SN Computer Science (2023) 4:107
https://doi.org/10.1007/s42979-022-01496-0

ORIGINAL RESEARCH

Efficient Order Batching Optimization Using Seed Heuristics and the Metropolis Algorithm

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

Received: 24 May 2022 / Accepted: 31 October 2022 / Published online: 17 December 2022

Abstract

Order Picking in warehouses is often optimized using a method known as Order Batching, which means that one vehicle can be assigned to pick a batch of several orders at a time. There exists a rich body of research on Order Batching Problem (OBP) optimization, but one area which demands more attention is computational efficiency, especially for optimization scenarios where warehouses have unconventional layouts and vehicle capacity configurations. Due to the NP-hard nature of the OBP, computational cost for optimally solving large instances is often prohibitive. In this paper, we compare the performance of two approximate optimizers designed for maximum computational efficiency. The first optimizer, *Single Batch Iterated* (SBI), is based on a Seed Algorithm, and the second, *Metropolis Batch Sampling* (MBS), is based on a Metropolis algorithm. Trade-offs in memory and CPU-usage and generalizability of both algorithms is analyzed and discussed. Existing benchmark datasets are used to evaluate the optimizers on various scenarios. On smaller instances, we find that both optimizers come within a few percentage points of optimality at minimal CPU-time. For larger instances, we find that solution improvement continues throughout the allotted time but at a rate which is difficult to justify in many operational scenarios. SBI generally outperforms MBS and this is mainly attributed to the large search space and the latter's failure to efficiently cover it. The relevance of the results within Industry 4.0 era warehouse operations is discussed.

Keywords Order picking · Order Batching Problem · Metropolis algorithm · Seed algorithm · Computational efficiency · Warehousing

Introduction

Order Picking is the process in which sets of products (orders) are retrieved from locations in a warehouse. *Order Batching* is a method in which vehicles can be assigned to pick several orders at a time. Order Batching can be formulated as an optimization problem known as the Order Batching Problem (OBP) [13] or the Joint Order Batching and Picker Router Problem (JOBPRP) [35], where the Picker Router Problem is a Traveling Salesman Problem (TSP) applied in a warehouse environment [28]. We consider the OBP and JOBPRP versions equivalent if solutions to the OBP are assumed to include TSP solutions (henceforth we use the term OBP to refer to this version). There are several other versions and focus areas in OBP's, including dynamicity, traffic congestion, depot setups and obstacle layouts. One focus area is optimization aimed toward maximum computational efficiency. As will be laid out in "Literature Review" section, computational efficiency has both direct and indirect impacts on the quality of warehouse operations. Authors generally consider it to be important, but there are significant differences in how CPU-times and timeouts are used. Although the variability of OBP versions and corresponding results concerning computational efficiency is high, we believe more research in this domain is warranted. We delimit our work to OBP's where the objective is to minimize aggregate distances, and as measurement of

This article is part of the topical collection "Advances on Operations Research and Enterprise Systems" guest edited by Marc Demange, Federico Liberatore and Greg H. Parlier.

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

Jacek Malec
jacek.malec@cs.lth.se

Volker Krueger
volker.krueger@cs.lth.se

[1] Department of Computer Science, Lund University, Lund, Sweden

[2] Kairos Logic AB, Lund, Sweden

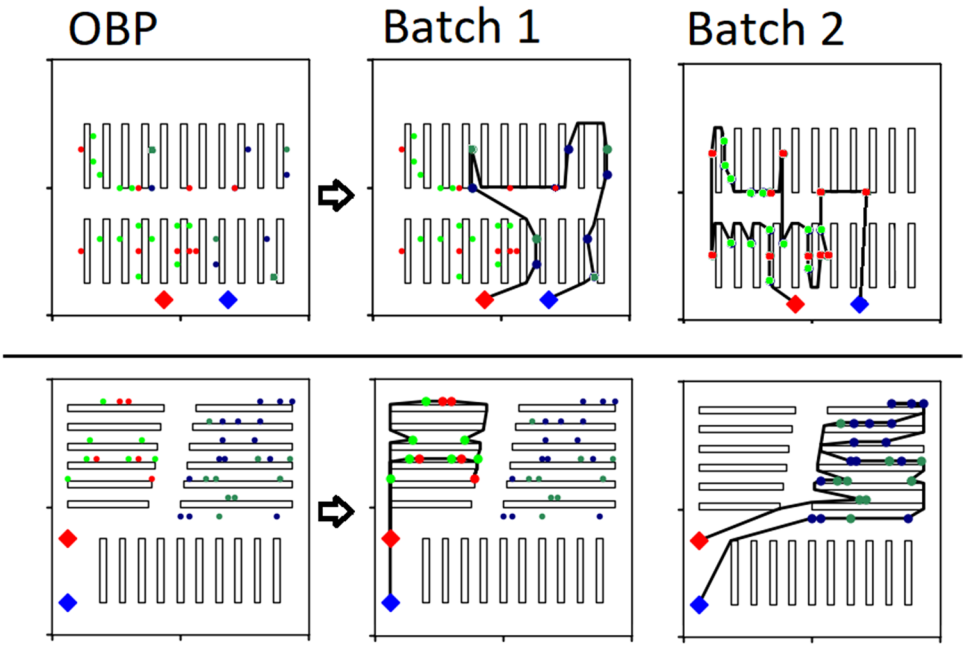

Fig. 1 Examples of the conventional (top) and a unconventional (bottom) layout warehouse, and OBP's with four orders from Oxenstierna et al. [26]. The colored diamonds denote origin and destination locations. The colored dots denote products and the orders which they belong to. In the solutions (right of the arrows), one vehicle is assigned to pick the red and lime orders and a second vehicle is assigned to pick the blue and green orders

computational efficiency we use the rate with which aggregate distance is reduced through CPU-time. We use the following two OBP optimizers: *Single Batch Iterated* (SBI) [25, 26] and *Metropolis Batch Sampling* (MBS) (introduced in this paper). We compare the aggregate distance result between the two optimizers, and also compare against results on public OBP benchmark datasets. We use a distance based OBP cost because this is the predominant Key Performance Indicator (KPI) in benchmark datasets. Although a KPI based on capital cost is what is mostly sought by warehouse management, it is more complex: there are a multitude of features that can go into capital, such as time-based aspects of work, traffic congestion, maintenance, ergonomics etc. A distance based KPI allows for a simpler model and a more generalized way in which to reproduce benchmark data and results.

