International Journal of Production Research
Vol. 00, No. 00, 00 Month 2012, 1–22

RESEARCH ARTICLE

On a selection and scheduling problem in automatic storage and retrieval warehouses

Antonio Alonso-Ayuso[a]*, Gregorio Tirado[b], Ángel Udías[a]

[a] *Universidad Rey Juan Carlos*; [b] *Universidad Complutense de Madrid*
(*Received 00 Month 200x; final version received 00 Month 200x*)

Warehousing is one of the main components of the supply chain and its optimization is crucial to achieve global efficiency. Warehouse operations involve receiving, shipping, storing and order picking, among others, and the coordinated optimization of all these different operations is highly complex. This paper approaches a selection and scheduling real problem that arises in an automatic storage/retrieval warehouse system involving the scheduling of forklifts pickup operations. The objective is to minimize the total loading time of the vehicles performing transportation, while respecting their departure due dates. This complex problem is approached through a two-phase decomposition method, combining both exact and heuristic procedures. The performance of the proposed solution method is evaluated through extensive computational results performed on several scenarios from a real case study built from data from a real mattress warehouse.

Keywords: logistics; scheduling; warehouse; order picking; heuristics.

1. Introduction

The area of logistics and transportation has undergone a great development during the last decades and it is one of the most important areas of concern for many companies. The new technologies becoming available introduce constant changes and induce new customer requirements, making it necessary to develop new adapted tools to deal with them and be able to compete in the market. Significant savings can be achieved if logistic processes are efficiently organized, and to achieve this all the components of the supply chain must be coordinated and optimized. Warehouses are one of these components and in fact warehousing management is among the most crucial factors to achieve efficiency (according to Lambert et al. 1998, there are more than 750,000 warehouse facilities all

*Corresponding author. Email: antonio.alonso@urjc.es

ISSN: 1478-6451 print/ISSN 1478-646X online
© 2012 Taylor & Francis
DOI: 10.1080/1478645YYxxxxxxx
http://www.informaworld.com

over the world). The organization of warehouse operations has been approached in the literature using several different strategies and has been receiving an increasing attention during the last years. Gu et al. (2007, 2010) present a comprehensive review of the state-of-art of warehouse research. Whereas the first paper focuses on warehouse operation problems related to the four major warehouse functions, i.e., receiving, storage, order picking, and shipping, the latter concentrates on warehouse design, performance evaluation, case studies, and computational support tools.

Among all warehouse operations, order picking –the process of retrieving articles from their storage locations in response to a specific customer request– is the most intensive labor activity in the warehouse, and a very capital-intensive operation in warehouses with automated systems (de Koster et al. 2007, Coyle et al. 1996, Tompkins et al. 2003, among others). These authors estimate that order picking can account for up to 65% of the operating costs of a warehouse, and thus their optimization is crucial to reduce costs. Several order picking methods have been proposed in the literature, as for instance single-order picking, batching and sort-after-pick, single-order picking with zoning and batching with zoning.

In most warehouses, the solution of the order picking optimization problem consists of generating an order picking sequence for each stack crane in order to reduce the total travel time. This is too complex to be solved to optimality in a reasonable time, and usual approaches apply shortest path algorithms (in terms of traveling time) that solve the TSP in polynomial time (see Ratliff and Rosenthal 1983 and Van deer Veen and Van Dal 1983, among others). Another possibility is to use simulation techniques, as in Chan and Chan (2011), where a simulation study of different storage and routing strategies to optimize picking operations is performed, or in Samaranayake et al (2011), where a numerical simulation is used to evaluate the performance of an integrated approach within the supply chain system of a global car company. Uncertainty in warehousing has also been considered in the literature, as in Kumar et al. (2011), where a warehouse scheduling problem with uncentainty regarding the demand of the customers is approached using a Fuzzy Artificial Immune System algorithm. However, the main data related to the problem approached in this paper is deterministic and known beforehand, and thus uncertainty is not considered.

Efficient organizations of the order picking operations frequently include order batching, i.e. the grouping of customer orders into picking orders (de Koster et al. 1999). For a recent review of order batching methods see, for example, Henn et al. (2011). Most approaches are focused on the minimization of picking times for a set of customer orders. Pan et al (2011) approach the coordination of batching and picking operations in a synchronized zone picker-to-part order picking system by using an analytical approximation model based on probability and queueing network theory. Chan and Cheng (2012) also solve a joint batch picking and picker routing problem, but now using metaheuristics. The authors apply a particle swarm optimization algorithm for the batching part and an ant colony optimization algorithm to design a good picking route for each batch. Öncan (2013), though, focuses only on the Order Batching Problem, in which the order picking routing policy is given. In this context, a genetic algorithm is used to find a good batching strategy for the customers.

However, nowadays on-time retrievals of customer orders have become important and then due dates for the customer orders appear. For instance, in distribution systems orders are typically carried to the customers by trucks, and thus the departure times of the trucks must be determined in order to ensure that the items are delivered in the required time period (Gademann et al. 2001). Furthermore, depending on the particular distances to the customers, different departure times arise, and the due dates for the customer orders originate from the corresponding departure times. Recently, Henn (2012) presents an algorithm for minimizing the total tardiness in picker-to-part warehouses taking into account these due-dates, and Mishra et al. (2011) use simulated annealing to minimize tardiness and the number of tardy jobs in a lot-size problem in warehousing, with the constraint of meeting all due dates dates while transferring the product from manufacturer to the warehouse and from the warehouse to the retailer.

The problem approached in this paper belongs to this family of order picking problems with due dates. We introduce a selection and scheduling real problem that arises in an automatic storage/retrieval warehouse system involving the scheduling of forklift pickup operations. The items stored in the warehouse are organized in cages that are managed by forklifts and classified by an automatic device. The decisions must be taken in two levels: firstly, a selection of some cages containing the items needed for satisfying the demand must be performed, and secondly the optimal sequence of extraction (picking) for classification is to be determined. The objective is to minimize the total loading time of the forklifts transporting the demanded items, that mainly depends on the transportation time from the shelves where the cages are stored to the classifying automatic machine that processes them. Moreover, there is a due time associated to each vehicle that must also be respected, making it not sufficient to only minimize the total loading time and increasing the complexity of the problem. However, the problem approached in this paper differs substantially from a TSP, since in each pick up operation the forklift only moves one cage simultaneously and the shortest path from each cage position to the input/output point is known a priori. In this case, the difficulty arises in the optimal selection of the cage from which each item reference is to be delivered to each specific destination. Since this item reference is also stored in many other cages in the warehouse, each of these cages could contain other daily demanded item references that could be demanded by vehicles with different departure deadlines. To the best of the authors' knowledge, a problem with the characteristics of the one approached in this paper has not been considered in the literature yet.

The remainder of this paper is organized as follows. Section 2 gives a detailed description of the problem. In Section 3 a two-phase formulation of the problem is presented, providing an exact method to solve phase 1 and a heuristic to solve phase 2. In Section 4 the considered case study is described in detail and the computational results obtained on this case study are discussed. Finally, Section 5 draws some conclusions from this work and raises some ideas for future research. Besides, the notation used is summarized in Appendix A and the pseudocode of the proposed solution method is provided in Appendix B.

2. Problem description

The problem presented in this paper is a selection and scheduling real problem in automatic storage/retrieval warehouse AS/RS systems. The warehouse works as a distribution center, it stores the products of the factory and sends the customers' orders by truck. The factory produces items with different characteristics and each item type is labeled with a different reference. Usually the quantity of each item reference in the warehouse is proportional to its demand, but there are situations in which this is not the case. The warehouse items are located in cages with limited capacity that can be partially filled or even empty. The warehouse is organized in parallel double-sided aisles with single deep racks that store the cages in different vertical levels that can be individually picked up by means of a forklift.

