Stochastic premarshalling of block stacking warehouses

Stochastic premarshalling of block stacking warehouses

Vittorio Maniezzo[a,*], Marco A. Boschetti[b], Walter J. Gutjahr[c]

[a] *DISI, Department of Computer Science and Engineering, University of Bologna, 47521 Cesena, Italy*
[b] *Department of Mathematics, University of Bologna, 47521 Cesena, Italy*
[c] *Dept. of Statistics and Operations Research, University of Vienna, 1090 Vienna, Austria*

Abstract

Warehouse premarshalling (also pre-marshalling or remarshalling) is the activity of reordering items in a storage location so that subsequent retrieval orders can be serviced with little or no need for further relocations. It has deep impact on warehouse efficiency. We are interested in a stochastic case, where pickup orders become known only at the moment when they are to be retrieved. The problem is framed in a business analytics settings, where a forecasting statistical model based on historic data generates the input of a two stage stochastic optimization module. Computational results both on artificial and real-world data confirm the effectiveness of the approach.

Keywords: Warehouse premarshalling, business analytics, stochastic optimization

1. Introduction

Warehouses contain goods, which could be arranged in many different ways. The most widespread storage strategy is stacking [20], which is both straightforward and robust. No racking or storage facility is required for this system and it can be employed in any warehouse with wide floor space. When stacking, homogeneous boxes (also known as blocks, pallets, ...) are piled up on one another in stacks, where each level of the stack is called a

*Corresponding author, vittorio.maniezzo@unibo.it

tier (first tier boxes are on the floor, second tier ones lay on first tier boxes and so on). The number of tiers is determined by the physical characteristics of the warehouse equipment and of the boxes, such as their crushability, the stability of the obtained stacks, the clearance height of the warehouse, and so on. In case they become too high, stacks can possibly be divided in successive substacks. The essential feature is that each stack can only be accessed following a LIFO policy, i.e., only the last inserted box can be retrieved first. The stacks are then arranged in parallel within the warehouse. Floor stacking (also named block stacking) is a very common warehousing strategy, as it implies the least setup costs and permits a good storage density. Typically it is a primary choice for storing building materials, ceramic or roof tiles and in general non-crushable palletized products.

Big warehouses are often partitioned into *areas*, which could in turn be partitioned into *blocks*. The storage location of a box is therefore given by the area (block), stack, and tier in which it is placed [41]. Usually, bigger companies make use of a Warehouse Management System (WMS), where each box is represented by a Stock Keeping Unit (SKU), which represents the smallest unit of a product that can be retrieved or added to inventory. In the framework of this work, we assume that SKUs make direct reference to boxes, i.e., there is no further consolidation level where several SKUs are contained in one same box.

Operating stack-arranged warehouses involves several different activities, where the most sensitive one is order picking, reputedly the most labor-intensive and time-consuming process in warehouse logistics [50]. Order picking refers to the retrieval of one or several boxes from their storage locations [13], and storage location assignment influences almost all key warehouse performance indicators [40, 39]. Among the main competitiveness KPI of a warehouse are therefore the time for retrieving the boxes requested by the customers and the amount of work to complete this task [10], which is

fundamentally affected by the stacking configuration and warehouse layout [54, 45].

Unfortunately, the LIFO policy imposed by stacking possibly implies the need to relocate unrequested boxes before getting access to the one targeted by the picking. An optimized planning strategy is thus crucial to minimize the number of relocation movements and to achieve overall logistic efficiency [12]. Planning has the objective of storing the boxes in locations where they will be readily accessible for future retrieval, with little need of relocations. Great attention has been devoted to this topic by the optimization community, but the high level of unpredictability inherent in the supply chains precludes to reliably maintain an optimized stack layout [56]. To face this further problem, one common procedure is *premarshalling* (alternatively denoted as pre-marshalling or sometimes remarshalling), which has the objective of reorganizing the warehouse in order to reduce the total number of relocations later on. Premarshalling is usually carried out in low activity periods, for example during night shifts when underworked forces are available, in order to minimize labor requests in the high activity periods. Thus, the Premarshalling Problem (PMP) asks to find the minimum number of relocations necessary to rearrange the warehouse boxes so that subsequent picking will need no further relocations. The PMP is NP-hard [7].

An additional difficulty comes when the orders that should drive premarshalling are not known with sufficient advance. This happens both because of the intrinsic uncertainty related to logistic processes and because more and more warehouses accept late orders, which induces the need of workload forecasting [51]. This paper presents a solution facing this last use case, where premarshalling has to be driven by forecasts made on the basis of past order history.

1.1. Contributions

This work presents a full Business Analytics case study on an actual warehouse management problem, thus focuses on the synergies between operations research methods, statistics and economics, applied to the analysis of stochastic warehouse management. The method we propose leverages on the interplay of information services and stochastic methods to get prescriptive directives that help reducing the impact of uncertainty.

We apply our methodology to a very significant application case, i.e., warehouses managed using the most widespread storage strategy, and we apply original statistic models, devised for this specific case, and an optimization methodology which can be framed in the area of matheuristics [38]. The approach put forth is based on historical data, and first defines the probability distribution of the picking requests, later using it for premarshalling optimization. The computational results show that the need of item relocations during picking decreases by a significant percentage.

Finally, we point out that the method we propose can have an even greater impact in practice, as it is not limited to stacked warehouses, but can be also applied to automated warehouses.

1.2. Related work

Large part of the relevant literature does not directly address the warehouse premarshalling problem, but the closely related container yard premarshalling, which shares the same core problem. This last is a problem arising in port yard management, where containers are stacked, and asks to find the minimal sequence of container movements to rearrange containers in the stacks, so that the resulting configuration is compatible with the time each container must leave the port.

Another closely related problem whose literature can be partly exploited is the *block relocation problem*, which considers also the removal from the warehouse of the boxes in the picking list.

These problems, and specifically premarshalling, are enjoying an increasing interest in recent years, even though contributions in the literature are mostly dedicated to deterministic settings, for which a number of heuristic and exact approaches have been presented. A survey thereof was proposed by Lehnfeld and Knust [35].

Exact methods are based on different approaches, which include integer programming formulations, dynamic programming, constraint programming and A* adaptations.

Parreño-Torres et al. [42] recently presented different families of integer programming models of the PMP, which proved computationally very effective. Lee and Hsu [34] presented a mixed-integer linear programming formulation of the PMP based on a multi-commodity network flow model with side constraints, which permits them to compute lower bounds and to solve small instances to optimality. Exposito-Izquierdo, Melian-Batista and Moreno-Vega [16] were primarily interested in a heuristic method, but to assess its effectiveness they also designed an exact code based on A*. This way of approaching the problem has then been used also by Tierney, Pacino and Voß [48], who extend it to IDA* (Iterative Deepening A*) and include novel branching and symmetry breaking rules. Prandstetter [43] proposes a dynamic programming approach based on a DP formulation, later embedded into a branch-and-bound framework in order to increase the computational effectiveness. The final exact procedure is also converted into a heuristic by means of heuristic state equivalence determination. Rendl, Prandstetter [44] presented a first constraint programming model, which is computationally dominated by dynamic programming, and extend it to a robust variant of the problem. Finally, Zhang, Jiang and Yun [55] present a heuristic-guided branch-and-bound, which integrates in the B&B a guiding heuristic, which eliminates numerous branches prior to evaluating the lower bound at the corresponding nodes.

The state of the art of exact approaches is represented by the branch and bound of Tanaka and Tierney [46], further expanded in [47], where lower bounds, dominance rules, an iterative deepening branch and bound search, a specific branching heuristic, and a greedy partial solution completion heuristic contribute to effectively solving testsets from literature, albeit much smaller than those presented in the present paper.

Another noteworthy contribution was made by de Melo et al. [14], who devised a unified integer programming model capable of representing both the PMP and the block relocation problem, permitting to obtain good solution performance on instances of both.

On the heuristic side, the range of different approaches is wide, too. Among the recent proposals, we find Caserta and Voß [8] who approached the PMP by a corridor method, which is an effective matheuristic approach [6]. Other methods that can be classified as a matheuristic were presented by Forster and Bortfeldt [5] for the closely related container relocation problem, and consisted in a heuristic tree search procedure based on a classification of possible moves and on a branching scheme using move sequences of promising single moves, and by Kim and Bae [30], who decomposed the PMP into a sequence of integer programming problems. More standard heuristics or metaheuristics were used by Lee and Chao [33] who relied on neighborhood search, by Huang and Lin [25], who used simple labeling procedures, by Gheith, Eltawil and Harraz [19], who implemented a genetic algorithm, and by Jovanovic, Tuba and Voß [27], who used multiple tailored heuristics. Recently, Wang et al. [52] further proposed a target-based heuristic of good performance.

An innovative approach has been proposed by Hottung et al. [23] with their Deep Learning Heuristic Tree, which uses deep neural networks to learn solution strategies and lower bounds through analyzing existing solutions of PMP instances. The networks are then integrated into a tree search

procedure to decide which branch to choose next and how to prune the search tree.

The only contribution we are aware of on a stochastic optimization model for a problem related to PMP was presented by Borjan et al. [4], who devised a mathematical model for a dynamic version of the container relocation problem, where stacking and retrieving periods overlap, and which considers uncertainty in container arrival and departure times. The problem details and the source of aleatority are different from the ones of interest to us, but the modeling approach is similar, in that the authors use a two stage stochastic optimization framework that differently from us relies on a model of the deterministic version of the problem.

Zhao and Goodchild [53] investigate the possible benefits of using (incomplete) information on truck arrival for relocating containers by yard cranes in order to quickly retrieve desired containers. There are some similarities to our model. The main differences are (i) that we assume that premarshalling (i.e., a more prolonged, connected series of re-locations) is done in preparation for expected demand, and (ii) that in our approach, a stochastic model is used to represent uncertain information on the set and temporal sequence of customer orders, whereas in [53], uncertainty concerns only the arrival sequence of trucks and is represented by partial information, an important special case being that one knows successively arriving groups of trucks but not the exact order of arrivals within each group.

