An algorithm for batching, sequencing and picking operations in a warehouse

(presented at the 6th IESM Conference, October 2015, Seville, Spain) © I^{4}e^2 2015

Manuel Bustillo[*], Borja Menéndez[†], Eduardo G. Pardo[‡], and Abraham Duarte[§]
[*†‡§]Universidad Rey Juan Carlos
[*†‡§]Dept. Computer Science
[*]Email: m.bustilloa@alumnos.urjc.es
[†]Email: borja.menendez@urjc.es
[‡]Email: eduardo.pardo@urjc.es
[§]Email: abraham.duarte@urjc.es

Abstract—**Order Batching is an optimization problem related to the picking process of orders in a warehouse. It consists in grouping the orders (each order is composed by a list of items to be collected) received in a warehouse in a set of batches of a maximum fixed capacity. Then, a route to collect the items in the same batch must be conformed. In this paper we tackle a variant of this problem known as Order Batching and Sequencing Problem where each order has a certain due date. This variant consists in grouping the orders into batches and then sequencing them, in such a way that the tardiness of each order (the extra time over the due date needed to collect it) is minimized. In this paper we propose a Variable Neighborhood Search algorithm to tackle the problem. Our approach outperforms previous attempts in the state of the art.**

I. Introduction

Warehouse management systems have become an essential part of the supply chain strategy. They mainly focus on moving and storing materials within a warehouse by performing different transactions (including shipping, receiving, and picking). The success of warehouse management strongly depends on how customer orders (containing a set of goods or items) are retrieved. The picking process consists in collecting articles (items) in a specified quantity before shipment to satisfy the orders of the customers. This picking process may consume up to 60% of the total time of all labor activities in the warehouse (Drury, 1988), which can suppose more than half of the total operating costs.

Given a set of orders received in the warehouse, there are two basic order-picking strategies: *strict-order picking* and *order batching*. In the first one, each picker collects all the items included in one order. Once he/she finishes, the picker continues with the second order and so on. In the order batching strategy, several orders are put together into batches. Then, each batch is assigned to a picker, who can retrieve the items of any order grouped into the assigned batch, and satisfies a capacity constraint (fixed by a maximum weight).

As it is well documented in the literature, reducing travel time of picking operations improves order picking productivity. This is why batching strategies are used in warehouses. According to De Koster et al. (1999), if batching and routing are simultaneously considered, the associated benefits can be considerably increased, reducing travel times more than 35%. Although in this paper we focus on the batching and sequencing strategies to minimize the total tardiness, the objective function is evaluated by means of the routing algorithm.

In this paper we deal with an optimization problem which is a variant of the Order Batching Problem (OBP). The OBP consists in minimizing the total time needed to collect all the orders received in a warehouse, when the batching policy is considered. An order is composed of several items, and these items are placed at specific pick locations in the warehouse. A set of orders grouped together conform a batch. Batches have a maximum capacity fixed in advance. This capacity can not be exceeded by the weight of the items in the orders grouped in the same batch. This problem has been proved to be $\mathcal{NP}$-hard for general instances, but it is solvable in polynomial time if each batch does not contain more than two orders (Gademann and Velde, 2005). Unfortunately, real warehouse instances does not usually fall into this category. Therefore, the OBP has been heuristically approached in the last few years. Among the most recent approaches to tackle the problem, Albareda-Sambola et al. (2009) proposed a Variable Neighborhood Descent (VND) algorithm that considered 6 different neighborhood structures. Henn et al. (2010) proposed two procedures. An Iterated Local Search (ILS) and a straightforward implementation of a Rank-Based Ant System. Later, Henn and Wäscher (2012) improved previous results by proposing two metaheuristic methods. A Tabu Search algorithm and an Attributed-Based Hill Climbing (ABHC) method with two genuine attribute sets. Öncan (2015) proposed several Mixed Integer Linear Programming (MILP) formulations, one for the batching problem and three for the routing problem. Additionally, the author proposed an Iterated Local Search with a Tabu Thresholding method. As far as we know, the most recent approach is Menéndez et al. (2015), where the authors explored different Variable Neighborhood Search strategies to tackle the problem.

The addressed variant in this paper, known as Order Batching and Sequencing Problem (OBSP), involves not only sorting the orders into batches and retrieving the items in each order from their location, but also achieving the time requirements of customers. These requirements mean that each order has to be delivered before a determined due date. From this perspective, it turns out that if an order is not delivered on time, it has associated a tardiness. This tardiness is the extra time over the due date needed to collect an order. The aim of the OBSP is then to minimize the sum of the tardiness of all orders received in the warehouse. Then, from an optimization point of view, we can identify three different problems: how to group orders into batches (batching), how to determine the order to collect the conformed batches (sequencing) and how to design the corresponding routes to collect them (routing). However, in this variant, instead of determining the quality of a solution by means of the total travel time (as in the case of the OBP), the order batching has to be evaluated with respect to the tardiness of the corresponding orders (Elsayed et al., 1993). It is also important to notice that, the orders received, must be retrieved by a single picker.