The picking process consists of picking up the cage from the rack and moving it to an input/output point (I/O point) were the retrieved cage is dropped off. In the I/O point, an automatic machine removes the demanded items and put them in the corresponding shipping bins. The not-demanded items are put again in the cage, that is refilled with new items from the factory and relocated in the warehouse. When a shipping bin is full, an automated guided vehicle (AGV) moves it from the I/O point to the dock, where a forklift truck place it inside the corresponding trucks. All forklift trucks are supposed to have the same maximum speed and the same picking capacity (number of cages that can be picked per time unit). The I/O point, with the automatic machine, is located near the dock bays. A limited number of trucks, that actually carry out the customer deliveries, can be loaded at the I/O point. See Figure 1 for a simple illustration of the warehouse.

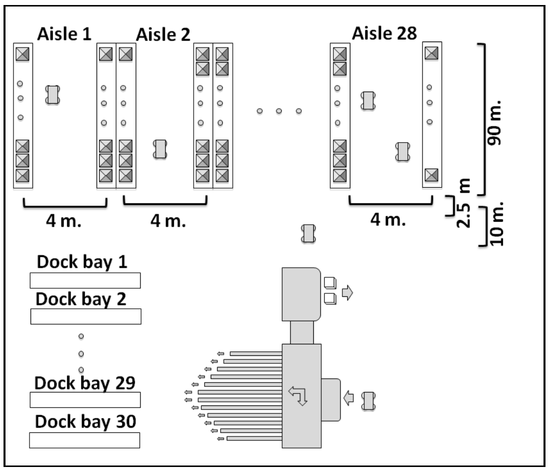


Figure 1.: Illustration of the warehouse

Every day, the warehouse has to deliver a set of orders that are known the day before. An order consists on a set of references and the amount of each reference that is demanded. Each order has a truck assigned. Depending on the destination, each truck has a departure deadline, in order to be able to arrive to its destination on time. A truck cannot depart until all its assigned items are loaded. Therefore it is necessary to plan and execute the loading procedures accurately in order to reduce the picking time and fulfill the delivery deadlines.

The cages are sequentially extracted from the warehouse to the I/O point and thus the extraction sequence will determine when the references are available at the I/O point. Besides, since one reference can be demanded by several trucks, when an item with a particular reference is to be managed at the I/O point it must also be decided to what truck it will be sent to, among all trucks demanding that reference. These decisions will determine when the last reference of each truck is extracted and thus when the tracks can departure. Note that each extracted cage may contain references demanded by different trucks, and thus a cage that is suitably sequenced for full-filling a truck could be a in a bad position for another truck.

It is also important to state clearly the differences between an item and a reference (also referred to as item reference). An item is a particular object in the warehouse, while a reference is a label given to an item to identify its type. Hence, there may be several items with the same reference. Note that one order consists of a set of references, not items, and thus there may be many different items in the warehouse that could be used to serve the demand of one particular order.

To sum up, the problem consists of choosing a set of cages from the warehouse containing enough references to meet all customers demands and designing an optimal cage picking sequence minimizing the time of the operation and respecting the due times of the trucks involved. In the next section we present a formulation of the problem and a solution method based on a 2-phase decomposition.

3. Formulation and solution method

Warehouse operations greatly enhance the productivity of the delivery process. However, there are some difficult problems that need to be solved, such as order sequencing, resource assignment for each line item, generating an order picking sequence for each crane or forklift, and replenishment planning. Usually, all tasks are strongly related and it would therefore be desirable to solve the four tasks together using a single model. However, such a model would be too complex, especially in real life applications, where large instances of the problems need to be solved quickly.

In the problem considered in this paper the order sequencing and resource assignment tasks are easily determined a priori, and so far the replenishment planning task is not a company concern, so order picking is the key issue to be optimized. Still, a model to generate an optimal order picking sequence for this problem is still too complex to be approached as a whole and to be solved to optimality in reasonable time, and thus it must be simplified to make it tractable. Fortunately, the two main decisions to be taken (cages to be extracted and sequence of extraction) can be easily separated, and then the problem naturally decomposes into two phases: in the first phase the cages to be extracted, containing enough items to meet the demand of each kind of product, are determined, and then in the second phase, once the set of cages to be picked is fixed, the optimal picking sequence is designed.

Just since an item reference can be requested in several destinations, even several times, it seems more interesting to solve firstly the cage selection problem that minimizes the total length of the routes to the I/O point, and afterwards the task sequencing problem for the selected cages and loading the trucks. The first phase is solved to optimality using an integer linear mathematical programming model and the second phase, that is more complex, is solved using heuristics. In what follows the two phases considered are detailed and the solution methods used are described. A schematic representation of the different alternatives available for each phase is presented at the end of this section (see Figure 4), and a detailed pseudocode of the complete solution method is provided in Appendix B.

3.1. *Phase 1: cage selection*

In this phase, we have to decide what cages are extracted from the warehouse so that the demand is met, considering the objective of minimizing the total travel time. We have a set $\mathcal{I}$ of cages and a set $\mathcal{K}$ of references. Cage i has u_{ik} units of reference k and needs a time t_i to be extracted, $\forall i \in \mathcal{I}, k \in \mathcal{K}$. This extracting time has two main components: the time to move cage i from the rack to the I/O point and the time of the cage exchange process. The objective is to select the cages to be extracted in order to meet the demand (say, d_k units for reference k, $\forall k \in \mathcal{K}$). This problem can be solved by using the following

0–1 Mathematical Programming Model, where the variables $x_i = 1$ if cage i is selected, and 0 otherwise.

The model formulation results:

$$\min \sum_{i \in \mathcal{I}} t_i x_i \tag{1}$$

$$\sum_{i \in \mathcal{I}} u_{ik} x_i \geq d_k \quad \forall k \in \mathcal{K} \tag{2}$$

$$x_i \in \{0,1\} \quad \forall i \in \mathcal{I} \tag{3}$$

Constraints (2) ensure that the quantity of items of each reference k in the extracted cages are sufficient to meet the demand. This model can be easily solved by using a plain solver as Cplex 12.3 or Gurobi in a few seconds. It produces a solution that provides the set of cages that are to be extracted to satisfy the demand, but not the sequence of extraction. The second phase of the algorithm is aimed to find a good extraction sequence allowing to assign a departure time for each truck that is compatible with its deadline.

In order to compare our proposal, we have considered two alternatives to obtain the initial set of cages: at random (select random cages until all demand is satisfied) or following a closest-first selection procedure (select cages with demanded references closest to the I/O point until all demand is satisfied). Note that the closest-first selection procedure is the most intuitive, and then, the most frequently used in real applications.

3.2. *Phase 2: cage picking sequencing*

Phase 1 provides a set of cages containing all demanded references and meeting the total demand, but it does not give any information about what cages to extract first. Once the initial set of cages is selected, phase 2 is aimed to find the optimal sequence that must be followed to extract those cages. At the beginning of the day, the orders of all trucks become open and they can start being loaded. Each truck has a latest departure time, determined to ensure that it arrives on time to its destination, but it is not allowed to depart until all its demanded items are received and thus the corresponding order is closed. The objective is to find a sequence of cages that allows each truck to depart on time and that minimizes the average loading time of the trucks (the average time the orders are open, or equivalently, the sum of loading times of all trucks).