Also Ku and Arthanari [32] are concerned with the goal to minimize necessary reshuffling moves in container terminals. They make a similar assumption as Zhao and Goodchild [53] by supposing that the container pickups (truck arrivals) can be grouped into "time windows", where the precedence relationship between pickups in the same group is unknown in advance. The relative retrieval order of containers with the same time window is assumed to be a random permutation. In combination with the

objective of minimizing the expected number of re-handles, this leads to a stochastic dynamic programming model that is solved by a search-based algorithm in a tree search space and by a heuristic technique. We also use a stochastic model, but as our focus is on premarshalling, it refers to the random pickup list rather than to the departure times.

As the two articles mentioned above, Galle et al. [17, 18] deal with the Container Relocation Problem (CRP) and focus therefore on the optimization of move sequences for the retrieval of a known set of items. Their stochastic extension of the CRP is a modification of the one presented in [53, 32], using again the assumption that the order of containers assigned to the same time window is a random permutation, but deviating from the previous models by assuming that this permutation will have already become known at the time when the retrieval of the corresponding containers starts. The authors derive bounds and develop exact and approximate solution algorithms for this model. A major difference to our work is again that in our problem of premarshalling type, already the set of items is uncertain (random).

Tierney and Voß [49] address the premarshalling problem by a robust optimization approach. Starting from the observation that exact retrieval times cannot be predicted, they investigate a version of the problem where each container has to be retrieved within a certain time interval. They solve their problem formulation to optimality and show that their method outperforms prior related approaches. While our present work also deals with premarshalling, we use the stochastic rather than the robust paradigm for addressing uncertainty, and we consider also the items to be retrieved as uncertain.

The same differences can also be stated with respect to the robust-optimization approach by Boge et al. [3]. They derive uncertainty sets from priority classes of items based on the swap-distance between permutations,

and define "robust" configurations by the constraint that under each scenario from the uncertainty set, there are no items blocking the retrieval of other items. They provide theoretical results, formulate mixed-integer programming optimization models, and report on computational experiments with benchmark instances from the literature.

Other related contributions are the robust optimization by Rendl and Prandtstetter [44], which solves by constraint programming a robust variant considering uncertainty on collecting times, and the simulated annealing of Kang, Ryu and Kim [28], which considers uncertainty on container weights and their effect on problem specific side constraints.

Finally, it is noteworthy to mention the works which try to estimate the optimal number of relocation moves in the PMP, which include the works of Kim [29], of Kang, Ryu and Kim [28] and of Kim and Hong [31] for the block relocation problem. All of these refer to the uncertainty of the resulting number of moves, not of the input data.

Contrary to this, the work of van Gils et al. [51] addresses from an application viewpoint the same uncertainty we were confronted with, in their case motivated by the observation that as warehouses accept late orders, the assumption of a completely known demand becomes questionable. The work presents a time series forecasting model to forecast the daily number of order lines, applied to a real life case study.

2. Business Analytics and Stochastic Model

We are interested in premarshalling optimization for stack-arranged warehouses, divided in blocks containing geometrically identical boxes. In the following we will focus on the independent reoptimization of each single block of the warehouse, that is, under the assumption that no movement is possible among different blocks. Furthermore, we assume that picking orders become known the moment they have to be satisfied, therefore pre-

marshalling optimization can only have a statistical basis.

Extensive historic data are usually available on the company WMS, including past picking lists, and these can be used to determine the order lines of the daily picking lists [11]. We envisage a solution architecture that is typical of *Business Analytics* applications, where "Business analytics (BA) refers to the skills, technologies, practices for continuous iterative exploration and investigation of past business performance to gain insight and drive business planning" [1]. In our case, data mining on historic data, on sufficiently long but significant time spans, can provide the past picking lists, therefore the picking frequency distribution over the currently available SKUs, where we remind that a SKU value is the accounting record of a physical box in storage.

The overall optimization method that we propose goes therefore through two stages.

The first stage solves a *premarshalling problem* implementing:

- a *statistical model*, which works on completely known past picking lists and identifies the probability distribution of SKU requests,
- an *optimization module* that works on the identified probability distribution functions and proposes a premarshalling scheme.

The second stage solves a *block relocation problem* induced by the next day picking list. We remark that in some actual cases, the picking list could become known only one box at a time, thus the overall problem could become a *multistage problem*, where the first stage is the premarshalling problem mentioned above, and each successive stage corresponds to a block relocation problem requiring to extract one single box. Notice that even the number of successive stages would not be known in advance, being a random variable itself.

The paper contributes a method only for the first stage. The second stage

Table 1: Notation

symbol	referent
n	number of boxes (items)
B	index set of boxes, $B = \{b_i\}$
S	index set of stacks
SKU	index set of SKUs
L	picking list, a sequence of SKU indices
n_S	number of stacks
cap_k	height (capacity) of stack $k \in S$
$h(k)$	number of boxes in stack k
n_k	number of boxes in stack k (alternative notation)
B_k	index set of boxes in stack k
L_h	a lineitem in L, a SKU index
n_{SKU}	total number of SKUs
g_s	number of boxes with SKU index s
SKU_i	SKU of box i
L	picking list, list of SKU
pri_i	priority index of boxes according to picking list L
p_i	probability of SKU i to appear in the picking list L

in case of two stage or the further stages in case of multistage are solved by the basic heuristic in use in the actual warehouses, and their solution is of interest only to determine the quality of the premarshalled configuration. Given this, we will continue to refer only to the two stage case.

The elements of the first stage premarshalling problem are formally introduced in the following, and the used notation is listed in table 1.

The warehouse block to rearrange contains n boxes (items), let $B = \{1, \ldots, n\}$ be the index set of all boxes in the block. In the following we will use both notations b_i and i, when this is not ambiguous, to denote the i-th box in the block. A block can also be seen as a collection of *stacks*, i.e., piles of boxes, let $S = \{1, \ldots, n_S\}$ be the index set of the stacks. The *width* of the block is given by the number of stacks, n_S. Each stack is arranged in a number of *tiers*, also named *levels*, the topmost of which defines the *height* h_k of its stack k, $k \in S$. An arbitrary distribution of the n boxes to the stacks in S is called a *layout* or a $configuration$.

The set B of boxes can be partitioned in two different ways:

- each box is part of a stack, let B_k, $k = 1, \ldots, n_S$, be the index set of the boxes contained in the k-th stack. Clearly, $B = \bigcup_k B_k$.
- each box $i \in B$ belongs to a set SKU_s, $s = 1, \ldots, n_{SKU}$, which contains the indices of all boxes with identical content, and which correspond to one same SKU in the accounting system.

Customer orders are received daily as a picking list L of SKUs to be retrieved, thus, given a lineitem L_h, $L_h \in L$, any box i such that $i \in SKU_{L_h}$ could satisfy it. When multiple boxes with a same SKU are requested, we assume to find multiple repetitions of that SKU in the list. The ordering of the list is significant. Picking lists are completely known for past data and are progressively uncovered for the next day. Furthermore, we assume, as it is generally the case in real-world practice, that external considerations (such as for example storage age), differentiate all boxes from one another and thus impose an ordering on the suitability of each box to satisfy any specific customer order. Any lineitem specifying a SKU can thus be translated into a request of a specific box.

A picking list L induces an ordering among boxes. Each box i will have a *priority index* $pri_i \in \{1, \ldots, |L| + 1\}$, so that boxes to be retrieved first will have a higher *priority* than those to be retrieved later. Boxes not contained in the picking list will all have the lowest priority value, i.e., $|L|+1$. Notice that it is common practice in the literature to indicate through the priority index the order in which the boxes will have to be retrieved, therefore boxes with high priority will have a low priority index value, which could be confusing at first. After premarshalling, the boxes with higher priority have to be on the top of boxes with lower priority, in any stack.

Boxes can be *moved* among the stacks. A move can only shift one item from the top of a stack to the top of another one (*relocation*). Moves that

would pick an item from the top of a stack and bring it outside the warehouse (*removal*), when it is requested in a picking list, are not considered in the PMP and would turn it into the *block relocation problem*.

The PMP asks to find a move sequence of minimum length, i.e. with the smallest number of moves, such that in the final layout no box is stored in a stack, having a lower priority box over it.

Further assumptions usually accepted for premarshalling are:

- All the boxes in the block have the same dimensions.
- The position of each box in the block is known in advance.
- Relocation is allowed only to other stacks within the same block.
- Relocated boxes can be put only on top of other stacks, i.e., no rearrangement of boxes within a stack is allowed.

We make one further operational assumption, derived from the real-world cases that motivated our research, that whenever it is necessary to relocate a box i, it is always possible to find a stack k to move it to with sufficient capacity and containing only boxes with lower priority, i.e., $pri_j \geq pri_i$, $\forall j \in S_k$. This comes from the fact that daily picking lists are much shorter than the number of stacks of the warehouse, and the occupancy ratio is usually significantly lower than 100%, therefore there surely is at least one stack k which is not affected by picking, or that contains lower priority items. Such a stack k will be called a *feasible stack* for the relocation of box i. Note that this operational setting is not shared with the related area of container terminals, in which there are typically fewer stacks per block and where the number of items to move as a ratio to the total number of items in the yard is higher.

Furthermore, we mention that in warehouses such as those we are dealing with, there is usually space for opening a new stack, which is simply done

by laying a pallet on some empty floor. However, this possibility is seen as a last resort, as suitable free space for a new stack is probably far from the stacks the operator is working on.

Since we do not assume the availability of the incoming picking list L at the time of the premarshalling, the block has to be rearranged without a precise knowledge of the orders that will later be received. A precise definition of the picking order is therefore not possible, though we can still define priorities among boxes containing items with the same SKU, for which the picking order is dictated by external considerations, such as storage age. Upon designing a suitable statistical model of the picking process, it is however possible to identify the best fitting distribution for random variables associated to each SKU h in the WMS, which tells the expected number of boxes $\widetilde{n}_h$ of each particular $h \in SKU$ that will be requested on the following day.

The WMS contains in fact the past history of picking lists, and this provides the basis for remarshalling. For example, a possible historic picking list could be (3,8,1,3,4), asking first for a box with content with SKU 3, then SKU 8 and so on.

