Computers & Operations Research 78 (2017) 500–512

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Variable Neighborhood Search strategies for the Order Batching Problem

Borja Menéndez [a], Eduardo G. Pardo [b], Antonio Alonso-Ayuso [a], Elisenda Molina [c], Abraham Duarte [a]

[a] Dept. Informática y Estadística, Universidad Rey Juan Carlos, Spain
[b] Dept. Sistemas Informáticos, Universidad Politécnica de Madrid, Spain
[c] Dept. Estadística, Universidad Carlos III de Madrid, Spain

ARTICLE INFO

Available online 17 February 2016

Keywords:
Order Batching Problem
Picking process
Variable Neighborhood Search

ABSTRACT

The Order Batching Problem is an optimization problem belonging to the operational management aspect of a warehouse. It consists of grouping the orders received in a warehouse (each order is composed by a list of items to be collected) in a set of batches in such a way that the time needed to collect all the orders is minimized. Each batch has to be collected by a single picker without exceeding a capacity limit. In this paper we propose several strategies based on the Variable Neighborhood Search methodology to tackle the problem. Our approach outperforms, in terms of quality and computing time, previous attempts in the state of the art. These results are confirmed by non-parametric statistical tests.

1. Introduction

The industry has found a very important issue in warehousing in the last few years, since it is a key part of the supply chain management. Warehousing primarily focuses on controlling the movement and storage of materials within a warehouse and processing the associated transactions, including shipping, receiving, and picking. Several policies have been proposed in the literature directed towards the improvement of warehouse operations. Some of them are suitable to be modeled as optimization problems [26]. On a strategic level, warehouse layout design is one of the most important decisions to be made (see Ghiani et al. [13]). It is a key component of further optimization tasks and it has a significant impact on order-picking and traveling distances in the warehouse [18]. From the tactical point of view, the main decision is the storage policy, that is, to decide where each product should be located. Traditional storage policies are random policy, demand-based storage, and class-based storage, among others. Finally, operational policies refer to order picking, i.e., the process of retrieving articles from their storage locations in response to a specific customer request. We refer the reader to Gu et al. [17] for a thorough review.

In this paper we deal with order picking operations, one of the most important processes in a warehouse. The cost of this process may constitute more than 50% of the total operating costs [3,7]. A warehouse receives every day several orders from its customers. Each order consists of a list of one or more items that have to be retrieved from the warehouse and shipped to a specific customer. Thus, items must be collected by a warehouse operator (picker). Several order picking strategies have been proposed in the literature. For instance, single-order picking, batching and sort-after-pick, single-order picking with zoning, or batching with zoning, among others. In particular, we focus on situations where several orders are put together into batches, satisfying a fixed capacity constraint. Then, each batch is assigned to a picker, who retrieves all the items included in those orders grouped into the corresponding batch in a single tour. This order-picking strategy is usually known as *order batching*.

According to De Koster et al. [5], it is possible to reduce the travel time up to 35%, simply by properly designing the routes of the order pickers. Moreover, if the two decisions concerning the order picking (i.e., batching and routing) are simultaneously considered, the associated benefits can be substantially increased. This fact provides the motivation for the study of the optimization problem known in the literature as the Order Batching Problem (OBP), where these decisions are combined. It basically consists in grouping a set of orders into batches (with a capacity limit) and then, for each batch, defining the route to collect the corresponding orders. Therefore, the objective function looks for the

E-mail addresses: borja.menendez@urjc.es (B. Menéndez), eduardo.pardo@upm.es (E.G. Pardo), antonio.alonso@urjc.es (A. Alonso-Ayuso), emolina@est-econ.uc3m.es (E. Molina), abraham.duarte@urjc.es (A. Duarte).

http://dx.doi.org/10.1016/j.cor.2016.01.020
0305-0548/

configuration of batches that minimizes the time needed to collect all of them. The OBP has been proved to be $\mathcal{NP}-$hard for general instances [10]. Nonetheless, it is solvable in polynomial time if each batch does not contain more than two orders [10]. Unfortunately, real warehouse instances do not usually fall into this category. Consequently, it has been heuristically approached in the last years [1,28].

The First-Come First-Served (FCFS) strategy might be the first heuristic approach implemented in warehouses to assign orders to batches. In particular, orders are sorted according to their arrival time to the warehouse. Following this sorting, they are added to the first batch while its capacity limit is not exceeded. The order that overtakes the capacity of the first batch is then assigned to the second one, which means that the first batch is ready to be collected. This logic is maintained until all orders are assigned to a batch. This strategy has been widely used due to its simplicity. In De Koster et al. [4] it is possible to find a survey of those methods where the authors proposed a classification. Among others, they highlighted "seed methods" [14,24,31] and "saving methods" [37]. We refer the reader to De Koster et al. [4] for a detailed description.

The first metaheuristic algorithm applied to the problem was described in Hsu et al. [25]. The authors proposed a Genetic Algorithm in order to deal with the Order Batching Problem for different batch structures and warehouse layouts.

Later, Albareda-Sambola et al. [1] proposed an algorithm based on the Variable Neighborhood Search methodology. Specifically, they proposed several neighborhoods in a Variable Neighborhood Descent (VND) scheme. In order to provide an initial solution to their method, they constructed a naïve solution where each order was assigned to a different batch and then the number of batches was reduced by joining batches. Once the batches were conformed, they applied VND, using three well-known routing strategies (S-Shape [16], Largest Gap [4], and Combined [35]) to evaluate the solution.

Henn et al. [22] proposed an Iterated Local Search and a Rank-Based Ant System algorithms to tackle the Order Batching Problem. Next, Henn and Wäscher [21] improved previous results by introducing two additional algorithms. The first one was a classical Tabu Search and the second one, an Attribute-Based Hill Climber. These authors also considered the Largest Gap and S-Shape routing methods to evaluate the solutions produced by their algorithms.

As far as we know, the most recent approach to tackle the problem was carried out by Öncan [30], where the author proposed an Iterated Local Search algorithm with a Tabu Thresholding method as the local search procedure. The initial solution for the algorithm was constructed with the savings algorithm by De Koster et al. [4]. The author considered S-Shape [16], Return [19], and Midpoint [19] routing strategies. He also proposed several Mixed Integer Linear Programming formulations for the OBP with these three routing strategies.

As it was aforementioned, the OBP is usually divided into two main subproblems: (i) batching and (ii) routing. In this paper we propose several strategies based on Variable Neighborhood Search (VNS) methodology to tackle the batching and, additionally, a new algorithm to tackle the routing. In Section 2 we visit the problem description in detail. In Section 3 we describe our procedure based on a Multi-Start VNS, whose main search strategies are a randomized procedure oriented to construct high quality but diverse solutions (Section 3.1), a Basic VNS to configure the corresponding batches (Section 3.2), and a post-optimization strategy based on a General VNS whose objective is to reduce the number of batches (Section 3.3). The routing strategy is described in Section 4. Finally, we experimentally compare our proposal with the current state of the art in Section 5 and present the most relevant findings in Section 6.

2. Problem description

The layout design of the warehouse is a key component of further optimization tasks and it has a significant influence on order-picking and traveling distances in the warehouse [18]. According to De Koster et al. [6] and Heragu [23], the design of the layout of a warehouse depends on several factors, such as length, width, and number of picking aisles; number of cross aisles and its shape; number of racks; and the position of the input and output gates in the warehouse. This paper is concerned with warehouse management at an operational level. Thus, decisions regarding the layout of the warehouse and the location of the different items are considered fixed throughout the process. We then focus on a typical warehouse layout with rectangular shape and two crossing aisles, which is suitable for pallet manipulation. In general, this problem can be parameterized in terms of the number of parallel aisles, picking positions, levels of each picking position, and batch capacity.

In Fig. 1 we show an example of this kind of warehouse layout, with rectangular shape and two main cross (horizontal) aisles, one at the front and another at the back of the warehouse. In this example the main parameters are set to 5 parallel (vertical) aisles with equal length and 9 picking positions per aisle at each side. For the sake of simplicity we consider that each picking position has only one level. In this figure it is also depicted a depot (located in the front aisle), which represents the input–output gate.

In this paper we focus on the order batching strategy where the main objective consists in minimizing the time needed to collect all the orders. This problem is traditionally divided into two different sub-problems: on one hand, grouping the orders into feasible batches (i.e., the sum of the weight of the items cannot exceed a capacity limit) and, on the other hand, finding the itinerary that a picker must follow to collect the items.

From a theoretical point of view, the OBP can be formulated as follows [10]. Let $\mathcal{O} = \{1, 2, \ldots, n\}$ be the set of customer orders, C the maximum allowed capacity for each batch, and c_j the capacity (weight, size, etc.) associated to order $j \in \mathcal{O}$. Let a_{ij} be a binary variable that takes on the value 1 if order j is included in batch i; otherwise this variable is set to 0. We denote with $\mathcal{B}$ as the set of all feasible batches, i.e., those where the capacity constraint is not

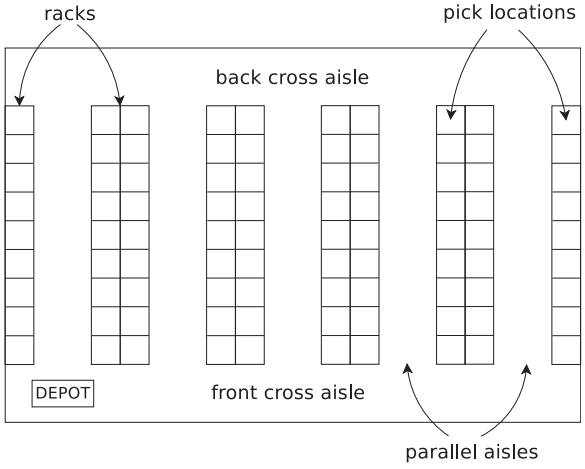

Fig. 1. Warehouse layout with rectangular shape and parallel aisles.

