)
W1_100_000	22, 308	10, 864	W3_100_000	39, 902	30, 959
W1_100_030	18, 371	6394	W3_100_030	33, 212	25, 539
W1_100_060	21, 676	9911	W3_100_060	41, 067	33, 276
W1_100_090	18, 313	6157	W3_100_090	33, 974	24, 111
W1_150_000	28, 612	18, 449	W3_150_000	54, 189	45, 495
W1_150_030	24, 386	13, 556	W3_150_030	46, 013	38, 616
W1_150_060	30, 770	18, 812	W3_150_060	58, 797	50, 833
W1_150_090	24, 101	12, 129	W3_150_090	46, 210	38, 277
W1_200_000	42, 091	29, 194	W3_200_000	72, 452	63, 494
W1_200_030	30, 810	18, 461	W3_200_030	60, 713	54, 808
W1_200_060	38, 485	27, 209	W3_200_060	79, 633	72, 394
W1_200_090	31, 405	20, 774	W3_200_090	57, 516	52, 072
W1_250_000	53, 101	40, 683	W3_250_000	93, 604	84, 912
W1_250_030	38, 490	26, 414	W3_250_030	74, 203	68, 954
W1_250_060	51, 255	40, 891	W3_250_060	99, 265	91, 559
W1_250_090	40, 606	30, 471	W3_250_090	76, 361	71, 208
W2_100_000	17, 203	6153	W4_100_000	110, 218	96, 620
W2_100_030	15, 535	2702	W4_100_030	92, 936	79, 951
W2_100_060	17, 131	4849	W4_100_060	94, 370	82, 276
W2_100_090	15, 069	2849	W4_100_090	77, 919	64, 560
W2_150_000	24, 227	13, 214	W4_150_000	155, 919	14, 1927
W2_150_030	21, 052	9595	W4_150_030	118, 893	105, 478
W2_150_060	24, 375	13, 561	W4_150_060	155, 998	141, 955
W2_150_090	21, 183	10, 110	W4_150_090	119, 539	106, 666
W2_200_000	33, 170	22, 498	W4_200_000	198, 530	186, 439
W2_200_030	26, 341	15, 274	W4_200_030	150, 094	136, 789
W2_200_060	31, 020	21, 247	W4_200_060	202, 348	188, 713
W2_200_090	26, 429	15, 433	W4_200_090	169, 074	154, 963
W2_250_000	38, 100	28, 017	W4_250_000	249, 690	236, 033
W2_250_030	33, 341	21, 956	W4_250_030	181, 508	168, 829
W2_250_060	41, 148	30, 372	W4_250_060	249, 863	235, 831
W2_250_090	34, 352	23, 382	W4_250_090	199, 738	185, 948

Appendix B: Results per instance (instance set Henn [18])

Table 14 Results per instance for the dataset introduced in [18] over the two considered objective functions