Figure 1 shows a sample instance, already used by [33] and [8]. The instance was proposed for the deterministic case, but in the context of this paper we interpret it as the configuration of a warehouse, consisting of 10 stacks, each with capacity 5. The number of boxes in stock is 35, yielding an occupancy ratio of 70%. Each box is specified by its contained SKU, so that, for example, there are 4 boxes containing SKU 1, 3 boxes containing SKU 2, etc.

When a list to service arrives, say (4,9,2,4,5), it will be necessary to determine which box must be retrieved for each requested SKU and to proceed with possible relocations. For example, the first SKU could be associated with the box at the fourth tier of the first stack, thus the topmost box of the

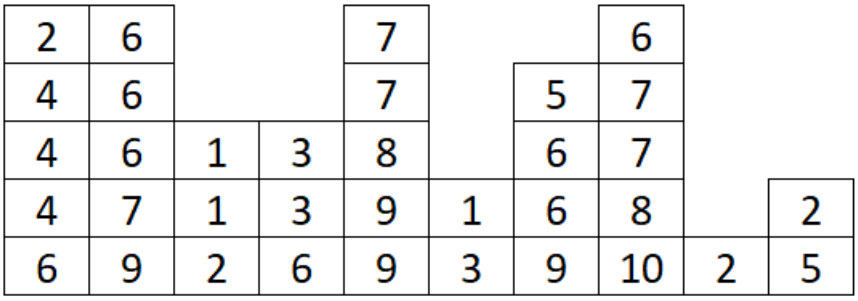

Figure 1: Storage configuration, instance adapted from Lee, Chao [33].

stack should be relocated. We remark that in our case, even after premarshalling, there could always be the need for relocations given the aleatory nature on the picking lists.

2.1. Mathematical model

The stochastic premarshalling problem problem can be modeled as a *two-stage stochastic programming problem*, where:

- *first stage decisions* are made on data known in advance, in our case these are the premarshalling moves acting on the known initial layout;
- a *random event*, in our case the arrival of a specific picking list L (one order at a time) imposes second stage decisions, as described in section 2;
- the *second stage decision*, or *recourse action*, copes with the actual case induced by the random event by handling the orders in the picking list, possibly further relocating some boxes.

As in all two stage processes, the cost is given by the sum of the costs induced by the two successive decisions. However, in our case, the first-stage decision, whose cost function is associated with the number of relocation moves, dictates activities to underwork staff. The first-stage cost term is therefore 0, subject to a constraint that the total number of relocation moves

must not exceed a threshold. This limit is determined by the duration of the underwork interval. Clearly, even if no direct cost is faced, it makes sense anyway to premarshal the warehouse by as few moves as possible. We can therefore consider a first stage cost which is lexicographically dominated by the second stage costs.

The relevant cost is in fact incurred in the second stage, which contributes to the objective function by the expected value of the handling cost, i.e., the expected number of relocation moves needed to pick all the boxes in L.

In the model, first stage variables, which are decided *before* the actual realization of the random parameters, correspond to premarshalling choices. Then, the random event (the disclosure of the picking list) occurs, after which the second-stage decision on the handling of the actual orders is made.

In the case of our problem, a full formulation of the two stages quickly becomes awkward because of the complex structure of the several problem elements involved.

For the first stage, we can refer to the time indexed multicommodity flow formulation with side constraints proposed by Lee and Hsu [34]. The formulation minimizes the total number of moves during premarshalling of a container yard, but at this level of abstraction the problem is the same as that arising in the warehouse. The formulation is linear, and makes use of 5 sets of 0-1 decision variables, which could correspond to our x variables. It is convenient for our application that the formulation requires to specify the set of allowed time intervals, in our case the maximum allowed number of moves.

The second stage is based on the solution of a block relocation problem. A mixed-integer linear formulation for this problem was presented for example by Exposito-Izquierdo et al. [16], again using 0-1 variables, which could correspond to our y ones.

The combination of the formulations for the two stages could result in a mathematical programming feasible way to approach the problem, but we deem it impractical, given the complexity of the models and the size of instances we aim to solve.

3. Statistical Model

The method for solving the PMP described in section 2 cannot currently be based on mathematical programming results. To get grounded directives we revert to a statistical guidance, based on WMS historic data. We study a statistical model of the PMP process of interest and obtain the expected cost of the recourse action for any given configuration. This is later used as a quality measure of each configuration we produce during the rearrangement process. By the explicit computation of the expected recourse cost, we avoid the necessity of resorting to a time-consuming combination of optimization with sampling, as it is frequently required for the solution of rich stochastic optimization problems (cf. [2, 21]).

To obtain the expected cost, we must fit the appropriate statistical distribution on historic pickup frequency data. If we consider day-long intervals, pickings could be modeled by a Poisson distribution: the number of times a specific SKU is picked in a day is discrete, pickings of different SKU are often independent of one another (for a discussion on this, see [24]), the rate at which picking orders are issued is constant, thus the probability of a picking in an interval is proportional to the length of the interval. To get to this, we first propose a Bernoulli model of the process, and then generalize it to a Poisson model, under the assumption of independence of SKU requests in the picking lists. This last assumption seems to be satisfied in all our study cases, but can be relaxed at the cost of adding some more complexity to the model.

A further assumption is that there is always room for relocating boxes,

i.e., the warehouse is never so fully occupied that there is no stack where to reposition a box which has to be relocated.

3.1. Bernoulli model

It is well known that a Poisson distribution is actually a limiting case of a Bernoulli process when the number of trials gets very large and the probability of success is small. In our case, this corresponds to saying that the picking probabilities should be defined for requests of each specific SKU (which could be modeled as a Bernoulli process), but with reference to time intervals much shorter than a day, thus inducing small success probabilities.

Initially, we work on a model according to the following assumptions:

- The occurrence of each item i ($i = 1, \ldots, n$) in the actual picking list L is Bernoulli-distributed with parameter p_i: item i occurs in the list with probability p_i. The p_i can easily be estimated from the historical data. Different items are assumed to occur independently from each other.

- The current configuration of the block is described as follows: the block, consisting of n_S stacks and containing n items, each occurring exactly once, has a completely known configuration, described by the numbers

$$a(j,k) = \begin{cases} \text{index of item on level } j \text{ in stack } k, & \text{if level } j \text{ is occupied} \\ 0 & \text{otherwise.} \end{cases}$$

  Therein, $1 \leq j \leq cap_k$ and $1 \leq k \leq n_S$. Clearly, in any feasible configuration we cannot have $a(j,k) > 0$ if $a(j-1,k) = 0$.

- The recourse action, i.e., the handling procedure for processing the picking list L is as follows: Stacks 1 to n_S are processed one after the other. From stack k, those items that belong to L are removed, and those items that have to be relocated during this procedure because

they lie above items from L although they do not belong to L themselves, are temporarily moved to another stack. The latter items are moved back to stack k again after the deepest item belonging to L has been removed.

3.1.1. Expected Cost of Recourse Action

Define for each $k = 1, \ldots, n_S$:

- the height of stack k,

$$h(k) = \max\{j : a(j,k) \neq 0\},$$

- the level of the deepest required item in stack k,

$$d(k) = \min\{j : a(j,k) \in L\},$$

  If no box of the stack is required, $d(k) = 0$.

- the number of required items in stack k,

$$r(k) = |\{j : a(j,k) \in L\}|.$$

Then it follows immediately from the description of the handling procedure that for fixed list L, one needs

$$2(h(k) - d(k) + 1) - r(k) \tag{1}$$

moves to properly rearrange stack k ($k = 1, \ldots, n_S$), reinserting in it the boxes that were relocated in other stacks.

For a given, fixed, configuration $a = [a(j,k)]$, the numbers $h(k)$ are constants. Let

$$X_{jk} = \begin{cases} 1, & \text{if } a(j,k) \in L, \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

for $1 \leq j \leq cap_k$, $1 \leq k \leq n_S$, and let

$$Y_{jk} = \begin{cases} 1, & \text{if } X_{j'k} = 0, \quad \forall j' \leq j, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

for $1 \leq j \leq cap_k$, $1 \leq k \leq n_S$.

Then, the number of required items in stack k is

$$r(k) = \sum_{j=1}^{h(k)} X_{jk} \quad (k = 1, \ldots, n_S) \tag{4}$$

and the lowest tier of the stack containing a required item is

$$d(k) = \sum_{j=1}^{h(k)} Y_{jk} + 1 \quad (k = 1, \ldots, n_S). \tag{5}$$

In the special case where stack k does not contain any element of L, formula (5) produces the value $d(k) = h(k) + 1$, which is consistent with a cost term of zero in equation (1).

By insertion in (1) and summation over all stacks, we get the following formula for the recourse cost:

$$Q = 2 \sum_{k=1}^{n_S} \left(h(k) - \sum_{j=1}^{h(k)} Y_{jk} \right) - \sum_{k=1}^{n_S} \sum_{j=1}^{h(k)} X_{jk}. \tag{6}$$

Finally, let us consider the list L and therefore the variables X_{jk} and Y_{jk} as *random*. In case of a Bernoulli process we have

$$E[X_{jk}] = p_{a(j,k)}$$

and

$$E[Y_{jk}] = P(X_{j'k} = 0 \ \forall j' \leq j) = \prod_{j'=1}^{j} (1 - p_{a(j',k)}),$$

the latter because of the assumed independence. This shows that the expected recourse costs are given by

$$E[Q] = 2 \sum_{k=1}^{n_S} \left(h(k) - \sum_{j=1}^{h(k)} \prod_{j'=1}^{j} (1 - p_{a(j',k)}) \right) - \sum_{k=1}^{n_S} \sum_{j=1}^{h(k)} p_{a(j,k)}. \tag{7}$$

As an aside, by inserting this explicit formula for the recourse function, a deterministic equivalent of premarshalling problem could be formulated and solved by any (exact or heuristic) method.

This model can now be easily extended to an actual stack handling procedure, where the items that have been moved to another stack in order to get access to the required items of the current stack k are *not* moved back to stack k later.