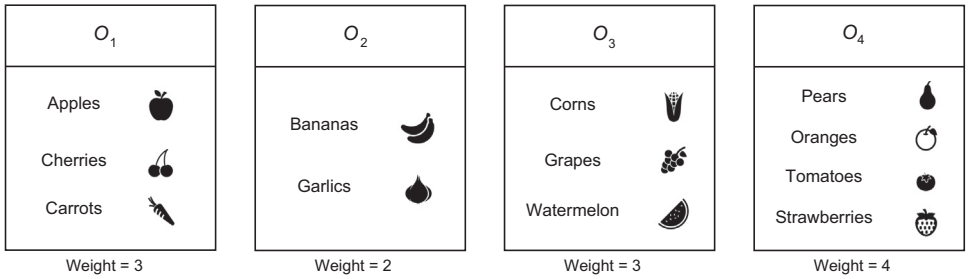

Fig. 2. Example of a set of orders.

violated. In mathematical terms:

$$\sum_{j \in \mathcal{O}} c_j \cdot a_{ij} \leq C \quad \forall i \in \mathcal{B}$$

Let d_i be the length of the picking route, which strongly depends on the routing strategy used in the resolution of the problem (see Section 4). If we define a binary decision variable x_i (with $i \in \mathcal{B}$), that establishes whether batch i is selected ($x_i=1$) or not ($x_i=0$), then the OBP problem can be formulated as follows:

$$\begin{aligned} &\min \sum_{i \in \mathcal{B}} d_i \cdot x_i \\ &\text{subject to} \\ &\sum_{i \in \mathcal{B}} a_{ij} \cdot x_i = 1 \quad \forall j \in \mathcal{O} \\ &x_i \in \{0,1\} \qquad \forall i \in \mathcal{B} \end{aligned}$$

where the constraints ensure that a set of batches is selected in a way that each order is included in exactly one of the chosen batches. It is worth mentioning that the number of feasible batches (i.e., the amount of binary variables) grows exponentially with the number of orders.

In Fig. 2 we show an example of a set of 4 orders,[1] $\{O_1, O_2, O_3, O_4\}$, received in a warehouse of vegetables, where each order contains, respectively, 3 items (apples, cherries, and carrots), 2 items (bananas and garlics), 3 items (corns, grapes, and watermelons), and 4 items (pears, oranges, tomatoes, and strawberries). For the sake of simplicity, we assume that all the items have the same weight and it is equal to 1 (i.e., the weight of each order is equal to the number of items in it). Without loss of generality, we indicate the weight of an order with the cardinality set notation. For instance $|O_1|=3$, $|O_2|=2$, $|O_3|=3$, and $|O_4|=4$ in the example depicted in Fig. 2.

Let us also consider that all batches have the same maximum capacity C which is equal to 6. Each picker then can collect on a single trip any number of orders (grouped on a single batch) whose summed weight is lower than or equal to 6. In Fig. 3a we present a possible configuration of batches. Specifically, the solution consists of 3 batches[2] with $B_1=\{O_1, O_2\}$, $B_2=\{O_3\}$ and $B_3=\{O_4\}$. Although this configuration is feasible (i.e., the summed weight of each batch is lower than 6), the batches have some space wasted. The free space of a batch B_i is denoted as $\overline{B_i}$. Specifically, B_1 has assigned orders whose summed weight is 5, leaving some free space ($\overline{B_1}=1$ unit). However, that free space cannot be occupied by another order since it would exceed the maximum capacity allowed. A similar situation occurs in the case of batches B_2 and B_3. In Fig. 3b we show a possible itinerary followed by the picker and composed by 3 closed routes. Each route starts from the depot, collects a set of items (from the orders packed in a single batch), and returns to the depot. For example, the route represented by the continuous line drives the picker to collect items belonging to orders O_1 and O_2 exclusively (batch B_1). Notice that the problem constraints forbid to collect an item of an order in a different batch (O_3 or O_4) but does not specify a sequence to collect the items of O_1 and O_2.

We assume that each picking position has a length/width equal to 1. Then, the picker traverses 9 units of length when he/she goes from the front to the back aisle. Similarly, going from one parallel aisle to the next one has an associated distance of 2 units of length. Therefore, ignoring the separation between shelves, the length of the route followed by the picker to collect B_1 (continuous line) is 46 units. Similarly, the length of the routes depicted to collect B_2 (dotted line) and B_3 (dashed line) are 32 and 38 units respectively. Thus, the objective function value associated to this solution is 116 units.

In Fig. 4a we depict a different batch configuration over the orders presented in Fig. 2. In this case, there are only two batches ($\{B'_1, B'_2\}$) where $B'_1=\{O_2, O_4\}$ and $B'_2=\{O_1, O_3\}$. This configuration is also feasible since the maximum capacity is not exceeded. In Fig. 4b we show two possible routes to collect the items within the two batches. In this case, the length of this two routes is 52 (continuous line) and 42 (dotted line) units, respectively. Therefore the value of the objective function for this solution is 94. Considering that we are facing with a minimization optimization problem, the solution presented in Fig. 4 is better than the one presented in Fig. 3.

3. Two-stage Variable Neighborhood Search

Variable Neighborhood Search (VNS) [29] is a metaheuristic which exploits the idea of neighborhood change in a systematic way, both descending to a local optimum or escaping from the basin of that local optimum. Unlike many other metaheuristics, the basic schemes of VNS and its extensions are simple and require few and, sometimes, no parameters. The fact of providing high quality solutions in simpler ways than other methods makes the methodology able to lead towards a more efficient and sophisticated new implementations.

In recent years, a large variety of VNS strategies have been proposed. We would like to highlight: Variable Neighborhood Descent (VND) explores the neighborhood in a deterministic way by using several local search procedures (in this case there is not shaking procedure); Reduced VNS (RVNS) explores solutions at random in each neighborhood by perturbing the solutions with the shaking procedure (in this case there is no local search procedure); Basic VNS (BVNS) combines deterministic and random exploration of the neighborhoods (by considering both, shaking and local search procedures). Other examples of more complex variants are Skewed VNS (SVNS), General VNS (GVNS), Variable Neighborhood Decomposition Search (VNSD), and Reactive VNS. Recently, the methodology has been extended to tackle

[1] We denote an order with O_j instead of j to clarify the meaning of the corresponding variable.

[2] We denote a batch with B_i instead of i to clarify the meaning of the corresponding variable.

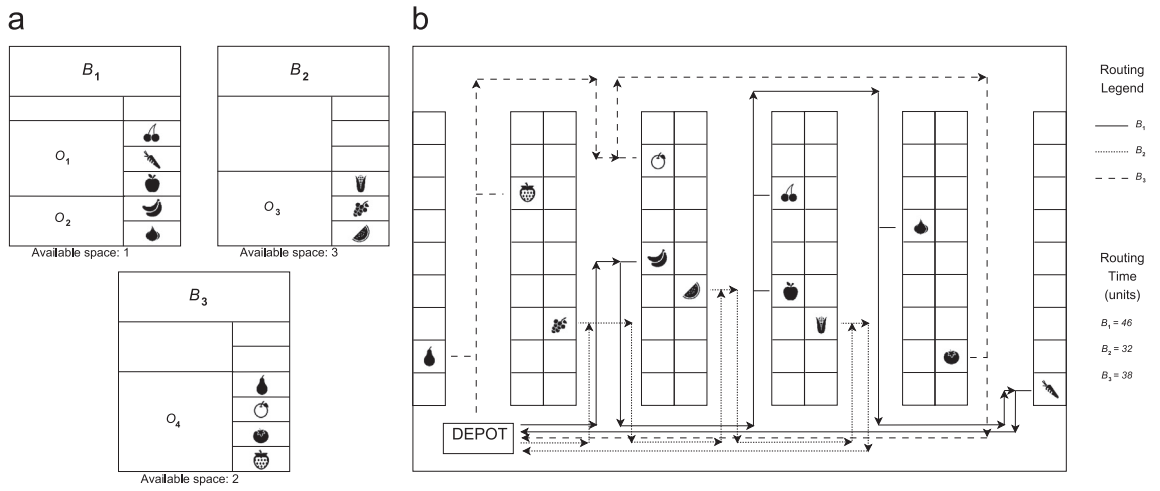

Fig. 3. Example of a set of batches from the orders in Fig. 2.

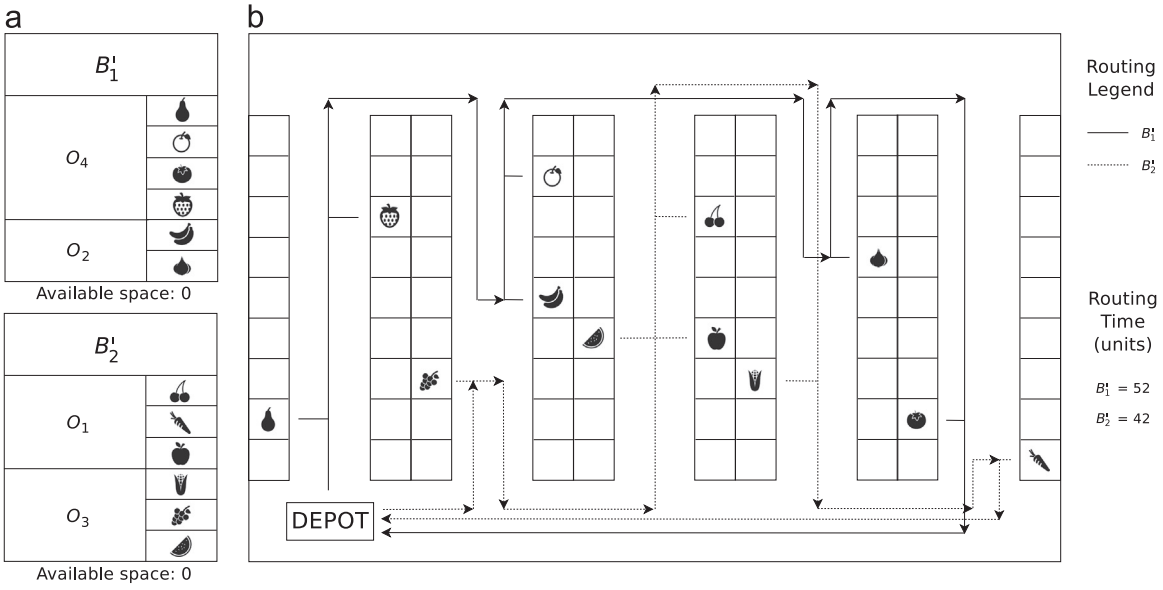

Fig. 4. Another different example of a set of batches from the orders in Fig. 2.

applications that require modern approaches, such as multi-objective optimization problems [9], parallel implementations [8], or change of the problem formulation [32]. See Hansen et al. [20] for a recent thorough review. We refer the reader to the previous recent and successful applications of different variants of VNS to hard optimization problems.

