Storage Assignment using Nested Annealing and Hamming Distances

Johan Oxenstierna[1,4][a], Louis Janse van Rensburg[3], Peter J.Stuckey[2][b] and Volker Krueger[1][c]
[1]*Dept. of Computer Science, Lund University, Lund, Sweden*
{johan.oxenstierna, volker.krueger}@cs.lth.se
[2]*Faculty of Information Technology, Monash University, Australia*
peter.stuckey@monash.edu
[3]*Optisol, 90 Sippy Downs Dr, QLD, Australia*
louis.jvr@optisol.com.au
[4]*Kairos Logic AB, Lund, Sweden*

Keywords: Storage Location Assignment Problem, Nested Annealing, Hamming Distances.

Abstract: The assignment of products to storage locations significantly impacts the efficiency of warehouse operations. We propose a multi-phase optimizer for a Storage Location Assignment Problem (SLAP) where solution quality is based on a distance estimate of future-forecasted order picking. Candidate assignments are first sampled using a Markov Chain accept/reject method. Future-forecasted pick-rounds are then modified according to the candidate assignments and solved as Traveling Salesman Problems (TSP). The model is graph-based and generalizes to any obstacle layout in 2D. Due to the intractability of the SLAP, methods are proposed to speed up search for strong solution candidates. These include usage of fast function approximation to find potentially strong samples, as well as restarts from local minima. Results show that these methods improve performance and that total travel distance can be reduced by as much as 30% within 8 hours of CPU-time. We share a public repository with SLAP instances and corresponding benchmark results on the generalizable TSPLIB format.

1 INTRODUCTION

The Storage Location Assignment Problem (SLAP) concerns the choice of locations for products in a warehouse. There are dozens of proposed versions and optimization methods for the SLAP (Charris et al., 2018). In this paper we consider SLAP optimization for a standard picker-to-parts scenario where obstacles can be laid out freely on a 2D plane and where vehicles (human-controlled or autonomous) may start and end their paths at any location. A candidate solution to the SLAP is an assignment of products to locations. We define the quality of a candidate solution as the aggregate travel distance needed to complete a given *picking-log*, i.e., a set of pick-rounds (sequences of product visits), added to the *reassignment distance* needed to move products to locations specified in the candidate solution. A pick-round is assumed equivalent to a *Steiner* Traveling Salesman Problem (TSP) (Valle et al., 2017) where the origin and destination locations may be different and where the same location may be revisited by one

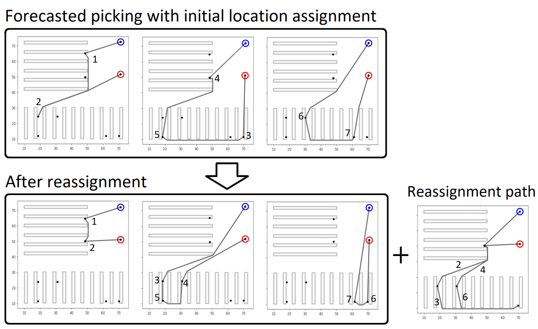

Figure 1: A SLAP example with three pick-rounds (TSP's) and an unconventional obstacle-layout. The initial baseline assignment (top) has a longer picking-log distance compared to a sample/candidate assignment (bottom left). The reassignment path needed to move the products according to the sample (bottom right), is longer than any possible savings concerning the picking-log, however (more pick-rounds are needed for savings).

or several vehicles. The aggregate TSP distance for a given assignment is obtained by solving all TSP's in the picking-log according to shortest distance. The reassignment distance is obtained by optimizing a sequence of sub-cycles in a single reassignment path. We refer to this model as the TSP-based SLAP.

In Section 2 we discuss existing literature on

[a] https://orcid.org/0000-0002-6608-9621
[b] https://orcid.org/0000000321860459
[c] https://orcid.org/0000-0002-8836-8816

the SLAP and strengths and weaknesses of various models, followed by a formulation of the TSP-based SLAP in Section 4. The proposed model is designed with three main assumptions: 1. The SLAP is *static*, meaning that the whole picking-log is given apriori. 2. The picking-log is limited in size, containing no more than a few hundred products. 3. Products cannot be swapped between pick-rounds. These assumptions can be criticized for simplifying a realistic SLAP. In a realistic SLAP scenario, strong location assignments can be assumed to vary dynamically based on variable demand. Also, there may be tens-of-thousands of products within a certain future-forecasted time-period, instead of a few hundred. Thirdly, the pick-rounds may change their product compositions through the future-forecasted time-period. One argument for the proposed model is that it is *layout-agnostic*, meaning that it makes no assumptions regarding how racks or other obstacles are laid out in the warehouse. Another argument for the model is that it poses a challenging problem even without the stated simplifications: The number of possible assignments of products to locations is factorial with regard to number of products (assuming a one-to-one relationship between products and locations). In order to find a strong assignment, an equilibrium point between two adversarial NP-hard problems must be found: 1. The minimization of TSP's in the picking-log, and 2. the minimization of the reassignment cost needed to move products to their assigned locations. A final argument for the choice of model is the lack of consensus regarding what should and should not be included in a basic version of the SLAP, for example with regard to for benchmark-instances (Charris et al., 2018). The TSP-based SLAP is our proposal for a basic version. We offer new public test instances on the generalizable TSPLIB format (Hahsler and Kurt, 2007) and we invite the community to discuss alternative formulations for a basic version of the SLAP.

In Section 5 we introduce our optimization algorithm. It is based on Simulated Annealing and a Hamming-distance location-swap heuristic. Approximate TSP optimization and restarts from local minima are proposed to improve computational efficiency (cost improvement through CPU-time). In Section 6 we introduce two datasets, including a publicly shared benchmark instance set, and corresponding computational results. All used instances are based on a bi-directional graph, meaning that no uni-directional travel conventions are assumed. Our contributions are summarized as follows:

1. A SLAP optimizer using a novel version of the Simulated Annealing algorithm and experiments to test its computational efficiency.
2. A publicly shared SLAP instance set on the TSPLIB format and corresponding solutions/results.

