SN Computer Science (2024) 5:477
https://doi.org/10.1007/s42979-024-02711-w

ORIGINAL RESEARCH

Storage Assignment Using Nested Metropolis Sampling and Approximations of Order Batching Travel Costs

Johan Oxenstierna[1,2] · **Jacek Malec**[1] · **Volker Krueger**[1]

Received: 4 April 2023 / Accepted: 14 February 2024
© The Author(s) 2024

Abstract
The Storage Location Assignment Problem (SLAP) is of central importance in warehouse operations. An important research challenge lies in generalizing the SLAP such that it is not tied to certain order-picking methodologies, constraints, or warehouse layouts. We propose the OBP-based SLAP, where the quality of a location assignment is obtained by optimizing an Order Batching Problem (OBP). For the optimization of the OBP-based SLAP, we propose a nested Metropolis algorithm. The algorithm includes an OBP-optimizer to obtain the cost of an assignment, as well as a filter which approximates OBP costs using a model based on the Quadratic Assignment Problem (QAP). In experiments, we tune two key parameters in the QAP model, and test whether its predictive quality warrants its use within the SLAP optimizer. Results show that the QAP model's per-sample accuracy is only marginally better than a random baseline, but that it delivers predictions much faster than the OBP optimizer, implying that it can be used as an effective filter. We then run the SLAP optimizer with and without using the QAP model on industrial data. We observe a cost improvement of around 23% over 1 h with the QAP model, and 17% without it. We share results for public instances on the TSPLIB format.

Keywords Storage location assignment problem · Order batching problem · Quadratic assignment problem · Metropolis algorithm · Warehousing

Introduction

Charris et al. [7] gives the following definition of a Storage Location Assignment Problem (SLAP): The "allocation of products into a storage space and optimization of the material handling (…) or storage space utilization [costs]". The relationship between material handling costs, on the one hand, and storage assignment, on the other, can be showcased in an example: If a vehicle needs to pick a set of products, its travel cost clearly depends on where the products are stored in the warehouse. At the same time, the development of an effective storage strategy needs to consider various features in material handling, such as vehicle constraints, traffic conventions and picking methodologies.

In this paper, we work with a version of the SLAP which is particularly generalizable. Kübler et al. [18], name this version the "joint storage location assignment, order batching and picker routing problem". The main characteristic of this version is the inclusion of two optimization problems in the SLAP:

1. The *Order Batching Problem* (OBP), where vehicles are assigned to carry sets of *orders* (an order is a set of products) [17].
2. The *Picker Routing Problem*, where a short picking path of a vehicle is found for the products that the vehicle is assigned to pick. The Picker Routing Problem is a Traveling Salesman Problem (TSP) applied in a warehouse environment [25].

This article is part of the topical collection "Innovative Intelligent Industrial Production and Logistics 2022" guest edited by Alexander Smirnov, Kurosh Madani, Hervé Panetto and Georg Weichhart.

✉ Johan Oxenstierna
johan.oxenstierna@cs.lth.se

Jacek Malec
jacek.malec@cs.lth.se

Volker Krueger
volker.krueger@cs.lth.se

[1] Dept. of Computer Science, Lund University, Lund, Sweden

[2] Kairos Logic AB, Lund, Sweden

Published online: 23 April 2024

Henceforth, we refer to this version as the OBP-based SLAP. A key advantage of using the OBP within the SLAP is the added flexibility and generality of the *order* on a conceptual level: For example, optimizing the OBP-based SLAP gives opportunity to also optimize the TSP-based SLAP [23]. When it comes to product locations, the sole difference between the OBP and the OBP-based SLAP is that locations for all products are assumed fixed in the former while, in the latter, they are assumed mutable (for a subset of locations in our case).

It is of scientific importance to be able to compare optimization approaches and solutions. For the SLAP, this is made difficult by the many versions of the problem. As the extensive literature review by Charris et al. [7] shows, there is little consensus regarding which versions are more important, or specifically, which features would represent a standardized version. Examples of such features are dynamicity, warehouse layout, vehicle types, cost functions, reassignment scenarios and picking methodologies. There is also a shortage of benchmark datasets for any version of the SLAP, which prevents the reproducibility of experiments [2, 16]. As part of our contribution for a standardized version, we suggest a modified TSPLIB format [26] (section "Datasets"). There are several ways in which to balance between simplicity, reproducibility and industrial applicability when developing SLAP versions and corresponding instances, however. From the generalization perspective, our model is advantageous in two main areas: Order-picking methodology and warehouse layout. But it is weak in two other areas: dynamicity and reassignment scenarios. We describe the meaning of these choices further in the light of prior work (section "Related Work") and in our problem formulation (section "Problem Formulation"). We invite the community to debate which features are more or less important for a standardized version.

In section "Optimization Algorithm", we introduce our SLAP optimizer. It is based on the Metropolis algorithm, a type of Markov Chain Monte Carlo (MCMC) method. A core feature of the optimizer is that the quality of a location assignment candidate is retrieved by optimizing an OBP. Due to the OBP's NP-hardness, it must be optimized in a way that trades off solution quality with CPU-time. For this purpose, we use an OBP optimizer with a high degree of computational efficiency [22]. Within the SLAP optimizer, the OBP optimizer is still computationally expensive, and we show that it can be assisted by fast cost approximations from a Quadratic Assignment Problem (QAP) model. Finally, we test the performance of the SLAP optimizer with and without inclusion of the QAP approximations. Cost improvements are around 23% over 1 h with the QAP model, and 17% without. In summary, we make three concrete contributions:

1. Formulation of an OBP-based SLAP optimization model and a corresponding benchmark instance standard.
2. QAP approximation model to predict OBP travel costs and experiments on generated instances to test whether the use of QAP approximations within a SLAP optimizer can be justified.
3. An OBP-based SLAP optimizer (QAP-OBP) and experiments on industry instances to test its computational efficiency. Comparison of results with and without usage of QAP approximations.

Related Work

This section goes through general strategies for conducting storage location assignment, as well as ways in which their quality can be evaluated. Various SLAP formulations and proposed optimization algorithms are covered. Our primary focus will be on the standard picker-to-parts arrangement. We specifically refer to the work of Kübler et al. [18], as their proposed model aligns with ours.

There exist numerous general strategies for conducting storage location assignment [7]. Three key strategies are Dedicated, Class-based and Random:

- *Dedicated* Each product is assigned to a specific location which never changes. This strategy is suitable if the product collection changes rarely and simplicity is desired. Additionally, human pickers can leverage this strategy by familiarizing themselves with specific products and their corresponding locations, which might speed up their picking [35].
- *Random* Each product can be assigned any available location in the warehouse. This is suitable whenever the product collection changes frequently.
- *Class-based (zoning)* The warehouse is partitioned into sections, and the products are classified based on their demand. Each class is assigned a zone. The outline of the zone can be regarded as dedicated in that it does not change, whereas the placement of each product in a zone is assumed to be random [21]. Class-based storage assignment can therefore be regarded as a middle ground between dedicated and random.

The quality of a location assignment is commonly evaluated based on some model of aggregate travel cost. For this purpose, a simplified simulation of order-picking in the warehouse can be used [7, 21]. Some proposals include the simulation of order-picking by the Cube per Order Index (COI) [15]. COI includes the volume of a product and the frequency with which it is picked (historically or future-forecasted). Products with high pick frequency and relatively low volume are subsequently assigned to locations close to

the depot. Since orders may contain products which are not located close to each other, COI is only adequate for order-picking scenarios where orders contain one product and vehicles carry one product at a time. This may be sufficient for pallet picking or when certain types of robots are used [3]. Mantel et al. [21], introduced Order Oriented Slotting (OOS) where the number of products in an order may be greater than one. A similar model to OOS is used by Fontana and Nepomuceno [10], Lee et al. [20] and Žulj et al. [37]. The picking cost of an order in OOS can in some cases be modeled using a Quadratic Assignment Problem (QAP) [21]. The QAP computes the sum of element-wise products of weights and frequencies [1] and for an order this can be translated into distances between products and how often they are picked. Nevertheless, a QAP on its own is often not sufficient to model a SLAP without extensive use of heuristics and constraints for warehouse layouts and picking methodologies [21]. For a layout-agnostic OBP-based SLAP, graph-based QAP techniques could be attempted, but hitherto they have only been applied on related problems [31, 36].

There is only limited research on SLAPs where vehicles are expected to carry multiple orders and where an Order Batching Problem (OBP) is integrated into the SLAP optimization process. One example is Xiang et al. (2018) and [33], who use this approach in a robotic warehouse where the vehicles are pods or mobile racks, which is not easily comparable to a picker-to-parts system. Another example is Kübler et al. [18], which we look closer at below.

Travel distance or time are commonly used to evaluate SLAP solution quality in the above mentioned models, but there are several alternatives and extensions. Lee et al. [20], for example, study the effect of location assignment and traffic congestion in the warehouse. Assigning too many products to locations close to the depot (the goal in common COI) may lead to traffic congestion, which should ideally be considered in an industrial model. Lee et al. [20], formulate Correlated and Traffic Balanced Storage Assignment (C&TBSA) as a multi-objective problem with travel cost on the one hand, and traffic congestion avoidance on the other. Larco et al. [19], include worker welfare in their evaluation of solution quality. If picking is conducted by humans who move products from shelves onto a vehicle, the weight and volume, as well as the height of the shelf the product is placed on, can have an impact on worker welfare. Parameters such as "ergonomic loading," "human energy expenditure," or "worker discomfort" [7] can be used to quantify worker welfare.