Small changes in the extraction sequences may lead to important changes in the loading times. Let us illustrate with a small example how the loading times of the vehicles change when the extraction sequence is modified. Let us assume we have selected 4 cages, say c_1, c_2, c_3, c_4, whose extraction times are equal to 1 hour each, containing references $\{1,1,1,1,2,7\}$, $\{1,2,2,3,3,8\}$, $\{1,2,3,4,5,6,8\}$, $\{1,3,4,6,6,8\}$, respectively, and there are 3 vehicles, say v_1, v_2 and v_3, which demanded references are $\{1,1,1,1,2\}$, $\{2,2,3,3,3,7\}$ and $\{1,2,3,4,5\}$, respectively. Some possible extraction sequences are the following:

- Extraction sequence $E_1 = (c_1, c_2, c_3, c_4)$: v_1 departs in 1 hour, v_2 in 2 hours and v_3 in 4 hours.
- Extraction sequence $E_2 = (c_2, c_3, c_4, c_1)$: just moving c_1 to the end, the loading times of all vehicles are now 4 hours.
- Extraction sequence $E_3 = (c_2, c_1, c_3, c_4)$: swapping the first two cages in E_1, now v_1

departs in 2 hours, v_2 in 2 hours and v_3 in 4 hours.

- Extraction sequence $E_4 = (c_3, c_2, c_1, c_4)$: swapping the first and third cages in E_1, now v_1 departs in 3 hours, v_2 in 3 hours and v_3 in 4 hours.

Solving to optimality the cage picking sequencing problem considered in this phase is computationally highly demanding, being prohibitive for realistic instances, and for this reason it will be approached using heuristics. The heuristics described next, which can provide good solutions using small computing time, consist of a constructive stage (Section 3.2.1) to design an initial sequence and an improving stage (Section 3.2.2) based on a local search procedure.

3.2.1. Sequencing constructive stage

The first stage in phase 2, the sequencing constructive stage, is aimed to sequence the elements of the initial set of cages already built to meet the demands of all trucks. This can be done simply at random, following a closest-first ordering policy (cages closest to the I/O point are extracted first) or using a more elaborated sorting procedure based on the departure times of the vehicles. The first two sequencing methods are straightforward, but, since departure times of the vehicles do not give an order of the cages, the third rule requires the definition of a set of cage priorities based on the departure times of the vehicles, which is done as follows.

Since vehicles with earlier departure times must be served more urgently, they are assigned a higher priority, and as a result the priority assigned to each cage is based, not only on its extraction time, but also on the priorities of the vehicles that can be served by the items contained in it. This follows the idea that closest cages should be assigned to vehicles with higher priority, so that they can be loaded earlier.

Let p_j denote the priority assigned to vehicle j according to its departure time (note that $p_j < p_k$ means that truck j has more priority than truck k). The priority q_i assigned to each cage i is calculated following the Priority Assignment procedure described in Figure 2.

```
Priority Assignment procedure
    Set $q_i = 0$ for each cage $i$.
    for all cages $i$:
        for all demanded items $l$ of cage $i$:
            1. Set $q_i = q_i + \frac{1}{p_j^2}$, where $j$ is the vehicle with the highest
               priority demanding item $l$
            2. Decrease by one unit the demand of vehicle $j$ for item $l$
        end for
    end for
```

Figure 2.: Phase 2. Improving procedure

The Priority Assignment procedure described above provides two different methods to calculate priorities:

- If step 2 is skipped, we obtain what is called the *Static Priority Assignment method.* Here priorities are assigned to cages without updating the demands of the vehicles during the assignment process, that remain unchanged. As a consequence, this assignment is not dependent on the order in which cage priorities are calculated.

- If step 2 is performed, we obtain what is called the *Dynamic Priority Assignment method*. Here, every time the priority of a cage is modified, the demand of the corresponding vehicle is updated, assuming that the demand of the last considered item from the current cage is already fulfilled. Therefore, the priority assigned to a cage depends on the order in which cages are considered, i.e. the initial cage sequence used for the calculation of priorities. Two variants have been considered in the construction of the initial sequence when applying the Dynamic Priority Assignment method: the cages are initially ordered at random or in increasing order of extraction time (closest-first ordering).

Then, the method based on departure times consists on sorting the given cages according to the priorities q_i calculated following any of the two priority assignment procedures described above.

Note that, if the initial set of cages is built by solving to optimality the mathematical programming model of phase 1, it does not exist any sequence in which one of the cages has no demanded references when it is extracted to be processed, since in that case the initial set would not be optimal. However, if the initial cage set is built randomly or following a closest-first selection policy, there may exist sequences in which one cage (or more) has no demanded references at the time of extraction (because the demanded references of that cage were already served by other cages). In that case, those cages with no demanded references are directly removed from the sequence and are not used any more.

As it was already pointed out earlier, the way items arriving at the I/O point are distributed among the bins that are to be loaded into the trucks is a relevant decision to make. For this purpose, the trucks are first sorted (usually in increasing order of departure time or simply at random) and then, every time an item is ready to leave the I/O point, it is loaded into the bin assigned to the first truck demanding that item.

3.2.2. Sequencing improving stage

Once the constructive stage is finished a complete feasible sequence is obtained, that is used as a starting point of the improving stage. This stage consists on a local search procedure based on swapping the positions of two cages in the sequence with the objective of improving the solution quality (according both to the sum of loading times or departure times feasibility). This Improving procedure is detailed in Figure 3.

Note that swapping the positions of only two cages in the sequence may change the loading times of all vehicles with assigned items belonging to cages within those positions (see the example at the beginning of Section 3.2), and thus the recalculation of the objective function value after performing the swap move is highly time consuming. As a consequence, the amount of running time needed to test all possible swaps to be performed on any initial solution is computationally prohibitive and thus only a small subset of cages are selected to take part on the evaluated swaps.

To explain how the cages to be swapped are selected, let $\hat{\Lambda} = (c_1, \cdots, c_n)$ be the initial (sorted) solution sequence of cages, $\Lambda = \{c_1, \cdots, c_n\}$ the initial (unsorted) set of cages and m the number of vehicles. Any swap s is defined by a pair of cages $s = \{c_i, c_j\} \in \Lambda \times \Lambda$, indicating that swap s corresponds to swapping the positions of cages c_i and c_j in the solution sequence. To define the reduced candidate subset of swaps that will be actually considered, several subsets of promising cages will be constructed as follows.

A straightforward way to construct a reduced subset of cages consists in just selecting some of them randomly from Λ. With this purpose, let $\Omega^r \subseteq \Lambda$ be a set of $\lfloor rn \rfloor$ cages randomly chosen from Λ. A more elaborated method is based on choosing some cages

```
Improving procedure
    for (each selected swap) do
        if (departure times are met in current solution) then
            if (departure times are met after swap) and (objective function is improved)
            then
                perform swap.
            else,
                reject swap
            end if
        else
            if (loading times are reduced after swap) or (objective function is improved)
            then
                perform swap.
            else,
                reject swap
            end if
        end if
    end for
```

Figure 3.: Phase 2. Improving procedure

that could be particularly interesting to be swapped so that they could lead to larger improvements in the solution quality. For this purpose, let $\Omega_t^j \subseteq \Lambda$ be the set containing the last t cages in the solution sequence $\hat{\Lambda}$ with items to be loaded to vehicle j and let $\Omega_t = \bigcup_j \Omega_t^j$ be the union of all those sets for all vehicles. This subset Ω_t of cages is designed in this way because the last cages to be delivered to each vehicle are the ones determining their earliest departure times, and as a consequence those cages provide a greater potential of improvement if their positions in the sequence are modified.