2 LITERATURE REVIEW

In this section we discuss how the SLAP has been described and optimized in previous work. We particularly refer to the extensive literature review by Charris et al. (2018). There are several strategies for conducting a storage location assignment. These include *Dedicated, Class-based and Random*.

- *Dedicated*: The locations of products are assumed to never change. This strategy is suitable if the collection of products does not change much through time. If human picking is used, this approach has the advantage that pickers can learn to associate products with locations, allowing for speed-ups in picking (Zhang et al., 2019).
- *Random*: Products can be assigned any location in the warehouse. This is particularly suitable if the collection of products changes frequently.
- *Class-based (zoning)*: Each product is assigned a class and the warehouse is divided into zones. Each zone contains one or several classes of products. Class-based storage can incorporate dedicated and random strategies for certain zones and/or classes (Mantel et al., 2007)

The quality of a location assignment can be modeled in several ways. Larco et al. (2017), for a human picking scenario, show that there exists a relationship between the height which products are placed on and worker welfare. Worker welfare can be quantified by estimating parameters such as "ergonomic loading", "discomfort" or "expenditure of human energy" (Charris et al., 2018). For autonomous vehicle or shuttle based storage and retrieval systems (AVS/R) there exists a model which has as objective to minimize "energy consumption" (Azadeh et al., 2019).

Another way to judge solution quality is through datamining, using computations such as support (pick frequency), confidence (affinity) and lift. These can also be used to propose SLAP candidate assignments (Kofler et al., 2014; Ming-Huang Chiang et al., 2014; Zhang et al., 2019). Datamining is primarily focused on the statistical analysis of products and their relationships, but it is often combined with order-picking in a SLAP.

A third proposal studies the effect of traffic congestion. Bottlenecks can be caused if too many products with high pick-frequency are placed close to depot, for example. Lee et al. (2020), propose

Correlated and Traffic Balanced Storage Assignment (C&TBSA), a multi-objective SLAP model which aims to minimize traffic congestion while also minimizing aggregate order -picking cost.

Order-picking has many variations, depending on obstacle layout, picking strategy and travel conventions (Charris et al., 2018; Mantel et al., 2007; Janse van Rensburg, 2019; Yu and Koster, 2009). Concerning obstacle layout, we distinguish between two types: *Conventional* and *Unconventional*. In the conventional layout, warehouse racks are assumed to be organized in Manhattan style blocks with parallel aisles and cross-aisles. Conventional layouts are used in the majority of research on both order-picking and the SLAP (Charris et al., 2018; Koster et al., 2007). The unconventional layout includes the "fishbone" and "cascade" layouts (Cardona et al., 2012; Charris et al., 2018), as well as all other layouts that are not conventional. Regardless of layout, the picking path of a vehicle can be formulated as a Traveling Salesman Problem (TSP) where paths cannot intersect obstacles (Henn and Wäscher, 2012). For conventional layouts, the TSP is often optimized using S-shape or Largest-Gap algorithms (Roodbergen and Koster, 2001). For unconventional layouts, Google OR-tools or Concorde have been proposed (Oxenstierna et al., 2022; Janse van Rensburg, 2019).

If a vehicle picks several orders at a time, an Order Batching Problem (OBP) can be formulated. In the OBP the objective is to assign sets of orders for the vehicles (an order is a set of products and a batch is a set of orders). The OBP can be optimized as a joint problem with the TSP (Gils et al., 2019; Valle et al., 2017). Proposals to use the OBP to estimate SLAP solution quality (OBP-based SLAP) include Kübler et al. (2020) and Xiang et al. (2018). Theoretically, the OBP allows for a strong simulation of travel in the warehouse, since it includes the search for product compositions in batch pick-rounds. Using an OBP within a SLAP also brings noteworthy challenges, however, since the OBP is highly intractable (Briant et al., 2020; Oxenstierna et al., 2022).

If batching is not included in the SLAP, heuristics such as Cube per Order Index (COI) (Kallina and Lynn, 1976) and Order Oriented Slotting (OOS) (Mantel et al., 2007) have been proposed. COI assumes that products with relatively high pick-frequency and low volume should be placed close to depot. COI does not include associations between products and is therefore mainly suitable for pick-rounds with few picks, such as pallet-picking or certain AVS/R systems (Azadeh et al., 2019). OOS, on the other hand, is specifically designed for scenarios where orders may contain more than one product. Mantel et al. (2007) introduce a Quadratic Assignment Problem (QAP) heuristic which computes distances between products and the number of times products appear in the same order. The quality of a candidate location assignment can then be estimated using QAP. Similar methods to OOS are used by Žulj et al. (2018), Fontana and Nepomuceno (2017) and Lee et al. (2020).

The SLAP usecase can be divided into two categories depending on the number of products that are to be moved. "Re-warehousing" is the case when a large proportion of products are moved, whereas a smaller proportion is moved in "healing" (Kofler et al., 2014). Movements can be conducted in many ways, each accompanied by a (re)assignment "effort". Kübler et al. (2020) propose the following (re)assignment effort scenarios:

i Product A is moved to an unoccupied location.

ii Product A swaps location with product B.

iii Product A is moved to a location occupied by product B. Product B is moved to a new location. If there is a product C occupying the new location the procedure continues until a final product is placed at an empty location.

Scenario (i) comes with the least (re)assignment effort and the effort grows through scenarios (ii) and (iii). Apart from travel distance, time used for product removal/placement on shelves and administrative times can be added to the effort computation (Kübler et al., 2020).

When it comes to optimization algorithms for the SLAP, both exact and non-exact methods have been proposed. The exact algorithms include dynamic programming, branch and bound algorithms and Mixed Integer Linear Programming (MILP) (Charris et al., 2018). The SLAP search space is often reduced in scope when exact solutions are sought. These include restricting the number of locations (Wu et al., 2014), number of products (Garfinkel, 2005; Liu, 1999) or by only working with conventional warehouse layouts (Boysen and Stephan, 2013).