In this paper we will tackle the batching and sequencing tasks (see Section III), since routing strategies have been deeply studied in the associated literature (see Section II). In particular, we tackle the version of the OBSP where warehouses have parallel aisles of equal length connected by two cross aisles (one at the front and one at the back). Warehouses considered have a depot located in the front cross aisle. In Figure 1 we illustrate an example of a warehouse layout with 5 parallel aisles, 10 shelves with 6 pick locations each, 2 cross aisles, and a depot located at the left bottom corner. Additionally, we have represented with darkened tiles the position of several items to be retrieved. Later, we will illustrate several routes to collect these items.

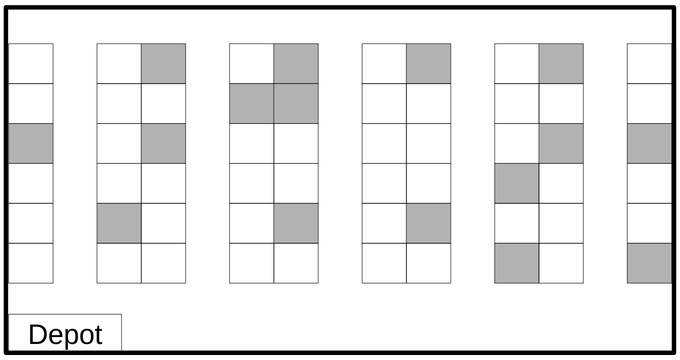


Fig. 1: Layout of a warehouse.

The OBSP has been recently studied in Henn and Schmid (2013), by using a modified version of the Iterated Local Search and the Attribute-Based Hill Climbing methods proposed in Henn et al. (2010) and Henn and Wäscher (2012), respectively. Hossein et al. (2013) proposed an hybrid approach based on combining a Weighted Association Rule Mining, a binary integer programming model, and two different Genetic Algorithms. The authors first determine the affinity among orders with respect to their due date, based on an Integer Linear Programming model. Then, they construct the batches and, finally, they apply two Genetic Algorithms: one to determine the sequence of the previously created batches and another one to construct the routes to collect the batches. The last attempt to address this problem has been driven by Chen et al. (2015). In that paper, the authors developed a new Genetic Algorithm to search near-optimal solutions of order batching and sequencing, and an Ant Colony Optimization algorithm for routing. However, these last two approaches only considereded very small test cases.

In this paper we propose an algorithm based on the Variable Neighborhood Search (VNS) methodology to address the OBSP. In particular, we center our attention in the batching and sequencing strategies, making use of previously defined algorithms for the routing. The rest of the paper is organized as follows. In Section II we introduce some of the most relevant routing strategies in the literature. Later, in Section III we present a VNS algorithm to tackle the batching and sequencing duties. Finally, computational experiments and the associated conclusions are presented in sections IV and V, respectively.

II. Routing strategies

The routing operation determines a sequence of steps to collect all the items in the same batch. Additionally, it also helps to determine the time needed to collect the orders. A picker retrieves the items in a given batch by following a specific route, which starts and finishes at the depot. The time employed in performing that route is known as *retrieving time* of a batch. It is computed as the sum of the *travel time* plus the *extraction time* which includes, as mentioned above, the time needed localize the item, the time needed to extract it from its pick location, the associated time employed in accelerating/decelerating the picking cart, etc. The routing problem, which actually determines the value of the *retrieving time*, can be exactly solved in polynomial time (see Ratliff and Rosenthal, 1983). However, as it is well documented, the optimal routes can be illogical for human pickers. Therefore, this kind of solutions are usually discarded in real situations, resorting to heuristic strategies.

The easiest routing strategy for warehouse workers is the S-Shape or *Traversal* strategy. It consists in traversing an aisle, that contains at least one item to be collected, from the front cross aisle to the back cross aisle (or the other way round). If the number of parallel aisles is odd, the last aisle is traversed until the farthest item from the front cross aisle. In Figure 2 we show an example of how a picker collects a set of items grouped in the same batch following the S-Shape strategy. Notice that the last aisle is not fully traversed since the number of aisles is odd. The retrieving time is computed as the time needed to perform this route and to collect all the required items.

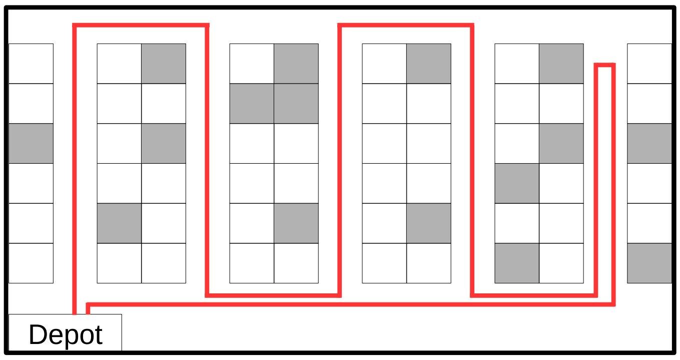


Fig. 2: Route obtained with a S-Shape strategy.