Considering these subsets of cages introduced above, the reduced candidate subset of swaps, denoted by Φ, can be constructed in different ways. In this paper we consider the following possibilities:

- **I1**: Φ is randomly generated such that $\Phi \subseteq \Lambda \times \Lambda$ and $|\Phi| = v$. This means that v swaps are chosen randomly among all possible swaps involving any pair of cages.
- **I2**: $\Phi = \Omega_t \times \Lambda$. In this case the last t cages assigned to each vehicle are swapped with all other cages in the sequence. Note that since $|\Omega_t| \leq tm$ and $|\Lambda| = n$, it holds that $|\Phi| \leq tmn$. In general, it is an upper bound and not an equality because a given cage could be one of the last t cages assigned to several vehicles and in that case $|\Omega_t| < tm$.
- **I3**: $\Phi = \Omega_t \times \Omega^r$. In this case the last t cages assigned to each vehicle are swapped with r cages randomly selected from the sequence. Note that since $|\Omega_t| \leq tm$ and $|\Omega^r| = \lfloor rn \rfloor$, it holds that $|\Phi| \leq tm\lfloor rn \rfloor$. Again, for the same reason as before, this is in general an upper bound and not an equality.

The strategy I1, I2 or I3 that is chosen and the values of v, r and t determine the number of swaps that are finally considered in the improving phase. In Section 4 it will be shown what values of the parameters are used and how each strategy performs when applied to a case study.

3.3. *Schematic representation*

A schematic representation of the complete solution method, providing the different strategies to be used at each step, is given in Figure 4. As stated previously, first the set of cages to be extracted is selected, then an initial cage-extraction sequence is constructed, and finally that sequence is improved by applying local search.

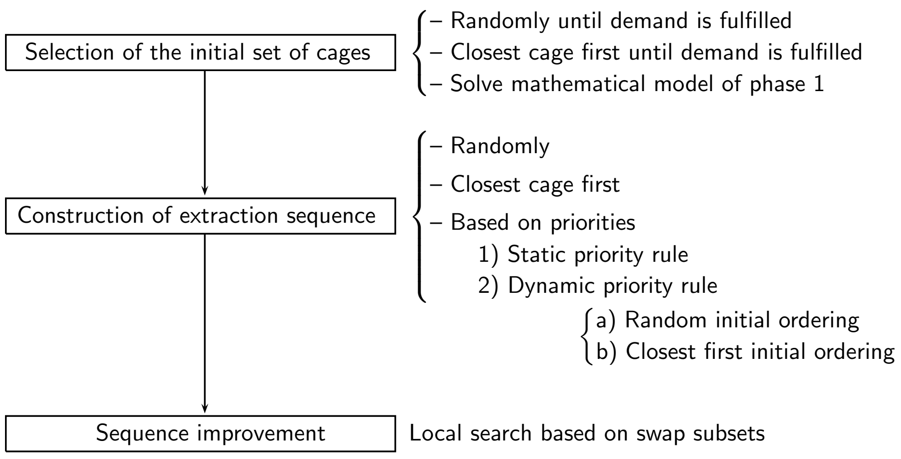

Selection of the initial set of cages
– Randomly until demand is fulfilled
– Closest cage first until demand is fulfilled
– Solve mathematical model of phase 1
Construction of extraction sequence
– Randomly
– Closest cage first
– Based on priorities
1) Static priority rule
2) Dynamic priority rule
a) Random initial ordering
b) Closest first initial ordering
Sequence improvement
Local search based on swap subsets

Figure 4.: Phase 2 solution method

4. Computational experience

4.1. *Case study*

In the case study considered in this paper the warehouse stores the items produced by the factory and deals with their organization and shipment to meet the requests of the customers. The main characteristics of this real mattress warehouse are listed in Table 1.

Aisle length	90 m
Centre distance between two aisles	4 m
Travel speed within aisles	2 m/s
Travel speed outside aisles	2 m/s
Additional time to enter or leave an aisle	0 s
Cage Capacity	8
Shipping Bin Capacity	8
Number of aisles	28
Number of cages per aisle	209
Number of cages in the warehouse	5852

Table 1.: Warehouse characteristics

The following assumptions have been made:

- All order information is known beforehand (references, departure times, etc.).
- The time required to pick a cage includes horizontal and vertical movement.
- Forklifts can only pick up one cage at the same time.
- The I/O point is fixed for all cases and central bottom located.
- The time required to relocate a cage to its original position is not considered.

Each item and storage location has a unique bar-code or chip to store the information of the storage location; however, at the moment, the storage process is continuously performed in a common random storage layout.

The warehouse is organized in 28 parallel double-sided 90-meter long aisles with single deep racks that contain 3 cages in different vertical levels that can be individually picked up by means of a forklift. The dock bays at the I/O point have capacity for 30 trucks.

On a daily basis, more than 4000 items must be processed and shipped from the warehouse to 30 different destinations (orders). The trucks that actually perform the deliveries have an homogeneous capacity of 154 items.

In our case study we have considered a total number of 46822 items in the warehouse with 2370 different item references, following an *ABC* distribution: on the one hand, there are many items (items of type A) corresponding to just a few references (typically up to 150 items for each of those few references), but on the other hand there are many references associated to very few items (items of type C, typically for more than half of the references there is only one single item available in the warehouse); the rest are items of type B.

For the generation of different realistic test instances based on this case study, three important factors have been considered:

- Customer Demand.
  - **Random (DR):** The probability of a reference being demanded is the same for all references, no matter how many items with that reference exist in the warehouse. Therefore, all references have the same probability of being demanded by a customer.
  - **ABC distribution (DABC):** The probability of a reference being demanded by a customer is proportional to the number of items of this reference in the warehouse.
- Storage Policy.
  - **Random (SR):** Cages are randomly distributed in the warehouse.
  - **ABC distribution (SABC):** The cages containing the most demanded references (A items) are stored close to the I/O point, whereas cages containing references with low demand (C items) are stored far away from it.
  - **Homogeneous cages closer (SHCC):** The more items with the same reference contained in a cage, the closest to the I/O point that cage is stored.
- Departure Time of Trucks.
  - **Discrete Increase (P1):** Trucks are divided into 4 groups (with 5, 10, 10 and 5 trucks) and the trucks belonging to the same group share the same departure time.
  - **Continuous Increase (P2):** The departure times are increased proportionally all over the day and only two trucks share each departure time.

For each level of the factors related to customer demand (2 possibilities) and storage policy (3 possibilities) we have generated 10 replicas, obtaining $(2 \times 10) \times (3 \times 10) = 600$ instances. Combining customer demand and storage policy we have $2 \times 3 = 6$ different scenarios (DR_SR, DR_SABC, DR_SHCC, DABC_SR, DABC_SABC, DABC_SHCC), and then we have 100 instances for each scenario. Besides, each of these instances is combined with the two departure times assignments considered, obtaining a total of $2 \times 600 = 1200$ instances (200 instances for each of the 6 available scenarios, 100 with P1 departure times and 100 with P2 departure times).

4.2. *Computational results*

In what follows the computational results obtained for the case study introduced above will be presented and analyzed. The mathematical model used in phase 1 has been solved using CPLEX 9 and the heuristics of phase 2 have been implemented in C#. All computations have been performed on an Intel Core2Duo 2GHz with 2Gb RAM.