In order to avoid unnecessary costs, the stack to which a not needed item is moved should be one that has already been accessed in a former iteration, or a stack from which no item will be required according to the current day's list. Its existence is ensured by the operational assumption made in the introduction of section 3, which includes the possibility of opening a new stack. Not moving items back will of course reduce the total cost, at the price of possibly leading to a more unbalanced distribution of the stack sizes. Our equations above can be modified in a straightforward way to address the just described alternative procedure. This produces expected recourse costs of

$$E[Q] = \sum_{k=1}^{n_S} \left(h(k) - \sum_{j=1}^{h(k)} \prod_{j'=1}^{j} (1 - p_{a(j',k)}) \right) \tag{8}$$

instead of (7).

3.2. Poisson model

We generalize now the model from the previous subsection by taking the possibility into account that different boxes contain the same content, i.e., that the sets $SKU_s \subseteq B$ $(s = 1, \ldots, n_{SKU})$ are not singletons. The assumption $\bigcup_{s=1}^{n_{SKU}} SKU_s = B$ is made, and the current distribution of the SKUs over the stacks is supposed to be known.

A priority pri_i is assigned to each box i as described in section 2. Smaller values of pri_i mean higher priorities.

Let $s_i = s$ if $i \in SKU_s$, in other words, s_i is the index of the SKU

contained in box i. Then box i can be uniquely represented by the pair (s_i, pri_i). Here, pri_i is only important for determining the order in which boxes i belonging to the same SKU are retrieved, thus we can assume without loss of generality that $pri_i \in \{1, \ldots, g_s\}$ with $g_s = |SKU_s|$ for all $s = 1, \ldots, n_{SKU}$, each value from $\{1, \ldots, g_s\}$ occurring exactly once as a priority of a box $i \in SKU_s$.

The list L is an ordered list of SKU indices, $L = (s^1, \ldots, s^m)$. Repetition is allowed. Since in the case of orders for the same SKU, boxes with higher priority (lower pri_i) have to be delivered first, we extend the elements L by the addition of priority values pri_i in the following way: the extended list is $\tilde{L} = ((s^1, pri^1), \ldots, (s^m, pri^m))$, where

$$pri^\mu = |\{\mu' \leq \mu : s_{\mu'} = s_\mu\}| \quad (\mu = 1, \ldots, m).$$

For example, list $L = (2, 7, 1, 7, 6, 5)$ could be extended to

$$\tilde{L} = ((2, 1), (7, 1), (1, 1), (7, 2), (6, 1), (5, 1)).$$

We remark that the priority given to different boxes containing the same SKU is derived from management considerations, usually the boxes with longest permanence in stock are to be retrieved first. What is important for us, is that there is a deterministic rule that permits to differentiate the desirability of boxes containing the same SKU.

In the extended context of this subsection, we need now a more refined way to describe a layout. Based on the numbers $a(j, k)$ introduced in subsection 3.1, we define

$$\alpha(j, k) = s_{a(j,k)} \quad \text{and} \quad \beta(j, k) = pri_{a(j,k)} \quad (j = 1, \ldots, cap_k;\ k = 1, \ldots, n_S)$$

with $s_0 = pri_0 = 0$ to denote empty locations. The numbers $\alpha(j, k)$ and $\beta(j, k)$ give the SKU index of the box in tier j of stack k and the priority of this box, respectively. The current layout is completely described by

the rectangular scheme of pairs $(\alpha(j,k), \beta(j,k))$ for $j = 1, \ldots, n$ and $k = 1, \ldots, n_S$.

Building on the example of Section 2, the layout of the last 5 stacks consists in the scheme below, where the stacks are columns and the tiers are rows, see figure 2.

α

5	0	0	6	0	0
4	0	5	7	0	0
3	0	6	7	0	0
2	1	6	8	0	2
1	3	9	10	2	5
	1	2	3	4	5

(a) Alpha values

pri

5	0	0	3	0	0
4	0	2	1	0	0
3	0	1	2	0	0
2	1	2	1	0	1
1	1	1	1	2	1
	1	2	3	4	5

(b) Beta (pri) values

Figure 2: Alpha and beta distributions

Here, we have

$h(k)$	2	4	5	1	2
$d(k)$	2	3	3	0	1
$r(k)$	1	1	2	0	2

The SKU sizes are $g_1 = 1, g_2 = 2, g_3 = 1, g_5 = 2, g_6 = 3, g_7 = 2, g_8 = 1, g_9 = 1, g_{10} = 1$.

Analogously to (2), we define

$$X_{jk} = \begin{cases} 1, & \text{if } (\alpha(j,k), \beta(j,k)) \in \tilde{L}, \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

for $1 \leq j \leq cap_k$, $1 \leq k \leq n_S$. From the numbers X_{jk}, the numbers Y_{jk} are derived by (3). The equations (4) and (5), expressing the number of required items in stack k and the tier index of the deepest required item in stack k, respectively, are still valid.

3.2.1. Model of the picking list

Let us turn now to the stochastic model for the list L. We assume that the number of orders of SKU s in L satisfies a *Poisson distribution* with parameter λ_s ($s = 1, \ldots, n_{SKU}$). The numbers of orders of different SKUs are assumed as independent.

Since there is no upper bound on a Poisson-distributed random variable, it can happen that the number of orders of SKU s exceeds the number g_s of existing boxes containing this SKU. In this case, the orders exceeding g_s can obviously not be satisfied. To represent this feature, we *prune* the actual list $\tilde{L}$ by removing all pairs (s^μ, pri^μ) with $pri^\mu > g_{s^\mu}$. The resulting pruned list is denoted by $\hat{L}$. Let

$$Z_s = |\{\mu : (s, pri^\mu) \in \hat{L}\}|$$

be the random variable representing the number of occurrences of an item belonging to SKU s in $\hat{L}$. Furthermore, consider the probability function of $\min(\text{Poisson}(\lambda), g)$,

$$\pi_{\lambda,g}(t) = \begin{cases} \frac{\lambda^t}{t!}\, e^{-\lambda}, & \text{if } t < g, \\ 1 - \sum_{\ell=0}^{g-1} \frac{\lambda^\ell}{\ell!}\, e^{-\lambda}, & \text{if } t = g, \\ 0, & \text{if } t > g, \end{cases} \tag{10}$$

where λ is the mean number of orders of the considered SKU, and $t, g \in \mathbb{N}_0$. Since Z_s obeys the distribution (10), we have $P(Z_s = t) = \pi_{\lambda_s, g_s}(t)$. Applying this to the SKU $s = \alpha(j,k)$ found on position (j,k), we get

$$E[X_{jk}] = P(X_{jk} = 1) = P((\alpha(j,k), \beta(j,k)) \in \hat{L}) = P(\beta(j,k) \leq Z_{\alpha(j,k)})$$

$$= \sum_{t=\beta(j,k)}^{g_{\alpha(j,k)}} \pi_{\lambda_{\alpha(j,k)}, g_{\alpha(j,k)}}(t). \tag{11}$$

In the example, the expected X_{jk} values, $E[X_{jk}]$, are reported in figure 3, where we set $\lambda_s = 1$ for all s.

5	0	0	0.08	0	0
4	0	0.26	0.63	0	0
3	0	0.63	0.26	0	0
2	0.63	0.26	0.63	0	0.63
1	0.63	0.63	0.63	0.26	0.63
	1	2	3	4	5

Figure 3: Expected X_{jk} values

Finally, a representation for $E[Y_{jk}]$ is needed. By (3),

$$E[Y_{jk}] = P((\alpha(j',k),\beta(j',k)) \notin \hat{L},\ \forall j' \leq j).$$

The event described in the argument of P on the right hand side says that no box at any position (j',k) below position (j,k) in stack k, including (j,k) itself, is ordered. The boxes on or below position (j,k) in stack k can be partitioned in parts, each of them corresponding to one specific SKU. For each of these parts, only the box with lowest pri value (highest priority) is relevant, since if this box is not in $\hat{L}$, than the other boxes of the same part are not in $\hat{L}$ either: $(s,pri) \notin \hat{L}$ implies $(s,pri') \notin \hat{L}$ for all $pri' > pri$. Therefore, we can focus on the pri-minimal elements of each part.

For fixed j and k, let

$$S_{jk} = \{\alpha(j',k) : 1 \leq j' \leq j\}$$

denote the set of SKU indices occurring at or below the considered position (j,k).

5	0	0	6,7,8,10	0	0
4	0	5,6,9	7,8,10	0	0
3	0	6,9	7,8,10	0	0
2	1,3	6,9	8,10	0	2,5
1	3	9	10	2	5
	1	2	3	4	5

Figure 4: Example S_{jk} values

For $s \in S_{jk}$, let

$$\bar{pri}_{j,k,s} = \min\{\beta(j',k) : \alpha(j',k) = s,\ 1 \le j' \le j\}$$

be the minimum priority value in the part belonging to SKU s. Then

$$(\alpha(j',k), \beta(j'k)) \notin \hat{L},\ \forall j' \le j \quad \Leftrightarrow \quad Z_s < \bar{pri}_{j,k,s},\ \forall s \in S_{jk}.$$

Since the events $Z_s < \bar{pri}_{j,k,s}$ refer to different SKUs s and independence holds, the probabilities of these events can be multiplied in order to get the joint probability. Thus, because of

$$P(Z_s < \bar{pri}_{j,k,s}) = \sum_{t=0}^{\bar{pri}_{j,k,s}-1} \pi_{\lambda_s,g_s}(t),$$

it follows that

$$E[Y_{jk}] = \prod_{s \in S_{jk}} \sum_{t=0}^{\bar{pri}_{j,k,s}-1} \pi_{\lambda_s,g_s}(t). \tag{12}$$

In the example, the expected Y values are reported in figure 5.

5	0	0	0.05	0	0
4	0	0.10	0.05	0	0
3	0	0.14	0.10	0	0
2	0.14	0.27	0.14	0	0.14
1	0.37	0.37	0.37	0.74	0.37
	1	2	3	4	5

Figure 5: Expected Y_{jk} values

By inserting (11) and (12) into

$$E[Q] = 2\sum_{k=1}^{n_S}\left(h(k) - \sum_{j=1}^{h(k)} E[Y_{jk}]\right) - \sum_{k=1}^{n_S}\sum_{j=1}^{h(k)} E[X_{jk}], \tag{13}$$

we get the expected costs of the recourse action for the Poisson case, which can be extended analogously to the model at the end of Subsection 3.1. In the example carried on, the final value is $E[Q] = 14.55$.