More commonly, non-exact heuristic or metaheuristic algorithms are used. Proposals include Particle Swarm Optimization (PSO) (Kübler et al., 2020), Genetic and Evolutionary Algorithms (Ene and Öztürk, 2011; Lee et al., 2020) and Simulated Annealing (Kofler et al., 2014; Zhang et al., 2019). The SLAP is often optimized in multiple phases using these methods. One example is to first generate candidate products for location assignments using datamining, and then evaluate various candidate assignments using order-picking optimization (Kofler et al., 2014; Wutthisirisart et al., 2015).

It is challenging to judge optimization results in previous work due to the multitude of variations in SLAP models (Charris et al., 2018). For results including reassignment costs, conventional warehouse layouts, dynamic picking patterns and meta-heuristic optimization, Kofler et al. (2014) report best savings around 21%. In a similar scenario, Kübler et al. (2020) report best savings around 22%. Excluding reassignment costs, Zhang et al. (2019) report best savings around 18% on simulated data with thousands of product locations, also using Simulated Annealing. In a similar setting, for a few hundred products and using a heuristic two-phase optimizer, Trindade et al. (2022) report best savings around 33%.

3 SIMULATED ANNEALING AND MODIFICATIONS

The proposed optimizer (Section 5) is based on Simulated Annealing (Algorithm 1). A *sample* function draws a sample x_{i+1} based on a desired distance to a previous sample x_i. The distance is given by some probability distribution $q(x_{i+1}|x_i)$, and the distribution is often chosen to be Normal, so that the distance between x_{i+1} and x_i is low with high probability (Mackay, 1998). The $cost^*$ function computes/retrieves the cost (f^*) of the new/previous sample (the first sample is retrieved from memory after the first iteration). The accept probability α^* is based on the solution-space distance function Δ (which outputs a negative value if the new cost is lower than the previous) and a temperature function T. The temperature enforces high variance at the beginning and high bias towards the end of optimization (weak new samples are more often accepted at high temperature) (Rajasekaran and Reif, 1992). Functions for temperature T and Δ are further discussed in Section 5.

The algorithm is a biased random walk and if proportionality between q and f^* is large, the random walk spends more time in regions of local minima. A known disadvantage of this type of Markov Chain Monte Carlo (MCMC) method is that each new sample is correlated to the previous one, risking convergence on weak local minima (Mackay, 1998). Several methods have been proposed to alleviate this problem, such as mode-jumping (Tak et al., 2018), Nested Annealing (Rajasekaran and Reif, 1992) and Basin Hopping (Wales and Doye, 1997). These methods split the search space into regions which are then subjected to local search. Another method is Simulated Annealing with Restart Strategy (SARS), which restarts the algorithm from a random new sample whenever a "non-improving" local minimum is found (Yu et al., 2021).

Algorithm 1 Simulated Annealing

x_i: Sample (candidate solution).
$f^*(x_i)$: Ground truth cost of sample x_i.
$q(x_{i+1}|x_i)$: Probability of distance between two samples.
Δ: Cost distance function.
N: Number of iterations.
T: Temperature function.
x_1: Initial sample (baseline).
for $i = 1,...,N$ **do**
    $t \leftarrow T(i)$
    $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$
    $f^*(x_i), f^*(x_{i+1}) \leftarrow cost^*(x_i, x_{i+1})$
    $\alpha^* \leftarrow exp(-c_1\Delta(f^*(x_{i+1}), f^*(x_i))/t)$
    $u \leftarrow \mathcal{U}(0,1)$ // random uniform
    if $u < \alpha^*$ **then** // sample *accepted*
        $x_i \leftarrow x_{i+1}$
    end if
end for

Christen and Fox (2005), propose a method which can make MCMC algorithms more computationally efficient, given that there exists a cost function f that can provide fast and reasonably accurate cost estimates of f^*. They propose to use f to reject new samples that are unlikely to yield an improvement in f^* over the previous sample. Using their modification, the common MCMC *accept* method is split into two parts: *Promote* (f^* evaluation for a sample with a strong f) and *accept* (update x_i for the next iteration to be a sample with a strong f^*). In the proposed algorithm (Section 5), we make use of this concept and split Simulated Annealing into *promotion* based on fast and less accurate TSP optimization in f and *acceptance* based on slow and more accurate TSP optimization in f^*.

4 PROBLEM FORMULATION

4.1 Objective Function

The objective in the TSP-based SLAP is to minimize the aggregate travel distance to:

1. Complete a given set of pick-rounds $\mathcal{B}$.
2. Carry out any proposed locations reassignments in a single reassignment path $\mathcal{R}$.

Each pick-round $b \in \mathcal{B}$ is a list of products. The set of all locations (including pick-locations, origin and destinations and obstacle corners in 2D Cartesian space) is denoted $\mathcal{L}$ and the set of all pick-locations is denoted $\mathcal{L}(\mathcal{P})$. The set of all products found in $\mathcal{B}$ is denoted $\mathcal{P}$. Each product $p \in \mathcal{P}$ is a tuple consisting of a unique key (Stock Keeping Unit), a location $l(p) \in \mathcal{L}(\mathcal{P})$ and a positive quantity. Each pick location is a tuple consisting of a unique key, a capacity and a location (represented as a node key in a graph). A product is located at strictly one location and a location stores strictly one product. A product is allowed to move from its initial location to a new one as long as the new location's capacity is not exceeded.