The SLAP can be categorized into two main groups based on the number of location assignments required. Either the assignment is a "re-warehousing" operation, which means that a large portion of the warehouse's products are (re)assigned [16]. Often, however, only a small subset of products are (re)assigned a location and this is called "healing" [16]. Solution proposals involving healing often look closely at different types of scenarios for carrying out initial assignments for new products in the warehouse, or reassignments for products already in the warehouse. Kübler et al. [18], propose four such scenarios.

1. *Empty storage location* A product is assigned to a previously unoccupied location.
2. *Direct exchange* A product changes location with another product.
3. *Indirect exchange 1* A product is moved to another location which is occupied by another product. The latter product is moved to a third, empty location.
4. *Indirect exchange 2* A product is moved to a new location which is occupied by a second product. The second product is moved to a new location which is occupied by a third product. The third product is moved to the original location of the first product.

The above scenarios are all associated with varying levels of effort, ranging from the lightest in scenario I, to the heaviest in IV. Kübler et al. quantify these efforts by including both physical and administrative times, which are transformed to effort terms by proposed proportionalities.

Concerning SLAP optimizers, proposals include models capable of obtaining optimal solutions, such as Mixed Integer Linear Programming (MILP), dynamic programming and branch and bound algorithms [7]. The warehouse environment is often simplified to a significant degree when optimal solutions are sought [7, 13, 16, 19]. The main simplification relates to order-picking using COI or OOS. Other simplifications involve limiting the number of products [13], number of locations [30], or by requiring the conventional warehouse rack layout [18]. The conventional layout assumes Manhattan style blocks of aisles and cross-aisles, and it is used almost exclusively in existing literature on the SLAP (we are only aware of two exception cases using the "fishbone" and "cascade" layouts [6, 7].

Most proposed SLAP optimizers provide non-exact solutions using heuristics or meta-heuristics. One example is multi-phase optimization where the first phase proposes possible locations for products, and the second phase carries out the assignments and evaluates them [32]. In Kübler et al. [18], a heuristic zoning optimizer is used to generate location assignments, and a Discrete Evolutionary Particle Swarm Optimizer (DEPSO) is used to optimize an OBP for the evaluation of the assignments. DEPSO is a modification of a standard PSO algorithm that addresses the risk of convergence on local minima and allows for a discrete search space. Other heuristic or meta-heuristic approaches include Genetic and Evolutionary Algorithms [9, 20], Ant Colony Optimization [34] and Simulated Annealing [35].

If TSP optimization is desired within a SLAP, *S-shape* or *Largest Gap* algorithms [28] are often utilized. For TSP-optimization on unconventional layouts with a pre-computed distance matrix, *Google OR-tools* or *Concorde* have been proposed [22, 27].

Evaluating the quality of results in prior work is challenging due to the variability of SLAP models. Below are a few examples where result quality is judged based on a percentage saving in travel distance or time: For conventional warehouse layouts, reassignment costs and dynamic picking patterns, Kofler et al. [16], report best savings around 21%. Kubler et al. (2020), report best savings around 22% in a similar scenario. Zhang et al. [35] report best savings around 18% on simulated data with thousands of product locations, but without reassignment costs. In a similar setting, for a few hundred products, Trindade et al. (2022) report best savings around 33%.

Nested Metropolis Sampling

The proposed optimizer (section "Optimization Algorithm") is based on a nested Metropolis algorithm first introduced by Christen and Fox [8]. The Metropolis algorithm is a type of Markov Chain Monte Carlo (MCMC) method, which first draws a sample x_{i+1} based on a desired feature distance (excluding costs) to a previous sample x_i. The distance is given by some probability distribution $q(x_{i+1}|x_i)$, and it is usually chosen such that the distance between x_{i+1} and x_i is low with a high probability (Mackay 1998). The *accept* probability is then computed based on some function that takes the costs of the new and previous samples as input [29]. Common Metropolis sampling assumes that there is only one cost function, f^*, and since we wish to include an approximation of this cost, f, we use a modification [8]. Nested Metropolis sampling is shown in flowchart form in Fig. 1.

After a first sample x_i has been initialized (i), a new sample x_{i+1} is generated (ii) and its cost approximated $f(x_{i+1})$ (iii). If the approximation is deemed strong enough (probabilistically) relative to $f(x_i)$, the sample is *promoted* (iv) to the next step where its ground-truth cost $f^*(x_{i+1})$ is computed (v). The accept filter (vi) is only used for promoted samples.

For a cost minimization problem, the promote and accept probabilities can be computed based on the following equations [8]:

$$\alpha(x_{i+1}|x_i) = min(1, f(x_i)/f(x_{i+1})) \tag{1}$$

$$\alpha^*(x_{i+1}|x_i) = min(1, f^*(x_i)/f^*(x_{i+1})) \tag{2}$$

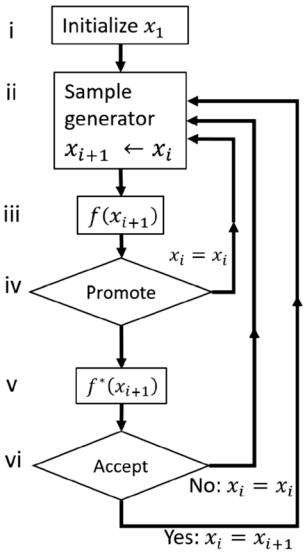

Fig. 1 Nested Metropolis Sampling. The inner loop computes a cheap (in terms of CPU-time) approximation of a sample cost and if the approximation is strong, the sample is promoted to the outer loop where an expensive ground-truth cost is computed

where $\alpha(x_{i+1}|x_i)$ denotes the promote probability and $\alpha^*(x_{i+1}|x_i)$ the accept probability.

Problem Formulation

Objective Function

The objective function in the OBP-based SLAP is based on the ones formulated in Henn and Wäscher [14] and Oxenstierna et al. [24], i.e., the minimization of cost in an Order Batching Problem (OBP):

$$f^*(x) = min \sum_{b \in \mathcal{B}} D^x(b) a_{vb}, v \in V, \mathcal{B} \subset 2^{\mathcal{O}} \tag{3}$$

where $\mathcal{O}$ denotes orders, where $\mathcal{B}$ denotes batches and where $D^x(b)$ denotes the distance of a TSP solution, i.e., the distance needed to pick batch b. Batch b is a set of orders and $v \in V$ denotes a vehicle. Each vehicle can carry one batch and the number of orders that can fit in the batch is governed by vehicle capacity (such as dimensions, bins, number of orders or products). a_{vb} denotes a binary variable set to 1 if vehicle v is assigned to pick b and 0 otherwise. Orders consist of products $\mathcal{O} \in 2^{\mathcal{P}}$, where each product $p \in \mathcal{P}$ is a tuple consisting of a unique key (Stock Keeping Unit), a Cartesian location $loc(p)$, and a positive quantity of how many p are available at $loc(p)$. The locations of all products are given by location assignment vector x, where the elements represent

products and the indices locations (each index is mapped to a Cartesian coordinate).

The mapping of location keys to coordinates and computation of distances between pairs of locations is based on a digitization pipeline for warehouses on any 2D obstacle layout and usage of the Floyd-Warshall graph algorithm. Details on this digitization pipeline and the OBP (including TSP-optimization for $D^x(b)$ and usage of vehicle capacity in a_{vb}) are beyond the scope of this paper, so for specifics we refer to Oxenstierna et al. [24] and Rensburg [27].

The difference between the OBP and the OBP-based SLAP mainly concerns product locations. In Oxenstierna, van Rensburg, et al. (2021) each product $p \in$ "has a [fixed] location", meaning that x in $f^*(x)$ is immutable. In the OBP-based SLAP, however, a subset of products $\mathcal{P}_s \subset \mathcal{P}$ do not have fixed locations, which means that some elements in x can change indices in the vector. The OBP-based SLAP objective consists of finding location assignment x such that the OBP in Eq. 3 is minimized:

$$\underset{x}{\operatorname{argmin}} \sum_{b \in \mathcal{B}} D^x(b) a_{vb}, v \in V, \mathcal{B} \subset 2^{\mathcal{O}} \tag{4}$$

This objective lacks reassignment costs and is therefore a version of the "empty storage location" scenario I in Kübler et al. [18] (section "Related Work"). Exclusion of reassignment costs is motivated for this scenario, since the initial location assignment of new products in a warehouse is not optional, but a requirement. The other of Kübler et al.'s scenarios are all reassignments. Contrary to the initial assignments that we work with, reassignments are optional and potential gains in travel cost must there be weighed against reassignment costs.