4. Optimization Model

The statistical model presented in section 3 provides the basis for solving stochastic premarshalling instances.

4.1. First stage

The problem of the first stage is the premarshalling problem PMP, which asks to rearrange the warehouse so that the picking of the list, later specified as a random event, will be made with as few relocations as possible. The data that can be used in the first stage are the initial warehouse configuration and the historic picking lists.

As mentioned in section 2, this first stage has no cost for the company, as it is performed by underwork staff in slack periods. However, as there is a constraint on the maximum time that can be dedicated to premarshalling, thus on the maximum number of movements that can be made, we are interested in rearranging the warehouse configuration using the least possible number of relocation moves. This maximizes the probability of being able to complete all premarshalling operations in the available time, besides having the least impact on the staff's usual activity.

The solution procedure that we propose for the first stage tries therefore to premarshal the warehouse using the least number of relocation moves. We propose two procedures, first a simple greedy heuristic based on the estimator of formula (13) and then a more involved heuristic, again using the estimator. In both cases, in order to compute the estimates, it is necessary to know λ_s, for each $s \in SKU$. These value will be computed using a maximum likelihood estimator applied to the historic picking lists, thus yielding the average number of occurrences of items belonging to each SKU in a picking list.

4.1.1. Local search

The pseudo code of solution algorithm is thus as follows. Let Ξ be the set of all feasible warehouse configurations and $\xi^{\mathbf{0}}$ be an initial warehouse

configuration, $\xi^0 \in \Xi$. Furthermore, if two feasible configurations ξ^h and ξ^k differ only by the position of one single box at the top of a stack, let μ_{hk} be the move that transforms configuration ξ^h into ξ^k and let Ξ^h be the subset of Ξ identified by all the moves (i.e., single box relocations) that can be applied to ξ^h.

A local search premarshalling heuristic based on the estimator of formula (13) is as follows.

```
Algorithm Local_Search (ξ⁰, maxiter)
1.   let i = iter = 0
2.   let minq = E[Q^i]
3.   repeat
4.     let minh = ∞
5.     for each ξ^h ∈ Ξ^i
6.       compute E[Q^h] by formula (13)
7.       if (E[Q^h] < minq) minq = E[Q^h], minh = h
8.     end // for each
9.     i = minh, iter = iter+1
10. until (minh = ∞ or iter ≥ maxiter)
11. return ξ^i
```

The algorithm starts with a solution to be carried to its local optimum, which is the initial incumbent solution. It computes the estimator on each solution in the neighborhood identified by the move of one single box, and if the solution in the neighborhood with lowest estimate is better than the incumbent one, it becomes the new incumbent solution.

At the core of the algorithm lays the computation at step 6, where different alternatives are ranked on the basis of formula (13) of Section 3.2. Note that the additive structure of expression (13) permits an efficient computation of the difference of the expected number of relocation moves between the configuration obtained after the move and that existing before it.

As an example of its usage, consider again the configuration of figure 2, to be rearranged following the picking list reported in Section 3.2. Let it be ξ^0. A possible box to move is that at the top of stack 3, and it could be moved on the top of any of the 4 remaining stacks, thus yielding four configurations included in Ξ^0. Figure 6 shows the 4 corresponding obtained configurations.

Notice that one iteration of algorithm *Local_Search* corresponds to one premarshalling move. As a consequence, satisfying the constraint on the maximum time that can be dedicated to premarshalling amounts to stopping

the algorithm after a given number of iterations.

We can now compute the expected number of relocations on each of these 4 configurations by means of formula (13), obtaining E[Q] = 14.388 for move 1, E[Q] = 14.438 for move 2, E[Q] = 13.283 for move 3, and E[Q] = 14.388 for move 4. Move μ_{03} produces therefore a configuration which leads to lower recourse costs (costs for retrieving the items in the second stage), and is therefore preferred over the three other ones.

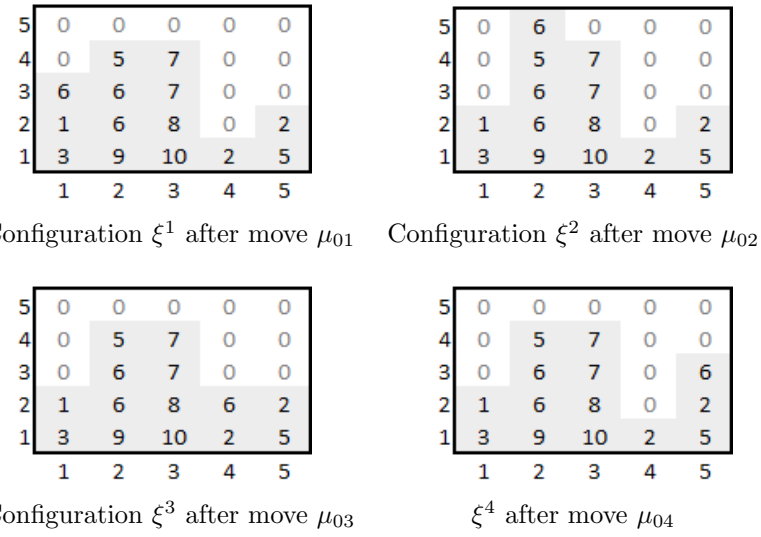

Figure 6: Alternative configurations after moving top box of stack 3 of fig. 2

4.1.2. Dynamic Programming matheuristics

The local search of section 4.1.1 is a straightforward option when the estimate of formula (13) is available, but its operational recommendations can be impractical in actual cases where the warehouse has hundreds of stacks, thus a front of hundreds of meters. In fact, since the reduction is made only with respect to the number of moves, but not with respect to their lengths, the final solution can be unacceptable in case of big blocks. The neighborhood to consider in step 5 could be restricted to a reasonable subset small enough to permit operational efficiency, but this at the cost of reducing the effectiveness of the algorithm.

In order to overcome the operational concerns of a basic greedy approach, we adapted to our case two methods from the literature, namely the *corridor method* by Caserta and Voß [9] and the *multiheuristic* by Jovanovic et al. [27].

The adaptation to our case, and the objective to improve the computational effectiveness over the greedy approach, was pursued by using all

historic data not only to estimate the λ coefficients of equation (10), but also in an attempt to capture some temporal information about when the order was placed during the one year span of the history. To that end, we explicitly considered in the code the sequence L of all orders available in the history, in the sequence they were actually received, and we designed a Dynamic Programming (DP) approach to the problem, whose structure is similar to the one proposed by Caserta and Voß [9] for the block relocation problem but which makes use of our estimator as a steering component.

The DP model is as follows. Let l_0 be the first unprocessed item in the sequence L, and let i be the topmost box in the stack where l_0 is stored, that is, $i = l_0$ if no box lays over l_0, $i \neq l_0$ otherwise. Box i will in any case be the next item considered for relocation.

The elements of the DP model are:

- **State**: the state s is defined as $s = (L_s, \xi^s)$, where L_s is the subsequence of items still to remove, and ξ^s is the configuration of the warehouse items. Let i be the next item to relocate and $k = 1, \ldots, n_S$ be the index of the stack in which the item is stored. From the configuration ξ^s it is easy to compute the set F_i of the stacks where it is feasible to relocate box i. Let $f(s)$ be a function that associates to each state s the expected minimal number of relocations necessary to move from the warehouse initial configuration to ξ^s

- **Stage**: A new stage is obtained by moving box i to a feasible destination stack $\sigma \in F_i$. Stages are defined by the number of box movements from the initial configuration, ξ^0.

- **State transition function**: let $s' = (L_{s'}, \xi^{s'})$ be the state obtained by moving item i to the least cost stack $\sigma \in F_i$. The new state cost is specified by the *forward recursion*:

$$f(L_{s'}, \xi^{s'}) = 1 + \min_{\sigma \in F_i} \{f(L_s, \xi^s)\} \tag{14}$$

  where $L_{s'} = L_s$ (thus $i' = i$ and $k' = k$) if $i \neq l_0$, otherwise $L_{s'} = L_s \backslash l_0$. The base of the recursion is $f(L, \xi^0) = 0$.

It is impractical to directly use the DP formulation to solve a big instance, as the DP algorithm quickly generates an exponential number of states even for small-sized instances. Therefore, to scale to real-world size, we need to resort to heuristics, an adapted corridor heuristic (see [8]) in our case. We note that the solution algorithm could be equivalently framed as a *beam search* or *VLSNS* algorithm, as matheuristics often overlap in specific application cases [37, 38], but the corridor is a much more established approach in the relocation literature.

Differently from [8] we make direct use of the DP formulation in our code, and we base all choices of the most promising box to relocate on expression (13), which permits to estimate the number of relocations needed to premarshal the current configuration. It is therefore possible to apply the estimation to each configuration that would be obtained after each possible relocation move, and prefer the move that forecasts a complete remarshalling with less expected relocations. Moreover, in our code the corridor is computed dynamically, obtaining a set of configurations that locally seem most promising.

After the corridor method has identified a solution, we start the multi-heuristic (adapted from [27]), which is not linked to the dynamic programming formulation, but it is applied as a local optimization, where the greedy cost functions is not that derived from the work of Exposito-Izquierdo [16] but again from formula (13).

More in detail, in the *corridor phase* we start from state s_0 corresponding to ξ^0 and $L_0 = L$. Let $S_0 = \{s_0\}$. We compute formula (13) for each feasible relocation stack for i, we keep only the δ best and compute a new state for each of them by means of the forward recursion formula (14). Let S_1 be the set of these δ newly generated states.

We then proceed by considering in turns each of the δ states of the set generated at the last stage. For each one that corresponds to a non request sequence, $L_s \neq \emptyset$, we compute formula (13) and expand only the δ best, storing in the set $S_{iStage+1}$ the non dominated ones. We then keep in $S_{iStage+1}$ only the δ overall best, an proceed expanding them until we obtain only states corresponding to a fully processed picking list.