We only work with CPU-times in the range 0–300 s. Results are compared with previous work by Aerts et al. [1] and Henn and Wäscher [16], who have proposed approximate optimization results for sets of smaller instance sizes. For smaller instances, we also assess results against optimal results on the Foodmart dataset [6]. For larger instances we use L09_251 [26]. As far as we are aware, there exists no standard benchmark format in OBP research, rendering experiment reproducibility difficult. Further discussions on how to represent key OBP features in reproducible data is highly relevant. Our research contributions are as follows:

1. An investigation into the importance of computational efficiency in OBP optimization.
2. Experiments regarding computational efficiency of two OBP optimizers on existing test-instances.

Literature Review

In this section, we first present how the OBP and some of its key features are formulated in the literature. Then we present commonly used OBP optimization algorithms and heuristics. Finally, we present how computational efficiency has been motivated and evaluated for different OBP models.

As several studies have pointed out, the Order Batching Problem (OBP) shares significant similarities with the more well-known Vehicle Routing Problem (VRP) [10, 34, 35]. Aerts et al. [1] distinguish three points of separation between the OBP and a common VRP:

1. *Order-integrity* constraint: In the OBP, products belonging to an order may only be picked by one vehicle, whereas there exists no concept of orders or order-integrity in the common VRP.
2. *Number of visits* constraint: In the OBP, the same location may be visited several times by various vehicles, whereas a location may only be visited once in the common VRP.
3. *Obstacle-layout*: In the OBP it is assumed that there exists an obstacle layout, whereas there is no such assumption in the common VRP.

Concerning the latter point, most of the research on the OBP assumes that the warehouse uses a *conventional* layout, which means racks are arranged with parallel aisles (between racks) and parallel cross-aisles (between sections of racks) [23]. If these conditions are not met the layout is *unconventional* (see Fig. 1).

Aerts et al. argue that the OBP can be modeled as a *Clustered* VRP (CluVRP) *with weak cluster constraints*. Weak cluster constraints mean that a vehicle may visit the locations in several clusters of locations in any sequence. The CluVRP was first introduced by Defryn and Sörensen [11] and according to Aerts et al. [1] it is equivalent to the OBP since clusters can be mapped as orders. In experiments they utilize this problem on a conventional layout warehouse and on OBP scenarios involving up to 100 orders.

For conventional layouts, proposed optimization algorithms include integer programming [35], clustering [20], datamining [9], dynamic programming [6] meta-heuristics and heuristics. Examples of meta-heuristics include Variable Neighborhood Search [1], Tabu Search [16], Ant Colony Optimization [21] and Genetic Algorithms [8]. The heuristic algorithms can be divided into three categories: priority rule-based algorithms, savings algorithms and seed algorithms [15]. Priority-rule-based algorithms build batches by sorting orders according to a heuristic, for example First-Come-First-Serve, First-Fit or Best-Fit. In savings algorithms batches with single orders are first initialized and evaluated. Then, pairs, triplets and larger batches of orders are constructed and the combination with the best total result is retrieved [16]. In seed algorithms batches are generated in two phases: Seed-selection and order addition. In the first phase, an initial seed order is selected. In order-addition orders are then added to the seed order. There are many choices for seed algorithms, with corresponding advantages and disadvantages depending on the usecase [17, 20, 31]. One example is the Sequential Minimal Distance (SMD) heuristic [32], where the sum of minimal distances between products in the seed order and remaining orders is computed:

$$\mathrm{SMD}(s,o) = \sum_{i \in s} \min_{j \in o} \left| d_{ij} \right|, o \in O, o \notin b, s \in b \tag{1}$$

where s denotes a seed order in batch b, where o denotes an order which does not exist in b, and where i and j denote products in order s and o, respectively.

Whenever there are more than two products in a batch, some form of TSP optimization is often used within the OBP optimizer. For conventional layouts, the highly efficient *S-shape* or *Largest Gap* algorithms are commonly used [14, 30]. We are not aware of any attempts to extend these to unconventional layouts. Given a distance matrix is provided, however, TSP's can be optimized reasonably fast using OR-tools [19] or Concorde [2, 3]. Concorde, for example, is almost guaranteed to find the shortest path of a TSP with 100 nodes in less than one second [3].

OBP models can be divided into static and dynamic: dynamic models are generally more realistic than static ones (incoming orders are there assumed to be known beforehand). The literature still tends to model OBP's as static since dynamicity incurs more complexity [31].

The importance of computational efficiency in OBP optimization can be derived from two types of factors. The first type has an immediate impact on operations in the warehouse. As an example, optimization should ideally be faster than the time it takes a vehicle to finish a picking round [14, 31]. Otherwise, vehicles must wait in an idle state at the depot while optimization finishes.

The second type concerns a larger perspective with flexible 4.0 industry-era integration and business utility. As an example, if an OBP optimization module is deployed on the cloud as a 3rd party software service (*SaaS*), a Warehouse Management System (WMS) client may be more interested in buying it if it is safe and simple to integrate. Longer CPU-times generally make it harder to set up a system (at least as a microservice) so that these two conditions are fulfilled [12]. Furthermore, rental and electricity cost of servers can naturally be assumed to rise with CPU-times [24].