Although reassignments should ideally be included in a complete SLAP model, a standardized SLAP needs to be a trade-off between simplicity and complexity. In the TSP-based SLAP [23] it is shown that the optimization of reassignments is NP-hard and not easily combined with order-picking optimization within a SLAP. The TSP-based SLAP includes reassignments, but uses the TSP instead of the OBP to optimize order-picking. The OBP-based SLAP excludes reassignments, but includes the OBP, a significantly more challenging problem than the TSP. As is often the case in literature on the SLAP, choice of optimization model depends on which features are considered more important for the usecase at hand.

Fast OBP Cost Approximation

One key difficulty with the OBP-based SLAP is that the OBP poses a highly intractable problem. Even for relatively small OBP instances, a significant amount of CPU-time is needed to obtain substantial cost improvements [18, 22]. In the case of the OBP-based SLAP, this means that it would require a large amount of CPU-time to minimize cost for many assignment candidates x (Eq. 4). To resolve this problem, we propose to include an approximation of $f^*(x)$:

$$f(x) = \sum_{p_1 \in \mathcal{P}} \sum_{\substack{p_2 \in \mathcal{P} \\ p_1 \neq p_2}} \sum_{l_1 \in \mathcal{L}_{\mathcal{P}}} \sum_{\substack{l_2 \in \mathcal{L}_{\mathcal{P}} \\ l_1 \neq l_2}} w_{p_1 p_2} d^x_{l_1 l_2} \times \times a(p_1, l_1) a(p_2, l_2) \tag{5}$$

where w denotes weight, where $d^x_{l_1 l_2}$ denotes distance between two locations l_1, l_2 and $a(p, l)$ a function which returns 1 if product p is located at location l and 0 otherwise. $f(x)$ is the element-wise summation of weights times distances. The cell values in the weight matrix represent the number of times two products, p_1, p_2, appear in the same order $o \in \mathcal{O}$. The (shortest) distances between all pairs of product locations are assumed pre-computed and stored in memory. We refer to Eq. 5 as the Quadratic Assignment Problem (QAP) model. Note that we never minimize it. For the $f(x)$ approximation to be of use, we proceed to discuss how its ability to predict $f^*(x)$ can be evaluated.

Assuming a dataset of finite samples with approximated and ground truth costs $\left(x, f(x), f^*(x)\right) \in X, |X| \in \mathbb{Z}^+$, $f(x), f^*(x) \in \mathbb{R}^+$, the predictive quality of $f(X)$ versus $f^*(X)$ is obtainable through softmax cross-entropy [4, 5]:

$$\mathbb{P}\left(f\left(x_i\right)\right) = \frac{e^{f(x_i)}}{\sum_{j=1}^{|X|} e^{f(x_j)}} \tag{6}$$

$$\mathbb{P}\left(f^*\left(x_i\right)\right) = \frac{f^*(x_i)}{\sum_{j=1}^{|X|} f^*\left(x_j\right)} \tag{7}$$

$$L = -\frac{1}{|X|} \sum_{\left(x_i, f(x_i), f^*(x_i)\right)} \mathbb{P}\left(f^*\left(x_i\right)\right) log \mathbb{P}\left(f\left(x_i\right)\right) \tag{8}$$

where $\mathbb{P}\left(f\left(x_i\right)\right)$ and $\mathbb{P}\left(f^*\left(x_i\right)\right)$ denote the probabilities of approximate and ground truth costs of sample x_i, respectively, where $\left(x_i, f\left(x_i\right), f^*\left(x_i\right)\right) \in X$. L is the *loss*, i.e., a distance heuristic between $f(X)$ and $f^*(X)$. This approach can be extended into Normalized Discounted Cumulative Gain (NDCG) [4].

$$NDCG = \frac{DCG}{IDCG} \tag{9}$$

$$DCG = \sum_{i=1}^{|X|} \frac{rel(\pi_{f(X)}(i))}{log_2\left(\pi_{f(X)}(i) + 1\right)} \tag{10}$$

$$IDCG = \sum_{i=1}^{|X|} \frac{rel(\pi_{f^*(X)}(i))}{log_2\left(\pi_{f^*(X)}(i) + 1\right)} \tag{11}$$

$\pi_{f(X)}$ is a *ranking* (an ordering of samples X according to their costs $f(X)$) and $rel(\pi_{f(X)}(i))$ is the relevance at rank $\pi_{f(X)}(i)$. *IDCG* denotes an *ideal* value, where $rel(\pi_{f^*(X)}(1)) > rel(\pi_{f^*(X)}(2)) > \cdots > rel(\pi_{f^*(X)}(|X|))$, i.e., the case when the relevance of a sample corresponds with how highly it is ranked. Bruch et al. [4] argue that NDCG is a stronger choice than softmax cross-entropy whenever cost is non-binary, which is the case in $f^*(x)$ (Eq. 3). In Fig. 13 (Appendix) an example is shown where NDCG is computed from $|X|$ samples.

In summary, we can quantify the predictive quality of the QAP model by its ability to rank a list of samples X against a ground truth ranking by the OBP optimizer. Since the nested Metropolis algorithm in section "Nested Metropolis Sampling" only stores two samples at any iteration, we modify the algorithm to instead work with more samples (section "Optimization Algorithm"). We also want to avoid the computation of $f^*(X)$ in each iteration, so in the optimization algorithm we only compute $f^*\left(argmin_x f(X)\right)$. In section "Experiments", we conduct an experiment to test the validity of using the NDCG-based $f^*\left(argmin_x f(X)\right)$ in SLAP optimization. In section "Datasets" we also discuss choice of datatype for the relevance values.

Optimization Algorithm

Overview

The proposed optimization algorithm includes three main modules: 1. a sample (location assignment) generator. 2. a fast cost approximator based on a model of the Quadratic Assignment Problem (QAP). 3. an Order Batching Problem (OBP) optimizer. In this paper, we mainly focus on how QAP approximations can be effectively utilized within the nested Metropolis sampler described in section "Nested Metropolis Sampling". In sections "Sample Generator" and "Promote and Accept Thresholds and Cost Computations", we therefore describe two main modifications. The final version (QAP-OBP) is shown in flowchart form in Fig. 2 and pseudocode in Algorithm 1.

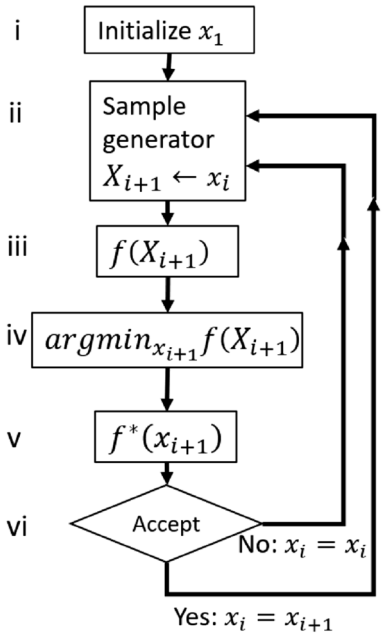

Fig. 2 QAP-OBP optimization algorithm

Algorithm 1 SLAP optimizer (QAP-OBP)

x, X: Sample(s) (location assignment(s)).
$f(X)$: Less accurate fast cost estimates (QAP).
$f^*(x)$: More accurate slow cost estimate (OBP).
N: Number of samples per iteration ($|X| = N$).
λ: Rate of change.
Δ: Cost-distance function.
α: Probability that sample x_{i+1} is *accepted*.
K: Number of iterations.
for $i = 1, ..., K$ **do**
    $X_{i+1} \leftarrow sample\_generator(N, \lambda, x)$
    $f(X_{i+1}) \leftarrow cost(X_{i+1})$
    $x_{i+1} \leftarrow argmin_{x_{i+1}} f(X_{i+1})$
    $f^*(x_{i+1}), f^*(x) \leftarrow cost^*(x_{i+1}, x)$
    $\alpha \leftarrow exp(-c_1 \Delta(f^*(x_{i+1}), f^*(x)))$
    $u \leftarrow \mathcal{U}(0, 1)$ // random uniform.
    if $u < \alpha$ **then** // sample *accepted*.
        $x \leftarrow x_{i+1}$
    end if
end for

Sample x contains both the assigned products (products already in the warehouse) and the unassigned products $\mathcal{P}_s$ (section "Problem Formulation"). x_1 is initialized such that products $\mathcal{P}_s$ are assigned locations randomly without replacement. Choices for iterations K, the cost distance function Δ and constant c_1 are discussed in section "Experiments".

Sample Generator

The input to the sample generator (step ii in Fig. 2) is a single sample x_i and the output is a list of new samples X_{i+1}. There are two main parameters in use by the sample generator. $N \in \mathbb{Z}^+$ dictates how many new samples are generated, i.e., $|X_{i+1}|$, and $\lambda \in \mathbb{R}^+$ dictates how much each new sample in X_{i+1} differs from x_i. The way N and λ are utilized to generate new samples is shown in Algorithm 2.