Instance	Objective function		Instance	Objective function	
	T (s)	Turnover (s)		T (s)	Turnover (s)
W5_abc1_40_29	21, 109	9848	W5_ran1_40_29	24, 689	12, 838
W5_abc1_40_30	17, 541	5133	W5_ran1_40_30	18, 023	8715
W5_abc1_40_31	17, 831	4788	W5_ran1_40_31	18, 815	6364
W5_abc1_40_32	16, 226	4488	W5_ran1_40_32	16, 393	5982
W5_abc1_60_37	31, 794	19, 729	W5_ran1_60_37	36, 681	25, 558
W5_abc1_60_38	23, 366	10, 978	W5_ran1_60_38	26, 418	14, 984
W5_abc1_60_39	21, 061	10, 985	W5_ran1_60_39	23, 674	12, 124
W5_abc1_60_40	18, 196	8538	W5_ran1_60_40	20, 759	9480
W5_abc1_80_61	40, 745	27, 372	W5_ran1_80_61	47, 254	34, 077
W5_abc1_80_62	32, 385	20, 458	W5_ran1_80_62	37, 688	24, 958
W5_abc1_80_63	26, 083	14, 713	W5_ran1_80_63	29, 133	17, 544
W5_abc1_80_64	24, 189	12, 530	W5_ran1_80_64	27, 565	15, 419
W5_abc1_100_69	45, 613	32, 404	W5_ran1_100_69	53, 976	41, 064
W5_abc1_100_70	33, 888	21, 849	W5_ran1_100_70	39, 367	27, 710
W5_abc1_100_71	30, 081	20, 595	W5_ran1_100_71	34, 027	23, 168
W5_abc1_100_72	28, 543	17, 174	W5_ran1_100_72	31, 446	19, 298
W5_abc2_40_9	21, 729	9279	W5_ran2_40_9	24, 660	14, 619
W5_abc2_40_10	18, 170	9377	W5_ran2_40_10	21, 073	8990
W5_abc2_40_11	15, 708	4829	W5_ran2_40_11	16, 051	5874
W5_abc2_40_12	15, 129	4177	W5_ran2_40_12	16, 623	5166
W5_abc2_60_17	29, 225	17, 812	W5_ran2_60_17	34, 631	22, 110
W5_abc2_60_18	25, 303	12, 760	W5_ran2_60_18	27, 725	17, 760
W5_abc2_60_19	20, 510	8120	W5_ran2_60_19	23, 565	13, 142
W5_abc2_60_20	18, 304	8467	W5_ran2_60_20	19, 750	8963
W5_abc2_80_45	38, 013	24, 949	W5_ran2_80_45	44, 605	31, 155
W5_abc2_80_46	28, 691	16, 423	W5_ran2_80_46	32, 758	21, 503
W5_abc2_80_47	25, 956	14, 817	W5_ran2_80_47	30, 911	18, 730
W5_abc2_80_48	23, 524	13, 018	W5_ran2_80_48	26, 983	16, 964
W5_abc2_100_53	49, 363	36, 092	W5_ran2_100_53	57, 097	43, 018
W5_abc2_100_54	35, 046	24, 269	W5_ran2_100_54	40, 093	27, 849
W5_abc2_100_55	34, 670	23, 567	W5_ran2_100_55	39, 278	28, 689
W5_abc2_100_56	28, 941	18, 828	W5_ran2_100_56	32, 025	20, 591