In the broader literature on the OBP, the second type of factors are rarely discussed. CPU times are chosen to be "tolerable" [20], "reasonable" [5] or "acceptable" [1, 31], but often lack in concrete explanations of what these terms entail. Some examples are provided below for how researchers have used CPU-times and timeouts in optimization experiments with OBP's.

For approximate optimization, Henn and Wäscher [16] use timeouts between 1 and 180 s for a heuristic optimizer and OBP's where 40–100 unassigned orders are to be batched. Aerts et al. [1], use timeouts between 1 and 60 s on the same instance set and propose a meta-heuristic algorithm specifically designed to terminate at around 60 s, since solution improvement is found to be insignificant beyond that point. Both Aerts et al. and Henn and Wäscher's algorithms come to within 5% of the best solution overall within the first 10% of optimization time. Scholz et al. [31] experiment with instances of similar size but in a dynamic setting and report a much lower efficiency: 70% of maximum allowed CPU-time is necessary to reach within 5% of best solution overall. Efficiency also decreases non-linearly with instance size in their results: for 10 orders, their optimizer needs 2 s, for 100 orders it needs 11 min, and for 200 orders 60 min. Henn [14] also presents an algorithm for dynamic OBP's and sets it to self-terminate after 60 s, partly due to operational considerations (to avoid vehicles from idling at the depot). Many publications do not present concrete results for timeouts or rate of solution improvement, or a low number of experiments [4, 7, 18]. Kulak et al. [20] and Li et al. [21], for example, present highly efficient meta-heuristic optimizers, but only on 5–10 instances, and do not include rate of solution improvement in their results. For authors presenting algorithms capable of finding optimal solutions to static OBP's, Henn and Wäscher [16] set timeouts between 2 and 1328 s for instances with up to 60 orders. Gademann et al. [13], set timeouts to 10–30 min for

up to 100 orders. Valle et al. [35] and Briant et al. [6], on the Foodmart dataset, present timeouts in the range 300 s to 2 h to obtain optimal results for 20–45 orders.

These examples show that computational efficiency in OBP optimization is difficult to judge generally. Choice of static or dynamic modeling, optimal versus approximative optimization, experimental setup, instance sizes and the technology level of used software and hardware, are all aspects that can have a complex effect on results in this regard.

Problem Formulation

We define the OBP objective as the assignment of batches to vehicles such that the aggregate distance needed to pick the batches is minimized. Each batch b consists of a set of one or several orders $b \in 2^{\mathcal{O}}$, where each $o \in \mathcal{O}$ is a subset of products $o \in 2^{\mathcal{P}}$. Each product $p \in \mathcal{P}$ is a set which includes a unique product identifier, an order identifier, weight w and volume vol, $w, vol \in \mathbb{R}^+$. The sum of weight, volume or number of orders in a batch can be retrieved with function $q(b), q \in w, vol, k$. The x, y location coordinates of all products is defined as set $\mathcal{L}_{\mathcal{P}}$, and the location of a product is retrievable with function $l(p)$. The locations of the products in an order are retrievable with function $l(o) = \cup_{p \in o} l(p)$, and all locations in a batch are retrievable with function $l(b) = \cup_{o \in b} l(o)$. We define a single origin location for all vehicles l_s, a single destination location l_d and a set of polygonal obstacle location sets $\mathcal{L}_{\mathcal{U}}$. The aggregate of all locations is $\mathcal{L} = \{l_s\} \cup \{l_d\} \cup \mathcal{L}_{\mathcal{P}} \cup \mathcal{L}_{\mathcal{U}}$.

We build undirected graph $G = (V, E)$. Each vertex in V represents a unique location in $\mathcal{L}$ and function $v(l)$ gives a vertex for a location. The vertices in batch b include the origin and destination vertices $v(b) = v\left(l_s\right) \cup v(l(b)) \cup v\left(l_d\right)$. E represents the set of all Euclidean edges between all locations that circumvent obstacles in $\mathcal{L}_{\mathcal{U}}$. Distance matrix D and shortest paths between all edges is computed using the Floyd-Warshall algorithm. How E and shortest paths can be constructed with polygonal obstacles is beyond the scope of this paper; for details see Ref. [29]. We also permit a surjective relationship of products to locations, i.e., several types of products can be stored at the same location and the location represents several real locations in the warehouse. This can be useful to help reduce the memory footprint of G. The path to pick batch b is retrievable with the following function:

$$T(b) = \{v_i\}_{i=1}^{n}, n = |v(b)|, \tag{2}$$

$$v_i = \begin{cases} v_s & i = 1 \\ v_k & 1 < i < n \\ v_d & i = n \end{cases} \tag{3}$$

and represents the solution to a Traveling Salesman Problem (TSP). The distance of $T(b)$ is retrievable with function $D(b) = \sum d_{T(b)_i T(b)_j}, i, j \in \mathbb{Z}^+, j = i + 1, i < |T(b)|$, where $d \in \mathbb{R}^+$ represents scalar entries in distance matrix D. Vehicles are defined as $m \in \mathcal{M}$ where each vehicle has capacities expressed in weight w, volume vol and number of orders k. The scenario where a vehicle m is assigned a batch, order, and/or product location is defined with binary variables x_{mb}, x_{mo} and x_{ml}, respectively. We then use the following OBP formulation:

$$\min \sum_{b \in B} D(b) x_{mb}, m \in M \quad s.t. \tag{4}$$

$$\sum_{m \in M} x_{mo} = 1, \forall o \in O \tag{5}$$

$$\sum_{l \in loc(o)} x_{ml} \geq x_{mo}, \forall o \in O, m \in M \tag{6}$$

$$q(b) \leq q(m) x_{mb}, b \in \mathcal{B}, \quad q \in w, vol, k, m \in \mathcal{M} \tag{7}$$

where (4) states the objective, i.e., minimize distances for all generated batches $\mathcal{B}$, where (5) enforces order-integrity, where (6) enforces all locations in all orders to be visited at least once and where (7) ensures vehicle capacities are never exceeded. Since this OBP is highly intractable we also use a less ambitious objective in the *single batch* OBP:

$$\underset{b \in \mathcal{B}}{argmin}\, D(b) \tag{8}$$

where the aim is to find a single batch for an already selected vehicle. For this case we also enforce the single batch to come as close as possible to vehicle capacity:$\exists q(q(b) + q(o) \geq q(m)), \forall o \in \mathcal{O}, o \notin b, q \in w, vol, k$.