In order to evaluate this solution, and for the sake of simplicity, in Figure 2 we assume that each picking position has a length/width equal to 1. Then, the picker traverses 6 units of length when he/she goes from the front to the back cross aisles. Similarly, going from one parallel aisle to the next one has an associated distance of 2 units of length. Notice that we do not consider the distance associated to the turn and the distance to leave/enter the depot. Then, in this example, the length of the route followed by the picker to collect all the items is 52 units.

The Largest Gap strategy is based on the idea of gap. A gap is defined in a parallel aisle as: the distance between two consecutive items within the aisle; or the distance between the front cross aisle and the nearest item in the parallel aisle; or the distance between the back cross aisle and the nearest item in the parallel aisle. The largest gap distance of a parallel aisle is the maximum of the previous defined distances. The Largest Gap strategy avoids to traverse the largest gap distance, performing a U-turn when the picker reaches the position of the item where the largest gap distance starts. In this routing strategy, the picker fully traverses both, the first and the last aisle that contain required items. The rest of the parallel aisles are partially traversed since the picker avoids to traverse the largest gap distance. In Figure 3 we illustrate the route conformed by the Largest Gap strategy for the same example depicted in Figure 1 and Figure 2.

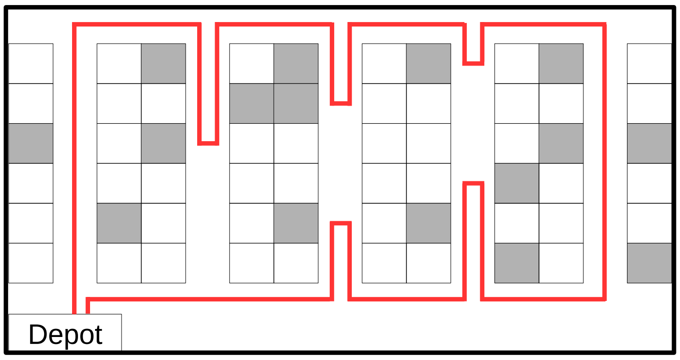


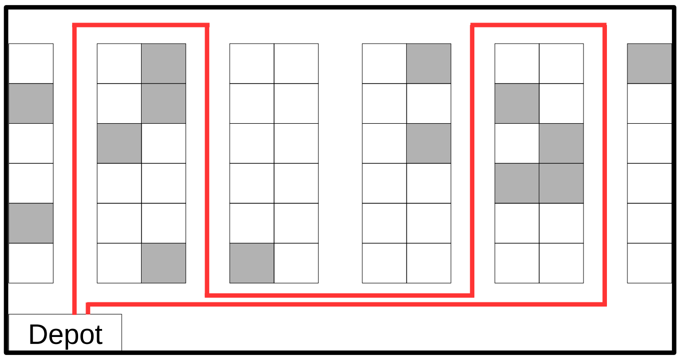
Fig. 3: Route obtained with a Largest Gap strategy.

As we can see in Figure 3, the first and the last aisles are fully traversed. The second aisle is only entered and exited from the back cross aisle (performing only one U-turn). Finally, the third and fourth aisles, are entered and exited from the back cross aisle and from the front cross aisle (performing two U-turns). In this case, the length of the route followed by the picker to collect all the items is 50 units. This factor indicates that the Largest Gap strategy for this configuration of items to retrieve is better than the S-Shape strategy.

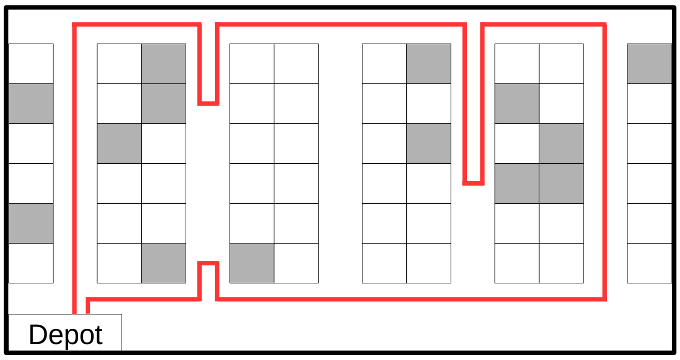
However, Largest Gap is not always better than S-Shape, since it depends on the particular instance (warehouse distribution and items to be collected). For example, in Figure 4 we show a different configuration of items to be collected into the same warehouse. In this case, the picker traverses 40 units if he/she chooses the S-Shape strategy (see Figure 4a), while the length of the route is 42 units if he/she chooses the Largest Gap strategy (see Figure 4b).

III. Batching and Sequencing Algorithm

Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) is a metaheuristic which exploits the idea



(a) Route obtained with a S-Shape strategy in a different order configuration.



(b) Route obtained with a Largest Gap strategy in a different order configuration.