Algorithm 2 Sample Generator

```
x: Sample (location assignment).
N: Number of new samples.
λ: Rate of change.
X_{i+1} ← list(): Empty list of new samples.
for i = 1, ..., N do
    x ← copy(x_i) // New sample.
    m ← Pois(λ) // Number of changes.
    P_m ← remove_m_products(x, m)
    for p ∈ P_m do
        q ← p.get_quantity()
        L ← possible_locations(x, q)
        loc(p) ← assign(x, L)
    end for
    X_{i+1}.add(x)
end for
```

Every time the sample generator is called, an empty list is first initialized. Then, for N iterations, a new sample x is generated by first copying x_i and then by computing m, the number of products for which the index in x can change. For m we use a truncated Poisson distribution with rate λ and upper bound $m \leq |\mathcal{P}_s|$. A uniform random selection of m products, $\mathcal{P}_m$, are then removed from x. For each $p \in \mathcal{P}_m$, a uniform random free index (either an empty location or an index holding a product in $\mathcal{P}_s$) in x is then selected such that the quantity (q) of the product does not exceed the location's capacity. After x has been filled, it is appended to X_{i+1}.

Promote and Accept Thresholds and Cost Computations

After a list of samples X_{i+1} has been generated (step ii in Fig. 2), their costs are approximated using the QAP model (iii). The sample with the lowest cost approximation is then always promoted (iv). Steps ii, iii and iv in both the nested Metropolis sampler and QAP-OBP (Figs. 1 and 2, respectively) are the same considering that the final output is a single promoted sample. There are advantages and disadvantages of both versions regarding how they conduct this selection. In the nested Metropolis sampler, the promote probability depends on the ratio of approximated cost between previous and new single samples. In QAP-OBP, the sample generator is instead set to output $N = |X_{i+1}|$ candidates, followed by argmin (compare step iv in Figs. 1 and 2). This modification simplifies evaluation of the QAP model's accuracy, since we can set up an experiment to compute OBP costs on the same samples (Fig. 5). Generating multiple samples could also facilitate parallelization, which, for future work, could reduce the QAP model's CPU-time. The main consideration, however, is that it simplifies the original algorithm for a particularly complex optimization scenario, where it cannot be expected to behave according to Christen and Fox's [8] performance guarantees. The problem with the original algorithm is that it assumes optimal ground-truth costs, but these are not generally available for OBPs [22] (as far as we are aware, there exists no proposal for how to obtain optimal results for but the smallest OBP instances within reasonable CPU-time). A relatively minor problem with the modification is that it requires tuning of the number of samples (N) that the sample generator is outputting each iteration. The reason we use a Metropolis algorithm instead of possibly more capable metaheuristic alternatives, is mainly due to implementation. The Metropolis algorithm does not have many parameters which could be tuned based on iterations K (such as the temperature in Simulated Annealing) and therefore, a time-based condition can be used instead of K to terminate the algorithm (we will use this in section "SLAP optimization with and without QAP approximation").

Concerning computation of $f^*(x)$ we use the *Single Batch Iterated* (SBI) optimizer and its main features are its high computational efficiency and its ability to handle warehouses with unconventional rack layouts [22]. OBP optimization and its internal use of TSP optimization, is beyond the scope of this paper, and we here treat SBI as a black-box which outputs a $f^*(x)$ for Eq. 3. The sample x with the lowest $f^*(x)$ found is always stored throughout the optimization procedure (sample storage is omitted in Figs. 1, 2 and the pseudo-code).

Datasets

For this paper, we have generated and shared instances in L17_533,[1] which are based on OBP instances in L6_203[2] and L09_251.[3] We also use data from a real warehouse (Aba

[1] https://github.com/johanoxenstierna/L17_533, collected 13–02–2023.

[2] https://github.com/johanoxenstierna/OBP_instances, collected 15–01–2023.

[3] https://github.com/johanoxenstierna/L09_251, collected 15–01–2023.

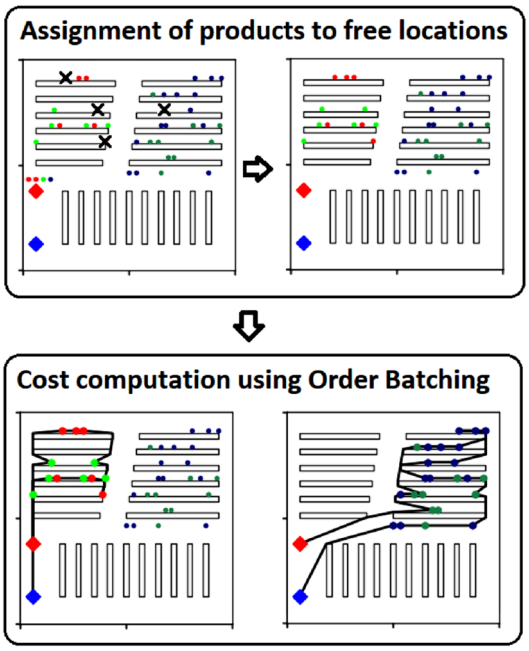

Fig. 3 Example storage assignment of four products and subsequent order-picking for the SLAP model used in the paper. Rectangles denote warehouse racks. Red and blue diamonds denote origin/destination for picking paths. Colored dots denote products and the four orders they belong to. Black crosses denote available locations for the new products. Note that products are often more spread out than what is shown in this example

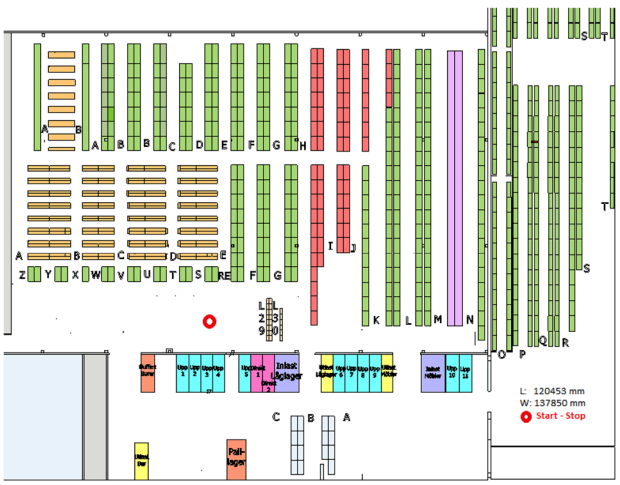

Fig. 4 Top-view of the Aba Skol AB warehouse. The picking zones are color-coded. The red circle denotes the most commonly used depot location

Skol AB). The generated instances use the TSPLIB format [26] with certain amendments for the SLAP, including 6 types of warehouse obstacle layouts, various depot configurations, vehicle capacities and orders (see Fig. 3 for an example of one of the layouts). L17_533 does not include any unidirectional travel rules, meaning that the distance between any two locations is equal both ways. The number of orders range between 4 to 1000 and number of products range between 10 to 3000. The products that are to be assigned a location, $\mathcal{P}_s$, are tagged as "SKUsToSlot" in the instance set. The "assignmentOptions" includes the available empty locations and how cost is to be computed (it is always set to the "empty storage location" scenario). For analysis, instances are categorized according to vehicle capacities, number of orders, products and parameters N and λ.

The industrial warehouse dataset (Fig. 4) contains 210,277 products in 37,014 orders collected using batch picking over a 4-month period. There are 1289 pick-locations (in the graph representation) and most batches exist within one of six picking zones, but 24.4% include picks from several zones. As with the generated instances, shortest distances and paths between any two locations are assumed equal. For a proof of concept, we select product subsets from this data to be of relevance to warehouse management and real-world utility, on the one hand, and comparability to the generated instances, on the other. We build 150 subsets from 3-week periods with selections of between 50–1800 products for $\mathcal{P}$ and between 10 and 225 corresponding products for $\mathcal{P}_s$. The subset selection is random apart from that the products in a subset must exist within the same 3 week period. Number of free locations is given on a *per-product* basis, since each product has specific constraints regarding where it can be placed, and on average it varies between 50 – 481 locations. For parameters N and λ, we explore suitable values on the generated instances within shorter optimization runs, followed by longer runs with chosen constants on the real dataset.

Experiments

Overview and Constants

The experiments are divided into two parts. The first part involves tuning the QAP model and comparing its ability to rank SLAP assignment samples against an OBP ground truth model and a random baseline (Fig. 5).