References

1. Albareda-Sambola, M., Alonso-Ayuso, A., Molina, E., De Blas, C.S.: Variable neighborhood search for order batching in a warehouse. Asia-Pac. J. Oper. Res. **26**(5), 655–683 (2009)
2. Chen, F., Wei, Y., Wang, H.: A heuristic based batching and assigning method for online customer orders. Flex. Serv. Manuf. J. **30**(4), 640–685 (2018)
3. Coyle, J.J., Bardi, E.J., Langley, C.J., et al.: The Management of Business Logistics, vol. 6. West Publishing Company Minneapolis, St Paul (1996)
4. De Koster, M.B.M., Van der Poort, E.S., Wolters, M.: Efficient orderbatching methods in warehouses. Int. J. Prod. Res. **37**(7), 1479–1504 (1999)
5. De Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: a literature review. Eur. J. Oper. Res. **182**, 481–501 (2007)
6. De Koster, R., Jan Roodbergen, K., van Voorden, R.: Reduction of walking time in the distribution center of de bijenkorf. In: New trends in distribution logistics, pp. 215–234. Springer, Berlin (1999)
7. Drury, J., Warehouse Operations Special Interest Group, Order Picking Working Party, Turnbull, B., Institute of Logistics (Great Britain): Towards More Efficient Order Picking. IMM monograph. Institute of Materials Management. ISBN 9781870214063 (1988). https://books.google.es/books?id=LbTKAAAACAAJ
8. Duarte, A., Pantrigo, J.J., Pardo, E.G., Mladenovic, N.: Multi-objective variable neighborhood search: an application to combinatorial optimization problems. J. Global Optim. **63**(3), 515–536 (2015)
9. Duarte, A., Pantrigo, J.J., Pardo, E.G., Sánchez-Oro, J.: Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J. Manag. Math. **27**(1), 55–73 (2013)
10. Feo, T.A., Resende, M.: Greedy randomized adaptive search procedures. J. Global Optim. **6**, 109–133 (1995)
11. Gademann, A.J.R.M., Van Den Berg, J.P., Van Der Hoff, H.H.: An order batching algorithm for wave picking in a parallel-aisle warehouse. IIE Trans. **33**(5), 385–398 (2001)
12. Gademann, N., Velde, S.: Order batching to minimize total travel time in a parallel-aisle warehouse. IIE Trans. **37**(1), 63–75 (2005)
13. Gibson, D.R., Sharp, G.P.: Order batching procedures. Eur. J. Oper. Res. **58**(1), 57–67 (1992)
14. Gu, J., Goetschalckx, M., McGinnis, L.F.: Research on warehouse design and performance evaluation: a comprehensive review. Eur. J. Oper. Res. **203**(3), 539–549 (2010)
15. Hall, R.W.: Distance approximations for routing manual pickers in a warehouse. IIE Trans. **25**(4), 76–87 (1993)
16. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)
17. Hansen, P., Mladenović, N., Moreno-Pérez, J.A.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
18. Henn, S.: Algorithms for on-line order batching in an order picking warehouse. Comput. Oper. Res. **39**(11), 2549–2563 (2012)
19. Henn, S., Koch, S., Doerner, K., Strauss, C., Wäscher, G.: Metaheuristics for the order batching problem in manual order picking systems. BuR Bus. Res. J. **3**(1), 82–105 (2010)
20. Henn, S., Schmid, V.: Metaheuristics for order batching and sequencing in manual order picking systems. Comput. Ind. Eng. **66**(2), 338–351 (2013)
21. Henn, S., Wäscher, G.: Tabu search heuristics for the order batching problem in manual order picking systems. Eur. J. Oper. Res. **222**(3), 484–494 (2012)
22. Ho, Y.-C., Tseng, Y.-Y.: A study on order-batching methods of order-picking in a distribution centre with two cross-aisles. Int. J. Prod. Res. **44**(17), 3391–3417 (2006)
23. Hsu, C.-M., Chen, K.-Y., Chen, M.-C.: Batching orders in warehouses by minimizing travel distance with genetic algorithms. Comput. Ind. **56**(2), 169–178 (2005)
24. Koch, S., Wäscher, G.: A grouping genetic algorithm for the Order Batching Problem in distribution warehouses. J. Bus. Econ. **86**(1–2), 131–153 (2016)
25. Menéndez, B., Bustillo, M., Pardo, E.G., Duarte, A.: General variable neighborhood search for the order batching and sequencing problem. Eur. J. Oper. Res. **263**(1), 82–93 (2017)
26. Menéndez, B., Pardo, E.G., Alonso-Ayuso, A., Molina, E., Duarte, A.: Variable neighborhood search strategies for the order batching problem. Comput. Oper. Res. **78**, 500–512 (2017)
27. Menéndez, B., Pardo, E.G., Duarte, A., Alonso-Ayuso, A., Molina, E.: General variable neighborhood search applied to the picking process in a warehouse. Electron. Notes Discrete Math. **47**, 77–84 (2015)
28. Menéndez, B., Pardo, E.G., Sánchez-Oro, J., Duarte, A.: Parallel variable neighborhood search for the min–max order batching problem. Int. Trans. Oper. Res, **24**(3), 635–662 (2017)
29. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