A SLAP solution candidate (also referred to as sample or assignment) is represented as permutation vector $x \in X$, where the elements are enumerated product keys and the indices are enumerated location keys. For an example warehouse with 3 locations, sample $x = [2,1,3]$ means that product 2 is assigned location 1, 1 assigned 2 and 3 assigned 3. Each x contains permutation integers in the range $[1,m]$, $2 \leq m \leq |\mathcal{P}|$ and each permutation has ground truth cost f^* (solution value) (see Equation 1). m denotes the number of products that are subject to location change, and it can be set manually to limit the search space (Section 5). Sample x_1 represents the baseline product location assignment (the inital locations of the products). The cost of all subsequent samples will be compared against the initial baseline cost of x_1. The minimization of the picking-log and reassignment distance is as follows,

$$\underset{x}{\operatorname{argmin}}((\sum_{b \in \mathcal{B}} D(b)) + \lambda D(\mathcal{R})) \tag{1}$$

The objective is to find a sample x such that *picking-log* distance $\sum_{b \in \mathcal{B}} D(b)$ and *reassignment* distance $D(\mathcal{R})$ are minimized. The factor λ allows us to weigh the two costs. Below we show how each of them is computed.

4.2 Picking-log distance

The distance of all pick-rounds in picking-log $\mathcal{B}$ is computed as $\sum_{b \in \mathcal{B}} D(b)$. $D(b)$ is the distance of the solution to the Traveling Salesman Problem (TSP) represented by product locations in b: $D(b) = d_{l(origin),l(p_1)} + d_{l(p_{|b|}),l(destination)} + \sum d_{l(p_i),l(p_j)}, j = i+1, 0 < i < |b|$, where $d_{l(p_i),l(p_j)}$ denotes the distance between the locations of $p_i, p_j \in b$, and where $d_{l(origin),l(p_1)}$ connects an origin location and $d_{l(p_{|b|}),l(destination)}$ a destination location to the path. The location of a product $l(p_i)$ is obtained from an index in the location assignment sample x. We assume shortest distances and corresponding shortest paths (needed if visualization is sought) between pairs of locations are queryable from Random Access Memory (RAM). All these shortest distances and paths are precomputed using the Floyd-Warshall algorithm on a bi-directed graph, using a warehouse digitization process beyond the scope of this paper (Janse van Rensburg, 2019). We allow the origin and destination locations in the pick-rounds to be any locations in $\mathcal{L}$ (this is sometimes referred to as Multi-Depot TSP or Dial-a-ride Problem). In Section 5 we describe how TSP optimization works for the multi-depot requirement.

4.3 Reassignment distance

Reassignment path $\mathcal{R}$ and its distance $D(\mathcal{R})$ is based on direct and indirect exchange scenarios (scenarios (ii) and (iii) (Section 2) with the following assumptions: Since there are an equal amount of products and locations in the formulated SLAP, scenarios (ii) and (iii) are a bijection of products and locations. We also assume three enumeration types for the bijection: Direct exchange, e.g. $x_1 = [1,2]$ to $x_2 = [2,1]$ (product 2 goes to location 1 and 1 goes to 2), indirect exchange, e.g. $x_1 = [1,2,3]$ to $x_2 = [3,1,2]$ (1 goes to 2, 2 goes to 3 and 3 goes to 1), or a combination of both. We also assume direct and indirect exchanges can be carried out in a single path without intermediate stops at the depot. Algorithm 2 shows how a single reassignment path can be built and optimized just from information in initial assignment x_1 and subsequent sample x_{i+1} (generated during optimization).

Algorithm 2 Reassignment Path Optimization

x_1: Initial assignment (baseline solution)
x: Sample obtained during optimization
$x_m \leftarrow copy(x)$
$D(\mathcal{R}_{best}) \leftarrow \infty$
for $i = 1, ..., K$ **do** // optimization iterations.
    $\mathcal{R} \leftarrow list()$
    while x_m not_empty **do**
        $r \leftarrow list()$
        while not_completed(r) **do**
            $r.add(x, x_m, x_1)$ // add to sub-cycle
        end while
        $\mathcal{R} += r$
    end while
    shuffle_and_flatten($\mathcal{R}$)
    $D(\mathcal{R}_{best}) \leftarrow$ update_best($\mathcal{R}, \mathcal{R}_{best}$)
end for

r denotes a sub-cycle of locations (sequence that starts and ends at the same location). $r.add(x, x_m, x_1)$ has two cases: 1. If r is empty, a random new element is removed from x_m and its initial location (the index for that product in x_1) is added to r. 2. If r is not empty, the new location of the last added product in r is first found in x and added to r. The product which sits at that "next" location is found in x_1, matched in and then removed from x_m. If the added location to r is equivalent to the first one in r, the sub-cycle is completed and r is added to $\mathcal{R}$. After x_m is emptied, $\mathcal{R}$ is first randomly shuffled and then flattened (the inner lists of subcycles are converted into a single list). The distance $D(\mathcal{R})$ is then computed as the sum of all location to location distances in $\mathcal{R}$, added with the distance from an origin depot location to the first location in $\mathcal{R}$ and the last location in $\mathcal{R}$ to a destination depot location. At each iteration, the update_best($\mathcal{R}, \mathcal{R}_{best}$) function updates the lowest minimum found by comparing distance $D(\mathcal{R})$ and distance $D(\mathcal{R}_{best})$. For Algorithm 1 and Algorithm 3 (below), $D(\mathcal{R})$ is included in the *cost** and *cost* functions.

$\mathcal{R}$ is a solution to a constrained, linked-list TSP where a product is dropped off and another product picked up at each location. The vehicle conducting the reassignment path is assumed to be able to carry the whole quantity of one product. A model of the reassignment path involving vehicle-capacities, enforcing return trips to depot when a product quantity exceeds capacity, is left for future work.

5 OPTIMIZATION ALGORITHM

5.1 SLAP Markov Chain Monte Carlo (MCMC)

We formulate Markovian sampling distribution q which is capable of proposing a distance from a sample x_i to a next sample x_{i+1} such that Equation 1 is minimized. For this to be possible, there must exist a proportionality between the cost expressed in Equation 1 and q. We hypothesize that such proportionality exists between the cost and a q involving a Hamming distance heuristic. Hamming distance measures the distance between permutations and it involves counting of non-identical elements between the permutations (Rathod et al., 2016). The following sampling distribution is then proposed (loosely based on bounds proposed by Christen and Fox (2005)):

$$q(x_{i+1}|x_i) = e^{-CH_d(x_i,x_{i+1})^P} \tag{2}$$

where C and P are hyperparameters in $\mathbb{R}^+$, and H_d denotes Hamming distance between two samples. The Hamming distance gives the number of location changes compared to the previous sample, and the number is determined by the q probability. In the remainder of this section, we propose to use this sampling function within Algorithm 1. We also propose methods which may improve computational efficiency (cost reduction through CPU-time) of Algorithm 1.