To improve efficiency, when an intermediate state appears, with a promising expected number of rehandles, we apply an iteration-constrained version of the *multiheuristic* by Jovanovic et al. [27], denoted MH, where the heuristic or random choices of the next box to move or of the stack to move it to, were deterministically based on formula (13). In case this finds a final solution, the corresponding number of moves may become an upper bound and thus a further dominance, as no state is expanded when the expansion would correspond to a number of moves higher than that of the current upper bound. The pseudo code of solution algorithm is thus as follows.

```
Algorithm CorridorHeuristic (δ, maxMHiter)
1.   Set l_0 and i.  Set iStage = 0 and S_0 = {s_0}
2.   repeat
3.     for each s ∈ S_iStage compute formula (13)
4.     Compute formula (14) for the δ best configurations of step 3.
5.     for each newly generated state s'
6.       if (L completed) check upper bound
7.       else
8.         if (E[Q](ξ^s') < maxMHiter)
```

```
9.            Apply modified multiheuristic to ξ^{s'}
10.           check upper bound
11.         end // if
12.         add s' to S_{iStage+1}
13.       end // if
14.     end // for each s'
15.     shrink S_{iStage} to its δ best states
16.     iStage = iStage + 1
17. until (S_{iStage} = ∅)
```

In lines 6 and 10 we check whether the newly computed solution is the best obtained so far, in which case we store it as the best so far, and its cost as the new upper bound.

Line 8 contains the application condition of the modified multiheuristic MH. The termination condition inside MH is the number of iterations, which must not exceed maxMHiter. We apply MH only if the expected number of relocations needed to premarshal the current configuration is less than maxMHiter. Line 8 makes use of formula (13) in a heuristic way. The local optimization is invoked only when it is expected to solve the problem in the allowed number of internal moves (i.e., iterations).

4.2. Second stage

The problem of the second stage is a Block Relocation Problem, BRP, where the boxes of a list L_2 have to be successively removed. The interest in solving the second stage is only for assessing the quality of the premarshalled solution obtained in the first stage, according with the actual procedures that are being implemented in the real-world case.

To assess the quality of a PMP solution, both for artificial and for real-world instances, we proceed as follows. The cost of a solution corresponds to the number of relocations that are needed in the second stage of the solution process to pick up all boxes of the next-day picking list. As our solution is to be compared to actual manual operations, we compute the manual operations needed to respond to the picking list the way they do in the warehouses which originated our data, which is to relocate blocking boxes into stacks at most 2 stacks apart from the original one. Therefore, both in the case of premarshalled and of non rearranged configurations, during the second stage for each box to pick up, in case it were blocked, we move each blocking box into a stack at most two stacks away, i.e., we choose among four possible relocations for each blocking box (in the following this will be referred to as *manual rule*). This corresponds to a basic heuristic for solving the BRP, justified by actual practice, which can result in a number a relocations higher than that estimated in formula (13) (which assumes the possibility to relocate in any stack of the warehouse). In no case we

assume the knowledge of the subsequent box to pick, when deciding where to relocate a box blocking the currently requested one.

The only difference in our validation code between the premarshalled and the non premarshalled cases, is that in case of non-rearranged configurations, the chosen stack is the smallest among the four, while in case of premarshalled ones, it is the stack which yields a configuration with smallest expected relocation cost according to formula (13). This last information corresponds to a datum that could easily be made available to stackcrane operators, in case it was computed.

The cost computed for an instance is given in any case by the number of manual relocations, computed for each picking list of a validation set. This last set consists of a set of independent lists, different from those used in the first stage.

5. Computational experiments

We implemented the algorithm of Section 4 in C# and ran it on a Windows 10 Intel i7 quad-core computer equipped with 32 Gb of RAM, with a time limit of 600 secs.

We were given 8 real-world instances for as many warehouse blocks, each one with a final warehouse configuration and 1 year of picking history. As the instances were large ones, 700 to 2300 boxes, we also designed a generator which could produce parametrized artificial instances, showing descriptive statistics similar to those of the real-world ones. We were concerned about structural similarity with real-world instances, as it is well known from practitioners, and progressively accepted in theory, what came to be known as the *"No Free Lunch"* theorem [26, 15], that the relative performance of heuristics on instances sharing a certain structure does not always transpose to instances with a different structure. It is therefore important to test on instances as much similar as possible to those that will be of interest in practice.

For the premarshalling tests, we used picking lists for validation different from those used for estimating the λ parameters, in order to avoid bias induced by already processed data.

5.1. Artificial instances

We generated scaled instances, showing descriptive statistics similar to those of the real-world ones. The generation was a three steps process:

1. determine the storage configuration of the warehouse.
2. determine the length of each pickup list.
3. determine the SKUs actually reported in each list.

To determine the storage configuration, a distribution fit was made on historic data. The fit was derived from a specific statistical model, based on a negative binomial distribution, which is an option to model count data [22]. To define the model, we used the real-world instances presented in Table 3, using half of them to fit the two distribution parameters and the other half for validation. Figure 7 shows the results on one of the instances, with the empirical and modeled frequency counts of the number of boxes of each SKU in stock.

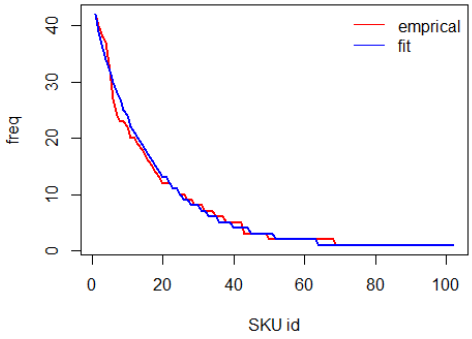

Figure 7: Fit of storage distribution

Similarly, we fitted the distribution of the available positions in each stack to determine the occupancy ratio to be allotted by means of the same approach used for computing the number of boxes of each SKU. Having these distributions, we used the generator in [16] to generate the storage configuration of the instances. Feasible instances could be obtained upon interpreting the *"priority"* parameter in the generator as the SKU identifier, which is possible since that parameter is dealt with as a label. The initialization of the SKU levels in stock was set to the negative binomial distributed values computed as described above. Figure 8 shows an example of an obtained configuration.

8	10	4	0	0	0	0	0	0	5	4	0	4	0	0	4	0	1	6	0	0
7	10	6	0	0	0	0	0	0	8	3	0	5	0	0	4	0	1	3	0	4
6	8	3	0	0	6	0	4	0	1	8	0	3	0	0	5	5	3	2	0	2
5	5	9	0	4	13	0	6	0	2	6	0	7	0	0	4	2	9	5	4	5
4	8	11	0	2	5	5	3	0	7	12	6	5	6	4	13	6	1	4	6	2
3	7	3	1	6	9	4	2	0	7	3	1	5	6	3	4	8	5	14	9	4
2	15	5	3	3	11	2	10	4	2	9	5	3	7	7	5	12	7	5	11	14
1	7	8	15	8	12	1	1	4	6	9	7	11	2	8	13	4	4	6	10	7
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Figure 8: Storage configuration, instance 20x8.

Table 2: Artificial instances

name	stacks	tiers	nbox	nsku	λ_p
testLC	10	5	35	9	3.0
test10x5	10	5	40	6	3.2
test20x8	20	8	120	15	4.9
test30x10	30	10	250	30	7.0
test40x12	40	12	400	60	9.0
test50x15	50	15	700	100	11.0

The second element to define is the counterpart of the historic picking list dataset.

In order to determine the length of the picking lists, we assume that the pickup requests in each day arrive following a Poisson distribution, since we clearly deal with a discrete distribution, where the requests occur independently, with the probability of getting a request in a time interval proportional to the length of the interval, and where the rate at which requests arrive is reasonably constant along the workshift.

We therefore fit Poisson distributions to our real-world data and we try to derive a relation that links the parameter of the Poisson distribution to the instance descriptions. (The Poisson parameter will now be denoted by λ_p to avoid confusion with the λ introduced in section 3.2.) One such relation, albeit loose, can be identified with respect to the number of boxes in the instance, which leads to generation of instances with characteristics reported in Table 2.

Finally, having determined the length of each pickup list, we have to dictate the specific SKUs in the list, which will be later translated to box id's in a company specific, deterministic way (in our case, we had to retrieve the oldest box of the SKU, unless a newer one with the same SKU was stored above the oldest one). By this procedure we generated sets of picking lists for each storage case consisting of 50 lists each.

The free parameters of the instance generator are the number of stacks n_S, the capacity of each stack cap_k, $k = 1, \ldots, n_s$, the number of stored boxes n, the number of picking lists and the average number of boxes requested in each picking list, λ_p. The generated instances, both configurations and picking lists, are available from [36].

5.2. Real-world case study

Further validation of the proposed methodology was carried out on real-world data. We were provided with extracts of WMS data relative to 1 year operation of 8 warehouses. The general descriptive statistics of these data were used as seeds for the generation of the instances described in subsection 5.1, in such a way that the biggest of the instances of subsection 5.1 was

Table 3: real-world instances

name	stacks	tiers	nbox	nsku	λ_p
instance1	51	15-24	737	102	11
instance2	69	12-18	999	113	15
instance3	71	18-30	1161	184	31
instance4	85	18-30	1735	304	30
instance5	88	24-30	1831	358	31
instance6	90	18-24	1852	371	28
instance7	92	18-24	1998	376	30
instance8	95	30	2330	362	64

comparable with the smallest of the real-world instances made available to us.

The characteristics of these instances are listed in Table 3. In this case the picking lists were the actual ones as derived from the WMS.

Using these data, we constructed two instance sets, one making use of the whole year's lists (which amounted to 291 working days lists) and one where we extracted the last 50 lists for each warehouse, roughly corresponding to two months activity, which was the horizon of interest for the analysis.

5.3. Estimates validation

A first set experiments is aimed at validating the quality of the estimator of section 3, upon which all our algorithms hinge. Figure 9 shows the boxplot of the validation, where the value estimated for each instance is represented by a gray rectangle, superimposed to the boxplot of the distribution of the actual number of rehandles that are necessary on the corresponding instance for each of the picking lists in the one-year dataset.

Note that the distributions of the number of rehandles is very skewed, being impossible to have less than 0 movements while a high number of them might be needed, therefore boxplots represent (as usual) medians and quartiles, the means being always higher than the median value, as the estimates suggest. Numerically, the mean number of rehandles per instance were on average 4.09% lower than their estimates on the 8 instances under study, with the interval of estimation errors ranging between -12.06% and +5.01%, which is a good fit for practical purposes.