30. Öncan, T.: MILP formulations and an iterated local search algorithm with Tabu thresholding for the order batching problem. J. Oper. Res. **243**, 142–155 (2015)
31. Öncan, T., Cağırıcı, M.: MILP formulations for the order batching problem in low-level picker-to-part warehouse systems. IFAC Proc. Vol. **46**(9), 471–476 (2013)
32. Pan, C.-H., Liu, S.-Y.: A comparative study of order batching algorithms. Omega **23**(6), 691–700 (1995)
33. Pardo, E.G., Mladenović, N., Pantrigo, J.J., Duarte, A.: Variable formulation search for the cutwidth minimization problem. Appl. Soft Comput. **13**(5), 2242–2252 (2013)
34. Pérez-Rodríguez, R., Hernández-Aguirre, A., Jöns, S.: A continuous estimation of distribution algorithm for the online order-batching problem. Int. J. Adv. Manuf. Technol. **79**(1), 569–588 (2015)
35. Petersen, C.G.: An evaluation of order picking routeing policies. Int. J. Oper. Prod. Manag. **17**(11), 1098–1111 (1997)
36. Petersen, C.G.: Routeing and storage policy interaction in order picking operations. Decis. Sci. Inst. Proc. **31**(3), 1614–1616 (1995)
37. Ratliff, H.D., Rosenthal, A.S.: Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Oper. Res. **31**(3), 507–521 (1983)
38. Roodbergen, K.J., Koster, R.D.: Routing methods for warehouses with multiple cross aisles. Int. J. Prod. Res. **39**(9), 1865–1883 (2001)
39. Roodbergen, K.J., Petersen, C.G.: How to improve order picking efficiency with routing and storage policies. In: G.R. Forger et al. (eds.), Progress in Material Handling Practice. Material Handling Institute, Charlotte, North Carolina, pp. 107–124 (1999)
40. Rosenwein, M.B.: A comparison of heuristics for the problem of batching orders for warehouse selection. Int. J. Prod. Res. **34**(3), 657–664 (1996)
41. Rubrico, J.I.U., Higashi, T., Tamura, H., Ota, J.: Online rescheduling of multiple picking agents for warehouse management. Robot. Comput. Integr. Manuf. **27**(1), 62–71 (2011)
42. Scholz, A., Schubert, D., Wäscher, G.: Order picking with multiple pickers and due dates—simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. Eur. J. Oper. Res. **263**(2), 461–478 (2017)
43. Scholz, A., Wäscher, G.: Order Batching and Picker Routing in manual order picking systems: the benefits of integrated routing. CEJOR **25**(2), 491–520 (2017)
44. Tang, L.C., Chew, E.P.: Order picking systems: batching and storage assignment strategies. Comput. Ind. Eng. **33**(3), 817–820 (1997). Selected Papers from the Proceedings of 1996 ICC&IC
45. Van Nieuwenhuyse, I., Colpaert, J.: Order batching in multi-server pick-and-sort warehouses. DTEW-KBI **0731**, 1–29 (2007)
46. Zhang, J., Wang, X., Chan, F.T.S., Ruan, J.: On-line order batching and sequencing problem with multiple pickers: a hybrid rule-based algorithm. Appl. Math. Model. **45**, 271–284 (2017)
47. Zhang, J., Wang, X., Huang, K.: Integrated on-line scheduling of order batching and delivery under b2c e-commerce. Comput. Ind. Eng. **94**, 280–289 (2016)
48. Zhang, J., Wang, X., Huang, K.: On-line scheduling of order picking and delivery with multiple zones and limited vehicle capacity. Omega **79**, 104–115 (2018)
49. Žulj, I., Kramer, S., Schneider, M.: A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. Eur. J. Oper. Res. **264**(2), 653–664 (2018)

Affiliations

Sergio Gil-Borrás[1] · Eduardo G. Pardo[2] · Antonio Alonso-Ayuso[2] · Abraham Duarte[2]

Sergio Gil-Borrás
sergio.gil@upm.es

Antonio Alonso-Ayuso
antonio.alonso@urjc.es

Abraham Duarte
abraham.duarte@urjc.es

[1] Universidad Politécnica de Madrid, C/Alan Turing s/n (Ctra. de Valencia, Km. 7), 28031 Madrid, Spain

[2] Universidad Rey Juan Carlos, C/Tulipán s/n, 28933 Móstoles, Madrid, Spain