5.2 TSP optimization and caching

In order to compute the quality of a SLAP solution candidate, TSP optimization is required. For optimal TSP solutions we use Concorde[1] (Applegate et al., 2002). For approximate TSP solutions we use OR-tools[2] (Kruk, 2018). In order to limit the CPU-time of OR-tools, its *solution_limit* parameter is set to 500, which is the maximum number of candidate TSP solutions that it is allowed to evaluate before terminating. Capability to handle multi-depot scenarios is added by modifying the input distance matrix by adding a dummy location whose distance is zero to the origin and destination, and whose other distances are set to infinite.

Given sampling function q, it is evident that only a subset of the pick-rounds in the picking-log are going to be affected by any given product to location assignment (pick-rounds will often not contain reassigned products). Instead of re-optimizing the same pick-rounds where no products have changed location, we instead cache optimal and approximate cost for each pick-round once computed. For any pick-round, the saved costs are then queried until one or several product locations are changed.

5.3 Nested Annealing

Algorithm 1 can potentially be made more computationally efficient if there exists a function f which can quickly estimate f^* (Section 3). The modification is shown in Algorithm 3.

After a sample x_{i+1} is generated, its cost is estimated using OR-tools. If the sample passes the *promote* filter, *cost** is computed using Concorde. The *cost* and *cost** functions include reassignment distance $D(\mathcal{R})$ (Algorithm 2). Since Algorithm 2 does not guarantee optimality for $D(\mathcal{R})$, *cost** does not guarantee optimality either, and hence we refer to f^* as "more accurate" rather than optimal. Note that hyperparameters $c_1, c_2 \in \mathbb{R}^+$ may be set differently. Christen and Fox (2005) suggest setting $c_1 > c_2$ so that the promotion of a sample is less likely than the acceptance of a promoted sample. The temperature function T is assumed to be a shifted and scaled reverse sigmoid (decreasing) that gives temperatures in range $[1,0]$. The pairwise solution-space distance

[1] https://math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm, collected 27-05-2022.

[2] https://developers.google.com/optimization/routing/tsp, collected 12-06-2022.

Algorithm 3 Nested Annealing (based on computational efficiency in cost estimation)

x_i: Sample (candidate solution)
$f(x_i)$: Less accurate fast cost estimate
$f^*(x_i)$: More accurate slow cost estimate
$q(x_{i+1}|x_i)$: Probability of distance between two samples
α: Probability that sample x_{i+1} is *promoted*
α^*: Probability that sample x_{i+1} is *accepted*
Δ: Cost distance function
N: Number of iterations
T: Temperature function
x_1: Initial assignment sample (baseline)
for $i = 1, ..., N$ **do**
  $t \leftarrow T(i)$
  $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$
  $f(x_i), f(x_{i+1}) \leftarrow cost(x_i, x_{i+1})$
  $\alpha \leftarrow exp(-c_1\Delta(f(x_{i+1}), f(x_i))/t)$
  $u \leftarrow \mathcal{U}(0,1)$ // random uniform
  if $u < \alpha$ **then** // sample *promoted*
    $f^*(x_i), f^*(x_{i+1}) \leftarrow cost^*(x_i, x_{i+1})$
    $\alpha^* \leftarrow exp(-c_2\Delta(f^*(x_{i+1}), f^*(x_i))/t)$
    $u \leftarrow \mathcal{U}(0,1)$
    if $u < \alpha^*$ **then** // sample *accepted*
      $x_i \leftarrow x_{i+1}$
    end if
  end if
end for

function Δ is assumed to be a shifted and scaled sigmoid that gives values in range $[0,1]$. Nested Annealing was first introduced by Rajasekaran and Reif (1992), but they do not use function approximation and base the nesting on variable set temperatures in local search regions. Algorithm 3 provides an alternative nesting strategy, based on a tradeoff between predictive speed and accuracy.

5.4 Restarts

Due to the large search space of the SLAP, the MCMC sampling function $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$, may benefit from occasional restarts (Section 2). Yu et al. (2021), propose restarts from randomly generated samples. Their test-problems do not include reassignment distances, however, and in the SLAP, randomly generated samples can be expected to have a significantly higher cost than x_1, due to reassignment distance $D(\mathcal{R})$. We thus propose restarts from local minima. The best minimum found through optimization is denoted x_{best} and it is used as restart sample with an increasing probability. Forcing restarts from x_{best} is motivated because its local neighbourhood cannot be extensively searched for in any but the smallest SLAP test-instances. Another minimum is denoted x_{lowR} and it is used as restart sample with a decreasing probability (distributions proposed in Section 6). Forcing restarts from x_{lowR} is designed to target a low reassignment distance $D(\mathcal{R})$. The first such local minimum is $x_{lowR} = x_1$, whose $D(\mathcal{R}) = 0$. $x_{lowR} = x_1$ can be assumed to be a strong local minimum, due to its lack of reassignment distance, but after $f^*(x_1)$ has been conclusively beaten, x_{lowR} is updated at regular intervals to a previously generated sample which has a relatively low f^* cost and $D(\mathcal{R})$. In Section 6 we present optimization results with and without restarts from x_{best} and x_{lowR}.

6 Experiments

6.1 Overview

We carry out experiments to investigate the following topics with regard to computational efficiency (cost improvement through CPU-time):

1. Utility of *Hamming-distance* based sampling (q).
2. Utility of *restarts*.
3. Algorithm 1 compared to Algorithm 3.
4. Other features (such as layout and number of products and pick-rounds).

All experiments are carried out using Intel Core i7-4700MQ, 2.40GHz, 4 cores and Python3 (with heavy use of Cython) and C.