A SLAP test-instance (orders with products) is first loaded (i) and x_1 initialized (products $\mathcal{P}_s$ are assigned free locations in x_1 randomly) (ii). Then, N location assignments, X_{i+1}, are generated according to Algorithm 2 (iii). The cost of the generated assignments is estimated using the QAP model and the OBP optimizer SBI (iv). The samples and costs are used to compute IDCG and DCG (v). IDCG is computed from the ranking of costs according to the OBP optimizer and DCG is computed from the ranking of costs according to the QAP model. A random DCG value is also pre-computed using the average of 10^6 random rankings. This random baseline represents

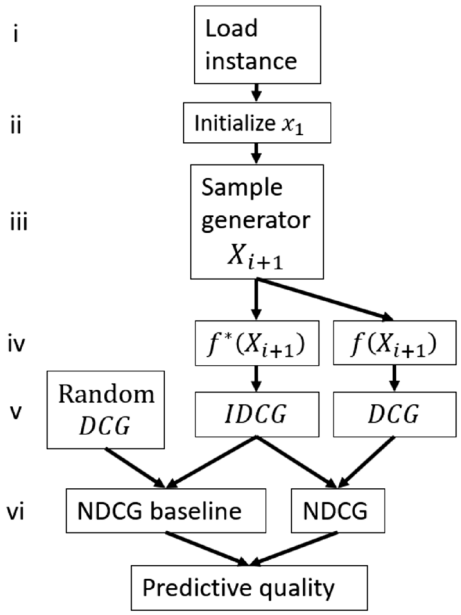

Fig. 5 Steps involved to obtain QAP predictive quality on samples generated from an instance

the case when $f(X_{i+1})$ and $argmin_{x+1} f(X_{i+1})$ (steps iii and iv in Fig. 3) cannot help produce a lower value in $f^*(x_{i+1})$ (step v) [11, 12]. Relevance values $rel(\pi_{f^*(X)})$ and $rel(\pi_{f(X)})$ are chosen to be the ordinal ranks of samples x according to respective cost functions. For N samples, the values are $rel(\pi_{f^*(X)}) = (\pi_{f^*(X)}(N), \pi_{f^*(X)}(N-1), \ldots, \pi_{f^*(X)}(1))$ and $rel(\pi_{f(X)}) = (\pi_{f(X)}(N), \pi_{f(X)}(N-1), \ldots, \pi_{f(X)}(1))$ (this corresponds to the set up shown in Fig. 13 in Appendix). The DCG value obtained from the QAP model is then used to compute NDCG according to Eq. 9 (vi). The predictive quality is finally calculated by subtracting the achieved NDCG value with the random NDCG baseline, with a positive value implying that the QAP model is stronger. We also record the CPU-time needed for the QAP model and the OBP-optimizer, respectively. The tuning of the QAP model concerns parameters N (number of samples) and λ (rate of change for the samples) to maximize NDCG. We further investigate whether NDCG is impacted by other factors, including warehouse layout and instance size. Instance size is used to provide a quantification of instance difficulty, and here we restrict it to number of orders, total number of products $|\mathcal{P}|$ and products which are to be assigned a location $|\mathcal{P}_s|$. The latter number, $|\mathcal{P}_s|$, is computed as 5–10% of $|\mathcal{P}|$ in the instance.

We proceed with a second experiments part, where we run the SLAP optimizer (Algorithm 1) on the industrial instances with and without the QAP model. For the experiments without the QAP model, $N = 1$ and lines 11 and 12 in Algorithm 1 are removed. This second part is carried out after suitable constants for N and λ values have been found on the L17_533. In order to find such constants, we run the steps in Fig. 5 for 10 N values ranged between 1–200 and 10 λ values set between 5–50% of $|\mathcal{P}_s|$. For the experiments to test N, we use λ = 15% of $|\mathcal{P}_s|$. For the experiment to test λ, we use $N = 50$. For the cost distance function Δ we use a scaled sigmoid, which is set to approach 1 when the ratio $f^*(x_i)/f^*(x_{i+1})$ exceeds 1.05. This means that sample x_{i+1} is unlikely to be accepted if its cost is 5% higher than that of x_i. For each instance, the global best OBP result is tracked and uploaded as the current best result. We refer to the documentation in L17_533 for further details. We use Intel Core i7-4710MQ 2.5 GZ 4 cores, 32 GB RAM, Python3, Cython and C.

Results

The Impact of Parameters *N* and λ on QAP Predictive Quality

Concerning N, we first observe that the average predictive quality of the QAP model is equivalent to the random baseline when $N = 1$ (Fig. 6). We further observe that mean predictive quality rises steadily until N is 20, after which it tapers off.

The result clearly shows that the QAP model is able to rank samples better than the random baseline (negative values imply the opposite). The positive initial trend could be impacted by the choice of ordinal relevance values $rel(\pi_{f(X)})$ for the NDCG computation (section "Overview and Constants"), which could favour the baseline for smaller N.

Concerning rate of change of new samples λ, the best results are achieved when it is set toward the lower end of the 5–50% range of $|\mathcal{P}_S|$ (Fig. 7). This provides some validation

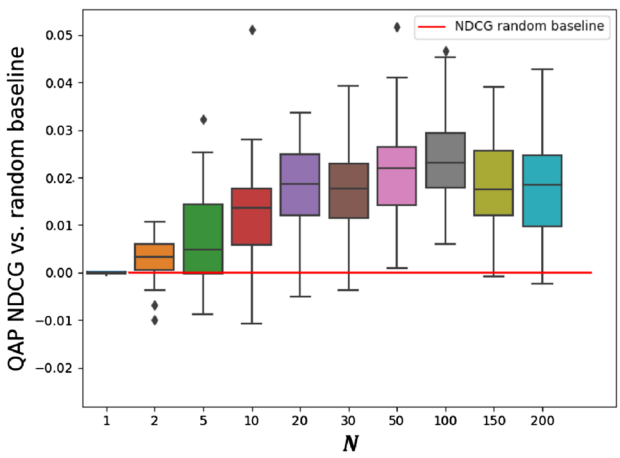

Fig. 6 Boxplot showing number of samples (N) against QAP predictive quality. The red line denotes the NDCG random baseline. The box edges show the first and third quartiles of the data (Q1, Q3) and the whiskers show (Q1 – 1.5 * IQR, Q3 + 1.5 * IQR), where IQR is the Inter Quartile Range

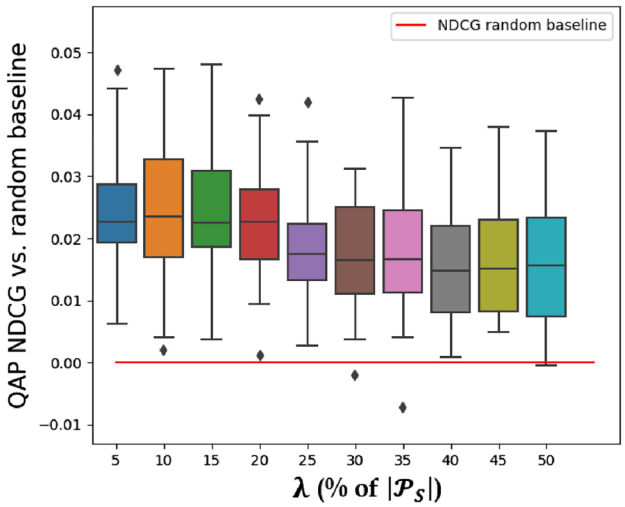

Fig. 7 How much new samples are changed compared to previous samples (λ) against QAP predictive power

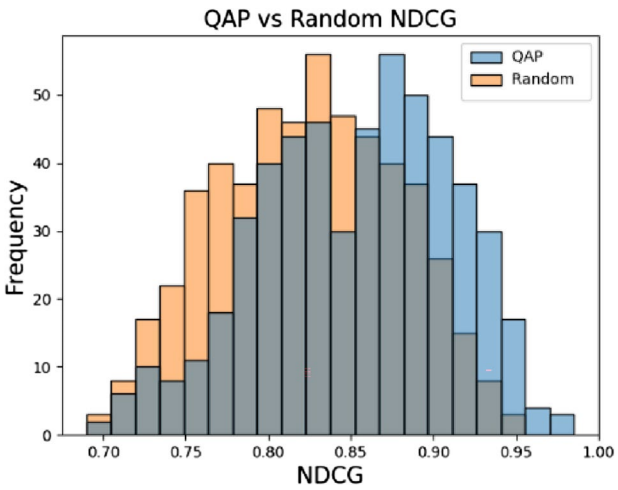

Fig. 9 Frequency distribution of NDCG values (20 bins) from QAP and random ranking of samples when $N = 20$ and $\lambda = 10\%$ (of $|\mathcal{P}_S|$)

for the use of a Metropolis algorithm, since it shows that a Markov Chain can be used to nudge samples closer towards lower costs. Otherwise, NDCG would be similar regardless of the x-axis in Fig. 7. This result is in line with Oxenstierna et al. [23], where a slightly stronger pattern is observed on the related TSP-based SLAP.

The Impact of Other Factors on QAP Predictive Quality

Results for all factors are shown in Tables 1, 2 and 3 (Appendix). We find that QAP predictive quality decreases as instance size increases (Fig. 8). This may be due to that the quality of $f^*(x)$ costs provided by the OBP optimizer

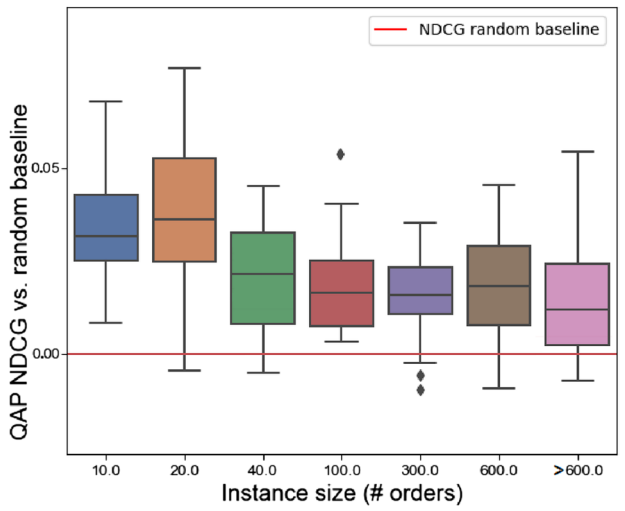

Fig. 8 Instance size in terms of number of orders, versus the predictive quality of the QAP model and the random baseline

decrease with instance size (they are sub-optimal, see section "Promote and Accept Thresholds and Cost Computations"), making analysis of results for larger instance classes more difficult in general. We find that the fraction of CPU-time required by the QAP model versus the OBP optimizer is between 0.006–0.019, or around 50–150 times faster. The difference is largest for the largest instances and smallest for the smallest instances (Table 2). We do not observe any relationship between QAP predictive quality and warehouse layout.