Phase 1

First the results obtained after solving phase 1 for the 600 instances considered are presented. Please note that phase 1 is focused on the cage selection problem, minimizing the total extraction time while ensuring that the demand of all customers is served, and thus the departure times of the vehicles are not taken into account. As a consequence, the initial set of cages obtained for each P1 and P2 instance would be identical.

The main results concerning the set of cages to be extracted on each scenario are given in Table 2. In the first two columns the instance scenarios considered are specified. In columns 3 and 4 the largest and smallest total extraction times, respectively, required on the instances of each scenario are provided, while columns 5 and 6 show the average and the standard deviation of the extraction times, respectively. In column 7, the average number of cages to be extracted for each scenario is presented, and finally column 8 shows the average running time (in seconds) needed to solve phase 1 on each case.

Storage	Demand	MAX	MIN	Average	sd	# cages	Cpu time
SR	**DR**	127118	117694	122693	2139.45	975.40	1.80
SR	**DABC**	74858	67648	71544	1532.82	838.00	77.87
SABC	**DR**	157671	148788	154327	2042.33	967.90	1.95
SABC	**DABC**	90572	83512	87926	1700.62	812.50	71.15
SHCC	**DR**	180290	174247	177293	1411.74	973.57	2.12
SHCC	**DABC**	101987	92781	98804	2268.73	847.00	4.74

Table 2.: Results concerning the set of cages to be extracted after solving phase 1

It can be observed that in DR scenarios the number of cages to be extracted is around 13% larger than in the corresponding DABC scenarios, being the total extraction time around 41% larger. This result was expected since in DR scenarios any reference is equally likely to be demanded, and thus there may be demanded items for which very few available cages exist and they have to be extracted even in the case they are placed very far away from the I/O point. It is also important to note that the smallest average extraction times are obtained for the SR scenarios, showing that using storing policies SABC or SHCC do not improve extraction times with respect to the random storing policy. This may seem surprising at first glance, but it should not be: it indicates that most abundant references are not a problem, because there exist many alternatives for obtaining them; however, scarce references are more significant, because there are very few alternative cages containing them and they should not be stored far away from the I/O point.

Phase 2: Sequencing constructive stage

In what follows we present the results obtained by solving phase 2 for a random selection of 10 instances for each scenario considered and each departure time combination ($10 \times 6 \times 2 = 120$ instances in total). Only a fraction of all instances is considered because phase 2 is computationally much more demanding than phase 1 and it would take too long to solve all 1200 instances. However, we believe that the 120 instances selected are a representative sample and are enough to obtain illustrative results for the given scenarios.

The rules chosen for the design of the cage picking sequence in phase 2 are described in Table 3. These rules are referred to as R0,. . . , R9 (see column 1) and correspond to the possible combinations among the different methods available for the selection of the initial cage set (column 2), the cage sequencing method (columns 3), the cage priority assignment rule (column 4), and the initial cage ordering (in columns 5). The methods used by these rules were described in Section 3.2 and outlined and linked together in the diagram presented in Figure 4.

Rule	**Initial cage set**	**Cage Sequencing**	**Priority assignment rule**	**Init. cage ordering**
R0	Random	Random	–	–
R1	Closest-first	Closest-first	–	–
R2	Closest-first	Priority rule	Static	Closest-first
R3	Closest-first	Priority rule	Dynamic	Closest-first
R4	Phase 1	Random	–	–
R5	Phase 1	Closest-first	–	–
R6	Phase 1	Priority rule	Static	Random
R7	Phase 1	Priority rule	Static	Closest-first
R8	Phase 1	Priority rule	Dynamic	Random
R9	Phase 1	Priority rule	Dynamic	Closest-first

Table 3.: Sequencing rules description

Table 4 shows the results obtained after applying rules R0-R9, based on the Sequencing Constructive Stage described in Section 3.2.1, to the 60 P1 instances (10 for each scenario) considered. In line 3 the results concerning phase 1 (only extraction time) are given, while lines 4-13 show the results concerning total loading times obtained using rules R0 to R9, respectively. Columns labeled by **X** provide the average total loading times for each scenario and columns labeled by **sd** contain the corresponding standard deviations. Average values over all scenarios (and for each rule) are given in column 15, while average values over all rules (and for each scenario) are given in the last line. Finally, the total loading time percentage deviations from the rule showing the best performance, that is R9, are shown in the last column.

The main conclusion that can be drawn from Table 4 is that the rule showing the best overall performance is R9, providing the smallest total extraction time in all scenarios, although it is remarkable that rules R3 and R7 also provide good results. This can be seen more clearly in Table 5 that shows, for each rule and for each scenario, the percentage deviations in the average total loading times from the results obtained with rule R9. Each rule is associated with one column and each scenario with one line. Besides, as it was already shown in Table 2 for extraction times, in Table 4 it can also be observed that the average total loading time is higher in DR scenarios than in the DABC ones. This result was again expected, due to the fact that in DR scenarios there are more scarce references demanded than in ABDC ones and extracting them usually takes a longer time.

The results concerning total loading times obtained applying rules R0-R9 to P2 instances are very similar to the ones obtained for P1 instances, showing again that R9 is clearly the rule with the best overall performance, and thus they are omitted in the pa-

	SR_DR		**SR_DABC**		**SABC_DR**		**SABC_DABC**		**SHCC_DR**		**SHCC_DABC**		**Avg.**	**% R9**
	X	**sd**	**X**	**sd**	**X**	**sd**	**X**	**sd**	**X**	**sd**	**X**	**sd**		
FOF1	122750	2489	71576	1669	154512	1336	87974	1683	177375	1543	98572	2116	118793	
R0	5963154	115743	4565850	68847	7060953	66071	4959563	54998	7851935	64208	5301379	93756	5950472	111.0
R1	4699579	164021	2537644	75054	6242736	94126	3364238	98717	7256563	121945	3702989	81424	4633958	64.3
R4	3689521	178618	2118221	47409	4582118	43710	2730682	416550	5259149	47971	2923271	60736	3550494	25.9
R5	3656841	187756	2068819	53421	4560632	38441	2699033	408017	5244924	52602	2867133	63905	3516230	24.7
R3	3436954	96707	1842281	47426	4470780	80282	2341741	48504	5267854	115934	2518500	52888	3313018	17.5
R8	3698650	187437	2119789	51353	4594200	43106	2717901	395382	5267811	49335	2910989	56775	3551557	25.9
R9	2907244	131752	1642037	35922	3677172	46472	2126636	318003	4297588	47272	2269519	30990	2820033	0.0
R2	5350867	96970	3532496	65302	5128003	76911	3357852	85269	5903848	72618	3954314	116174	4537897	60.9
R6	3698862	181012	2117724	51956	4588214	37763	2712842	395013	5268383	44279	2922530	60635	3551426	25.9
R7	3416029	271493	1879389	49307	4204594	23939	2398615	367449	4813693	52901	2544260	39262	3209430	13.8
Avg.	3694586	146727	2226893	49788	4478538	50196	2681553	235417	5146284	60964	2910314	59878		

Table 4.: Constructive stage results with rules R0-R9 on P1 instances

	SR_DR	**SR_DABC**	**SABC_DR**	**SABC_DABC**	**SHCC_DR**	**SHCC_DABC**
R0	105.1	178.1	92.0	133.2	82.7	133.6
R1	61.7	54.5	69.8	58.2	68.9	63.2
R2	84.1	115.1	39.5	57.9	37.4	74.2
R3	18.2	12.2	21.6	10.1	22.6	11.0
R4	26.9	29.0	24.6	28.4	22.4	28.8
R5	25.8	26.0	24.0	26.9	22.0	26.3
R6	27.2	29.0	24.8	27.6	22.6	28.8
R7	17.5	14.5	14.3	12.8	12.0	12.1
R8	27.2	29.1	24.9	27.8	22.6	28.3