5.4. Computational results, 2 stages

The first test we made was aimed at determining whether the combination of the two heuristics we implemented, namely, the Corridor Method (CM) and the MultiHeuristic (MH), which make up the algorithm *CorridorHeuristic* (CH) of section 4.1.2, performed better than the local search of section 4.1.1 and of any of the two alone.

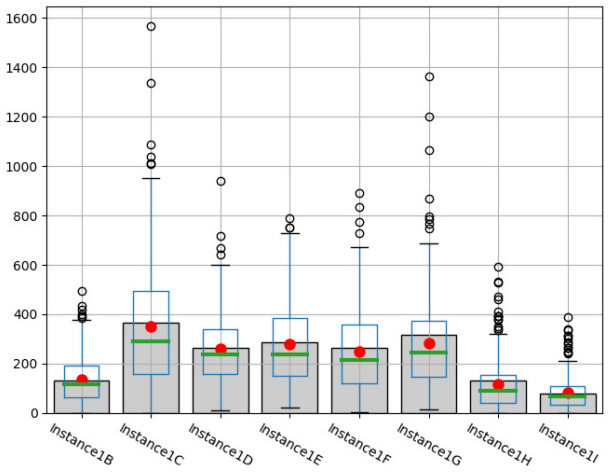

Figure 9: Estimate validation. The second axis shows the number of requested rehandles. Gray boxes: estimates of the means; red dots: actual means; boxplots: actual values.

We use for testing the two artificial instances together with the smallest and biggest real-world instance, obtaining coherent outcomes. All instances are tested on their 50 picking lists datasets, with the lists used for premarshalling different from the lists used for validation, as described in Section 2. Table 4 shows the quantitative results, where we report:

- name: the name of the instance of the block, or the quantity type.
- n.box: the number of box to remarshal.
- manual: the total number of relocations made in order to pick all boxes from the lists, using the *manual rule* described at the beginning of this section on all lists of the validation set.
- LS: data relative to the local search algorithm, here denoted LS.
- CH: data related to the CH algorithm.
- CM: data related to CM alone.
- MH: data related to MH alone.

For each instance and for each algorithm, we report also:

train : number of relocations made by the algorithm for remarshalling the warehouse on the basis of the lists in the *training set*.

val : number of relocations needed by the *manual rule*, after remarshalling, to pick all the boxes from the picking lists of the *validation set*.

Table 4: Comparison among methods

name	n.box	manual	LS	CH	CM	MH
testLC	35	253				
train			3	27	77	34
val			112	109	226	148
gap%			55.73	56.92	10.67	41.50
t.cpu			0.40	0.58	0.15	0.69
test50*15	700	3362				
train			64	410	1103	560
val			2491	2388	3294	2883
gap%			28.66	31.61	5.67	17.44
t.cpu			39.09	15.01	13.66	1.09
instance1	737	3760				
train			50	1432	5023	2112
val			2362	2316	2810	2448
gap%			37.18	38.40	25.27	34.89
t.cpu			40.07	16.22	14.18	3.59
instance8	2330	15978				
train			218	7344	15569	8620
val			12927	12081	13530	12443
gap%			19.09	24.39	15.32	22.12
t.cpu			2058.48	32.85	21.68	36.00

gap% : percentage ratio (manual - val)/manual.

t.cpu : cpu time, in seconds.

The parameter *maxIter* is set for all algorithms to the value of 5000, a value that never causes termination of LS but is the active termination condition in all other cases.

Notice how local search, albeit permitting at each step a relocation of a blocking box anywhere in the warehouse, is in all cases less effective than the CH heuristic, despite the cpu time, which is rapidly increasing with the size of the instance due to the computational impact of the computation of the estimator on higher numbers of configurations. On the contrary, all other heuristic approaches limit the number of stacks to consider, therefore the cost of the computation of the estimators.

As an example of the trace of a test, on the big real-world instance8, Figure 10 shows the trace of the decrease of the E[Q] estimator value along one run of LS and the average number of actual relocation moves which are used in the best solution of the 50 lists verification set in the case of LS (LSbest) and in the case of the *CorridorHeuristic* (CHbest). The latter value is smaller than the former one. We deem this result to be ascribable to the possibility of the CH heuristic to implicitly consider also some temporal information about when the orders were issues (as mentioned in section

4.1.2), thus, for example, to favor the picking of SKUs which were often requested later over those that were often requested at the beginning of the training horizon.

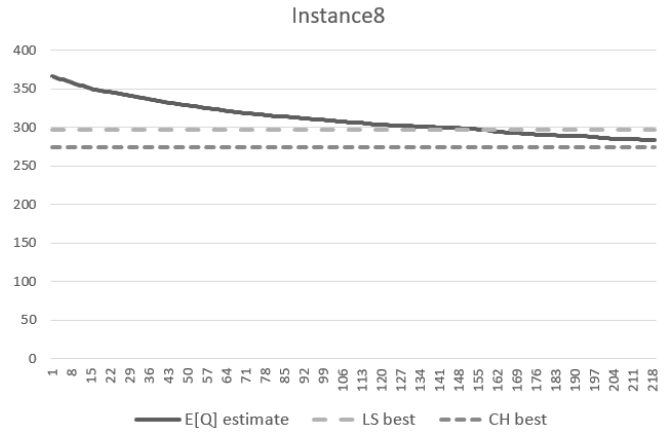

Figure 10: LS trace and LS and CH effectiveness comparison.

These preliminary tests provide a confirmation of the usefulness of the hybridization of the corridor method with the multiheuristic, a result that was consistently obtained in all tests we eventually made. All subsequent results are obtained by this method.

2 months scenarios:. Another set of tests was made both on artificial and on real-world instances on the basis of 50 picking lists, i.e., roughly two months of picking history. The results for the artificial instances are reported in Table 5 and those for the real-world ones are in Table 6.

In case of manual operations we report the number of relocations needed to service either the lists used for training ('training') or the validation ones ('validation'). In case of *CorridorHeuristic*, we report the number of relocations needed to premarshal according to the training set ('training'), the number of relocation to service all lists in the validation set ('validations') and the percentage gap on the number of relocations to be made using the manual rule, computed between the non-premarshalled case (Manual $\Longrightarrow$ validation) and the premarshalled one (CH $\Longrightarrow$ validation).

The column 'training' in the manual operations reports the number of relocations needed by the *manual rule* to service all picking lists of the training set, provided to as a basis for premarshalling, as such it should not be directly compared with any other column (the 'training' column for CH reports the number of relocation moves needed to service the lists in the training set after warehouse CH premarshalling). It has been included to ascertain that training and validation are structurally coherent.

Both in Table 5 and 6 it is apparent the benefit that can be achieved by premarshalling using historic data. The variance of the benefit is high,

Table 5: Artificial instances, 50 lists

		Manual		CH		
name	n.pallet	training	validation	training	validation	gap
10x5	40	181	199	15	97	51.26%
20x8	120	529	464	19	296	36.21%
30x10	250	991	1091	86	708	35.11%
40x12	400	1766	1991	211	1124	43.55%
50x15	700	2610	3362	410	2388	31.61%

Table 6: real-world instances, 50 lists

		Manual		CH		
name	n.pallet	training	validation	training	validation	gap
instance1	737	4123	3760	1432	2316	38.40%
instance2	999	6180	5328	2399	3182	40.28%
instance3	1161	5786	5126	2962	2256	55.99%
instance4	1735	12497	15103	4375	8508	43.67%
instance5	1831	11143	11906	5484	9147	23.17%
instance6	1852	14751	12860	6548	8379	34.84%
instance7	1998	12656	13987	6076	10306	26.32%
instance8	2330	18811	15978	7344	12081	24.39%

deriving from the high randomness of the list content, but in any case the benefit is remarkable.

1 year scenarios. A further test was made to assess the impact of the quantity of historic data on the final outcome. In this case, we use all data that were provided to us for the real-world instances, i.e., one year of picking history, corresponding to 291 working days. As the picking lists are applied to the initial warehouse configuration we were provided with, we remove from them all SKUs that were not present in the stock, which amount to ca. 5% of the historic picking orders.

The results are reported in Table 7. They are not significantly different from those of Table 5, even though on average somewhat worse (31.96% average gap versus 35.88%). This is possibly due to a shift on the distribution of the SKU requests along the year, so that very old data induce a bias that can be avoided considering only the more relevant, recent ones.

5.5. Computational results, multistage

Subsection 5.4 reports about the improvements that can be achieved on the first day of operation after performing all remarshalling relocations. Actually, the warehouse operations would consist in a first deep remarshalling,

Table 7: real-world instances, 291 lists

name	n.pallet	Manual validation	CH training	CH validation	gap
instance1	737	22878	1457	17103	25.24%
instance2	999	32494	2886	19830	38.97%
instance3	1161	31447	3330	20050	36.24%
instance4	1735	82108	4203	48265	41.22%
instance5	1831	72911	6383	54623	25.08%
instance6	1852	80650	6595	54462	32.47%
instance7	1998	75873	7786	57310	24.47%
instance8	2330	97696	8099	61595	36.95%

Table 8: Multistage operation

operation	1	2	3	4	5	6	7	8	9	10	Avg
Premarshal	14	3	1	2	0	2	1	1	1	1	
Validation	93	99	124	97	135	64	144	60	61	96	97.3
Manual	170	209	176	190	202	203	220	156	222	233	198.1

then in a sequence of days where lesser impact remarshalling would be performed overnight, as the warehouse configuration is not deeply affected by a single day of work.

Table 8 reports a validation in such a multistage setting considering a row of 10 days in the case of instance 10x5. The final gap between the number of relocations in the case of premarshalled configurations is 50.88%, which is coherent with the result reported in Table 5, but it is evident the decrease of the number of relocations needed to premarshal the warehouse.

The same holds true for the data reported in Table 9, which refers to 10 days of operations on the instance 50x15. Here the final gap is 20.04%, lower than that of the first day alone, but the substantial decrease of premarshalling relocation is still evident.