Optimization Algorithms

Single Batch Iterated (SBI)

SBI (Algorithm 1) is a heuristic multi-phase optimizer. In the core of the algorithm unassigned orders $\mathcal{O}$ are iteratively sent as input to the *SMD* (Sequential Minimal Distance) function, together with distance matrix D, a randomly chosen available vehicle and a semi-stochastic seed order index. The *SMD* function builds a single batch b by first selecting a seed order according to the seed index and then adds orders to it according to minimal distances (Eq. 1). Batch b is then removed from the set of unassigned orders and the procedure repeats until all orders have been batched into $\mathcal{B}$. An approximate solution to the OBP (Eq. 4) is thus obtained by pre-selecting

vehicles and approximately solving a *single batch* OBP for each vehicle (Eq. 8).

Algorithm 1: *Single Batch Iterated (SBI)*

$y^* \leftarrow \infty$
$\hat{y} \leftarrow \infty$
for $i = 1, ..., N$ **do**
  $\mathcal{O}_s \leftarrow \mathcal{O}$
  $\mathcal{B} \leftarrow \{\}$
  while $\mathcal{O}_s$ **do**
    $m \leftarrow \text{select\_vehicle}(\mathcal{M})$
    $b \leftarrow \text{SMD}(\mathcal{O}_s, m, D, i)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup b$
  end
  $y \leftarrow TSP(\mathcal{B}, D)$
  $\text{update\_best}(y, \hat{y}, y^*)$
end

Number of iterations (N) is used here for brevity and in the implementation ("Experiments" section) a time-based condition is used to stop the outer loop. The purpose of the i index is to reduce the probability that the same solution is obtained multiple times, and, if used, it can be set to $N = |\mathcal{O}|$, for example. The paths to visit all locations in batches $b \in \mathcal{B}$, $T(b)$ and their distance, $\sum_{b\in\mathcal{B}} D(b)x_{mb}$, is computed using the OR-tools TSP optimization suite[1] (in the *TSP* function). OR-tools is set to finish quickly using a number-of-iterations parameter, which is set to grow linearly with number of vertices in the TSP. In the update_best function, the aggregate distance between the new OR-tools cost (y) is compared against the best OR-tools cost obtained so far ($\hat{y}$). If the new cost is lower, the TSP's are optimally solved using Concorde[2] and if this is better than the previous Concorde best, the result is stored as the new best in y^*.

Since the number of SMD computations between orders is approximately cubic to number of orders, $|\mathcal{O}| \sum (|\mathcal{O}| - i), i \in [|\mathcal{O}| - 1]$, we use an SMD order–order enumerated matrix which is populated through the optimization procedure: if SMD between two orders does not exist in the matrix, it is computed and pushed to the matrix. Once the value is stored it is subsequently queried. Caching SMD's reduces number of calls to SMD from cubic to square, at an insignificant increase in memory usage (~25 megabytes for 5000 orders assuming 8 bits per cell in the matrix). It should be noted that this only works for an SMD algorithm where the seed is defined as a single order, which cannot provide more than a noisy estimate of the subsequent TSP solution distance for batches with more than two orders. We still deem pairwise order–order SMD caching suitable, since distance estimates are inaccurate even if SMD's for larger collections of orders are computed (TSP optimization is required for accurate estimates). Caching could also be used to store all generated single batches and their solved TSP's in a hash tree or equivalent, to prevent the same TSP to be optimized twice (*memoization*). We leave an implementation of this for future work.

One potential issue with SBI is its reliance on the SMD heuristic. Although SMD makes sure the distance between orders is always minimized for a given batch, the number of orders to select from decreases through the single-batch while-loop in Algorithm 1. Hence, the last batch which is created in the while-loop can be assumed to be of worse quality in terms of distance minimization relative to the first.

Metropolis Batch Sampling (MBS)

MBS is a heuristic multi-phase optimizer which uses distance matrix D, the Concorde TSP solver (in the *TSP* function below) and the SMD heuristic to compute distance between orders. The main difference between SBI and MBS is that the latter only uses the SMD function to produce an initial solution. A Metropolis algorithm [22] is then used to improve on it using the following procedure:

Algorithm 2: Metropolis Batch Sampling

$\mathcal{B}_1 \leftarrow \{\}$
while $\mathcal{O}$ **do**
  $m \leftarrow \text{select\_vehicle}(\mathcal{M})$
  $b \leftarrow \text{SMD}(\mathcal{O}, m, D)$
  $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup b$
end
$y_1 \leftarrow TSP(\mathcal{B}_1, D)$

for $i = 1, ..., N$ **do**
  $\mathcal{B}_{i+1} \leftarrow \text{new\_sample}(\mathcal{B}_i)$
  $y_{i+1} \leftarrow TSP(\mathcal{B}_{i+1}, D)$
  $\alpha \leftarrow min(1, \gamma(y_i/y_{i+1}))$
  $u \leftarrow \mathcal{U}(0, 1)$ // random uniform
  if $u \leq \alpha$ **do**
    $\mathcal{B}_i \leftarrow \mathcal{B}_{i+1}$
    $y_i \leftarrow y_{i+1}$
end

The upper while-loop is equivalent to the one in Algorithm 1. The lower for-loop consists of a Metropolis algorithm

[1] https://developers.google.com/optimization/routing/tsp, collected 13–09-2021.