Overall, the result provides evidence that QAP approximations of OBP costs within an OBP-based SLAP optimizer may be justified. Its predictive quality may decrease with instance size, relative to the OBP optimizer (Fig. 7), but its relative usage of CPU-time also decreases. Another way to visualize the performance difference between the QAP model and the random baseline is through a frequency distribution (Fig. 9).

SLAP Optimization With and Without QAP Approximation

We report results from running the QAP-OBP SLAP optimizer (section "Optimization Algorithm") on the industrial dataset with and without the use of QAP approximations. Apart from general settings (section "Overview and Constants"), K is set to 10^8 and the algorithm is set to terminate after 60 min (which, given maximum OBP and QAP CPU-times, ensures iterations never exceed K). λ is set to 10% of $|\mathcal{P}_S|$ and $c_1 = 1$. N is set to 20, which means that the QAP model will have a relatively small impact on overall CPU-time. N could theoretically be set to a much larger number, but this may not necessarily yield

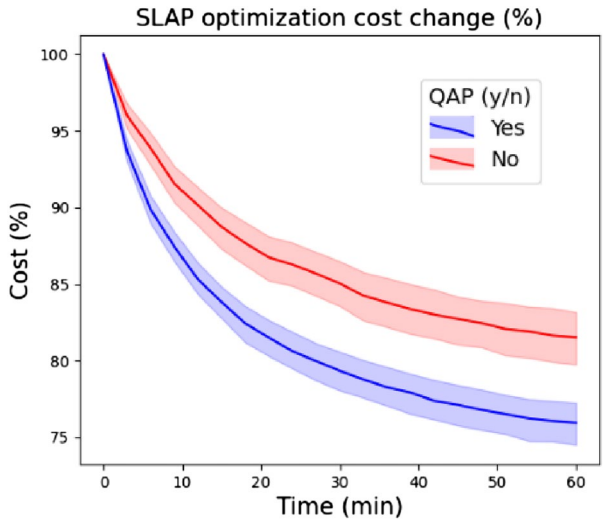

Fig. 10 SLAP optimization cost improvements with and without the QAP model during 1 h. The shaded areas denote 95% confidence intervals

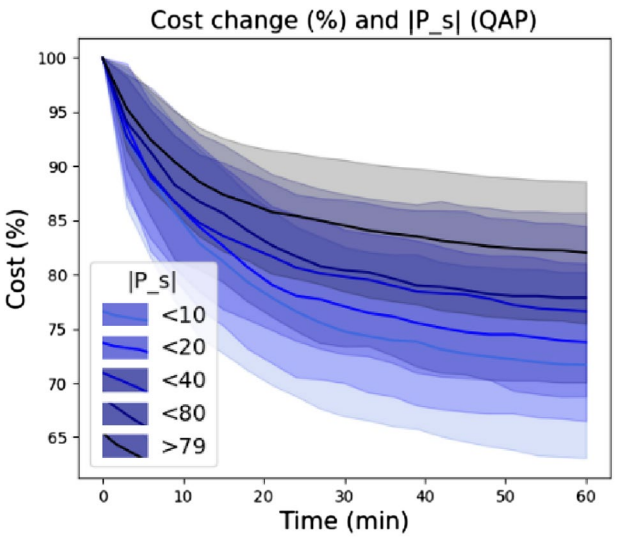

Fig. 11 QAP-OBP SLAP cost improvement using QAP approximations for 5 categories of instance sizes (in terms of $|\mathcal{P}_s|$). Shaded areas denote data within 1 standard deviation

better results. The QAP model in the form of Eq. 5 likely needs to be further developed before its extended use can be motivated. One risk with setting N to a large number is that the SLAP optimizer will spend too much time in search regions with a low QAP cost, rather than in regions with a low OBP cost.

In Fig. 10, we see that Algorithm 1, on average, improves cost by around 23% in 1 h. Without QAP approximations, cost improves by around 17%.

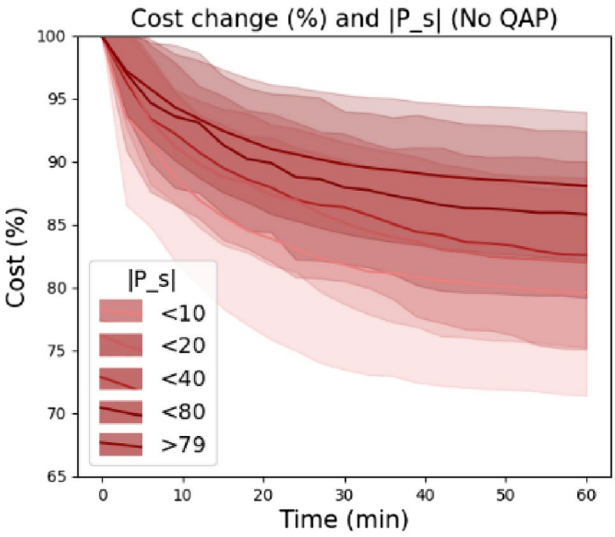

Fig. 12 Same as Fig. 11, but without using QAP approximations

The size of the instances has a significant impact on computational efficiency. In Figs. 11 and 12, we see that the impact of instance size, in terms of number of products that are assigned a location, $|\mathcal{P}_s|$, has a similar effect on computational efficiency regardless of whether the QAP is used. The stronger performance of the smaller instances can largely be attributed to more samples being generated within the 60 min. On average, cost improvement continues throughout the time, which is explainable due to the large SLAP search space.

Conclusion

In this paper, we:

- formulate an optimization model for the Storage Location Assignment Problem (SLAP), where the costs of assignments are evaluated using Order Batching Problem (OBP) optimization.
- share generated SLAP test instances, with the goal to standardize formats and comparability between solution approaches.
- propose a Quadratic Assignment Problem (QAP) model to quickly approximate OBP costs in SLAP optimization. The QAP model is tested and tuned on the generated instances.
- propose a SLAP optimizer (QAP-OBP), which we test on industrial instances with a 1 h optimization timeout.

Within the QAP-OBP optimizer, the QAP and OBP modules are utilized in a Metropolis algorithm, where samples are modified by a variable amount each iteration. The

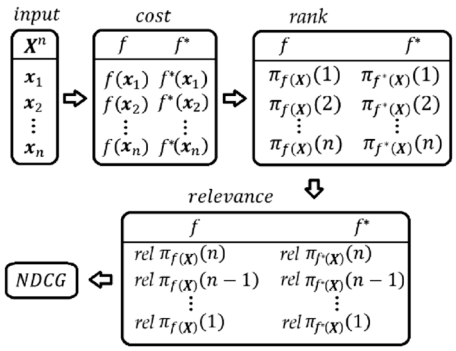

Fig. 13 NDCG procedure flowchart

algorithm is nested such that OBP costs are only computed for samples with a relatively strong QAP cost approximation.

In order to motive the use of the QAP model within the algorithm, experiments are first conducted to test its predictive quality against costs obtained by the OBP optimizer and a random baseline. Results show that QAP predictive quality is stronger than the baseline, and that they are around 50–150 times faster to compute than the cost obtained through OBP optimization.

We then proceed to run the SLAP optimizer with and without the QAP approximations. We find that the optimizer performs better when using the QAP approximations, with cost improvements of around 23% after 1 h. This result is in line with results in related work on SLAPs that are less difficult in some regards (for example concerning warehouse layouts), but more difficult in others (dynamicity or larger number of products).

For future work, the parameter which controls the number of samples that should be approximated by the QAP model for every OBP cost computation, N, could be tuned. The QAP computations could be significantly sped up by the use of parallelization and Graphical Processing Units (GPU), extending its utility within the SLAP optimizer for larger N. Also, alternative optimization approaches could be explored. These include meta-heuristic techniques such as Simulated Annealing or Particle Swarm Optimization. The QAP cost approximator could also be developed for a Machine Learning approach and used in a similar fashion as the weak estimators in boosting or aggregate bootstrapping. The factorial search space remains a fundamental problem for learning, however. Finally, we invite discussions into how to best represent SLAP features in public benchmark data and which features to choose for a standardized version of the problem.

Appendix

NDCG flowchart: the below example shows how Normalized Discounted Cumulative Gain (NDCG) can be computed from input permutations (products to locations), approximated (f) and ground truth (f^*) values. Note that $f(X)$ denotes a sorting of X according to the cost valuation of elements in the *cost* step. Also note that relevance values can be formulated in several ways.