Fig. 4: S-Shape and Largest Gap strategies for the same configuration of items to be collected.

of neighborhood change in a systematic way. The aim is to descend to a local optimum or, alternatively, to escape from the basin of attraction from that local optimum. The original metaheuristic has been widely evolved with many extensions. For instance: Reduced VNS (RVNS) explores solutions at random in each neighborhood by perturbing the solutions with the shaking procedure; Variable Neighborhood Descent (VND) explores the neighborhood in a deterministic way by using several local search procedures; Basic VNS (BVNS) combines deterministic and random explorations of the neighborhoods; and, General VNS (GVNS) presents a similar schema to BVNS but changing the Local Search to explore the solution space with a VND. See Hansen et al. (2008) for a thorough review of the most common approaches. More advanced implementations of the methodology include: a variant based on changes of the formulation of the problem to deal with hard min-max/max-min optimization problems with flat landscape (G. Pardo et al., 2013); a variant to tackle Multi-Objective optimization problems (Duarte et al., 2014); or even a parallel implementation of this metaheuristic Duarte et al. (2013).

In this paper we propose a General Variable Neighborhood Search for the Order Batching and Sequencing Problem. The pseudocode of this algorithm is presented in Algorithm 1. This scheme has three input parameters: the initial solution (S), the largest neighborhood to be explored (k_{max}), and the maximum computing time (t_{max}). The influence of the parameter k_{max}

on the search is studied in Section IV. The parameter t_{max} is set in order to match the computing time of previous approaches in the state of the art. Finally, the construction of the initial solution does not belong to the GVNS scheme. In fact, it is usual in the VNS community that the initial solution is constructed at random. However, as it is well documented in the related literature, a more elaborated constructive procedure might improve considerably the quality of the best solution found. In Section III-A, we describe the constructive procedure proposed for the OBSP.

The GVNS algorithm starts by initializing k to the first neighborhood (step 3), and then it enters in the main loop (Steps 5, 6, and 7). First, the current solution, S, is perturbed in the shake procedure by applying a random move in the k-neighborhood (see Section III-C). Next, the obtained solution, S', is improved within a VND procedure, which considers two different neighborhoods (see Section III-B). The VND is deeply described in Section III-D. Finally, we use a standard implementation of the neighborhood change method. In particular, this procedure determines which neighborhood is the next one to be explored. If the perturbed and improved solution (S'') outperforms S, the GVNS method considers S'' as the new incumbent solution ($S \leftarrow S''$) and resets the search to the first neighborhood ($k = 1$). Otherwise, the current solution is not updated, but a larger neighborhood is explored ($k = k+1$). These three steps are repeated until the largest neighborhood (k_{max}) is explored without finding an improvement. Then, if the maximum computing time (t_{max}) has not been exceeded, the GVNS performs a new iteration.

Algorithm 1 GVNS algorithm.

```
 1: procedure GVNS($S, k_{max}, t_{max}$)
 2:     repeat
 3:         $k \leftarrow 1$
 4:         repeat
 5:             $S' \leftarrow Shake(S, k)$
 6:             $S'' \leftarrow VND(S')$
 7:             $NeighborhoodChange(S, S'', k)$
 8:         until $k > k_{max}$
 9:         $t \leftarrow CPUTime()$
10:     until $t \geq t_{max}$
11:     return $S$
12: end procedure
```

A. Constructive method

As it has been previously mentioned, VNS usually takes a random feasible solution, but a more elaborated constructive procedure might improve considerably the quality of the best solution found. Based on this idea, the initial solution is constructed by means of the Earliest Due Date (EDD). This constructive algorithm first sorts all orders according to their due date in such a way that orders with a closer due date come first. Next, orders are successively assigned to batches following the previously defined organization ensuring that the capacity constraint is not violated. In this sense, the first batch is filled out until the next order coming does not fit into the batch. Then, a new empty batch is created, the previous order is assigned to this batch, and so on.

B. Neighborhood structures

A solution to the OBSP is represented as a list of m batches, i.e., $S = \{B_1, B_2, \ldots, B_m\}$, where each batch B_j contains an unfixed number of orders that do not overcome the capacity of the picking cart (i.e., $|B_j| \leq C$, where $|B_j|$ denotes the weight of the batch B_j and C denotes the capacity of the cart), and each order is formed by an unfixed number of items that must be retrieved. The free space of a batch B_j is denoted as $\overline{B_j}$. According to this solution representation, an insert move, denoted by $Insert(S, O_i, B_j, B_k)$, produces a new solution S' where the order O_i is removed from its current batch (B_j) and it is 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., $|O_i| \leq \overline{B_k}$).

In Figure 5 we illustrate the $Insert$ move with an example. In this case, the initial configuration (Figure 5a) of the batches includes two batches: B_j (with three orders, O_1, O_2, and O_3) and B_k (with two orders, O_4 and O_5). Given this configuration, we illustrate how to perform $Insert(S, O_2, B_j, B_k)$. This move tries to insert the order $O_2 \in B_j$ in B_k. Once the move has been performed, the new configuration of B_j is $B_j = \{O_1, O_3\}$ and the new configuration of B_k is $B_k = \{O_2, O_4, O_5\}$ (see Figure 5b).

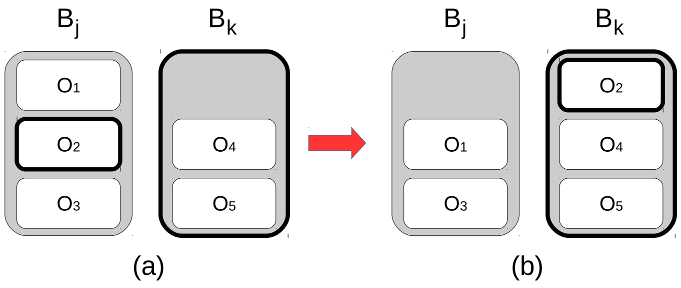


Fig. 5: Example of an *Insert* move.