[2] http://www.math.uwaterloo.ca/tsp/concorde/index.html, collected 16–09-2021.

where each new sample is drawn from a previous one. The function new_sample($\mathcal{B}_i$) uses the following stationary distribution to describe the probability for a given new sample:

$$q(B_{i+1}|B_i) = e^{-2CH_d(B_i,B_{i+1})^P} \quad (9)$$

where C and P are constants and where the $H_d(\mathcal{B}_i, \mathcal{B}_{i+1})$ function denotes the number of swapping operations needed to obtain $\mathcal{B}_{i+1}$ from $\mathcal{B}_i$. A swapping operation is defined as a switch of position of two orders in an enumerated set of batches. Since number of swaps to go from $\mathcal{B}_i$ to $\mathcal{B}_{i+1}$ is always equal to number of swaps to go from $\mathcal{B}_{i+1}$ to $\mathcal{B}_i$, the q distribution is symmetrical, i.e., $q(\mathcal{B}_{i+1}|\mathcal{B}_i) = q(\mathcal{B}_i|\mathcal{B}_{i+1})$. A swap is only permitted if vehicle weight and volume capacity constraints are not broken.

The TSP's of the batches in the new sample are then solved using Concorde ("Literature Review" section) in the *TSP* function, and the aggregate cost is stored in y_{i+1}. The accept probability α is computed based on the following balance condition [33]: If $y_{i+1} < y_i$ the new sample is always accepted. If $y_{i+1} \geq y_i$ the sample may still be accepted if a uniform random value is less than α. α depends on the ratio y_i/y_{i+1}.

Contrary to SBI, the search space of MBS is guaranteed to include the global optimum, provided the sampling function can output any $\mathcal{B}$ that does not break constraints and enough computational time. This may just as well be a liability, however, since the search space may be too large for the algorithm to see optimization gains within reasonable time. We add bias parameter $\gamma \in \mathbb{R}^+$ to allow for experiments where the search space of the algorithm is more restricted. Without the use of γ, the probability is high that the algorithm steps away from the SBI local minimum $\mathcal{B}_1$ in the very first iteration (which is likely to happen if y_1/y_2 is close to one).

The best sample is assumed to be stored throughout the optimization procedure (sample storage is omitted from the pseudo-code). Just as with Algorithm 1, a number of iterations parameter $N \in \mathbb{Z}^+$ is shown as stopping condition in Algorithm 2 for brevity, but in the experiments in "Experiments" section, a CPU-time condition is used instead. Establishing a suitable N for converge is possible by studying covariance of samples, but it is challenging in the OBP case: $\mathcal{B}_i$ is a set of orders where the orders may contain a variable number of products at variable locations. Heuristics would thus be needed to quantify covariance between two samples.

Experiments

Benchmark Datasets

The publicly shared OBP datasets Foodmart,[3] L6_203[4] and L09_251[5] are used for experimentation. Foodmart was introduced by Valle et al. [35] and models a warehouse with a conventional layout and a maximum of 8 aisles and 3 cross-aisles. A feature in Foodmart is that vehicles carry bins and that vehicle capacity is expressed as a volume unit per bin. If an order cannot fit in a single bin, splitting it between different bins is permitted. SBI and MBS are not designed for this feature (it constitutes an extra bin packing problem within the OBP), so a greedy heuristic module is attached to the optimizers for the Foodmart experiment (for details see Ref. [25]).

L6_203 and L09_251 model scenarios for one conventional and up to six unconventional warehouse layouts and multiple depots. In these instances, vehicle capacity is expressed in number of orders and total number of orders generally range between 4 and 50. All but 6 Foodmart instances also fit within this range. For total number of orders in the range 50–1000 we use L09_251.

Number of orders only gives a rough idea of how much CPU-time might reasonably be needed to optimize an OBP instance. Number of products per order and vehicle capacities are further examples of features that have a considerable impact. To classify instances by size, we use the amount of computational time needed to obtain the SMD baseline solution: 0–2, 2–4, 4–7 or > 7 s. The resulting number of instances for the four classes are as follows: 0–2 s: 335, 2–4 s: 179, 4–7 s: 91, > 7 s: 56. The maximum permitted CPU-time for the 0–2 s instances is set to 20 s and 300 s for the remaining ones. For all our experiments we use Intel Core i7-4710MQ 2.5 GZ 4 cores, 16 GB RAM.

Experiment Results

Aggregations of all results are presented in Figs. 2, 3, Tables 1 and 2 (Appendix). In Fig. 2, the relative improvement rates from the baseline are shown for the two optimizers and four instance size classes. The shades around the lines represent 95% confidence intervals.

In terms of rate of solution improvement, SBI performs stronger than MBS across all instance size classes, and the difference grows with instance size. The MBS results shown are for parameter values $C = 0.1$, $P = 1$, $\gamma = 10$ ("Metropolis Batch Sampling (MBS)" section), retrieved from early testing. A γ value of 1 (the standard Metropolis algorithm), yields weaker results.

[3] https://pagesperso.g-scop.grenoble inp.fr/~cambazah/ batching/, collected 19–05-2022 (135 instances).

[4] https://github.com/johanoxenstierna/OBP_instances, collected 19–05-2022 (257 instances).

[5] https://github.com/johanoxenstierna/L09_251, collected 19–05-2022 (269 instances).