As it is well documented in the literature, a multi-start strategy can overcome situations where the problem presents a solution space with a steep landscape [27] since, in each iteration, the algorithm starts the search from a different but high-quality solution.

Algorithm 1. MS-VNS($cons$, $iters$, $k_{\max}$, α).

1: $S_{best} \leftarrow \varnothing$
2: **for** $i \in \{1 \ldots cons\}$ **do**
3: $S \leftarrow InitSolution(\alpha)$
4: $S' \leftarrow BVNS(S, k_{\max}, iters)$
5: $S'' \leftarrow GVNS(S')$
6: **if** $f(S'') < f(S_{best})$ **then**
7: $S_{best} \leftarrow S''$
8: **end if**
9: **end for**
10: **return** S_{best}

In Algorithm 1 we show the pseudo-code of the proposed Multi-Start VNS (MS-VNS) for the Order Batching Problem. Each iteration starts by generating a different initial solution S (step 3) with the method described in Section 3.1. Then, that solution is improved with a Basic VNS (step 4) described in Section 3.2. Next, a post-optimization strategy based on General VNS is applied to this solution to improve it (step 5). The MS-VNS method tests whether the incumbent solution improves upon the current best solution (step 6) or not. After performing the established number of iterations, this method returns the best found solution (step 10).

3.1. Initial solutions

The procedure to construct high-quality but diverse initial solutions starts by considering a naïve solution, where each order is placed in a different batch. Then, this solution is improved with a local search procedure. Obviously, if we improve this naïve solution with a deterministic local search, based on either the first or the best improvement strategy, we will always obtain the same local optimum. In order to produce different solutions, we improve the naïve solution by considering an Elite Candidate List (ECL)

within the local search method. This strategy was originally introduced in the context of Tabu Search [15]. It first builds a Master List (ML) by examining all feasible moves that improve the current solution (i.e., removing an order from its current batch and including it in a different one with enough free space). Then, the ECL is constructed by the solutions obtained with those moves in ML that improve a given value of threshold *th*. The original proposal of Glover and Laguna [15] performed the best available move in the ECL (even deteriorating the quality of the current solution). In our design, the quality of the solution is improved by randomly selecting an improving move whose associated value ranges from *th* to the best one.

Let S be a solution to the OBP whose objective function value is denoted by $f(S)$ (computed as the sum of routes needed to collect all the orders). ML contains all the feasible solutions S' obtained by applying an improving move (considering only those insert moves defined in Section 3.2) to S whose objective function value, $f(S')$, is better than $f(S)$.

The ECL is then constructed as a subset of the best solutions in ML. More formally:

$$ECL = \{S \in ML : f(S) < th\}$$

where *th* is the threshold parameter computed as:

$$th = I_{\min} + \alpha \cdot (I_{\max} - I_{\min})$$

and $I_{\max}$ and $I_{\min}$ are the values of the objective function associated to the improving moves that produces the maximum and minimum decrement in the objective function, respectively. The parameter $\alpha \in [0, 1]$ controls the size of the ECL. In other words, if $\alpha = 0$ then ECL only contains the best solution. On the other hand, if $\alpha = 1$, $ECL = ML$.

In Algorithm 2 we show the pseudo-code of this procedure. The initial naïve solution, where each order is placed in a different batch, is constructed in Step 1. Then, in Step 3, we construct the Master List (ML) with those moves which produce better feasible solutions (notice that a move is considered feasible when it drives to a new solution where the capacity constraint is not violated). Steps 4, 5, and 6 compute the values $I_{\max}$, $I_{\min}$, and *th*, respectively. Elite Candidate List (ECL) is built by selecting a subset of the best elements in ML (step 7). A solution is randomly selected from the ECL (step 8). This method performs iterations until no further improvements are available.

Algorithm 2. InitSolution(α).

1: $S \leftarrow NaiveSolution()$
2: **while** $ThereAreImprovingMoves(S)$ **do**
3: $ML \leftarrow ConstructMasterList(S)$
4: $I_{\max} \leftarrow \max_{S \in ML} f(S)$
5: $I_{\min} \leftarrow \min_{S \in ML} f(S)$
6: $th = I_{\min} + \alpha \cdot (I_{\max} - I_{\min})$
7: $ECL \leftarrow ConstructEliteCandidateList(ML, th)$
8: $S \leftarrow GetAtRandom(ECL)$
9: **end while**
10: **return** S

3.2. Basic Variable Neighborhood Search for batching configuration

In this paper we propose the use of the Basic VNS (BVNS) schema as the batching strategy. The pseudo-code of this algorithm is depicted in Algorithm 3. It receives a solution, S, the largest neighborhood to be visited, $k_{\max}$, and the maximum allowed time, $t_{\max}$, as input parameters. In our case, we redefine $t_{\max}$ as the maximum number of iterations. BVNS perturbs the incumbent solution with the shake method to generate a new solution in the current neighborhood (step 4). Then, the solution is improved with a local search method (step 5). Finally, if there has been an improvement, the incumbent solution is updated and the search restarts from the first neighborhood. Otherwise, it continues with the next one (this fact is determined in *NeighborhoodChange* procedure in step 6). The method ends when the maximum allowed time is reached. Then, it returns the best solution found during the search process.

Algorithm 3. BVNS (S, $k_{\max}$, $t_{\max}$).

1: **repeat**
2: $k \leftarrow 1$
3: **while** $k \leq k_{\max}$ **do**
4: $S' \leftarrow Shake(S, k)$
5: $S'' \leftarrow Improve(S')$
6: $NeighborhoodChange(S, S'', k)$
7: **end while**
8: $t \leftarrow CPUTime()$
9: **until** $t > t_{max}$
10: **return** S

The general schema presented in Algorithm 3 is then customized for the OBP. Specifically, given a solution composed by q batches (i.e., $S = \{B_1, B_2, \ldots, B_q\}$), the insert move, denoted by $Insert(O_i, B_j, B_k)$, produces a new solution S' where the order O_i is removed from its current batch (B_j) and included in B_k. As it was previously mentioned only feasible moves are considered. Then, O_i is admitted in B_k if and only if B_k has enough free space (i.e., $|O_i| \leq \overline{B_k}$). The set of solutions reachable from S by applying an insert move is then denoted by $N_I(S)$.

We introduce an additional move, denoted by $Swap(O_i, B_j, O_l, B_k)$, that produces a new solution S' where the order O_i is removed from its current bath (B_j) and included in B_k. Simultaneously, the order O_l is removed from its current batch (B_k) and included in B_j. Again, only feasible moves are considered. Then, an order O_i is admitted in a batch B_k (respectively, O_l in B_j) if and only if each batch has enough free space (i.e., $|O_i| \leq \overline{B_k} + |O_l|$ and $|O_l| \leq \overline{B_j} + |O_i|$). The set of solutions reachable from S by applying a swap move is then denote by $N_S(S)$.

The shake procedure (step 4 of Algorithm 3) makes a random perturbation to the solution within the k-th neighborhood. This perturbation consists of a combination of k moves selected at random among the feasible moves that can be performed in the neighborhood. In this case, we consider a combined neighborhood defined by insert and swap moves. The shake procedure is an efficient strategy to get out from a basin of attraction. Given a solution S and the k-th neighborhood $N_k(S)$, it usually generates, at random, a solution $S' \in N_k(S)$. In the OBP, we define $N_k(S)$ as the set of solutions that can be reached by applying k consecutive times either swap or insert moves. In order to favor the diversification of the search process, the corresponding move is chosen at random. Notice that our method only considers feasible moves, discarding those random selections that drive to an unfeasible solution.

The *Improve* method explores a neighborhood in search for either the first or the best improving move. At the end of the process it is guaranteed that the best solution found is a local optimum with respect to the considered neighborhood. In this paper, we propose a local search strategy based on a nested exploration of $N_I(S)$ and $N_S(S)$. Therefore, in each iteration of the local search the best possible move is performed in the composed neighborhood (i.e., selecting either the best insert or swap available move). It performs iterations until no further improvement is found in the composed neighborhood, returning the best solution found, which is a local optimum with respect to both neighborhoods.

3.3. Post-optimization strategy based on an alternative objective function

As it is well documented in the related bibliography, there is a strong correlation between the number of formed batches and the total travel time [4]. We have experimentally observed that, in general, the smaller the number of batches, the shorter the time needed to collect them. Then, once the method gets stuck in a local optimum, we can determine whether the number of batches can be reduced or not. Given a solution to the OBP with q batches (i.e., $S=\{B_1, B_2, \ldots, B_q\}$), the total wasted space, W, is computed as:

$$W = \sum_{i=1}^{q} \overline{B_i}$$

If $W < C$ (being C the maximum capacity of a batch), it is not possible to reduce the number of batches since there is no enough free space to redistribute the orders in such a way that a batch gets empty. Otherwise, there is an opportunity to reduce the number of batches. In fact, this reduction can be estimated as:

$$\lambda = \left\lfloor \frac{W}{C} \right\rfloor$$

being λ the number of batches that can be eventually reduced. Therefore, if $\lambda=0$, no further reduction in the number of batches is possible.

The local optimum returned by the method proposed in Section 3.2 might have batches with enough wasted space to eliminate, at least, one of them. With the aim of performing a reduction in the number of batches, we propose the use of an alternative formulation to escape from basins of attraction. As it is documented in Pardo et al. [32], the use of alternative formulations (i.e., by using different objective functions) within a VNS schema might be helpful when the search process gets stuck. In particular, this alternative objective function is the number of full batches (i.e., batches without wasted space). Therefore, those moves able to increase the number of full batches are considered as improving moves, even if the original objective function is deteriorated.

The reduction of the number of batches has some similarities with the well-known bin packing problem [12]. This $\mathcal{NP}-$hard problem [11] can be efficiently solved by using advanced metaheuristics [2]. However, both problems are not completely equivalent since, in the OBP context, we look for eliminating only the batch with the lowest number of orders by redistributing them among the other batches and maintaining, as much as possible, the orders already included in the other batches. On the contrary, the bin packing problem adapted to the context of OBP would maximize the number of full batches, ignoring the orders that each batch had. In addition, this is an experimental fact not a theoretical one. Indeed, we have observed that in some instances of the OBP it is better to have a larger number of batches (with a smaller percentage of occupation), than a smaller number of full batches [4]. Therefore, we propose a problem-specific procedure to minimize the alternative objective function instead of using previous methods for the bin packing problem.