Considering the $Insert$ move previously introduced, we can define the neighborhood N_1 of a solution S as:

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

where $O_i \in B_j$; $1 \leq i \leq n$, being n the number of orders; $1 \leq j \leq m$, $1 \leq k \leq m$, and $j \neq k$, being m the number of batches.

We now introduce an additional move, the swap move, denoted by $Swap(S, O_i, B_j, O_k, B_l)$. Given a solution S, the $Swap$ move produces a new solution S' where the order O_i is removed from its current batch (B_j) and then it is inserted in the batch B_l. Simultaneously, the order O_k is removed from its current batch (B_l) and inserted in B_j. This move is performed if and only if S' is a feasible solution (i.e., $|O_i| \leq \overline{B_k} + |O_k|$ and $|O_k| \leq \overline{B_j} + |O_i|$).

In Figure 6 we illustrate the $Swap$ move with an example. In this case, in the initial configuration the batch B_j is composed by $\{O_1, O_2, O_3\}$ while B_l is composed by $\{O_4, O_5\}$ (see Figure 6a). In particular, we show how to perform $Swap(S, O_2, B_j, O_5, B_l)$. This move involves the orders

$O_2 \in B_j$ and $O_5 \in B_l$, and results in a new configuration of batches where B_j is composed by $\{O_1, O_3, O_5\}$ and B_l is composed by $\{O_2, O_4\}$ (see Figure 6b).

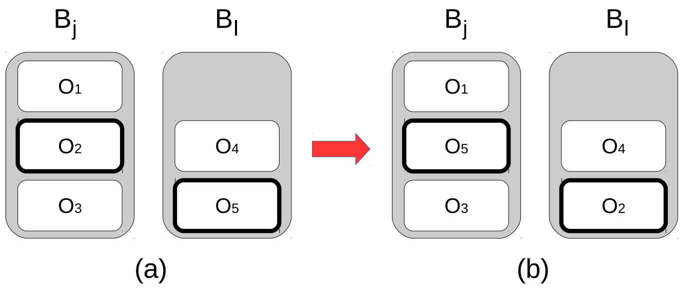


Fig. 6: Example of a *Swap* move.

Considering the *Swap* move previously introduced, we can define the neighborhood N_2 of a solution S as:

$$N_2(S) = \{S' \leftarrow Swap(S, O_i, B_j, O_k, B_l)\}$$

where $O_i \in B_j$, $O_k \in B_l$; $1 \leq i \leq n$, $1 \leq k \leq n$, being n the number of orders; $1 \leq j \leq m$, $1 \leq l \leq m$, and $j \neq k$, being m the number of batches.

C. Shake strategy

The shake stage is usually introduced in VNS as an effective strategy to scape from a basin of attraction. Given a solution S, the shake procedure randomly generates a new solution S' in the k-th neighborhood (denoted as $N_k(S)$) by applying k moves. More precisely, $N_k(S)$ contains the set of solutions that can be reached by applying k consecutive moves.

In the OBSP, the move applied for the shake strategy is based on *Swap* moves. In particular the shake procedure applies a random *Swap* k consecutive times.

D. Improvement methods

Local search procedures are heuristic methods designed with the aim of reaching a local optimum. Starting from a given feasible solution, these procedures explore a determined neighborhood. In each iteration, the method replaces the current solution if a neighbor solution improves the objective function. The search ends when all neighbor solutions has an associated objective function worse than the current one (i.e., larger objective function value in a minimization problem). The end of the search means that a local optimum has been found.

There exists two classical strategies to explore a neighborhood: best improvement and first improvement. In the former, the associated neighborhood is completely explored by a fully deterministic procedure, performing the best associated move. In the latter, the first improvement strategy, the method tries to avoid the exploration of the whole neighborhood by performing the first improving move encountered during the exploration. In general, iterations performed in the first improvement strategy are more efficient than those in the best improvement one, since the former only evaluates a part of the neighborhood, while the latter explores it completely. Although the time needed to perform a best improvement strategy is usually larger, we can not ensure that the obtained improvement with this strategy is better than the one obtained with a first improvement strategy.

In this paper, we propose a Variable Neighborhood Descent method as the local search procedure. This VND method explores the two aforementioned neighborhoods (see Section III-B) with a first improvement strategy. In both neighborhoods, N_1 and N_2, only feasible moves that improve the value of the objective function of the solution are considered. Given the neighborhood N_1, we define a local search procedure LS1 as the one that explores N_1 using a first improvement strategy. This local search starts from a random point and performs $Insert$ moves until no further improvement is achieved. Similarly, we define LS2 as the local search procedure that explores N_2 also using a first improvement strategy. Again, this local search starts from a random point, but in this case it performs $Swap$ moves until a local optimum is reached.