Table 5.: Percentage deviations in loading times from the best (rule R9) for P1 instances

per. However, as expected, the results concerning departure time violations are different in P1 and P2 instances. Table 6 shows, for each scenario and each rule, the average sum of times that vehicles exceed their deadlines and the average number of trucks violating their deadlines, both for P1 (Table 6a) and P2 instances (Table 6b). It can be observed that average time violations and number of trucks violating deadlines are quite higher in P1 instances than in P2 instances; these differences in the results regarding deadline violations are mainly due to the distribution of time limits for P1 and P2 instances, because P1 deadlines take only 4 possible values and there are large time jumps between ones and the others, while P2 deadlines are increased in a more continuous fashion and this helps in loading the trucks on time in an organized way. Again, it is remarkable that SR scenarios are harder to solve than DABC ones also according to the deadline satisfaction: for P2 instances, all rules but R0 and R2 find solutions satisfying all deadlines in DABC instances, while in DR scenarios only rule R9 (and R3 and R7 in some cases) is able to find solutions not violating any deadline.

Figure 5 illustrates more clearly the average departure deadlines compliance applying several rules to different scenarios with the two deadline configurations used. Figure 5a is related to P1 instances of scenario SHCC_DABC, while Figure 5b refers to P2 instances of scenario SHCC_DA. They compare the deadline of each truck (in white) with the average departing time in the solutions obtained using rules R0, R1 and R9 (in different grey colors). It can be observed that rules R0 and R1 (together with R4 and R5, that show a similar behavior but are not included in the graphic to avoid confusion) provide solutions with many trucks departing after their deadlines. In fact, in these solutions the departure times of all trucks lie in a very small time period, indicating that most of them are finished to be loaded at the same time and thus deadline information is not taken into account properly when making scheduling decisions. However, the figures also show that the solutions provided by rule R9 do verify all departure deadlines, and it can be clearly observed how the departure times of the trucks are different and tend to increase as the deadline also increases (see the border between the darkest area corresponding to R9 and the following one corresponding to R1 on both Figues 5a-b). Rules R8, R2 and R6 perform a bit better than R0-R5 thanks to the use of some Priority assignment procedure (Static or Dynamic), but are still clearly outperformed by rules R3 and R7, that show the best overall performance after R9 and are able to adjust loading times to departure deadlines quite well (again, they are not included in the graphic to avoid confusion). The explanation for this could be that R3 and R7 share several of the key features of R9, that seem to be the Closest-first initial ordering policy and the use of a Dynamic Priority Assignment procedure to perform the cage sequencing based on priorities.

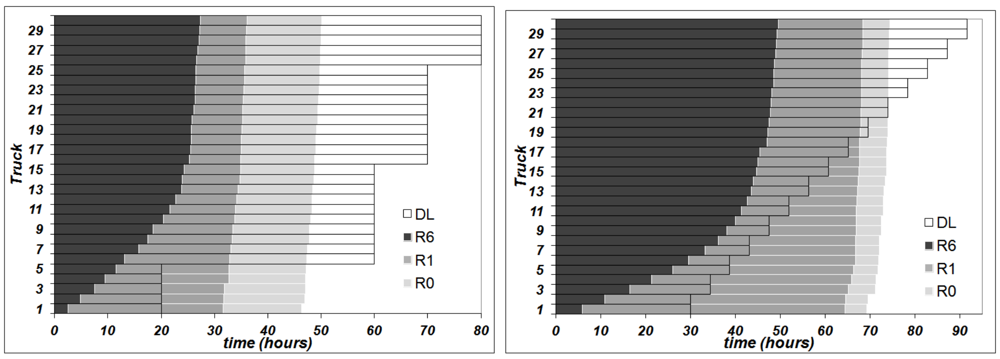
	SR_DR		**SR_DABC**		**SABC_DR**		**SABC_DABC**		**SHCC_DR**		**SHCC_DABC**	
	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks
R0	167.88	5.0	104.31	5.0	268.59	15.0	118.33	5.0	411.46	25.0	135.22	5.0
R1	103.28	5.0	7.41	4.3	175.14	5.0	43.77	5.0	293.91	15.5	57.03	5.0
R2	94.73	4.8	27.96	2.0	66.79	5.9	18.28	3.4	66.15	11.3	17.12	4.0
R3	8.28	2.2	0.00	0.0	26.97	3.5	0.00	0.0	52.30	4.1	0.00	0.0
R4	66.90	5.0	1.97	0.6	106.93	5.0	22.40	5.0	138.28	5.0	32.25	5.0
R5	61.82	5.0	51.92	0.4	93.92	4.5	18.13	5.0	135.65	5.0	26.88	5.0
R6	68.07	5.0	1.46	0.7	97.14	4.5	21.47	5.0	139.26	5.0	32.20	5.0
R7	13.19	2.8	0.00	0.0	28.47	2.7	1.21	0.3	45.47	4.0	0.00	0.0
R8	67.95	5.0	1.74	0.5	98.06	4.5	21.88	5.0	139.17	5.0	31.41	5.0
R9	0.00	0.0	0.00	0.0	1.35	0.9	0.00	0.0	7.67	2.0	0.00	0.0

(a) P1 instances

	SR_DR		**SR_DABC**		**SABC_DR**		**SABC_DABC**		**SHCC_DR**		**SHCC_DABC**	
	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks	time	# of trucks
R0	168.93	12.9	38.67	6.0	317.66	17.5	67.51	8.0	457.45	21.1	85.04	9.8
R1	46.74	7.4	0.00	0.0	192.56	14.0	0.12	0.2	339.72	18.2	1.66	1.1
R2	84.34	10.5	6.12	5.7	60.58	15.4	3.61	6.4	58.56	21.0	11.91	14.0
R3	0.00	0.0	0.00	0.0	1.76	1.3	0.00	0.0	20.86	7.1	0.00	0.0
R4	6.25	2.1	0.00	0.0	44.14	6.0	0.00	0.0	92.73	10.0	0.00	2.1
R5	2.83	1.9	0.00	0.0	39.63	6.0	0.00	0.0	88.77	9.8	0.00	0.0
R6	5.95	2.0	0.00	0.0	44.27	6.0	0.00	0.0	94.53	10.0	0.00	0.0
R7	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0	9.88	4.8	0.00	0.0
R8	6.46	2.0	0.00	0.0	44.35	6.0	0.00	0.0	94.32	10.0	0.00	0.0
R9	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0	0.00	0.0

(b) P2 instances

Table 6.: Time (in hours) that deadlines of trucks are violated



(a) P1 instances (b) P2 instances

Figure 5.: Illustration of departure time of trucks using R0, R1 and R9

Phase 2: Sequencing improving stage

Once a first feasible extraction sequence is constructed, the improving stage described in Section 3.2.2 is applied in order to reduce loading times and deadline violations. With the aim of considering only a small amount of promising swaps, the improving stage makes use of different reduced swaps subsets, obtained by combining different types of cages subsets (I1, I2, I3) of different sizes, as described in Section 3.2.2. The nine combinations tested in this experimentation are shown in Table 7 (where n is the number of cages selected to be extracted).