We maximize the alternative objective function with a General Variable Neighborhood Search (GVNS). The pseudo-code of this procedure is equal to the BVNS presented in Algorithm 3 but changing the local search with a VND.

We have experimentally observed that it is difficult to *navigate* through the search space of the OBP by only performing smooth moves. Therefore, in some situations, insert and swap moves together with perturbations are unlikely able to escape from a depth basin of attraction. We propose two additional moves, denominated $Swap_2$ and $Swap_3$, to perform more *aggressive* changes in the current solution. In order to reduce the size of the search space, we only focus on the batch with the largest wasted space, $B^\star$. The four moves are intended to redistribute the orders in $B^\star$ among the rest of the batches in such a way that we increase the number of full batches.

The first neighborhood, $N_1(S)$, is based on feasible insert moves that try to reduce the number of orders in $B^\star$ by eliminating the wasted space of a different batch. More precisely, this neighborhood contains solutions generated by removing one order $O_i \in B^\star$ and inserting it in a batch B_k such that the size of O_i is equal to $\overline{B_k}$. In mathematical terms:

$$N_1(S) = \{S' \leftarrow Insert(O_i, B_j, B_k) : \ |O_i| = \overline{B_k}\}$$

The second neighborhood, $N_2(S)$, is based on feasible swap moves that eliminate the wasted space of a batch. In particular, swapping $O_i \in B_j$ with $O_l \in B_k$ is defined as:

$$N_2(S) = \{S' \leftarrow Swap(O_i, B_j, O_l, B_k) : \ |O_i| = \overline{B_k} + |O_l|\}$$

The third neighborhood is based on a more complex move, denoted as $Swap_2$, that considers two orders $\{O_i, O_{i'}\} \in B^\star$ and swaps them with a different one $O_l \in B_k$. This move is only performed if O_i and $O_{i'}$ completely cover the wasted space of B_k (i.e., $\overline{B_k}$) when removing O_l. Notice that the previous fact guarantees that there is enough space in $B^\star$ to allocate O_l. The neighborhood $N_3(S)$ is then defined as:

$$N_3(S) = \{S' \leftarrow Swap_2(O_i, O_{i'}, B_j, O_l, B_k) : \ |O_i| + |O_{i'}| = \overline{B_k} + |O_l|\}$$

The fourth neighborhood, $N_4(S)$, is based on a move denominated $Swap_3$. Specifically, given the orders $O_i \in B^\star, O_m \in B_n$, and $O_l \in B_k$, this move removes O_i and O_m from their corresponding batches, inserting them in B_k. Simultaneously, the order $O_l \in B_k$ is moved to $B^\star$. The neighborhood $N_4(S)$ is defined as:

$$N_4(S) = \{S' = Swap_3(O_i, B^\star, O_m, B_n, O_l, B_k) : \\ |O_i| + |O_m| = \overline{B_k} + |O_l| \wedge |O_l| < \overline{B^\star} + |O_i|\}$$

These neighborhoods are explored with a Variable Neighborhood Descent procedure. As it is customary, neighborhoods are explored from the smallest and fastest to evaluate (N_1) to the largest (N_4). Specifically, VND starts by obtaining a local optimum with respect to the first neighborhood. Then, instead of abandoning the search (as a local search procedure), it resorts to the following neighborhood searching for an improvement. If so, the search starts again by considering N_1. Otherwise, the VND explores the next neighborhood. The method ends when it does not find an improving solution in N_4, returning the best solution found.

Once we obtain a local optimum with respect to the alternative objective function, the aforementioned GVNS perturbs the corresponding local optimum with a shake procedure. We propose a more elaborated shake procedure where, instead of augmenting the size of the perturbation with the same move operator, we use different move operators. In particular, k_{max} is equal to 4, and the sequence of feasible perturbations is: *Insert* ($k=1$), *Swap* ($k=2$), $Swap_2$ ($k=3$), and $Swap_3$ ($k=4$). The neighborhood change operator changes from k to $k+1$ if and only if there are no feasible moves in k. Once a perturbation is actually performed, k is set to 1 to test whether there are new available feasible moves in $k=1$. The GVNS method ends when it overtakes the maximum allowed time or, alternately, when there are no feasible moves for the secondary objective function.

4. Routing algorithm for the Order Batching Problem

As it was aforementioned, finding a solution to the Order Batching Problem implies to solve two different sub-problems: a batching problem, which consists of grouping the orders into

batches; and a routing problem, which determines the route followed by the picker to collect the items (belonging to one or more orders and packed in a single batch). Therefore, the routing actually determines the value (quality) of a solution.

The routing problem in warehouses with parallel aisles has been intensively studied in the last years. In fact, Ratliff and Rosenthal [34] proposed an optimal procedure based on dynamic programming for rectangular warehouses without intermediate cross aisles. This algorithm requires a linear number of computations with respect to the number of parallel aisles. This procedure can also be used in more complex warehouses (with intermediate cross aisles) but its efficiency decreases. Unfortunately, this algorithm is barely used in practice since the proposed itineraries could imply illogical routes for human pickers, as stated in Petersen [33] and Hall [19]. In other words, these routes might be hard to follow for human pickers, since they could imply to move forward and backward in the same crossing aisle.

Heuristic routing procedures are usually preferred in real situations since the identification of non-confusing routes for (human) pickers is even more important than either, the optimality of the corresponding routes or the efficiency of the algorithm [33]. See Hall [19], Petersen [33] and Roodbergen and Petersen [36] for a thorough review. Perhaps, the three most used routing strategies in the related literature are S-Shape [16], Largest Gap [4], and Combined strategies [35].

The S-Shape or Traversal strategy (Fig. 5a) leads to a route in which the visited aisles (i.e., those that contain at least an item that must be picked) are totally traversed. The remaining aisles are skipped. Thus, aisles are visited in the shape of an S. In this route, the picker starts from the depot and then enters to the left-most (or, alternatively, the right-most) aisle containing items (whichever is the closest). The picker then enters an aisle from one end and leaves the aisle from the other end. Notice that if the items are located in an odd number of aisles, the last one is only traversed from the front cross aisle to the farthest item. Once the picker collects the last item, he/she returns to the depot.

In the Largest Gap strategy (Fig. 5b) the picker enters and exits to each visited aisle from the same end with the exception of the first and last aisle, which are completely traversed from front to back and back to front, respectively. As in the previous strategy, aisles that do not contain items to retrieve are skipped. This strategy is based on the "gap concept", which is defined in this context as: the distance between any two adjacent pick locations with items that must be retrieved; the distance between the first pick location and the front aisle; or the distance between the last pick location and the back aisle. Therefore, the Largest Gap is the maximum of these gaps and represents the portion of the aisle that the picker does not traverse. If the Largest Gap is between two adjacent pick locations, the picker performs a return route from both ends of the aisle. Otherwise, a return route from either the front or back aisles is used.

The third routing method is usually denominated as Combined strategy [35], which can be considered as a combination of S-Shape and Largest Gap methods. The original heuristic creates order picking routes that visit every aisle that contains items exactly once. The aisles are visited sequentially from left to right and then the picker returns to the depot. It performs a small dynamic programming algorithm, consisting of choosing the best option between fully traversing the aisle (S-Shape) or enter and exit from the same side (Largest Gap).

We propose an alternative routing algorithm based on the Combined strategy. In particular, this new procedure starts by considering the solution found by the Largest Gap method. In Fig. 6 we show an example of this initial solution.

The procedure then determines whether it is convenient to traverse an isolated aisle with the Largest Gap or the S-Shape strategy. Considering again the example depicted in Fig. 6, we show in Fig. 7 the distances that the picker would cover if the aisles were traversed with either strategy. Since the warehouse layout is rectangular, traversing an aisle with the S-Shape strategy has always the same associated traveling distance and it is equal to 15. On the other hand, the length of the route when traversing the aisle with the Largest Gap strongly depends on the item distribution. For example, it is better to traverse the second aisle (a_2) with the Largest Gap strategy (14 vs. 15), while the third aisle (a_3) is better to be traversed with the S-Shape strategy (18 vs. 15). In general, our routing procedure labels as LG all those aisles better to be traversed with the Largest Gap strategy (i.e., a_2, a_4, a_7, and a_{10}). The remaining aisles are labeled as SS (i.e., a_3, a_5, a_6, a_8, and a_9). Notice that the first and last aisles are mandatory labeled as SS (a_1 and a_{11}).

In a first simple approach, the routing algorithm should route again the aisles labeled as SS to be traversed by the S-Shape strategy. However, this option is not always available since it could produce even infeasible solutions. For example, the aisle a_3 depicted in Fig. 6 is recommended to be traversed with the S-Shape strategy (see Fig. 7). Although this would be the ideal situation, this change is not possible since the next aisle (a_4) requires to collect items starting from the back aisle, but the picker would be located at the front aisle. Thus, it is not possible to collect the items of aisle a_3 with the S-Shape strategy.

In general, the aisles that might produce this problem are those that require to collect items starting from the back aisle (aisles a_2, a_4, and a_7 in Fig. 6). We denote them as "critical aisles". Then, if a solution has not "critical aisles", there would be no problem in routing again SS aisles with the S-Shape strategy. Additionally, if there are an even number of aisles labeled as SS between any two pairs of "critical aisles", the SS aisles can also be routed again with the S-Shape strategy without generating any infeasible solution.

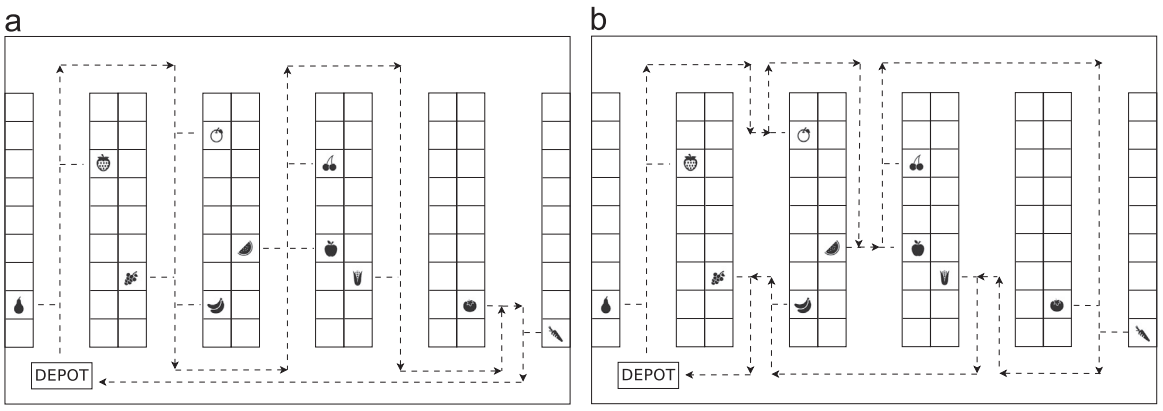

Fig. 5. S-Shape (a) and Largest Gap (b) routing strategies.