Table 1 Summary of instances and results for different types of warehouse layouts

	NoObstacles	SingleRack		NR1		L9_251
	Conventional	TwelveRacks		NR2		
Num Orders avg.	41	18	20	14	16	19	548
Num Procucts avg.	128	63	89	49	45	88	464
Num available locations % avg.	10	10	10	10	10	10	5
Num depots	2	2	2	2	2	2	2
CPU-time OBO model avg.	0.086	0.029	0.052	0.03	0.03	0.059	1.72
CPU-time QAP model avg.	0.001	0.0006	0.0008	0.0004	0.0004	0.0008	0.014
NDCG difference between baseline and prediction							
Aggregates for n=(5, 10, 20, 30, 50, 100)							
Grand total avg.	0.03	0.02	0.03	0.02	0.01	0.04	0.01
Std. avg.	0.02	0.02	0.01	0.01	0.01	0.02	0.02
Vehicle Capacity Orders 2-4	0.01	0.02	0.03	0.01	0.02	0.05	0.01
4-6	0.01	-	-	0.04	-0.01	0.02	0.02
6-8	0.05	-	-	0.01	0.00	0.04	-0.01
>8	0.03	-	-	-	-	-	0.01
Num orders 5-20	0.03	0.00	0.02	0.01	-0.01	0.05	-
20-40	0.03	0.02	0.05	0.02	0.03	0.03	-
100	0.03	-	-	-	-	-	0.03
500	0.02	-	-	-	-	-	0.01
>500	-	-	-	-	-	-	0.01
Num products 10-20	0.04	0.04	0.02	0.02	-0.01	0.06	-
20-100	0.03	0.01	0.04	0.02	0.02	0.03	-
100-500	0.01	-	-	-	-	-	0.02
500-1000	-	-	-	-	-	-	0.04
>1000	-	-	-	-	-	-	0.01

Also an aggregate of the results concerning the predictive quality of the QAP model

Table 2 Summary of results with regard to instance size

num_products	num_orders	num_samples	λ (% of $\lvert\mathcal{P}_S\rvert$)	N	CPU-time OBP (s)	CPU-time QAP (s)	$\lvert\mathcal{P}_S\rvert$ (%)	Impr. $f(x_1)$	NDCG performance versus baseline
8	5.00	5021.74	22.08	54.88	0.031	0.001	10.00	16.07	0.031
14	9.13	11203.65	22.81	54.95	0.120	0.002	10.00	15.70	0.022
23	11.82	29037.20	25.98	55.00	0.023	0.001	10.00	15.40	0.037
32	15.45	21914.16	25.97	55.24	0.170	0.003	10.00	16.14	0.029
41	18.87	19745.97	26.00	55.00	0.165	0.002	10.00	15.80	0.028
50	23.46	19970.10	26.00	55.00	0.202	0.003	10.00	16.51	0.028
60	23.74	15425.63	25.86	55.03	0.311	0.005	10.00	13.11	0.047
69	25.53	11653.50	26.18	55.00	0.421	0.006	10.00	14.98	0.035
78	24.93	8744.61	26.04	55.30	0.404	0.005	10.00	16.60	0.026
87	29.16	10957.86	25.85	55.09	0.538	0.007	10.00	14.30	0.026
96	29.60	11323.38	26.07	54.52	0.524	0.006	10.00	12.00	0.026
106	24.17	10505.19	26.00	55.00	0.587	0.007	10.00	14.65	0.018
115	28.00	10280.07	25.96	54.97	0.631	0.007	10.00	11.35	0.037
124	33.00	18550.38	26.06	55.00	0.704	0.008	10.00	12.04	0.024
133	30.33	18496.34	26.05	54.51	0.568	0.006	10.00	12.76	0.024
142	33.00	31796.75	26.00	55.06	0.789	0.007	10.00	13.51	0.023
151	34.75	21580.00	25.85	55.41	0.864	0.008	10.00	9.18	0.029
161	38.67	13668.20	25.93	54.52	1.427	0.014	10.00	9.90	0.022
170	38.33	20419.26	26.00	55.24	1.014	0.009	9.12	11.57	0.021
179	32.75	10290.98	25.99	55.37	1.057	0.010	8.94	12.30	0.025
188	34.67	19598.84	25.96	55.00	1.528	0.013	8.56	13.03	0.020
197	32.00	9319.42	25.91	55.24	0.882	0.007	7.08	9.62	0.019
216	44.00	9140.06	26.00	54.60	1.751	0.014	6.98	10.40	0.010
225	37.50	11584.69	25.85	54.76	1.274	0.010	6.07	12.00	0.033
234	36.00	11511.74	26.00	55.22	2.173	0.016	4.99	11.80	0.011
252	39.00	9913.05	25.99	55.31	2.234	0.017	5.00	7.59	0.018
262	44.00	8519.34	25.85	54.80	2.642	0.020	4.94	7.14	0.017
289	39.50	9355.62	26.09	54.55	2.552	0.019	4.91	8.90	0.017
317	161.00	9672.15	25.80	55.00	2.427	0.017	5.00	6.57	0.026
326	162.00	8000.55	26.00	55.15	2.657	0.019	5.10	5.26	0.019
381	206.00	7495.94	27.19	55.17	2.592	0.018	5.05	6.00	0.015
390	207.00	8630.29	26.14	55.00	2.671	0.017	4.95	5.70	0.011
399	205.00	9639.09	26.45	54.63	2.393	0.015	5.00	4.44	0.014
418	210.50	8200.00	26.98	55.00	2.727	0.016	5.00	5.10	0.024
427	222.00	7546.00	26.67	54.55	2.726	0.016	5.07	4.80	0.013

These results exclude instances with more than 435products. All values are averages over instances with a certain number of products (num_products)

Table 3 Results on 60 min optimization runs

P_s	num_products	num_orders	CPU-time OBP Mean	CPU-time OBP Std	CPU-time QAP Mean	CPU-time QAP Std	Cost improvements (starting at 100%) at 20, 40 and 60 minutes using QAP (left) and not using QAP (right)					
11	104.46	22.61	0.05	0.02	0.001	0.001	80.09	74.55	71.02	86.69	80.80	76.76
16	150.10	40.29	0.09	0.08	0.001	0.001	82.91	77.77	73.96	86.01	81.61	79.00
21	178.05	57.62	0.13	0.09	0.002	0.001	81.02	75.66	73.54	87.83	82.27	79.42
26	244.86	56.95	0.20	0.08	0.002	0.001	83.63	75.76	72.53	88.94	84.60	80.82
31	278.14	144.71	0.32	0.21	0.002	0.000	82.76	77.81	74.20	88.75	83.42	81.98
36	362.20	196.95	0.34	0.09	0.003	0.002	85.37	78.61	76.09	89.66	88.05	85.39
41	444.35	239.29	0.48	0.21	0.004	0.001	82.33	74.37	70.19	91.87	83.43	81.79
46	428.29	269.85	0.49	0.12	0.004	0.001	83.03	80.15	77.92	89.33	84.83	82.50
51	470.26	276.92	0.69	0.31	0.005	0.004	85.30	81.27	79.40	88.88	85.68	83.37
56	539.26	338.38	0.75	0.36	0.005	0.002	83.23	80.41	75.53	89.01	86.01	83.77
61	546.20	331.63	0.70	0.21	0.006	0.003	86.85	83.15	80.87	88.72	85.01	81.98
66	640.93	416.04	1.09	1.13	0.007	0.003	83.96	78.54	75.55	90.22	86.51	85.09
71	673.02	427.68	1.05	0.28	0.007	0.003	82.98	77.39	74.44	89.51	86.43	85.00
76	699.86	380.58	1.09	0.34	0.009	0.006	85.05	80.76	78.98	87.42	84.84	82.05
81	787.48	405.63	1.16	0.30	0.009	0.004	82.96	78.28	75.77	92.84	89.89	88.58
86	770.31	434.21	1.13	0.41	0.010	0.004	87.32	82.69	79.06	89.71	85.25	82.91
91	919.05	607.36	1.68	0.62	0.012	0.006	90.08	89.08	85.66	89.68	83.39	82.30
96	912.42	515.95	1.53	0.48	0.012	0.004	86.84	83.13	81.29	89.15	84.57	82.55
101	917.49	566.34	1.64	0.82	0.012	0.005	84.79	82.14	80.11	92.82	88.33	85.33
106	949.39	567.17	1.66	0.51	0.013	0.006	85.80	82.62	79.47	88.27	84.05	81.20
111	1023.24	631.35	1.76	0.39	0.014	0.005	84.35	78.47	74.59	91.22	85.68	83.50
116	1043.54	559.44	1.92	0.76	0.014	0.005	85.56	81.51	80.51	92.50	87.85	86.51
121	1106.57	612.55	1.95	0.52	0.016	0.006	85.21	81.16	78.50	89.58	87.67	85.84
126	1151.81	610.52	1.99	0.42	0.017	0.006	87.32	84.13	83.13	91.30	87.66	84.82
131	1186.47	670.41	2.13	0.44	0.018	0.006	83.13	80.57	78.97	91.58	86.88	84.73
136	1265.07	741.22	2.26	0.47	0.020	0.008	88.33	83.93	81.70	90.50	86.06	83.93
141	1284.32	832.35	2.42	0.22	0.020	0.006	84.03	77.73	74.07	90.46	87.84	85.83
146	1348.17	736.48	2.43	0.70	0.021	0.003	84.03	75.75	74.30	89.77	87.39	86.31
151	1373.92	816.92	2.77	0.42	0.022	0.007	87.94	83.77	77.67	91.30	88.70	86.76
156	1352.87	797.75	2.48	0.52	0.021	0.006	87.99	86.20	84.51	90.02	88.48	87.43
161	1407.88	793.72	2.79	0.56	0.022	0.007	83.97	78.49	74.63	91.40	87.29	85.13
166	1462.75	720.25	2.96	1.20	0.023	0.005	81.70	77.92	75.87	89.61	85.19	84.19
171	1657.88	826.32	2.85	0.42	0.029	0.008	87.25	83.74	82.74	94.56	88.99	87.99
176	1565.06	821.50	2.95	0.39	0.027	0.007	86.95	84.11	80.61	95.82	90.60	89.60
181	1501.56	868.61	2.92	0.30	0.026	0.008	87.56	86.56	83.43	84.92	83.27	82.27
186	1596.48	801.25	3.82	3.67	0.027	0.006	84.91	80.92	77.31	92.59	91.55	87.35
191	1777.02	887.87	3.57	0.73	0.032	0.009	88.37	81.85	77.94	93.43	88.74	87.65
196	1562.60	794.37	2.75	0.19	0.028	0.001	87.99	86.99	85.99	81.05	77.50	76.50
201	1855.64	932.25	3.20	0.32	0.031	0.001	92.85	81.41	80.41	98.00	79.73	77.78
206	1808.08	920.67	3.41	0.41	0.032	0.009	90.44	82.14	81.14	91.85	87.87	86.87
211	1743.06	854.71	3.78	0.78	0.037	0.011	88.04	82.46	80.37	91.73	86.22	83.41
221	1782.87	883.77	4.17	0.79	0.033	0.008	83.68	77.54	74.95	91.47	84.01	81.03