6. Conclusions

This paper presents the first optimization approach to stochastic premarshalling of warehouses, where randomness is associated to the picking

Table 9: Multistage operation

operation	1	2	3	4	5	6	7	8	9	10	Avg
Premarshal	410	3	8	6	8	14	8	16	7	5	
Validation	2374	2388	3020	3033	3144	2824	2956	2506	3104	2014	2736.3
Manual	3362	3492	3678	3316	3384	3370	3641	3079	3593	3308	3422.3

lists.

We describe a business analytics application, where the *statistical module* permits to fit probability distributions of picking lists and to estimate the number of relocation moves, and the *optimization module* permits to suggest sequences of relocation moves leading to optimized configurations.

The results were validated both on artificially generated and on real-world instances, with results that show that, even in actual application settings, a significant reduction of the number of the relocation moves needed to service a picking list can be obtained by a low-effort premarshalling process, to be carried out in underwork periods.

References

[1] Bartlett, R. *A Practitioneŕs Guide To Business Analytics: Using Data Analysis Tools to Improve Your Organizatioń s Decision Making and Strategy*. McGraw-Hill, 2013.

[2] Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing 8*, 2 (2009), 239–287.

[3] Boge, S., Goerigk, M., and Knust, S. Robust optimization for premarshalling with uncertain priority classes. *European Journal of Operational Research 287*, 1 (2020), 191–210.

[4] Borjan, S., Manshadi, V., Barnhart, C., and Jaillet, P. Dynamic stochastic optimization of relocations in container terminals. working paper,submitted, MIT, 2013.

[5] Bortfeldt, A., and Forster, F. A tree search procedure for the container relocation problem. *Computers and Operations Research 39*, 2 (2012), 299–309.

[6] Boschetti, M., Maniezzo, V., Roffilli, M., and Bolufé Röhler, A. Matheuristics: Optimization, simulation and control, in hybrid metaheuristics:. In *6th International Workshop, HM 2009 Proceedings*, M. Blesa, L. C.Blum, A. Gaspero, M. Roli, and A. S. Sampels, Eds. Springer, Berlin Heidelberg, 2009, pp. 171–177.

[7] Caserta, M., Schwarze, S., and Voss, S. Container rehandling at maritime container terminals. In *Handbook of Terminal Planning*, J. B ose, Ed., vol. 49 of *Operations Research/Computer Science Interfaces*. Springer, New York, 2011, pp. 247–269.

[8] Caserta, M., and Voss, S. A corridor method-based algorithm for the pre-marshalling problem. evoworkshops, volume 5484 of. *Lecture Notes in Computer Science* (2009), 788–797.

[9] Caserta, M., Voss, S., and Sniedovich, M. Applying the corridor method to a blocks relocation problem. *OR Spectrum 33*, 4 (2011), 915–929.

[10] Chan, F., and Chan, H. Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of classbased storage. *Expert Systems with Applications 38*, 3 (2011), 2686–2700.

[11] Chen, M., and Wu, H. An association-based clustering approach to order batching considering customer demand patterns. *Omega 33*, 4 (2005), 333–343.

[12] Choe, R., Park, T., Oh, M.-S., Kang, J., and Ryu, K. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. *Journal of Intelligent Manufacturing 22*, 2 (2011), 201–217.

[13] de Koster, R., Le-Duc, T., and Roodbergen, K. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research 182*, 2 (2001), 481–501.

[14] de Melo da Silva, M., Toulouse, S., and Wolfler Calvo, R. A new effective unified model for solving the pre-marshalling and block relocation problems. *European Journal of Operational Research 271*, 1 (2018), 40–56.

[15] Droste, S., Jansen, T., and Wegener, I. Optimization with randomized search heuristics: the anfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science 287*, 1 (2002), 131–144.

[16] Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, M. Premarshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications 39*, 9 (2012), 8337–8349.

[17] Galle, V., Manshadi, V., Boroujeni, S. B., Barnhart, C., and Jaillet, P. The stochastic container relocation problem. *Transportation Science 52*, 5 (2018), 1035–1058.

[18] Galle, V., Manshadi, V. H., Boroujeni, S. B., Barnhart, C., and Jaillet, P. The stochastic container relocation problem. *Transportation Science 52*, 5 (Oct. 2018), 1035–1058.

[19] Gheith, M., Eltawil, A., and Harraz, N. Solving the container pre-marshalling problem using variable length genetic algorithms. *Engineering Optimization 48*, 4 (2016), 687–705.

[20] Gu, J., Goetschalckx, M., and McGinnis, L. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research 177*, 1 (2007), 1–21.

[21] Gutjahr, W. J., and Pichler, A. Stochastic multi-objective optimization: a survey on non-scalarizing methods. *Annals of Operations Research 236*, 2 (2016), 475–499.

[22] Hilbe, J. *Modeling Count Data*. Cambridge University Press, 2014.

[23] Hottung, A., Tanaka, S., and Tierney, K. Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research 113* (2020).

[24] Hu, J., Zhang, C., and Zhu, C. s,s inventory systems with correlated demands. *INFORMS Journal on Computing 28*, 4 (2016), 603–611.

[25] Huang, S., and Lin, T. Heuristic algorithms for container premarshalling problems. *Computers & Industrial Engineering 62*, 1 (2012), 13–20.

[26] Igel, C., and Toussain, M. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms 3* (2004), 313–322.

[27] Jovanovic, R., Tuba, M., and Voss, S. A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research 25*, 1 (2017), 1–28.

[28] Kang, J., Kwang, R. R., and Kap, H. K. Deriving stacking strategies for export containers with uncertain weight information. *J Intell Manuf 17* (2006), 399–410.

[29] Kim, K. Evaluation of the number of rehandles in container yards. *Computers Ind. Eng. 32*, 4 (1997), 701–711.

[30] Kim, K., and Bae, J. Re-marshalling export containers in port container terminals. *Computers and Industry Engineering 35* (1998), 655–658.

[31] Kim, K., and Gyu-Pyo, H. A heuristic rule for relocating blocks. *Comput. Oper. Res 33*, 4 (2006), 940–954.

[32] Ku, D., and Arthanari, T. S. Container relocation problem with time windows for container departure. *European Journal of Operational Research 252*, 3 (2016), 1031–1039.

[33] Lee, Y., and Chao, S. A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research 196* (2009), 468–475.

[34] Lee, Y., and Hsu, N. Y. An optimization model for the container pre-marshalling problem. *Computers & Operations Research 34* (2007), 3295–3313.

[35] Lehnfeld, J., and Knust, S. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research 239*, 2 (2014), 297–312.

[36] Maniezzo, V. Stochastic premarshalling problem instances. http://astarte.csr.unibo.it/data/, 2020. retrieved July 4.

[37] Maniezzo, V., Boschetti, M. A., and Stützle, T. *Matheuristics: algorithms and implemntations*. Springer, to appear, 2020.

[38] Maniezzo, V., Stützle, T., and Voss, S. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming 1st ed.* Springer Publishing Company., 2009.

[39] Muppani, V. R., and Adil, G. K. Efficient formation of storage classes for warehouse storage location assignment: A simulated annealing approach. *Omega 36*, 4 (2008), 609–618.

[40] Pan, C.-H., and Liu, S.-Y. A comparative study of order batching algorithms. *Omega 23*, 46 (1995), 691–700.

[41] Park, T.-K., and Kim, K. H. Comparing handling and space costs for various types of stacking methods. *Computers & Industrial Engineering 58*, 3 (2010), 501–508.

[42] Parreno-Torres, C., Alvarez-Valdes, R., and Ruiz, R. Integer programming models for the pre-marshalling problem. *European Journal of Operational Research 274*, 1 (2019), 142–154.

[43] Prandtstetter, M. A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem. Tech. rep., 2013.

[44] Rendl, A., and Prandtstetter, M. Constraint models for the container pre-marshaling problem. In *ModRef 2013: 12th International Workshop on Constraint Modelling and Reformulation*, G. Katsirelos and C.-G. Quimper, Eds. 2013, pp. 44–56.

[45] Sculli, D., and Hui, C. F. Three dimensional stacking of containers. *Omega 16*, 6 (1988), 585–594.

[46] Tanaka, S., and Tierney, K. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. *European Journal of Operational Research 264*, 1 (2018), 165–180.

[47] Tanaka, S., Tierney, K., Parreno-Torres, C., Alvarez-Valdes, R., and R.Ruiz. A branch and bound approach for large pre-marshalling problems. *European Journal of Operational Research 278*, 1 (2019), 211–225.

[48] Tierney, K., Pacino, D., and Voss, S. Solving the pre-marshalling problem to optimality with a* and ida*. *Flexible Services and Manufacturing Journal 29*, 2 (2017), 223—-259.

[49] Tierney, K., and Voss, S. Solving the robust container pre-marshalling problem. In *International Conference on Computational Logistics*. Springer, Cham, 2016, pp. 131–145.

[50] Tompkins, J., White, Y., Bozer, E., and Tanchoco, J. *Facilities planning*, 4 ed. John Wiley & Sons, 2010.

[51] van Gils, T., Ramaekers, K., Caris, A., and Cools, M. The use of time series forecasting in zone order picking systems to predict order pickers' workload. *International Journal of Production Research* (2016).

[52] Wang, N., Jin, B., and Lim, A. Target-guided algorithms for the container pre-marshalling problem. *Omega 53* (2015), 67—77.

[53] Wenjuan, Z., and Anne, V. G. The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E: Logistics and Transportation Review 46*, 3 (2010), 327 – 343.

[54] Zhang, G., Tatsushi, N., Sarina, D. T., Keisuke, O., and Xindan, L. An integrated strategy for a production planning and warehouse layout problem: Modeling and solution approaches. *Omega 68* (2017), 85 – 94.

[55] Zhang, R., Jiang, Z.-Z., and Yun., W. Y. Stack pre-marshalling problem: a heuristic-guided branch-and-bound algorithm. *International Journal of Industrial Engineering 22*, 5 (2015), 509–523.

[56] Zhao, W., and Goodchild, A. V. The impact of truck arrival information on container terminal rehandling. *Transportation Research Part E: Logistics and Transportation Review 46*, 3 (2010), 327–343.