Once the two previous neighborhoods are defined, we propose the combination of both of them in a VND method. In particular, in Algorithm 2 we show the pseudocode of this procedure. This method receives as input parameters, the initial solution S and the value of k_{max} (in this case, $k_{max} = 2$). Notice that the neighborhoods are sorted in such a way that N_1 is explored first and N_2 is explored in second position. As it is common in the VNS community, neighborhoods are explored from the smallest and fastest to explore to the largest one. In this case, N_1 (based on insert moves) is usually shorter than N_2 (based on swap moves), since the number of batches is normally smaller than the number of orders, the order within the batch is not taken into account, and, additionally, only feasible moves are considered. This method finishes when there are not improvements in any of the neighborhoods.

Algorithm 2 VND algorithm.

```
1: procedure VND($S, k_{max}$)
2:     $k \leftarrow 1$
3:     repeat
4:         $S' \leftarrow ChooseBestNeighbor(S, k)$
5:         $NeighborhoodChange(S, S', k)$
6:     until $k > k_{max}$
7:     return $S$
8: end procedure
```

IV. COMPUTATIONAL EXPERIMENTS

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 algorithm identified in the state of the art (Henn and Schmid, 2013). We have implemented our algorithms in Java 7 and the experiments were run on an AMD Phenom II X6 1050T with 2.8 GHz and 4 GB of RAM with Ubuntu 14.04 64 bit OS. The section also inclues a detailed description of the instances used in the experimentation.

A. Instances

We have considered a set of instances previously used in the literature for this optimization problem. In particular we have selected the `Set HS` (Henn and Schmid, 2013). This set contains 4800 instances whose main features are described below. In order to have an easier understanding of the instances, we have divided the description of them in three different parts: warehouse layout, item distribution, and customer orders.

- *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), while the width is 1.5 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 LU to move from the current aisle to the next one. The depot is located 1.5 LU 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 contain 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.
- *Customer orders*: This set of instances considers 4 different sizes of orders ($n = \{20, 40, 60, 80\}$), 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 45 and 75.

The time needed by an order picker to collect all the items in a batch is the sum of three times: the time needed to walk through the warehouse (collecting time), the time spent picking the items (picking time) and the time required to process the batch at depot (depot time). The collecting time equals to the total distance gone through divided into the move speed of 0.48 LU/second. The picking time is 6 seconds per item and, the depot time is 3 minutes, no matter the amount of orders in the batch. These parameters are set according to the configuration of the instances.

This set of instances also considers the *Modified Traffic Congestion Rate* (MTCR) parameter. High values of MTCR mean that the due dates of the orders are very close one to each other, while low values of MTCR mean that the due dates are more scattered. The MTCR values considered are 0.50, 0.55, 0.60, 0.65, 0.70 and 0.75.

We have experimentally tested that it is not necessary to use the whole set of instances, since a small representative subset can produce similar results. Therefore, we have selected a subset of 96 instances from the previously described `Set HS`. In particular, we have selected one instance per type in order to perform our experiments.

B. Preliminary experiments

In this section we adjust the parameters of our algorithm (value of k_{max}, local search to use, and selection of the routing algorithm) by conducting several preliminary experiments with 12 out of the 96 instances selected.

We first compare the performance of the VND (see Section III-D), with respect to the local search procedures in isolation (LS1 and LS2). In order to perform this comparison, we constructed a solution with the EDD and then we applied each improvement method (LS1, LS2 and VND) to the same initial solution.

In Tables I and II we summarize, for each improvement method, the average value of the objective function (*Tardiness*), the average execution time in seconds (*CPU t (s)*), the average deviation from the EDD constructed solution (*Dev. (%)*), the number of times the method has obtained the best solution (*#Best*) and the number of times the method has obtained a tardiness value of 0 (*#Zeros*). In Table I we coupled each algorithm with the S-Shape routing strategy and in Table II we coupled each algorithm with the Largest Gap routing strategy. Results in that tables confirm that the VND procedure compares favorably with respect to simple local search methods when using both: S-Shape and Largest Gap routing strategies.

Local search	**LS1**	**LS2**	**VND**
Tardiness	28462	20665	16603
CPU t (s)	<1	<1	<1
Dev. (%)	21.13%	31.76%	38.75%
#Best	3	4	11
#Zeros	1	1	1

TABLE I: Comparison of LS1, LS2 and VND with the S-Shape routing strategy.

Local search	**LS1**	**LS2**	**VND**
Tardiness	17012	17418	12040
CPU t (s)	<1	<1	<1
Dev. (%)	14.10%	17.96%	26.68%
#Best	3	4	9
#Zeros	1	1	1

TABLE II: Comparison of LS1, LS2 and VND with the Largest Gap routing strategy.

As we can see in the tables, VND achieves the maximum deviation from the EDD method in both, the S-Shape and the Largest Gap comparisons. In particular, VND obtained 38.75% in the S-Shape comparison and 26.68% in the Largest Gap one, meanwhile LS1 obtained 21.13% and 14.10%; and LS2 obtained 31.76% and 17.96%, in the S-Shape and Largest Gap comparisons, respectively. If we pay attention to the number of best solutions found, again VND also achieves the maximum number of best solutions found in the experiment (11 versus 3 and 4 in the S-Shape comparison and 9 versus 3 and 4 in the Largest Gap comparison). It is worth mentioning that the results of LS2 in isolation seem to be consistently better than the ones by LS1. Therefore, we select VND as our improvement strategy for the next experiments.