Acknowledgements This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We also convey thanks to Kairos Logic AB for software.

Funding Open access funding provided by Lund University. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP).

Declarations

Conflict of interest The authors declare that they have no conflict of interest. This article does not contain any studies with human participants or animals performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Abdel-Basset M, Manogaran G, Rashad H, et al. A comprehensive review of quadratic assignment problem: variants, hybrids and applications. J Ambient Intell Human Comput. 2018. https://doi.org/10.1007/s12652-018-0917-x.
2. Aerts B, Cornelissens T, Sörensen K. The joint order batching and picker routing problem: modelled and solved as a clustered vehicle routing problem. Comput Oper Res. 2021;129: 105168. https://doi.org/10.1016/j.cor.2020.105168.
3. Azadeh K, De Koster R, Roy D. Robotized warehouse systems: developments and research opportunities. ERIM report series research in management Erasmus Research Institute of Management. ERS-2017-009-LIS. 2017.
4. Bruch S, Wang X, Bendersky M, Najork M. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In: Proceedings of the 2019 ACM SIGIR international conference on the theory of information retrieval (ICTIR 2019). 2019. pp. 75–8.
5. Cao Z, Qin T, Liu T-Y, Tsai M-F, Li H. Learning to rank: from pairwise approach to listwise approach. In: Proceedings of the 24th international conference on machine learning, vol. 227. 2007. pp. 129–36. https://doi.org/10.1145/1273496.1273513.
6. Cardona LF, Rivera L, Martínez HJ. Analytical study of the fishbone warehouse layout. Int J Log Res Appl. 2012;15(6):365–88.
7. Charris E, Rojas-Reyes J, Montoya-Torres J. The storage location assignment problem: a literature review. Int J Ind Eng Comput. 2018;10.
8. Christen JA, Fox C. Markov Chain Monte Carlo using an approximation. J Comput Graph Stat. 2005;14(4):795–810.
9. Ene S, Öztürk N. Storage location assignment and order picking optimization in the automotive industry. Int J Adv Manuf Technol. 2011;60:1–11. https://doi.org/10.1007/s00170-011-3593-y.
10. Fontana ME, Nepomuceno VS. Multi-criteria approach for products classification and their storage location assignment. Int J Adv Manuf Technol. 2017;88(9):3205–16.
11. Freund Y, Iyer R, Schapire RE, Singer Y. An efficient boosting algorithm for combining preferences. J Mach Learn Res. 2003;4(Nov):933–69.
12. Freund Y, Schapire RE. Experiments with a new boosting algorithm. 1996.
13. Garfinkel M. Minimizing multi-zone orders in the correlated storage assignment problem. School of Industrial and Systems Engineering, Georgia Institute of Technology. 2005.
14. Henn S, Wäscher G. Tabu search heuristics for the order batching problem in manual order picking systems. Eur J Oper Res. 2012;222(3):484–94.
15. Kallina C, Lynn J. Application of the cube-per-order index rule for stock location in a distribution warehouse. Interfaces. 1976;7(1):37–46.
16. Kofler M, Beham A, Wagner S, Affenzeller M. Affinity based slotting in warehouses with dynamic order patterns. Advanced methods and applications in computational intelligence. 2014. pp. 123–43.
17. de Koster R, Le-Duc T, Roodbergen KJ. Design and control of warehouse order picking: a literature review. Eur J Oper Res. 2007;182(2):481–501.
18. Kübler P, Glock CH, Bauernhansl T. A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses. Comput Ind Eng. 2020;147: 106645.
19. Larco JA, de Koster R, Roodbergen KJ, Dul J. Managing warehouse efficiency and worker discomfort through enhanced storage assignment decisions. Int J Prod Res. 2017;55(21):6407–22. https://doi.org/10.1080/00207543.2016.1165880.
20. Lee IG, Chung SH, Yoon SW. Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations. Comput Ind Eng. 2020;139: 106129. https://doi.org/10.1016/j.cie.2019.106129.
21. Mantel R, Schuur P, Heragu S. Order oriented slotting: a new assignment strategy for warehouses. Eur J Ind Eng. 2007;1:301–16.
22. Oxenstierna J, Malec J, Krueger V. Efficient order batching optimization using seed heuristics and the metropolis algorithm. SN Comput Sci. 2022;4(2):107.
23. Oxenstierna J, Rensburg L, Stuckey P, Krueger V. Storage assignment using nested annealing and hamming distances. In: Proceedings of the 12th international conference on operations research and enterprise systems—ICORES. 2023. pp. 94–105. https://doi.org/10.5220/0011785100003396.
24. Oxenstierna J, van Rensburg LJ, Malec J, Krueger V. Formulation of a layout-agnostic order batching problem. In: Dorronsoro B, Amodeo L, Pavone M, Ruiz P, editors. Optimization and learning. Berlin: Springer International Publishing; 2021. p. 216–26.
25. Ratliff H, Rosenthal A. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Oper Res. 1983;31:507–21.
26. Reinelt G. TSPLIB—a traveling salesman problem library. INFORMS J Comput. 1991;3:376–84.
27. Rensburg LJ. Artificial intelligence for warehouse picking optimization—an NP-hard problem [Master's Thesis]. Uppsala University. 2019.
28. Roodbergen KJ, Koster R. Routing methods for warehouses with multiple cross aisles. Int J Prod Res. 2001;39(9):1865–83.
29. van Ravenzwaaij D, Cassey P, Brown SD. A simple introduction to Markov Chain Monte-Carlo sampling. Psychon Bull Rev. 2018;25(1):143–54. https://doi.org/10.3758/s13423-016-1015-8.

30. Wu J, Qin T, Chen J, Si H, Lin K. Slotting optimization algorithm of the stereo warehouse. In: Proceedings of the 2012 2nd international conference on computer and information application (ICCIA 2012). 2014. pp. 128–32. https://doi.org/10.2991/iccia.2012.31.
31. Wu X, LuWuZhou JSX. Synchronizing time-dependent transportation services: reformulation and solution algorithm using quadratic assignment problem. Transport Res Part B Methodol. 2021;152:140–79. https://doi.org/10.1016/j.trb.2021.08.008.
32. Wutthisirisart P, Noble JS, Chang CA. A two-phased heuristic for relation-based item location. Comput Ind Eng. 2015;82:94–102. https://doi.org/10.1016/j.cie.2015.01.020.
33. Yang N et al. Evaluation of the joint impact of the storage assignment and order batching in mobile-pod warehouse systems. Math Probl Eng. 2022;2022.
34. Yingde L, Smith JS. Dynamic slotting optimization based on SKUs correlations in a zone-based wave-picking system. In: IMHRC proceedings, vol. 12. 2012.
35. Zhang R-Q, Wang M, Pan X. New model of the storage location assignment problem considering demand correlation pattern. Comput Ind Eng. 2019;129:210–9. https://doi.org/10.1016/j.cie.2019.01.027.
36. Zhou F, De la Torre F. Factorized graph matching. IEEE Trans Pattern Anal Mach Intell. 2016;38(9):1774–89. https://doi.org/10.1109/TPAMI.2015.2501802.
37. Žulj I, Glock CH, Grosse EH, Schneider M. Picker routing and storage-assignment strategies for precedence-constrained order picking. Comput Ind Eng. 2018;123:338–47. https://doi.org/10.1016/j.cie.2018.06.015.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.