Swap subset	**Improving strategy**	**Parameter value**	**# swaps evaluated**
IS0	I1	$v = 5000$	5000
IS1	I2	$t = 1$	$30n$
IS2	I2	$t = 3$	$90n$
IS3	I2	$t = 4$	$120n$
IS4	I2	$t = 6$	$180n$
IS5	I3	$t = 1$, $r = 0.5$	$15n$
IS6	I3	$t = 3$, $r = 0.5$	$45n$
IS7	I3	$t = 4$, $r = 0.5$	$60n$
IS8	I3	$t = 6$, $r = 0.5$	$90n$

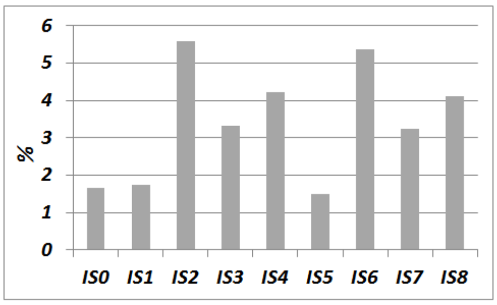
Table 7.: Reduced swap subsets description

The improving stage is the most running time demanding phase (it can take hours to be run) and thus the experimentation has been performed only on a small set of 7 instances of different types representing different scenarios that have been randomly selected from the original test bed. Table 8 shows the percentage of reduction in the total loading times obtained after applying the improving stage with the 9 reduced swap sets on the selected instances, starting from the initial solution obtained applying the constructive sequencing stage with R9 (that showed the best overall performance). It can be observed that reduced sets IS2 and IS6 provide the largest improvements in all considered instances, improving the total loading times an average of more than 5%. However, running times should also be taken into account when evaluating the performance of each reduced swap set selection. For this purpose, Figure 6 provides a comparison between the percentage of loading time improvements achieved by each strategy (Figure 6a) and the running time required to be executed (Figure 6b). These figures show that, although reduced sets IS2 and IS6 provide similar average improvements, IS6 requires half the running time than IS2, letting us conclude that IS6 shows the best overall performance.

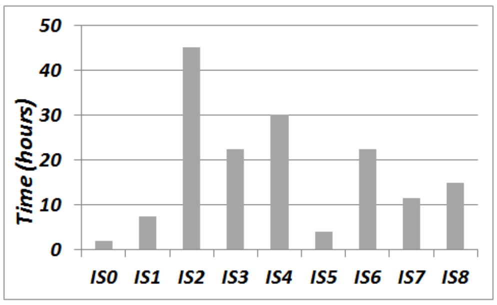
IS6 improving stage can be run in around 20 hours in a standard PC, and since orders are usually known every day 24 hours before the loading plan must be designed, it could actually be executed on a daily basis. However, if this time is excessive, reduced sets IS7 and IS8 provide improvements between 3-4% with running times between 10-15 hours, while reduced sets IS0 and IS5 provide improvements between 1-2% with running times between 2-4 hours. Therefore, the improving phase to be used should be chosen according to the running time available for computation.

	SR1 DR4	**SR8 RA6**	**SR0 DABC0**	**SABC2 DR4**	**SABC2 DR9**	**SHCC6 DR5**	**SHCC9 DR9**	**average**
IS0	2.06	1.54	1.94	1.28	1.63	1.70	1.40	1.65
IS1	2.06	1.69	2.25	0.91	1.92	1.87	1.41	1.73
IS2	5.46	4.98	6.06	5.54	6.45	5.54	4.98	5.57
IS3	2.60	3.44	4.41	2.71	3.40	4.00	2.74	3.33
IS4	3.93	3.70	4.91	3.95	4.43	4.60	3.98	4.21
IS5	1.43	1.49	1.81	0.68	1.99	1.70	1.28	1.48
IS6	4.63	4.25	6.74	4.70	6.90	4.65	5.75	5.37
IS7	2.22	3.15	4.95	1.89	3.86	3.20	3.37	3.23
IS8	3.42	3.34	5.28	2.90	5.49	3.76	4.60	4.11
average	3.09	3.06	4.26	2.73	4.01	3.45	3.28	

Table 8.: Improvement (%) over R9 for each reduced swap set



(a) Improvement (%)



(b) Computing times

Figure 6.: Comparison of improvement (%) and running time for different reduced swap sets

5. Conclusions and future work

This work approaches a typical logistics problem in many warehouses related to order picking, which is usually the most intensive labor activity in the warehouse. In this problem, decisions concerning what items should be extracted to serve customers demand, what extraction sequence should be followed to minimize loading times and what cages should be assigned to each truck to meet departure due times must be carefully planned and coordinated to achieve efficiency. Two main objectives are considered: reduce the total operational time to extract the cages and load the items in the trucks and meet the due times associated to the departures of those trucks. Given the huge number of items that must be processed in the warehouse every day, as well as the nature of the resulting problem, the application of exact algorithms in this context is computationally prohibitive. Therefore, in this paper we have focused on a heuristic solution approach based on a 2-phase decomposition of the problem, that is shown to be able to provide large improvements over the planning method that logistics companies usually use on their daily warehouse operations. The case study considered shows that significant reductions in operational time are achieved, together with a much better compliance with the departure deadlines of the trucks.

Several of the proposed improving mechanisms are able to improve both total operational time and departure deadline violations of the initial solutions built on the constructive stage; however, it is important to note that the computational complexity of some of those improving methods may be excessive to be run in a daily basis. Therefore, one interesting research line to be undertaken in the future deals with the development of this improving phase, exploring new subsets of exchanges to expand the local search process, designing more efficient techniques for the evaluation of candidate moves and considering alternative metaheuristic frameworks such as variable neighborhood search, tabu search or genetic algorithms (see for example Chan and Kumar 2009, where metaheuristics such as tabu search and simulated annealing are applied to other warehouse scheduling problems).

In the formulation of the problem approached, some assumptions had to be made in order to make it tractable, and their consideration could also lead to other interesting lines of future research. For instance, the proposed model considers that the forklifts can move freely through the warehouse and they cannot interfere with each other. However, two or more forklifts could meet at the same time traveling by the same aisle, causing a congestion and delaying the distribution of the cages. Besides, the location to where picked cages are moved back after being managed by the I/O point is not part of the problem approached, but a correct relocation of the extracted cages could facilitate significantly the picking operations of the following day. Furthermore, the way new items are placed in the relocated cages is also important for the plans to be followed during the following days, and that is also outside the scope of the problem considered in this paper.

Acknowledgements

This research has been partially supported by projects OPTIMOS2 MTM2009-14039-C06-03, TIN2009-07901 from the Spanish Ministry of Science and Innovation and RIESGOS-CM from Comunidad de Madrid (Spain).

References

[1] Chan FTS and HK Chan (2011). Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage. *Expert Systems With Applications*, 38(3), 2686-2700.

[2] Chan, LK and CY Cheng (2012). Joint order batching and picker routing using two-phased algorithm. *Proceedings of the Asia Pacific Industrial Engineering & Management Systems Conference*, V. Kachitvichyanukul, H.T. Luong, and R. Pitakaso Eds.

[3] Chan, FTS and V Kumar (2009). Hybrid TSSA algorithm-based approach to solve warehouse-scheduling problems. *International Journal of Production Research*, 47(4), 919-940.

[4] Chen, ZL (2009). Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58, 130-148.

[5] Coyle, JJ, EJ Bardi and CJ Langley (1996). *The management of Business Logistics*. South Western College Publishing.