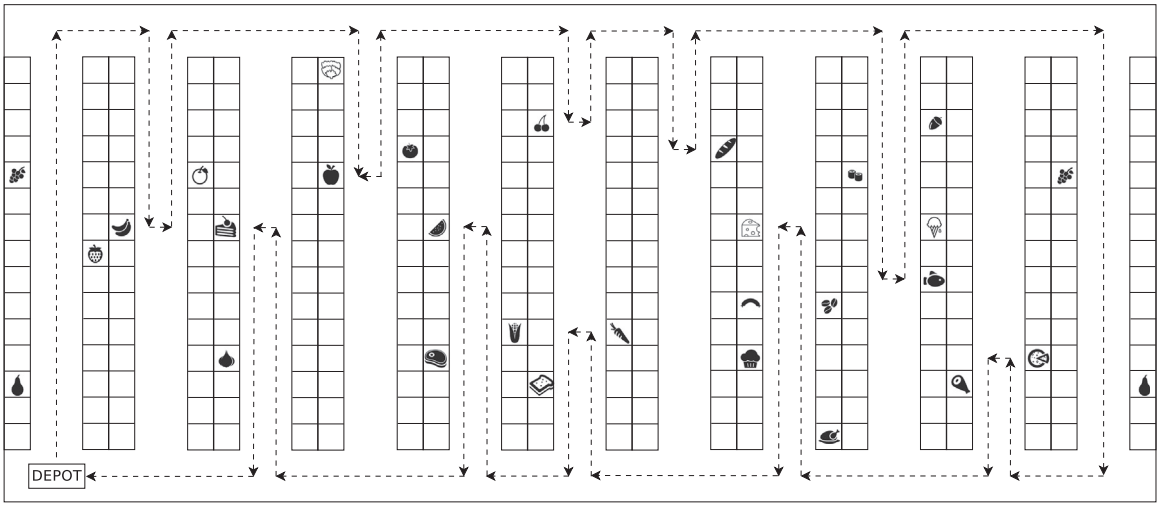

Fig. 6. First step in the proposed routing algorithm.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
LG	-	14	18	10	18	16	8	18	18	8	-
SS	15	15	15	15	15	15	15	15	15	15	15

Fig. 7. Distances of traversing an aisle with Largest Gap or S-Shape strategies.

The logic behind this strategy resides on the fact that traversing an even number of aisles with the S-Shape method does not change the crossing aisle (front or back) in which the picker is located. For example, In Fig. 8a we show an example with two critical aisles (a_4 and a_7 in Fig. 6), where there are 2 aisles labeled as SS (a_5 and a_6). In Fig. 8b we show how aisles a_5 and a_6 can be routed again with the S-Shape strategy without generating an infeasible solution.

On the other hand, if the number of SS aisles between two "critical aisles" is odd, it is not possible to route them again with the S-Shape strategy (the picker would be located in the opposite crossing aisle). In this situation, the routing method decides whether it is better to relabel an aisle from LG to SS or, alternatively, to relabel an aisle from SS to LG (in order to obtain an even number of SS aisles between two critical aisles). We evaluate both alternatives and select the one that minimizes the total travel time. For example, the solution shown in Fig. 6 has 1 SS aisle between a_2 and a_4. Therefore, it is not possible to route them again directly. In Fig. 9a we show the corresponding solution when a LG aisle is transformed into SS and in Fig. 9b we depict the solution when an SS aisle is transformed into LG. Considering only the involved aisles (a_2, a_3, and a_4), the partial solution depicted in Fig. 9a is better since the route has a length of 40 while the route depicted in Fig. 9b has a length of 42, not taking into account the time needed to go from an aisle to the next.

For the sake of completeness, we depict in Fig. 10 the final routing obtained with our proposed method. The total length of this route is 146, which compares favorably with the distance obtained by classical methods. Specifically, S-Shape, Largest Gap, and classical Combined strategies obtain, respectively, 165, 158, and 160 (these values are obtained not taking into account the time needed to go from an aisle to the next).

5. Experimental results

In this section we present the experiments performed to empirically study the influence of the proposed strategies and then to compare our best variant with the best algorithms identified in the state of the art (Albareda-Sambola et al. [1], Henn and Wäscher [21], and Öncan [30]). We have implemented our algorithms in Java 6 and they were run on an Intel QuadCore with 2.5 GHz and 6 GB of RAM with Xubuntu 14.04 64 bit OS.

We describe in this section the instances we used for the experiments (Section 5.1); our preliminary experiments to configure our algorithm and to illustrate the performance of the proposed strategies (Section 5.2); and the final experiments to compare the best methods in the state of the art with our best proposal (Section 5.3).

5.1. Description of the instances

We have considered two sets of instances previously used in this optimization problem. All of them are available at the website http://www.optsicom.es/obp/. We have divided the description of each set of instances in three different parts: warehouse layout, item distribution, and customer orders.

`Set HW` [21]. This set contains 2560 instances whose main features are:

- *Warehouse layout*: It consists of 900 storage locations, where each one stores a different article (item). The warehouse has 10 aisles with 90 storage locations each (45 on either side of each aisle). The length of each storage location is set to 1 length unit (LU). When the picker leaves an aisle, it is assumed that he/she moves 1 LU (from either the first or the last storage location to the cross aisle). Finally, the picker spends 5 LUs to move from the current aisle to the next one. The depot is located 1.5 LUs away from the first storage location in the leftmost aisle.
- *Item distribution*: There are two different scenarios: (1) ABC distribution and (2) random distribution. In the first one, items can be grouped into three classes. The first one contains very demanding items (Class A), where 10% of the articles represent 52% of the demand. The second class contains items with medium demand (Class B), where another 30% of the articles accounts for 36% of the demand. Finally, Class C contains items with low demand and represents the final 60% of the articles that represents 12% of the demand. Articles of Class A are stored in the first aisle, articles of Class B in the second, third and fourth aisles, and articles of Class C in the remaining 6 aisles. Notice that items are randomly located within a demand class. In the second scenario, items are randomly distributed among the storage locations.

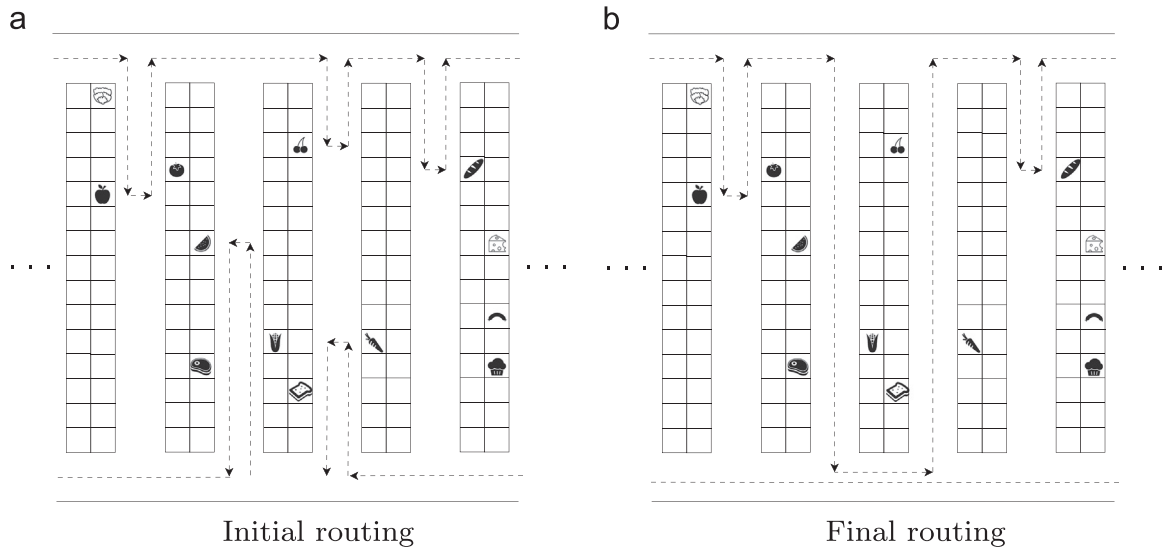

Fig. 8. Even number of SS aisles between two critical aisles. (a) Initial routing. (b) Final routing.

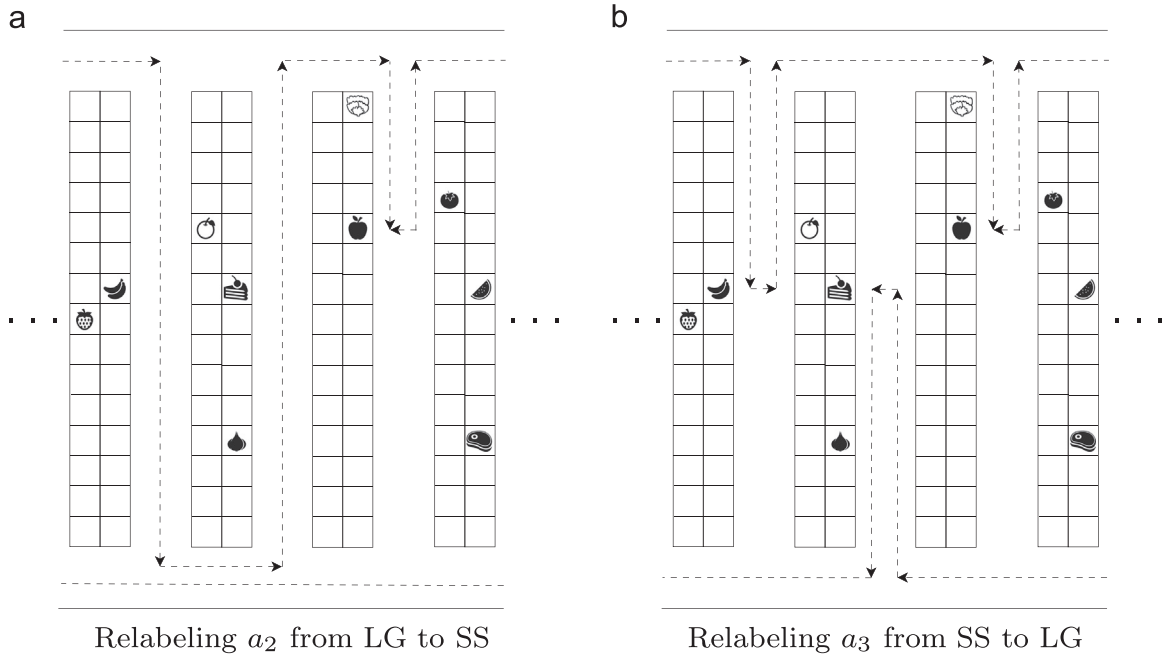

Fig. 9. Odd number of SS aisles between two critical aisles.