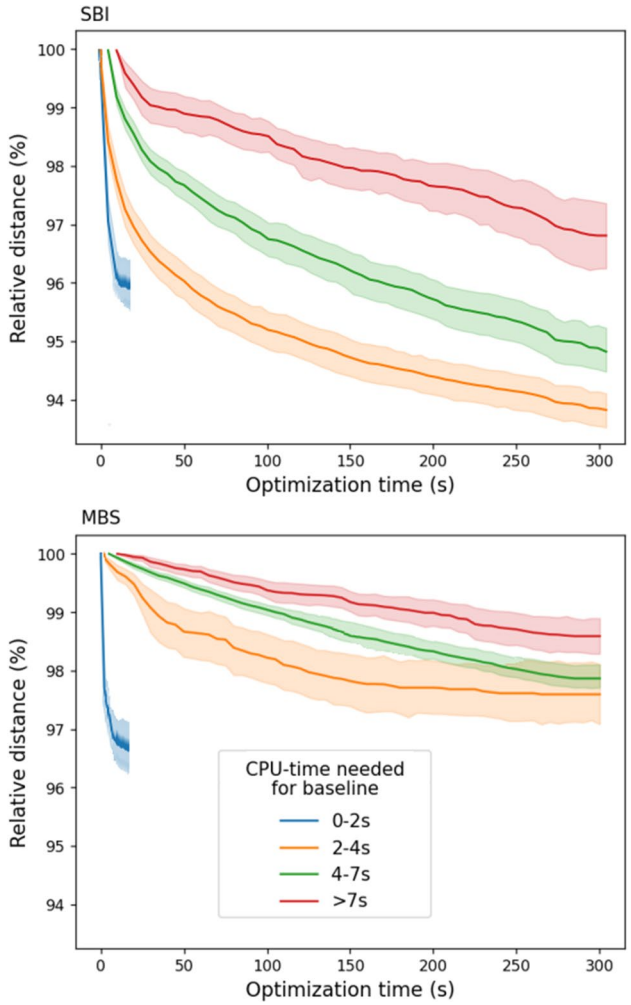

Fig. 2 Optimization time versus relative OBP distances in percentages, for four instance size classes (661 total instances). The smallest instances (blue) end at a 20 s timeout

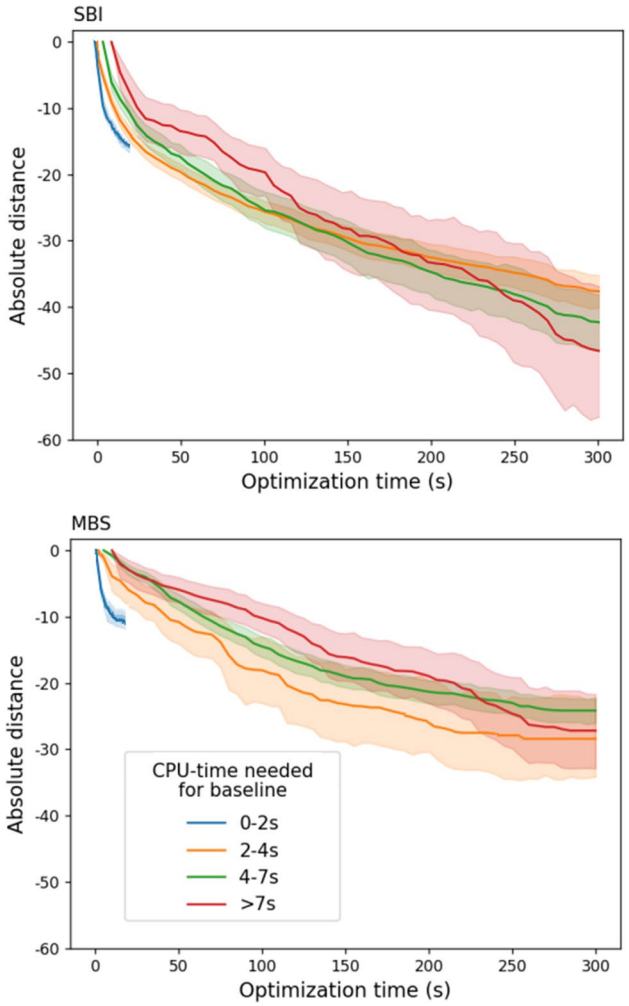

Fig. 3 Optimization time versus standardized absolute distance savings, for four instance size classes

Overall, the solution improvement rates for the two smaller instance classes (blue and orange) corroborate those of Henn and Wäscher [16] and Aerts et al. [1]: improvements are significant in the initial stage of optimization (1–4% improvement over baseline within the first 10% of optimization) and then taper off. In our case all instances with up to 100 orders require no more than 2 s to obtain the SBI baseline.

The Foodmart instances (all except 6) fit within the smallest class and there we compare our strongest results against optimal results in Briant et al. [6]: on average a gap to optimality of 2.3% is achieved after a maximum of 10 s. The average gap between the baseline solution and the best solution found is 3.2% on Foodmart. On generated instances in L6_203 the corresponding gap is 3.5%. For the larger two instance classes (> 4 s to find a baseline solution), the pattern is similar, but more time is needed to reach the same percentage improvement over the baseline.

In terms of absolute distance rate of improvement, we first standardize the data such that the average pick round is of similar length between the three datasets. The absolute distance improvements for the four instance size classes are shown in Fig. 3. One observable pattern in Fig. 3 is that larger OBP's tend to see more solution improvement. One explanation for this is that the probability of finding a strong baseline decreases with larger instances, and possible optimization gains can therefore be assumed to also be larger. The red curves (> 7 s), for example, yield weaker results in the beginning and stronger results in the end, relative to the green curves (4–7 s). It takes around 4 min to get there, however. Since there are only 56 instances in the red class, more data would be needed to investigate this pattern further and to narrow the confidence intervals.

The best achieved rates of solution improvement decrease to less than < 1%/minute after the initial gains taper off (after 30–60 s for the larger instances in Fig. 2). In terms of standardized distance, this is on average equivalent to around 18% of the length of a single batch TSP solution (~ 12 standardized distance units).

As discussed in "Literature Review" section, judgment of results in light of previous work is challenging due to the high variability of OBP models. Overall, we believe 1% / min is a slow rate of improvement and that it would be difficult to justify in many OBP scenarios, especially when considering the advantages of short CPU-times discussed in "Literature Review" section.