6.2 Parameters

For all experiments, the number of products open for location reassignment (m) is set to be equivalent to the number of products in the test-instance. The number of reassignment path optimization iterations (K in Algorithm 2) is set to 300. After optimization has completed, the reassignment path is re-optimized with K set to 10000. The accept probability computation is set to be equivalent between Algorithm 1 and 3 ($c_2 = 1$ and equivalent Δ and T functions). The Δ function is set to approach 1 when the ratio of the distance between a new sample and a previous sample exceeds 1.05: If a new sample has a distance 5% higher than the previous sample, it is unlikely to be promoted and/or accepted. c_1 in Algorithm 3 is set to 2, which makes it more difficult for a sample to be promoted than accepted once promoted. The reverse sigmoid probability distribution q, which gives the number of location changes between a new and a previous sample, is set to approach zero when number of location changes exceeds 20. For all experiments where a restart strategy is used, sample x_{i+1} can be built from either x_i, x_{best} or x_{lowR} (Section 5). The probability to pick one of the latter two is governed by a sigmoid and reverse sigmoid, respectively, with probabilities in range $[0, 0.2]$ and $[0.2, 0]$, stretched over N iterations. In all iterations where neither of the latter two is picked, x_i is used (no restarts). The total number of iterations and/or CPU-time is set depending on dataset (see below). λ is set to 1 in all experiments.

6.3 Datasets

The following two datasets are used:

- 266 TSPLIB instances[3] modified for the SLAP and shared in a public repository[4]. These instances include 6 different types of warehouse layouts (including one with no obstacles), all modeled as bi-directional graphs (without uni-directional travel-conventions). The number of products open for location reassignment vary between 5-427 in these instances. The initial locations for all products (baseline assignment x_1) in these instances is selected using a random uniform distribution. Solution proposals are uploaded for each of these instances using Algorithm 3 after a maximum of 20000 iterations (N). Experiments to test utility of Hamming distances and restarts are conducted on this dataset.
- A real warehouse with a conventional layout and without any uni-directional travel conventions. The provided picking log for this warehouse includes 260 unique products and 260 product locations. There are 200 pick-rounds and most products are picked in several pick-rounds. The experiments where Algorithm 1 and 3 are compared are run on this dataset. Algorithm 1 and 3 are run 10 times each on this dataset, with varying random seeds and a maximum CPU-time set to 8 hours. For a discussion regarding how often storage (re)assignments can be expected to be conducted (which may guide the choice of optimization time), see Section 2.

In both datasets, the capacity of all locations is assumed to be identical, meaning that any product can be placed at any location. We compare costs of samples against the baseline x_1, where each product is

[3] https://github.com/johanoxenstierna/OBP/instances, collected 19-10-2022.

[4] https://github.com/johanoxenstierna/L40_266, collected 14-11-2022

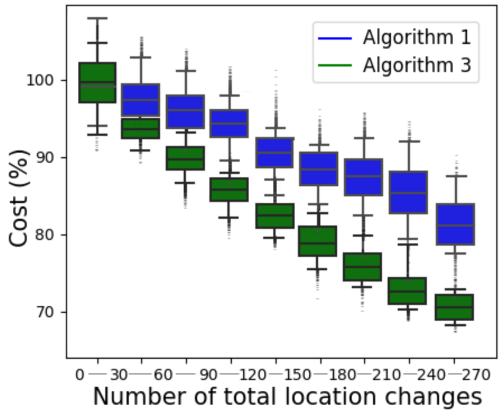

Figure 2: The total number of product location reassignments needs to be large to achieve the best total travel costs in $f^*(x_{best})$.

fixed to its initial location, where optimal picking costs are computed in $D(\mathcal{B})$ and where $D(\mathcal{R}) = 0$.

6.4 Experimental results

6.4.1 Utility of Hamming-distance based sampling

Results show that many location reassignments are needed to reach the best reductions in travel cost (Figure 2). Also, more reduction in cost is achieved when the Hamming distance (number of location changes) between a previous sample and a new one is relatively low (Figure 3). On average, the cost of sample $f^*(x_{i+1})$ is more reduced compared to a previous sample $f^*(x_i)$ if fewer location changes are attempted. This result empirically validates the Hamming distance distribution $q(x_{i+1}|x_i)$ and its bias toward fewer location changes (Equation 2).

6.4.2 Utility of *Restarts*

Aggregated results with and without *restarts* (Section 5) are shown in Figure 4. Given the same amount of optimization iterations ($N = 30000$) on the real warehouse dataset, the best results for both Algorithm 1 and 3 are obtained using restarts. Restarts enforce revisits to local minima with relatively short total travel costs f^* or reassignment paths $D(\mathcal{R})$ (Section 5). Since fewer reassignments mean that fewer pick-rounds contain products whose locations change, TSP optimization CPU-time is significantly lower when restarts are used. This is achieved by the caching of TSP costs (Section 5). Furthermore, few reassignments mean that the optimization of the reassignment path requires less CPU-time to reach a strong solution. As can be observed, Algorithm 1 and 3 without

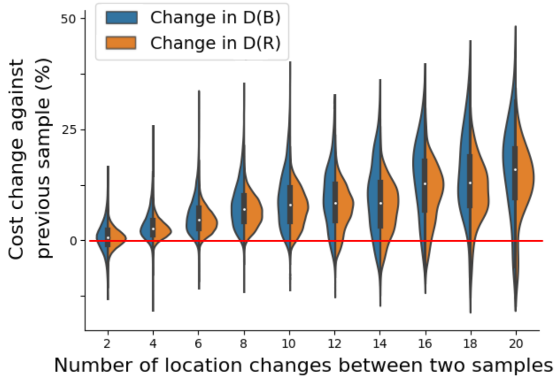

Figure 3: Distribution (violin) plot showing number of location changes against picking-log distance $D(\mathcal{B})$ (blue) and reassignment distance $D(\mathcal{R})$ (orange) when moving from a previous sample to a new sample. The mean cost of both $D(\mathcal{B})$ and $D(\mathcal{R})$ increase when more location changes are attempted in new samples. This plot excludes any x_i and x_{i+1} pairs where either were restarts back to a local minimum.