- *Customer orders*: This set of instances considers 4 different sizes of orders ($n = \{40, 60, 80, 100\}$), where the number of items per order is uniformly distributed in $\{5, 6, \ldots, 25\}$. The capacity of the picking device C (defined as the maximum number of items which can be assigned to a batch) has been fixed to 30, 45, 60 and 75.

`Set AAMD` [1]. This set contains 2400 instances whose main features are:

- *Warehouse layout*: It considers four different warehouses: W1, W2, W3, and W4. The first one has 4 parallel aisles with 60 storage locations (30 per side) and a total aisle length of 50 m. The distance between two consecutive aisles is 4.3 m and the picker speed is set to 0.6 m/s. W2 contains 10 aisles (20 storage locations on each side) with a length of 10 m. The separation between aisles is 2.4 m and the picker speed is also 0.6 m/s. The third warehouse has 25 aisles (25 storage locations per side) of length 50 m. The distance between two consecutive aisles is 5 m, and the picker speed 2 m/s. Finally, W4 has 12 aisles (16 storage locations per side) of 80 m length. The distance between two consecutive aisles is 15 m and the picker speed is 1 m/s.
- *Item distribution*: This set of instances also considers an ABC distribution and a random distribution. In this case, Class A contains 20% of the articles representing 80% of the demand. Class B) contains 20% of the articles representing 10% of the demand. Finally, Class C contains the remaining 60% of the articles that represents 10% of the demand. Items belonging to each class are assigned to aisles according to this percentages. The random distribution is equivalent to the aforementioned one.
- *Customer orders*: The order size for W1, W2, W3, and W4 is respectively $U(1, 7)$, $U(2, 10)$, $U(5, 25)$, and $U(1, 36)$, where U indicates a uniform distribution. The weight of each item is set to 1 for W1, W2, and W3. In the fourth warehouse, W4, the

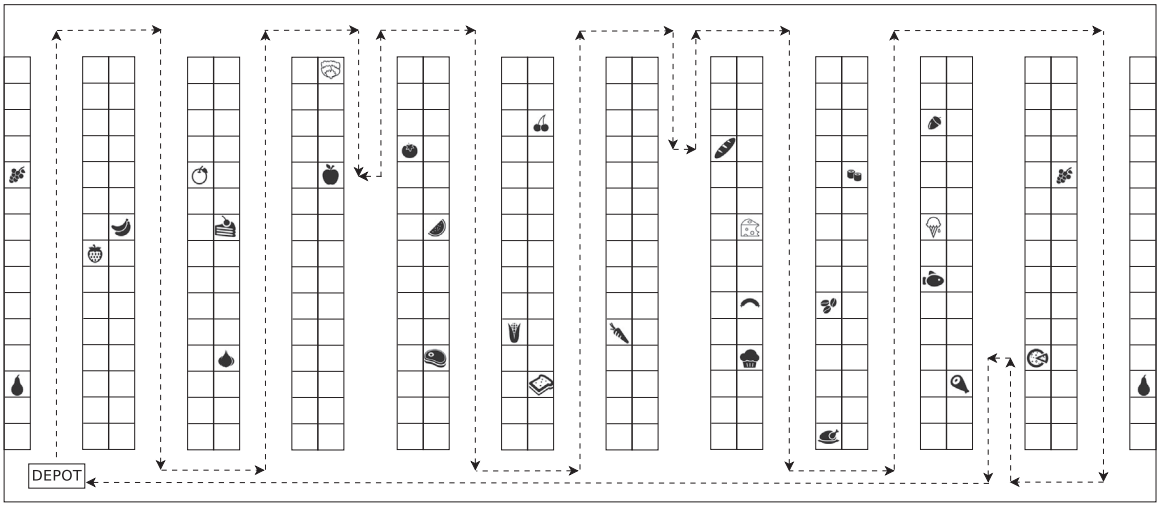

Fig. 10. Final routing.

weight is randomly generated according to a uniform distribution $U(1,3)$. Finally, the capacity of the picking device C is set to 12, 24, 150, and 80 for each warehouse, respectively.

The number of instances of each set is really huge (totalizing 4960 instances). We have empirically tested that there are no significant differences between using the whole dataset or a representative subset. In particular, the `set HW` contains 64 groups of 40 similar instances. Similarly, the `set AAMD` has 80 groups of 30 similar instances. Therefore, we propose to use only one instance per group to ease future comparisons, reducing the original dataset from 4960 to 144.

We have divided our experimentation into two different parts: preliminary experimentation and final experimentation. The preliminary experiments were performed to show the merit of the proposed search strategies. We consider a representative subset of 10% of the whole set of instances, with different properties.

5.2. Preliminary experimentation

In the first preliminary experiment we test the influence of the size of the Elite Candidate List, ECL (see Section 3.1), by considering three different values of α. In particular, given the set of all available improving moves, ML, the size of the ECL is determined as $|ECL| = \alpha \cdot |ML| + 1$ with $\alpha = \{0.01, 0.05, 0.10\}$. We increase the size of the ECL in one unit to guarantee the local optimality of the improved solution (i.e., if the ML contains, at least, one improving move, it is always applied independently of the value of α). We additionally test how the number of restarts affects in the quality of the obtained solution and the computing time by considering three different values ($cons = \{1, 50, 100\}$). We report in Table 1, for each pair of values, the average percentage deviation, Dev. (%), computed for each instance and each variant, as the difference between the value obtained with the corresponding variant and the best solution found in this experiment, divided by the best solution found in this experiment. We also report the computing time in seconds, CPUt (s).

Given a number of constructions, α does not substantially affects the performance of the method since small values of α implies to perform a move close to the best improving one and large values of α represents moves close to the first improving one. Therefore, we experimentally test that it is not possible to establish the superiority of one strategy over the other. On the other hand, when we increase the number of restarts, the quality is also improved. However, $\alpha = 0.01$ drives to a marginal improvement.

Table 1
Results for the constructive phase.

α	Cons	Dev. (%)	CPUt (s)
0.01	1	2.97	0.14
	50	1.12	6.40
	100	1.02	12.22
0.05	1	3.58	0.16
	50	0.69	6.61
	100	0.38	12.50
0.10	1	3.78	0.14
	50	0.72	5.96
	100	0.57	11.82

This fact can be partially explained by the fact that this value of α does not allow the method to generate different solutions. In particular, if the number of improving moves is less than 100, the ECL contains only 1 available move. Considering that the execution times with different values of α and the same number of constructions are very similar, we then select $\alpha = 0.05$ since it obtains the best values for more than one restart.

The second preliminary experiment is devoted to test the influence of the k_{max} parameter in the performance of the BVNS described in Section 3.2. In order to adjust this value to each specific problem, it is set as a percentage of the number of orders in a particular instance ($k_{max} = \{0.50n, 0.75n, 1.00n\}$, being n the number of orders).

It is expected that the larger the value of k_{max}, the larger the computing time and, hopefully, the lower the objective function value. In Table 2 we show the average deviation and the computing time for the three different values of k_{max}. In the table it is shown that $k_{max} = 1.00n$ obtains the best results in terms of quality but using the highest computing time. On the other hand, $k_{max} = 0.50n$ is the fastest BVNS variant but it produces lower quality results. Then, we select $k_{max} = 0.75n$ by the trade off between quality and computing time.

The next experiment is devoted to evaluate the relevance of the post-optimization strategy performed by a GVNS procedure (see Section 3.3). Fig. 11 illustrates the evolution of the original objective function (sum of the length of the routes) and the alternative objective function (number of full batches) on a representative instance. At the beginning of the process, the local optimal solution (ensured by the BVNS) contains 9 full batches (first iteration) with a value of the original objective function of 12 300. The GVNS

Table 2
Results for the BVNS phase.

k_{max}	0.50*n*	0.75*n*	1.00*n*
Dev. (%)	0.19	0.06	0.01
CPUt (s)	267.02	314.41	364.46

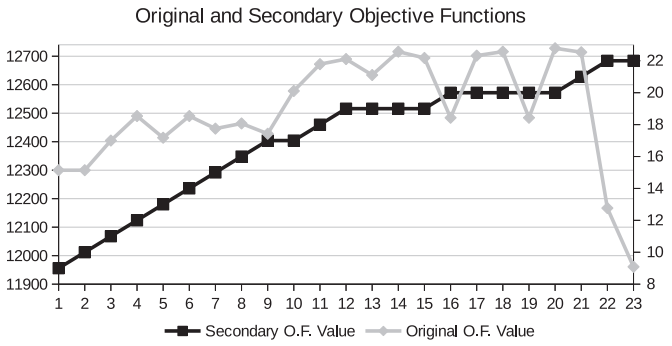

Fig. 11. Alternative and original objective function.

procedure then increases the number of full batches (from 9 to 17) in the following 9 iterations. These moves, in general, considerably deteriorates the quality of the objective function (from 12 300 to 12 428). In the ninth iteration, the GVNS gets stuck, and shakes the current solution to escape from a local optimum with respect to the alternative objective function. After that, the post-optimization method is able again to produce better outcomes. This process is repeated until the 22nd iteration, where the original objective function value is dramatically improved (from 12 714 to 12 167). This reduction is explained by the fact that a batch is totally emptied by redistributing its orders among the remaining batches. At the end of the process, the solution has improved in a 2.83% (from 12 300 to 11 961) and the number of full batches has been increased from 9 to 22. Notice that this post-optimization process cannot be further applied since the solution has 23 batches, meaning that the GVNS has reached the maximum number of full batches.