There are several possible reasons why MBS performs worse than SBI. One is that the relatively general application of the Metropolis algorithm faces a search space which is far too large for it to adequately sample within the allotted time. Even with a high amount of extra bias, imposed through manually added parameter γ, MBS is not able to find and improve on samples faster than SBI. The purely heuristic and more biased SBI optimizer has no global optimum guarantee due to the SMD heuristic, but it instead guarantees that each sample is a relatively strong local minimum. Using the semi-stochastic seed index within the SMD function ("Single Batch Iterated (SBI)" section) also makes sure that the SBI local minimums are uncorrelated to some extent. Another possible advantage of SBI is that it uses an approximate TSP optimizer to filter out promising samples before solving them optimally. Tests show that both TSP optimizers (Concorde and OR-tools) perform relatively similar in terms of CPU-time on the given instance set: its TSP's are often quite short (5–20 locations) and there is a significant amount of static CPU-time software overhead relative to the actual TSP optimization for these cases. The main optimization result is that the SMD heuristic proves useful, at least in terms of computational efficiency, the way it is used within the SBI optimizer and for the OBP version at hand.

Conclusion

We investigated computational efficiency in approximate Order Batching Problem (OBP) optimization. In previous work, computational efficiency has not been given enough attention, especially when considering unconventional warehouse layouts and vehicle types [1]. It is an important topic that affects operational costs both directly and indirectly, however. In experiments we studied the computational efficiency of two approximate optimizers, Single Batch Iterated (SBI) and Metropolis Batch Sampling (MBS). They both begin by obtaining an initial solution using the Sequential Minimal Distance (SMD) heuristic. SBI then improves on this solution by rerunning the SMD selection procedure using a semi-stochastic seed-order index, whereas MBS improves on the initial solution using a Metropolis algorithm.

For OBP instances with up to 100 orders and a few seconds of CPU-time, both optimizers yield distances only a few percentage points higher than results obtained at timeout (or optimal results where such are available). The result corroborates previous research claims: fast approximate optimization is a practicable choice in common OBP scenarios [5, 20].

For larger instances, with 100–1000 orders, more time is required to obtain similar optimization gains. The standardized absolute distance saved through the optimization procedure grows similarly for all instance sizes. In SBI's case this can be explained since weak batches (with products located far from each other) are only constructed whenever there are few orders left to select from (SMD prevents this in other cases). This phenomenon occurs an equal number of times regardless of instance size and the amount of possible solution improvement in larger instances is thus relatively low. MBS does not face this particular issue, but on the other hand it has no mechanism to reduce the vast search space. MBS generally performs worse than SBI within the 5 min timeout, particularly on larger instances.

Regardless of instance size, we conclude that that the value in spending significantly more CPU-time to obtain a result a few percentage points better than a baseline, must be weighed against the less measurable and indirect costs that come with lower computational efficiency. Although several authors have problematized large CPU-time requirements for OBP optimization [5, 20, 34], it is challenging to judge optimization efficiency generally due to the large variability of OBP use cases ("Literature Review" section).

For future work, we believe the investigation can be widened to include more optimizers. MBS could be replaced by similar but more biased MCMC algorithms, such as Simulated Annealing [27] or Basin Hopping [36]. Heuristics to add even more bias to these algorithms might be needed, however. Examples include mode-jumping [33] and restarts [37], which prevent convergence on local minima. Also, the number of required samples (N) for convergence could be estimated for various MCMC algorithms by calculating covariance between generated samples. That way, maximum CPU-time can be set in a more informed manner. For SBI, the Sequential Minimal Distance (SMD) heuristic could be replaced by alternatives which may be more suitable for the unconventional layout. We believe there are significant savings to be made in optimization if more memory is allocated to store and reuse parts of expensive computations. Modeling of OBP's and data-driven performance evaluation are also of primary importance. Currently, there exists no standard format for OBP benchmark datasets and this poses a serious threat to scientific reproducibility. Since there are many possible versions of OBP's, the community needs to discuss how a standard format for OBP

benchmark data can be designed to balance realism with simplicity and reproducibility.

Appendix

See appendix Tables 1 and 2.

Table 1 Aggregation of SBI test-instance results into categories based on number of orders in the OBP's