If we compare the deviation with respect to the EDD solution obtained with S-Shape and Largest Gap, it seems that S-Shape is a better routing strategy. However, this is not necessarily true, since the deviation is calculated with respect to the EDD solution routed with the corresponding routing strategy. In fact, if we have a look to the average tardiness obtained by the methods, it is considerably better in the case of the Largest Gap.

The next experiment consists in determining the influence of the parameter k_{max} on the performance of GVNS, fixed the local search procedure as the VND method. The execution time in this next experiment depends on the number of orders (100 ms $\times$ number of orders in the instance) and the results are compared with respect to the EDD method. In particular, we consider $k_{max} = \{10, 15, 20\}$. In Table III we show the associated results for each routing strategy. We present the same rows as the ones in Table I. For each routing strategy, the best k_{max} value is 15 since it obtains the best value for the deviation from the EDD, the minimum tardiness value and the maximum number of best solutions found. It also achieves the highest value for #Zeros for both routing strategies.

Routing	**S-Shape**			**Largest Gap**		
$\mathbf{k_{max}}$	**10**	**15**	**20**	**10**	**15**	**20**
Tardiness	11590	11383	11672	9895	10032	10116
CPU t (s)	3.36	3.37	3.42	3.30	3.32	3.36
Dev. (%)	53.72%	53.82%	53.70%	47.99%	49.43%	49.36%
#Best	9	11	9	10	10	10
#Zeros	1	1	1	2	3	3

TABLE III: Influence of k_{max} on the performance of the GVNS.

C. Final experiments

Once we have identified the best parameters and strategies among our proposed variants, the final experiment is intended to compare the performance of our best proposal with the best previous approach in the state of the art. In particular, we have selected VND as improvement strategy and k_{max} = 15 to compose our best algorithm. We have conformed a GVNS method with the previous parameters and we have compared it with the ILS algorithm described in Henn and Schmid (2013) in both contexts: S-Shape and Largest Gap routing strategies.

For this final comparison we considered the full set of 96 instances selected. The algorithms ran for the same time, which is set by the ILS algorithm with a maximum time of 300 seconds per instance.

In tables IV and V we report the results obtained by the ILS algorithm compared to our proposed GVNS over the whole set of instances. In Table IV we refer to the results obtained using the S-Shape routing strategy and in Table V we refer to the results obtained using the Largest Gap routing strategy. In these tables we present the same rows as in previous tables.

The first conclusion that we can extract by analyzing these results is that the new proposed procedure outperforms the previous method in the state of the art. This experiment shows that for the algorithms running with the S-Shape routing strategy there exist no significant differences. Although ILS performs better in terms of deviation from the EDD (53.50%) than the GVNS (53.13%), our proposed algorithm obtains a higher number of best solutions (75, while ILS obtains 74). In order to confirm the previous observations, we have performed a test of Wilcoxon. The obtained p-value of 0.981 indicates that there are not significant differences between both methods.

Algorithm	**ILS**	**GVNS**
Tardiness	15654	15697
CPU t (s)	9.26	8.60
Dev. (%)	53.50%	53.13%
#Best	74	75
#Zeros	11	11

TABLE IV: Best methods on `set HS` with the S-Shape routing strategy.

Having a look at Table V, it seems that in this case the differences between the methods are larger. The deviation from the EDD achieved by the GVNS (50.00%) is better than the one achieved by the ILS (49.44%). Additionally, GVNS was able to achieve 86 best values while ILS was only able to achieve 69. Again, in order to confirm our observation, we performed a test of Wilcoxon. The obtained p-value of 0.001 indicates that there are not significant differences between the algorithms.

Algorithm	**ILS**	**GVNS**
Tardiness	13952	13622
CPU t (s)	111.21	97.31
Dev. (%)	49.44%	50.00%
#Best	69	86
#Zeros	14	14

TABLE V: Best methods on `set HS` with the Largest Gap routing strategy.

In Tables VI and VII we present the results divided by groups depending on the MTCR parameter. In particular, we report the average deviation with respect to the EDD method for each algorithm using the S-Shape routing strategy (Table VI) and Largest Gap routing strategy (Table VII). From this perspective, it is interesting to point out that the performance increase of ILS and GVNS with respect to EDD is higher when the MTCR grows up.

MTCR	**ILS**	**GVNS**
0.50	38.36%	35.95%
0.55	36.27%	36.48%
0.60	66.85%	67.30%
0.65	60.03%	60.09%
0.70	57.37%	57.31%
0.75	62.14%	61.83%

TABLE VI: Average deviation with respect to the EDD considering the S-Shape routing strategy.