[6] Dekker, R, R de Koster, KJ Roodbergen and H Van Kalleveen (2004). Improving order-picking response time at Ankor's warehouse. *Interfaces*, 34(4), 303-313.

[7] Gademann, N, J Berg and H Hoff (2001). An order batching algorithm for wave picking in a parallel-aisle warehouse. *IIE Transactions*, 33, 385-398.

[8] Gu, JX, M Goetschalckx and LF McGinnis (2010). Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203, 539-549.

[9] Gu, JX, M Goetschalckx and LF McGinnis (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 117, 1-21.

[10] Henn S (2012). Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. *Flexible Services and Manufacturing Journal* (in press). DOI 10.1007/s10696-012-9164-1.

[11] Henn S, S Koch and G Wascher (2011). Order Batching in Order Picking Warehouses: A Survey of Solution Approaches. Working Paper No. 01/2011. Faculty of

Economics and Manegement, Otto-von-Guericke-Universitat Magdeburg, Magdeburg, Germany.

[12] Hsieh, L-F and L Tsai (2006). The optimum design of a warehouse system on order picking efficiency. *International Journal of Advanced Manufacturing Technology*, 28, 626-637.

[13] De Koster, R, KJ Roodbergen and R Van Voorden (1999). Reduction of walking time in the distribution center of De Bijenkorf. In: Speranza MG, P Stähly (Eds.) *New Trends in Distribution Logistics*, 215-234. Springer, Berlin.

[14] De Koster, R, T Le-Duc and KF Roodbergen (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182, 481-501.

[15] Kumar V, N Mishra, FTS Chan, and A Verma (2011). Managing warehousing in an agile supply chain environment: an F-AIS algorithm based approach. *International Journal of Production Research*, 49(21), 6407-6426.

[16] Lambert, DM, JR Stock and LM Ellram (1998). *Fundamentals of Logistics Management.* McGraw-Hill, Singapore.

[17] Mishra N, V Kumar, N Kumar, M Kumar and MK Tiwari (2011). Addressing lot sizing and warehousing scheduling problem in manufacturing environment. *Expert Systems With Applications*, 38(9), 11751-11762.

[18] Öncan, T (2013). A Genetic Algorithm for the Order Batching Problem in Low-Level Picker-to-Part Warehouse Systems. *Proceedings of the International Multi-Conference of Engineers and Computer Scientists* Vol I, Hong Kong.

[19] Pan, L, JZ Huang and SCK Chu (2011). Order batching and picking in a synchronized zone order picking system. *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management* Singapore, 156-160.

[20] Petersen, C and R Schmenner (1999). An evaluation of routing and volume-based storage policies in an order picking operation. *Decision Science* 30(2), 481-501.

[21] Ratliff, HD and AS Rosenthal (1983). Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations Research*, 31(3), 507-521.

[22] Samaranayake P, T Laosirihongthong and FTS Chan (2011). Integration of manufacturing and distribution networks in a global car company - Network models and numerical simulation. *International Journal of Production Research*, 49(11), 3127-3149.

[23] Tompkins, JA, JA White, YA Bozer, E. Frazelle and JMA Tanchoco (2003). *Facilities Planning.* Wiley, New Jersey.

[24] Van der Veen, JAA and R van Dal (1983). Solvable Cases of the No-Wait Flow-Shop Scheduling Problem. *Operational Research Society*, 42(11), 971-980.

Appendix A. Notation

Warehouse	
TSP	Travelling Salesman Problem
AS/RS system	Automatic Storage/Retrieval System
I/O point	Input/Output point
AGV	Automated Guided Vehicle
item	any particular object of the wharehouse
reference	label to identify the type of an item
cage	set of items stored together in the warehouse
forklift truck	vehicles for the transportation of cages
Phase 1. Mathematical Model	
$\mathcal{I}$	set of cages in the warehouse
$\mathcal{K}$	set of references
$\mathcal{V}$	set of vehicles
u_{ik}	number of units of reference k stored on cage i
t_i	extraction time of cage i
d_k	demand of references of type k
x_i	1 if cage i is selected, 0 otherwise
Phase 2. Cage picking sequencing	
closest first policy	cages closest to the I/O point are picked first
p_j	priority of vehicle j
q_i	priority of cage i
Static Priority	demands are not updated while assigning priorities
Dynamic Priority	demands are updated while assigning priorities
$\Lambda = \{c_1, \cdots, c_n\}$	initial (unsorted) solution sequence of cages (selected in Phase 1, $\Lambda_s \subset \mathcal{I}$)
$\hat{\Lambda} = (c_1, \cdots, c_n)$	initial (sorted) solution sequence of cages
Ω^r	set of cages randomly chosen from Λ
$\Omega_t^j \subseteq \Lambda$	set containing the last t cages in $\hat{\Lambda}$ with items to be loaded to vehicle j, $\Omega_t = \sum_{j\in\mathcal{V}} \Omega_t^j$
Φ	reduced candidate subset of swaps
I1	swapping strategy: a few swaps are chosen at random ($\Phi = \Lambda \times \Lambda$)
I2	swapping strategy: last cages are swapped with all others ($\Phi = \Omega_t \times \Lambda$).
I3	swapping strategy: last cages are swapped with some of the others ($\Phi = \Omega_t \times \Omega^r$)
Computational experiments	
DR	uniform demand of references
DABC	demand proportional to the number of items for each reference
SR	random distribution of cages in the wharehouse
SABC	cages with most demanded references closer to the I/O point
SHCC	cages with homogeneous items closer to the I/O point
P1	discrete increase in departure times of vehicles
P2	continuous increase in departure times of vehicles
R0-R9	sequencing rules, described in Table 3
IS0-IS9	reduced swap subsets, described in Table 7

Appendix B. Pseudocode of the solution method

In what follows a pseudocode of the complete solution method proposed is presented. Note that the first stage of Phase 2 consists on creating an initial sequence of cages, $\hat{\Lambda}$. The pseudocode considers that $\hat{\Lambda}$ is built based on the priority rule. If the sequence is built randomly or using the closest cage first rule, step 2.2 can be skipped and step 2.3 must be properly adapted.

```
Phase 1. Determine $\Lambda \subset \mathcal{I}$, initial set of cages, by solving model (1)-(3).
Phase 2. Sequencing constructive stage:
    2.1. Assign priorities to vehicles (based on their departure times), $p_j, \forall j \in \mathcal{V}$
    2.2. Assign priorities to cages in $\Lambda$
        Set $q_i = 0$ for each cage $i$.
        For all cages $i$:
            For all demanded items $l$ of cage $i$:
                1. Set $q_i = q_i + \frac{1}{p_j^2}$, where $j$ is the vehicle with the highest
                   priority demanding item $l$
                2. Decrease by one unit the demand of vehicle $j$ for item $l$
            end for
        end for
    2.3 Build $\hat{\Lambda}$, the initial sequence of cages (based on their priorities).
    2.4 Improve the current sequence of cages $\hat{\Lambda}$
        For (each selected swap) do
            If (departure times are met in current solution) then
                If (departure times are met after swap) and (obj function is
                   improved)
                then
                    perform swap and update $\hat{\Lambda}$
                else,
                    reject swap
                end if
            else
                if (loading times are reduced after swap) or (obj function is
                  improved)
                then
                    perform swap and update $\hat{\Lambda}$
                else,
                    reject swap
                end if
            end if
        end for
    2.5 Stopping criteria
        If (maximum computing time is reached) or (no swaps were performed in the last
           iteration)
        then
            STOP. $\hat{\Lambda}$ is the proposed solution.
        else,
            repeat step 2.4
        end if
```