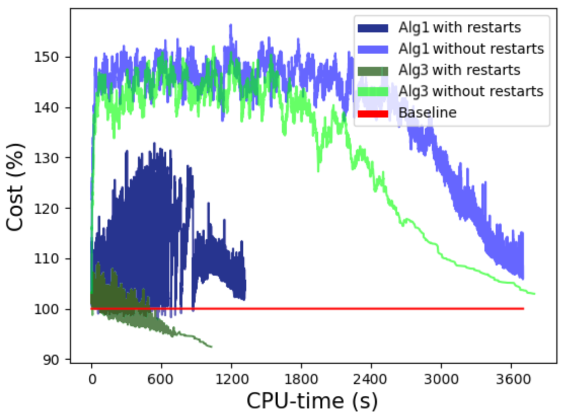

Figure 4: Algorithm 1 and Algorithm 3 with and without *restarts* for 30000 iterations on the real warehouse dataset. The costs shown are for $f^*(x_{i+1})$.

restarts (lighter blue and green) quickly jump up in cost. This is mainly attributed to the relatively low cost in initial assignment x_1, where $D(\mathcal{R}) = 0$, which is never revisited once stepped away from and never improved on (without restarts).

6.4.3 Algorithm 1 compared to Algorithm 3

Algorithm 3 (Nested Annealing with restarts) outperforms Algorithm 1 (Simulated Annealing with restarts) within the given CPU-time (Figure 5). The Markov chain in Algorithm 3 is more biased compared to the one in Algorithm 1, due to more samples being rejected. Algorithm 1 accepts more samples with high cost, which may lead to the discovery of more attractive search regions, if given more CPU-time than 8 hours.

Figure 5: Aggregate CPU-time against shortest total travel cost ($f^*(x_{best})$) on the real warehouse dataset (20 optimization runs): Blue is Algorithm 1, green is Algorithm 3 and red is the cost of baseline assignment x_1 (100%). The shadowed areas represent 95% confidence intervals.

6.4.4 Other features

Aggregate averages of results on the generated instances and Algorithm 3 are shown in Table 1 (Appendix). The elements for columns $f(x_i)$, $f^*(x_i)$, $f(x_{i+1})$, $f^*(x_{i+1})$, $f^*(x_{best})$, $D(R)_1$ $D(R)_{300}$ are all shown as percentages against the distance of the baseline cost $f^*(x_1)$ (100%). $D(R)_1$ and $D(R)_{300}$ denote the distance of the reassignment path after Algorithm 2 has been run for 1 and 300 iterations, respectively. The rows are aggregated averages based on number of products shown in column 1, from a total of 5279885 samples on the generated instance dataset (with 3-12 minutes CPU-time on each instance). One interesting result in Table 1 is that the predictive quality of $f(x_i)$ is almost identical to $f^*(x_i)$. OR-tools delivers very strong solutions to the given picking-log $\mathcal{B}$, which is explainable since pick-rounds $b \in \mathcal{B}$ rarely exceed 15 locations in length. This means that the strong performance of Algorithm 3 with restarts (Figure 4), may be achievable by Algorithm 1 set up with restarts and with a higher *accept* threshold c_1 (instead of using the *promote* filter), at least for the used datasets.

No correlation was found between the warehouse layout (the six ones in the generated instance set) and features such as total cost improvement, reassignment distance and/or number of final proposed location reassignments. This is explainable since both TSP-optimizers (OR-tools and Concorde) and the reassignment path optimizer (Algorithm 2) are *layout-agnostic* (Section 1).

7 CONCLUSION

An optimization model for the Storage Assignment Location Problem (SLAP) was proposed. In the TSP-based SLAP, products cannot be swapped between pick-rounds and future-forecasted picking is assumed to be static. Furthermore, the warehouse rack layout is assumed to have any configuration in 2D. An optimizer based on Simulated Annealing, to provide solutions to the TSP-based SLAP, was proposed. The optimizer generates assignment samples using a Hamming distance function and two *accept* filters. A *restart* heuristic, which forces occasional revisits to local minima, is also used. Since products cannot be reassigned to new locations for free, the distance of a *reassignment path* is added to the cost of any generated sample.

Two datasets were used to evaluate the proposed optimizer: A real warehouse dataset and a set of publicly shared test-instances on a generalizable format. The modifications to standard Simulated Annealing were found to be motivated and the best cost savings, of around 30%, were achieved after 8 hours of CPU-time. Overall this result is in line with results in prior work where strong assumptions are made with regard to warehouse layout (but where dynamicity may be assumed or where number of products is larger) (Kofler et al., 2014; Kübler et al., 2020; Trindade et al., 2022).

For future work, heuristics to increase bias in the proposed algorithm could be investigated. These include zoning, where products are set up to be drawn to certain areas in the warehouse. Another heuristic is λ (Section 4), and it can be defined to be adjusted dynamically (instead of being a constant), to potentially improve optimization performance and/or to reflect a more realistic division between the cost of picking and reassignment of products. For example, λ could be set to start at a low value and then to grow linearly. Setting it to a low value initially would prevent many samples from being rejected due to the clear relationship between few number of reassignments and low cost reduction (Figure 2). Another proposal involves analysis of the future-forecast picking log and how it relates to potential savings. Zhang et al. (2019) and Kofler et al. (2014) use datamining heuristics to show that reassignment "potential" is correlated to the way in which products in pick-rounds are distributed. It is challenging to make use of such heuristics to make concrete proposals for reassignments in a Markov chain, however. The TSP-based SLAP is highly intractable, even though it is a simplification of storage assignment in realistic use-cases.

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.

REFERENCES

Applegate, D., Cook, W., Dash, S., and Rohe, A. (2002). Solution of a Min-Max Vehicle Routing Problem. *INFORMS Journal on Computing*, 14:132–143.