Despite of the results shown in Tables IV and V, it is fair to say that the Largest Gap routing strategy is the best routing algorithm for this set of instances. This is shown in Table VIII, where data from Table IV and Table V is mixed. In this table we compare the four previous algorithms (ILS+SS and GVNS+SS for the algorithms that use the S-Shape method, ILS+LG and GVNS+LG for the algorithms that use the Largest Gap method). In this case we do not report the deviation

MTCR	ILS	GVNS
0.50	38.96%	39.08%
0.55	33.38%	34.67%
0.60	60.97%	61.21%
0.65	57.52%	57.65%
0.70	51.34%	52.39%
0.75	54.46%	55.00%

TABLE VII: Average deviation with respect to the EDD considering the Largest Gap routing strategy.

against EDD since the comparison point is different in the case of S-Shape and Largest Gap. However, instead of that value we reported the deviation against the best found solution (*Dev. Best (%)*) by any of the methods (notice that the higher the deviation, the worse the algorithm). When the objective function value is zero, the deviation can not be reported, so we also reported the number optimum values found (#Zeros). Results in Table VIII help us to confirm that the proposed algorithm (GVNS) using the Largest Gap routing strategy is the best strategy since it achieves the lowest deviation with respect to the best value of the experiment (1.25%) and the highest number of best values found (79 times out of 96 instances).

Algorithm	ILS+SS	GVNS+SS	ILS+LG	GVNS+LG
Tardiness	15654	15697	13952	13622
Dev. Best (%)	44.75	56.35	3.40	1.25
#Best	18	16	64	79
#Zeros	11	11	14	14

TABLE VIII: Comparison of the best four methods considering the deviarion with respect to the best solution found.

V. Conclusions

In this paper we present a General Variable Neighborhood Search (GVNS) algorithm to tackle the Order Batching and Sequencing Problem (OBSP). Specifically, we make use of the well-known constructive method Earliest Due Date (EDD). The GVNS explores two different neighborhoods coupled in a Variable Neighborhood Descent method. The algorithm was paired with two well-known routing methods: S-Shape and Largest Gap. After testing the best configurations for our method, we compared it with the best previous algorithm in the state of the art, over a set of instances referred in the literature. This comparison was statistically supported, and it favors our proposal.

VI. Acknowledgements

This research has been partially supported by the Spanish Ministry of "Economía y Competitividad", and "Comunidad de Madrid" with grants ref. TIN2012-35632-C02 and S2013/ICE-2894. The authors thank Prof. Henn for sharing with them his results, code and instances.

References

M. Albareda-Sambola, A. Alonso-Ayuso, E. Molina, and C.S. De Blas. Variable neighborhood search for order batching in a warehouse. *Asia-Pacific Journal of Operational Research*, 26(05), 2009.

T-L. Chen, C-Y. Cheng, Y-Y. Chen, and L-K. Chan. An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, 159:158–167, 2015.

R. de Koster, K.J. Roodbergen, and R. van Voorden. Reduction of walking time in the distribution center of de bijenkorf. In *New trends in distribution logistics*, pages 215–234. Springer, 1999.

J. Drury. Towards more efficient order picking. *IMM monograph*, 1, 1988.

A. Duarte, J.J. Pantrigo, E.G. Pardo, and J. Sánchez-Oro. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. *IMA Journal of Management Mathematics*, In press, 2013.

A. Duarte, J.J. Pantrigo, E. G. Pardo, and N. Mladenović. Multi-objective variable neighborhood search: an application to combinatorial optimization problems. *Journal of Global Optimization*, pages 1–22, 2014.

E. A. Elsayed, M-K. Lee, S. Kim, and E. Scherer. Sequencing and batching procedures for minimizing earliness and tardiness penalty of order retrievals. *The International Journal of Production Research*, 31(3):727–738, 1993.

E. G. Pardo, N. Mladenović, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5):2242 – 2252, 2013.

N. Gademann and S. Velde. Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE transactions*, 37(1):63–75, 2005.

P. Hansen, N. Mladenović, and J. A. Moreno. Variable neighbourhood search: methods and applications. *4OR*, 6 (4):319–360, 2008. ISSN 1619-4500. doi: 10.1007/s10288-008-0089-1.

S. Henn and V. Schmid. Metaheuristics for order batching and sequencing in manual order picking systems. *Computers & Industrial Engineering*, 66(2):338–351, 2013.

S. Henn and G. Wäscher. Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Operational Research*, 222(3):484–494, 2012.

S. Henn, S. Koch, K. Doerner, C. Strauss, and G. Wäscher. Metaheuristics for the order batching problem in manual order picking systems. *BuR Business Research Journal*, 3 (1), 2010.

A. H. Hossein, T. Shahrooz, G. Pezhman, M. Saman, M. Zameri, and K-Y. Wong. Order batching in warehouses by minimizing total tardiness: a hybrid approach of weighted association rule mining and genetic algorithms. *The Scientific World Journal*, 2013, 2013.

B. Menéndez, G. E. Pardo, A. Duarte, A. Alonso-Ayuso, and E. Molina. General variable neighborhood search applied to the picking process in a warehouse. *Electronic Notes in Discrete Mathematics*, 47:77–84, 2015.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

T. Öncan. Milp formulations and an iterated local search algorithm with tabu thresholding for the order batching problem. *Journal of Operational Research*, 2015.

H. D. Ratliff and A. S. Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521, 1983.