5.3. Final experimentation

In this section, we undertake a comparative analysis among our MS-VNS procedure and the current best algorithms for the Order Batching Problem: a Variable Neighborhood Descent (VND), an Attribute-Based Hill Climbing (ABHC), and an Iterated Local Search with Tabu Thresholding (ILST), [1,21,30], respectively. We consider two versions of the Ant-Based Hill Climbing: ABHC with the S-Shape routing strategy (ABHC+SS) and ABHC with the Largest Gap routing strategy (ABHC+LG) since there are no significant differences between them. The source code of VND, ABHC+SS, and ABHC+LG was kindly provided by the authors. Then, these algorithms have been executed in the same conditions as ours. However, we could not obtain the source code of ILST (to guarantee the same hardware conditions). We have re-implemented it but, unfortunately, our version produces significantly worse solutions than those published in Öncan [30]. Therefore, we have taken the results from the original source. For this reason, we first compare the MS-VNS with the best methods proposed in Albareda-Sambola et al. [1], Henn and Wäscher [21] (executed in the same computer) and then with the method presented in Öncan [30].

Table 3 compares the performance of VND, ABHC+SS, ABHC+LG, and MS-VNS on the whole set of Albareda-Sambola et al. [1] (with 80 instances). We report on each main row the average deviation with respect to the best-known solution, Dev. (%), the number of times that each method matches the best known solution, #Best, and the computing time in seconds, CPUt (s). We show these results on each warehouse (W1, W2, W3, and W4) and a summary on column Total. In this table, we show that our method systematically produces better outcomes than the competitors in all warehouses. In particular, the total average deviation is 0.13% which compares favorably to its competitors (3.18%, 6.51%, and 8.55%). According to the number of best solutions found, the analysis is similar. Specifically, our method obtains the best solution in 70 instances (out of 80). The computing time of ABHC+SS, ABHC+LG, and MS-VNS are similar (about 350 s on average). However, the VND method is considerably faster than the others. We have conducted an additional experiment to test the performance of our method in a short-time horizon. In particular, we executed MS-VNS for the same time as VND (6 s on average). Our method outperforms VND on 60 instances (out of 80), improving the average quality in 0.94%.

Table 3
Best methods on Albareda-Sambola et al. [1] set.

Statistics	Algorithm	W1	W2	W3	W4	Total
Dev. (%)	VND	1.95	2.41	5.89	2.48	3.18
	ABHC+SS	6.79	6.50	1.45	11.28	6.51
	ABHC+LG	0.58	1.88	23.54	8.21	8.55
	MS-VNS	0.10	0.01	0.02	0.38	0.13
#Best	VND	0	1	0	1	2
	ABHC+SS	1	0	2	0	3
	ABHC+LG	4	1	0	0	5
	MS-VNS	15	18	18	19	70
CPUt (s)	VND	0.85	2.98	10.64	9.42	5.97
	ABHC+SS	381.01	377.41	440.00	352.22	387.66
	ABHC+LG	413.43	581.21	693.47	602.10	572.55
	MS-VNS	113.36	286.98	787.18	200.72	347.06

We applied the non-parametric test of Friedman for multiple correlated samples to the best solutions obtained by each of the 4 methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 1 is assigned to the best method and rank 4 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated *p*-value or significance will be small. The resulting *p*-value is lower than 0.001. If we consider a level of significance of 0.05, it clearly indicates that there are statistically significant differences among the 4 methods tested. The rank values produced by this test are 1.15 (MS-VNS), 2.68 (VND), 2.94 (ABHC+LG), and 3.24 (ABHC+SS).

In Table 4 we report the results of the same four methods on the whole set of Henn and Wäscher [21] (with 64 instances). We show in this table the same statistics. We separate again the results by warehouse type (ABC_1, ABC_2, Ran_1, and Ran_2) and the summary on column Total. This experiment shows again the superiority of our method in this benchmark. Specifically, it achieves the smallest average deviation 0.19% and the largest number of best solutions found (56 out of 64), which are better than those obtained by VND (8.48% and 0), ABH+SS (6.26% and 8), and ABHC+LG (13.23% and 0). As in the previous experiment MS-VNS, ABHC+SS, and ABHC+LG runs for much more time than VND. Then, we execute our method the same time than VND (i.e., about 0.5 s). In that time our method outperforms the VND in all instances, improving the quality in 5.8%. Notice that in the same 0.5 s our method is also able to obtain better results than ABHC+SS (defeating it in 46 instances out of 64).

We confirm the superiority of our method by applying the non-parametric test of Friedman. The corresponding *p*-value is lower than 0.001, indicating that there are statistically significant differences among the four methods tested. The associated rank

Table 4
Best methods on Henn and Wäscher [21] set.

Statistics	Algorithm	ABC_1	ABC_2	Ran_1	Ran_2	Total
Dev. (%)	VND	9.60	8.66	8.35	7.32	8.48
	ABHC+SS	5.00	9.93	2.00	8.10	6.26
	ABHC+LG	18.02	8.44	12.05	14.43	13.23
	MS-VNS	0.05	0.00	0.70	0.00	0.19
#Best	VND	0	0	0	0	0
	ABHC+SS	1	0	7	0	8
	ABHC+LG	0	0	0	0	0
	MS-VNS	15	16	9	16	56
CPUt (s)	VND	0.44	0.46	0.59	0.72	0.55
	ABHC+SS	10.45	12.30	14.71	12.18	12.41
	ABHC+LG	41.19	43.24	75.18	47.96	51.89
	MS-VNS	39.38	37.69	47.51	50.68	43.81

values are 1.13 (MS-VNS), 2.44 (ABHC+SS), 2.92 (VND), and 3.52 (ABHC+LG).

We finally compare our method with the Iterated Local Search with Tabu Thresholding, ILST, presented in Öncan [30]. The author considers three different routing strategies: S-Shape, Midpoint, and Return. Experimental results reported in Öncan [30] show that the third routing strategy is not competitive. Therefore, we only include in our comparison the ILST coupled with the S-Shape (ILST+SS) and Midpoint (ILST+MP).

As it was aforementioned, we do not have the source code of these procedures. Then, we have directly used the results reported in Öncan [30]. The computer used to execute ILST+SS and ILST+MP is a QuadCore, 3.16 GHz with 32 GB RAM, while our computer is an Intel QuadCore, 2.5 GHz with 6 GB RAM. In any case, the computing time in this experiment must be considered as an indicative value.

The set of instances used in Öncan [30] is a subset of Henn and Wäscher [21]. In particular, it contains two different warehouses, ABC_1 and Ran_1, where the number of orders n ranges from 20 to 100 with increment of 20, and picker capacities C have values of 30, 45, 60, and 75.

In Table 5 we show the experimental results on the first warehouse, with an ABC distribution and 200 instances. We report for each pair (number of orders, picker capacity) the average percentage deviation (Dev. (%)) and the computing time (CPUt (s)) for the three considered methods. These results are averaged over 10 instances. Attending to the average results, our method consistently produces better outcomes. In particular, the average percentage deviation is 0.04% in 33.7 s, while ILST+SS and ILST+MP obtain a deviation of, respectively, 8.73% (in 73.58 s) and 9.50% (in 75.72 s). If we analyze the results by considering the 20 problem scenarios (number of batches and picker capacity) our method also improve the results of the competitors in 19 of those configurations (highlighted with bold fonts). According to the CPU time, our method is the fastest in 11 configurations, while ILST+SS and ILST+MP are the fastest in 4 and 5 problem scenarios, respectively. We test whether there are statistically significant differences among the three compared methods or not. The resulting p-value is lower than 0.001 and the rank values are 1.05 (MS-VNS), 2.45 (ILST+SS), and 2.50 (ILS+MP), confirming the competitiveness of our method.

Experimental results on the second warehouse are reported in Table 6. Our method obtains better average percentage deviation in 19 out 20 different scenarios (highlighted with bold fonts), being the fastest algorithm in 10 of them. When considering this complete benchmark, the average percentage deviation is 0.01% in 43.80 s, while ILST+SS and ILST+MP obtain, respectively, 6.39% in 81.61 s and 16.66% in 80.13 s. The associated test of Friedman has a p-value lower than 0.001, and the rank values are 1.05 (MS-VNS), 2.20 (ILST+SS), and 2.75 (ILST+MP).

To detect the differences among our method and the best previous methods (VND [1], ABHC+SS [21], and ILST+SS [30]), we also conducted a test of Wilcoxon. Let R_+ be the sum of ranks for the functions on which our method outperforms the competitor, and R_- be the sum of ranks for the opposite. Ranks corresponding to zero differences are split evenly among the sums. If $\min\{R_+, R_-\}$ is lower than or equal to the critical value, Wilcoxon's test detects

Table 5
Comparison with Iterated Local Search with Tabu Thresholding [30] on 200 ABC instances.

Configuration		ILST+SS		ILST+MP		MS-VNS	
n	C	**Dev. (%)**	**CPUt (s)**	**Dev. (%)**	**CPUt (s)**	**Dev. (%)**	**CPUt (s)**
20	30	17.28	1.00	6.71	1.60	**0.00**	**0.79**
	45	3.68	1.30	3.24	**1.20**	**0.00**	1.83
	60	9.09	**1.10**	14.21	1.50	**0.00**	2.72
	75	4.38	**1.20**	14.13	1.50	**0.00**	3.10
40	30	9.38	9.40	**0.00**	11.50	0.73	**2.90**
	45	7.09	8.70	7.56	11.90	**0.00**	**7.26**
	60	3.19	7.70	9.25	**7.40**	**0.00**	12.62
	75	3.53	**7.80**	13.54	9.80	**0.00**	18.39
60	30	12.05	18.60	2.13	18.20	**0.00**	**7.88**
	45	9.10	**16.50**	8.95	19.40	**0.00**	18.24
	60	7.69	20.80	13.10	**20.60**	**0.00**	34.40
	75	7.21	19.30	16.91	**17.10**	**0.00**	44.11
80	30	13.91	52.00	3.75	59.80	**0.00**	**16.20**
	45	7.65	67.40	5.75	55.40	**0.00**	**35.00**
	60	9.29	58.20	12.62	74.60	**0.00**	**57.98**
	75	9.48	69.80	16.07	**69.10**	**0.00**	79.89
100	30	11.37	287.40	4.74	291.80	**0.00**	**28.27**
	45	10.73	278.70	9.25	286.40	**0.00**	**53.74**
	60	11.22	269.60	13.54	256.60	**0.00**	**99.25**
	75	7.26	295.10	14.54	299.00	**0.00**	**149.46**
Average		8.73	74.58	9.50	75.72	0.04	33.70

Table 6
Comparison with Iterated Local Search with Tabu Thresholding [30] on 200 random instances.