Azadeh, K., De Koster, R., and Roy, D. (2019). Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science*, 53.

Boysen, N. and Stephan, K. (2013). The deterministic product location problem under a pick-by-order policy. *Discrete Applied Mathematics*, 161(18):2862 – 2875.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L., and Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research*, 285(2):497 – 512.

Cardona, L. F., Rivera, L., and Martínez, H. J. (2012). Analytical study of the Fishbone Warehouse layout. *International Journal of Logistics Research and Applications*, 15(6):365–388.

Charris, E. et al. (2018). The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10.

Christen, J. A. and Fox, C. (2005). Markov Chain Monte Carlo Using an Approximation. *Journal of Computational and Graphical Statistics*, 14(4):795–810.

Ene, S. and Öztürk, N. (2011). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology*, 60:1–11.

Fontana, M. E. and Nepomuceno, V. S. (2017). Multi-criteria approach for products classification and their storage location assignment. *The International Journal of Advanced Manufacturing Technology*, 88(9):3205–3216.

Garfinkel, M. (2005). *Minimizing multi-zone orders in the correlated storage assingment problem*. PhD Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology.

Gils, T. v., Caris, A., Ramaekers, K., and Braekers, K. (2019). Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research*, 277(3):814–830.

Hahsler, M. and Kurt, H. (2007). TSP – Infrastructure for the Traveling Salesperson Problem. *Journal of Statistical Software*, 2:1–21.

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

Janse van Rensburg, L. J. v. (2019). Artificial intelligence for warehouse picking optimization - an NP-hard problem. Master's thesis, Uppsala University.

Kallina, C. and Lynn, J. (1976). Application of the Cube-per-Order Index Rule for Stock Location in a Distribution Warehouse. *Interfaces*, 7(1):37–46.

Kofler, M., Beham, A., Wagner, S., and Affenzeller, M. (2014). Affinity Based Slotting in Warehouses with Dynamic Order Patterns. (Advanced Methods and Applications in Computational Intelligence):123–143.

Koster, R. d., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481 – 501.

Kruk, S. (2018). *Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools*. Apress.

Kübler, P., Glock, C., and Bauernhansl, T. (2020). A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses. 147:106645.

Larco, J. A., Koster, R. d., Roodbergen, K. J., and Dul, J. (2017). Managing warehouse efficiency and worker discomfort through enhanced storage assignment decisions. *International Journal of Production Research*, 55(21):6407–6422.

Lee, I. G., Chung, S. H., and Yoon, S. W. (2020). Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations. *Computers & Industrial Engineering*, 139:106129.

Liu, C.-M. (1999). Clustering techniques for stock location and order-picking in a distribution center. *Computers & Operations Research*, 26(10):989–1002.

Mackay, D. J. C. (1998). Introduction to Monte Carlo Methods. In *Learning in Graphical Models*.

Mantel, R. et al. (2007). Order oriented slotting: A new assignment strategy for warehouses. *European Journal of Industrial Engineering*, 1:301–316.

Ming-Huang Chiang, D., Lin, C.-P., and Chen, M.-C. (2014). Data mining based storage assignment heuristics for travel distance reduction. *Expert Systems*, 31(1):81–90. Publisher: Wiley Online Library.

Oxenstierna, J., Malec, J., and Krueger, V. (2022). Analysis of Computational Efficiency in Iterative Order Batching Optimization. In *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES,*, pages 345–353. SciTePress.

Rajasekaran, S. and Reif, J. H. (1992). Nested annealing: a provable improvement to simulated annealing. *Theoretical Computer Science*, 99(1):157–176.

Rathod, A. B., Gulhane, S. M., and Padalwar, S. R. (2016). A comparative study on distance measuring approches for permutation representations. In *2016 IEEE international conference on advances in electronics, communication and computer technology (ICAECCT)*, pages 251–255. IEEE.

Roodbergen, K. J. and Koster, R. (2001). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, 39(9):1865–1883.

Tak, H., Meng, X.-L., and Dyk, D. A. v. (2018). A Repelling–Attracting Metropolis Algorithm for Multimodality. *Journal of Computational and Graphical Statistics*, 27(3):479–490.

Trindade, M. A. M., Sousa, P., and Moreira, M. (2022). Ramping up a heuristic procedure for storage location assignment problem with precedence constraints. *Flexible Services and Manufacturing Journal*, 34.

Valle, C., Beasley, J. E., and da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, 262(3):817–834.

Wales, D. J. and Doye, J. P. K. (1997). Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *Journal of Physical Chemistry A*, 101:5111–5116.

Wu, J., Qin, T., Chen, J., Si, H., and Lin, K. (2014). Slotting Optimization Algorithm of the Stereo Warehouse. In *Proceedings of the 2012 2nd International Conference on Computer and Information Application (ICCIA 2012)*, pages 128–132. Atlantis Press.

Wutthisirisart, P., Noble, J. S., and Chang, C. A. (2015). A two-phased heuristic for relation-based item location. *Computers & Industrial Engineering*, 82:94–102.

Xiang, X., Liu, C., and Miao, L. (2018). Storage assignment and order batching problem in Kiva mobile fulfilment system. *Engineering Optimization*, 50(11):1941–1962.

Yu, M. and Koster, R. B. M. d. (2009). The impact of order batching and picking area zoning on order picking system performance. *European Journal of Operational Research*, 198(2):480 – 490.

Yu, V. F., Winarno, Maulidin, A., Redi, A. A. N. P., Lin, S.-W., and Yang, C.-L. (2021). Simulated Annealing with Restart Strategy for the Path Cover Problem with Time Windows. *Mathematics*, 9(14).

Zhang, R.-Q. et al. (2019). New model of the storage location assignment problem considering demand correlation pattern. *Computers & Industrial Engineering*, 129:210–219.

Žulj, I., Glock, C. H., Grosse, E. H., and Schneider, M. (2018). Picker routing and storage-assignment strategies for precedence-constrained order picking. *Computers & Industrial Engineering*, 123:338–347.