Configuration		ILST+SS		ILST+MP		MS-VNS	
n	C	**Dev. (%)**	**CPUt (s)**	**Dev. (%)**	**CPUt (s)**	**Dev. (%)**	**CPUt (s)**
20	30	17.19	1.80	12.40	1.00	**0.00**	**0.95**
	45	**0.00**	**1.00**	7.91	1.20	0.15	2.37
	60	5.61	1.80	21.22	**1.30**	**0.00**	4.30
	75	2.18	**1.30**	23.68	1.70	**0.00**	4.13
40	30	5.87	8.90	4.14	9.10	**0.00**	**3.71**
	45	6.15	9.60	14.78	12.50	**0.00**	**8.58**
	60	2.56	11.70	18.82	**11.40**	**0.00**	17.15
	75	2.37	**9.90**	23.77	12.80	**0.00**	26.48
60	30	7.98	16.50	5.76	24.60	**0.00**	**9.54**
	45	5.63	**20.60**	14.97	21.40	**0.00**	26.37
	60	5.65	**17.50**	20.91	22.80	**0.00**	48.16
	75	6.64	**18.80**	27.37	23.40	**0.00**	51.47
80	30	11.45	74.60	9.28	73.60	**0.00**	**18.31**
	45	6.16	74.30	13.75	68.70	**0.00**	**46.68**
	60	6.32	**63.40**	20.37	69.40	**0.00**	76.82
	75	6.79	**51.20**	25.15	62.10	**0.00**	102.33
100	30	9.21	331.70	7.18	284.60	**0.00**	**32.03**
	45	6.94	316.30	14.71	279.20	**0.00**	**75.10**
	60	8.21	322.80	22.21	291.10	**0.00**	**134.07**
	75	4.95	278.50	22.84	330.60	**0.00**	**187.57**
Average		6.39	81.61	16.56	80.13	0.01	43.80

Table 7
Results of the test of Wilcoxon.

Algorithms	R_-	R_+	$p-$Value	Significant
MS-VNS vs. VND	146	10 294	0.001	Yes
MS-VNS vs. ABHC+SS	389	10 051	0.001	Yes
MS-VNS vs. ILST+SS	1	819	0.001	Yes

significant differences between the algorithms, which means that an algorithm outperforms its opponent.

In Table 7 we summarize the results of this statistical test with a level of significance 0.05, where the values of R_+ (associated to our method) and R_- (associated to the competitors) of the test are specified in the second and third columns. The fourth column reports the p-value associated to each experiment and the last column indicates whether the test of Wilcoxon found statistical differences between these algorithms or not. In particular, if $\min\{R_+, R_-\}$ is lower than or equal to the critical value [38] in this experiment, this test detects that an algorithm outperforms its opponent. In particular, if this fact occurs and, simultaneously, $R_- = \min\{R_+, R_-\}$, then MS-VNS is better than the competitor. Notice that the confidence of the test is always determined by the p-value. These results complement the ones reported in previous tables. Specifically, the test of Wilcoxon again confirms the superiority of MS-VNS over VND, ABHC+SS, and ILST+SS.

6. Conclusions

Our goal was to develop a state-of-the-art solution method for the Order Batching Problem. We accomplished this goal with an implementation of a two-stage Variable Neighborhood Search that computational experiments show to be superior to the solution methods reported in the literature. We developed a number of mechanisms that we believe can be helpful in similar problem settings. Specifically, we propose a procedure to construct high-quality but diverse initial solutions based on an Elite Candidate List (ECL) within the local search method. The initial solution is then improved with a Basic Variable Neighborhood Search where the shake and the improvement methods consider a nested exploration of two different neighborhoods, returning a local optimum with respect to both of them. Finally, the local optimum is further improved with a post-optimization strategy based on a General Variable Neighborhood Search that relies on an alternative objective function. We additionally proposed a new routing strategy based on the ideas of the Combined strategy. We performed an extensive computational testing over a large set of instances, considering different warehouse layouts. Experimental results show that the proposed algorithm outperforms the best methods identified in the state of the art. We also conducted statistical tests to confirm the significance of the obtained results.

Acknowledgment

This research has been partially supported by the Spanish Ministry of “Economía y Competitividad” and “Comunidad de Madrid” with Grants refs. TIN2012-35632-C02, TIN2015-65460-C2-2-P, and S2013/ICE-2894. The authors thank Prof. Henn and Prof. Alonso-Ayuso for providing us with their Ant-Based Hill Climbing method and Variable Neighborhood Descent, respectively, for the Order Batching Problem.

References

[1] Albareda-Sambola M, Alonso-Ayuso A, Molina E, De Blas CS. Variable neighborhood search for order batching in a warehouse. Asia-Pac J Oper Res 2009;26(05).
[2] Alvim Adriana CF, Ribeiro Celso C, Glover Fred, Aloise Dario J. A hybrid improvement heuristic for the one-dimensional bin packing problem. J Heuristics 2004;10(2):205–29.
[3] Coyle JJ, Bardi EJ, Langley CJ. The management of business logistics, vol. 6. Minneapolis/St. Paul: West Publishing Company; 1996.
[4] De Koster MBM, Van der Poort ES, Wolters M. Efficient order batching methods in warehouses. Int J Prod Res 1999;37(7):1479–504.
[5] 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. Springer Berlin Heidelberg; 1999. p. 215–34.
[6] De Koster R, Le-Duc T, Roodbergen KJ. Design and control of warehouse order picking: a literature review. Eur J Oper Res 2007;182(2):481–501.
[7] Drury J. Towards more efficient order picking. IMM Monogr 1988;1.
[8] Duarte A, Pantrigo JJ, Pardo EG, Sánchez-Oro J. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J Manag Math 2013. http://dx.doi.org/10.1093/imaman/dpt026.
[9] Duarte A, Pantrigo JJ, Pardo EG, Mladenovic N. Multi-objective variable neighborhood search: an application to combinatorial optimization problems. J Glob Optim 2015;63(3):515–36.
[10] Gademann N, Velde S. Order batching to minimize total travel time in a parallel-aisle warehouse. IIE Trans 2005;37(1):63–75.
[11] Garey MR, Johnson DS. Computers and intractability: a guide to np-completeness; 1979.
[12] Garey MR, Graham RL, Ullman JD. An analysis of some packing algorithms. In: Combinatorial algorithms (Courant computer science symposium, no. 9; 1972); 1973. p. 39–47.
[13] Ghiani G, Laporte G, Musmanno R. Introduction to logistics systems planning and control. John Wiley & Sons; 2004.
[14] Gibson DR, Sharp GP. Order batching procedures. Eur J Oper Res 1992;58(1):57–67.
[15] Glover F, Laguna M. Tabu search. Norwell, MA, USA: Kluwer Academic Publishers; 1997 [ISBN 079239965X].
[16] Goetschalckx M, Ratliff HD. Order picking in an aisle. IIE Trans 1988;20(1):53–62.
[17] Gu J, Goetschalckx M, McGinnis LF. Research on warehouse operation: a comprehensive review. Eur J Oper Res 2007;177(1):1–21.
[18] Gu J, Goetschalckx M, McGinnis LF. Research on warehouse design and performance evaluation: a comprehensive review. Eur J Oper Res 2010;203(3):539–49.
[19] Hall RW. Distance approximations for routing manual pickers in a warehouse. IIE Trans 1993;25(4):76–87.
[20] Hansen P, Mladenović N, Moreno P. Variable neighbourhood search: algorithms and applications. Ann Oper Res 2010;175(1):367–407.
[21] Henn S, Wäscher G. Tabu search heuristics for the order batching problem in manual order picking systems. Eur J Oper Res 2012;222(3):484–94.
[22] 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 2010;3(1).
[23] Heragu SS. Facilities design, iuniverse. NE: Lincoln; 2006.
[24] 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 2006;44(17):3391–417.
[25] Hsu C-M, Chen K-Y, Chen M-C. Batching orders in warehouses by minimizing travel distance with genetic algorithms. Comput Ind 2005;56(2):169–78.
[26] Karasek J. An overview of warehouse optimization. Int J Adv Telecommun Electrotech Signals Syst 2013;2(3):111–7.
[27] Martí R, Moreno-Vega JM, Duarte A. Advanced multi-start methods. In: Handbook of metaheuristics. Springer US, 2010. p. 265–81.
[28] Menéndez B, Pardo EG, Duarte A, Alonso-Ayuso A, Molina E. General variable neighborhood search applied to the picking process in a warehouse. Electron Notes Discret Math 2015(47):77–84.
[29] Mladenović N, Hansen P. Variable neighborhood search. Comput Oper Res 1997;24(11):1097–100.
[30] Öncan T. Milp formulations and an iterated local search algorithm with tabu thresholding for the order batching problem. Eur J Oper Res 2015.
[31] Pan C-H, Liu S-Y. A comparative study of order batching algorithms. Omega 1995;23(6):691–700.
[32] Pardo EG, Mladenović N, Pantrigo JJ, Duarte A. Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 2013;13(5):2242–52.
[33] Petersen CG. An evaluation of order picking routing policies. Int J Oper Prod Manag 1997;17(11):1098–111.
[34] Ratliff HD, Rosenthal AS. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Oper Res 1983;31(3):507–21.
[35] Roodbergen KJ, De Koster R. Routing methods for warehouses with multiple cross aisles. Int J Prod Res 2001;39(9):1865–83.
[36] Roodbergen KJ, Petersen CG. How to improve order picking efficiency with routing and storage policies. In: Progress in material handling practice; 1999. p. 107–24.
[37] Rosenwein MB. A comparison of heuristics for the problem of batching orders for warehouse selection. Int J Prod Res 1996;34(3):657–64.
[38] Wilcoxon, Frank. Individual comparisons by ranking methods. Biometrics bulletin 1945;1(6):80–3.