# Orders	# Products	# Batches in solution	Baseline distance	Time to obtain baseline (s)	Improvement percentage, after					
					5 s	10 s	30 s	60 s	120 s	300 s
4–5	15.93	1.09	123.25	0.03	2.99	3.82	–	–	–	–
6–7	27.56	1.80	143.12	0.18	3.23	3.47	–	–	–	–
8–9	39.43	3.78	153.77	0.21	3.47	5.07	–	–	–	–
10–11	71.91	4.16	192.00	0.37	3.71	4.40	–	–	–	–
12–15	75.68	4.90	191.47	0.33	3.95	3.95	–	–	–	–
16–20	110.15	5.58	232.89	0.35	4.19	4.58	–	–	–	–
21–25	145.51	5.29	266.29	0.33	4.42	4.78	–	–	–	–
26–30	161.80	6.52	284.31	0.49	4.66	4.88	–	–	–	–
31–40	221.98	6.60	348.57	0.48	4.90	5.53	–	–	–	–
41–50	272.45	7.24	429.48	0.43	5.14	5.14	–	–	–	–
51–60	306.42	8.10	471.01	0.56	5.38	5.38	–	–	–	–
61–70	331.37	8.94	509.58	0.57	5.62	5.62	–	–	–	–
71–80	360.15	8.80	544.29	0.65	5.86	5.86	–	–	–	–
81–90	390.53	9.65	592.94	0.77	6.10	7.93	–	–	–	–
91–100	439.70	9.46	667.07	0.68	3.03	3.62	3.90	–	–	–
101–110	379.97	10.74	632.89	0.54	2.84	3.40	3.83	–	–	–
111–120	396.38	12.58	745.79	0.79	2.65	3.21	3.74	3.75	–	–
121–130	418.98	11.15	859.19	1.22	2.43	3.03	3.62	3.66	–	–
131–140	437.76	13.41	972.50	1.36	2.23	2.81	3.53	3.61	3.92	–
141–150	463.67	13.22	1091.84	1.11	2.02	2.63	3.42	3.49	3.94	–
151–160	473.43	14.57	1200.46	1.51	1.82	2.43	3.33	3.45	4.00	–
161–170	494.32	13.62	1305.03	1.77	1.64	2.46	3.24	3.37	4.07	5.24
171–180	517.94	14.35	1425.75	1.93	1.42	2.55	3.11	3.32	4.12	5.41
181–190	539.55	13.75	1543.44	1.91	1.21	1.83	3.01	3.23	4.16	5.81
191–200	555.13	14.41	1649.98	2.06	1.01	1.28	2.94	3.14	4.20	5.71
201–250	662.07	16.16	1853.66	2.10	0.81	1.43	1.53	3.11	4.22	5.54
251–300	802.01	19.34	2087.13	2.22	0.61	1.46	1.51	2.69	3.92	4.92
301–350	989.22	22.95	2369.19	2.41	0.60	1.31	1.52	2.13	3.03	4.52
351–400	1326.53	24.11	2805.68	3.16	0.74	1.15	1.38	1.56	1.75	4.95
401–450	1438.58	26.00	3006.29	2.69	0.53	0.53	0.77	1.01	1.25	3.46
451–500	1581.54	27.48	3249.08	3.30	0.20	0.50	0.58	0.60	1.65	2.73
501–550	1694.52	27.80	3453.46	4.07	0.24	1.02	1.23	1.45	2.70	3.89
551–600	1989.84	29.74	3843.07	4.38	0.30	1.58	1.79	2.03	2.27	2.50
601–650	2149.81	35.85	4093.18	5.32	–	1.06	1.56	2.27	2.60	3.14
651–700	2316.35	39.98	4348.14	5.02	–	1.32	1.36	1.38	1.42	1.43
701–750	2660.47	46.27	4796.14	5.92	–	0.89	1.01	1.12	1.33	1.59
751–800	3019.32	50.12	5248.71	6.05	–	0.45	0.98	1.61	2.12	2.64
801–850	3154.56	55.04	5472.26	6.26	–	0.61	0.63	0.65	1.74	1.75
851–900	3290.80	59.79	5708.32	7.30	–	1.04	1.08	1.10	1.11	1.87
901–950	3497.78	64.09	6199.37	7.80	–	0.29	0.47	0.67	1.60	1.97
951–1000	3760.57	71.14	6568.42	8.30	–	0.80	0.96	1.46	1.78	2.54

Within each category the average over all results is shown. Whenever the specified time is not relevant or the optimizer failed to obtain a result within its scope, a minus sign (−) is shown. The distances shown are standardized

Table 2 Aggregation of MBS test-instance results into categories based on number of orders in the OBP's

# Orders	# Products	# Batches in solution	Baseline distance	Time to obtain baseline (s)	Improvement percentage, after					
					5 s	10 s	30 s	60 s	120 s	300 s
4–5	15.93	1.09	123.25	0.02	3.00	3.54	–	–	–	–
6–7	27.56	1.80	143.12	0.19	2.95	3.04	–	–	–	–
8–9	39.43	3.69	155.02	0.19	3.32	3.45	–	–	–	–
10–11	71.91	4.07	188.36	0.41	3.39	3.70	–	–	–	–
12–15	75.68	4.99	196.83	0.34	3.01	3.13	–	–	–	–
16–20	110.15	5.59	233.63	0.37	2.23	2.87	–	–	–	–
21–25	145.51	5.87	275.70	0.40	3.23	3.63	–	–	–	–
26–30	161.80	6.52	284.31	0.39	3.95	3.93	–	–	–	–
31–40	221.98	6.60	348.57	0.37	2.47	3.16	–	–	–	–
41–50	272.45	7.24	429.48	0.36	3.06	3.14	–	–	–	–
51–60	306.42	8.71	491.63	0.47	2.77	3.66	–	–	–	–
61–70	331.37	8.94	509.59	0.55	2.59	3.02	–	–	–	–
71–80	360.15	8.84	531.03	0.40	1.95	2.40	2.53	–	–	–
81–90	390.53	9.60	582.97	0.95	2.56	2.95	2.98	–	–	–
91–100	439.70	9.47	667.08	0.76	2.28	2.39	3.69	–	–	–
101–110	379.97	11.76	644.23	0.61	1.88	2.16	2.40	2.59	–	–
111–120	396.38	12.51	748.19	0.89	1.47	2.29	2.61	2.75	–	–
121–130	418.98	11.25	854.24	1.41	2.14	2.20	2.54	2.88	2.91	–
131–140	437.76	13.47	971.66	1.38	1.69	1.77	1.98	2.06	2.13	–
141–150	463.67	13.25	1109.84	1.02	1.25	1.37	1.85	2.05	2.22	–
151–160	473.43	14.58	1200.46	1.23	1.58	1.89	1.91	2.15	2.18	–
161–170	494.32	13.57	1334.75	1.59	1.40	1.92	2.02	2.05	2.14	2.65
171–180	517.94	14.35	1425.75	1.98	1.23	2.31	2.15	2.50	2.59	2.68
181–190	539.55	13.78	1552.34	2.06	1.18	1.60	1.73	2.22	2.28	2.41
191–200	555.13	14.52	1660.20	2.00	0.95	1.30	1.50	1.59	1.85	2.05
201–250	662.07	16.16	1853.66	2.35	0.63	1.03	1.08	1.66	1.91	2.32
251–300	802.01	20.00	2192.61	2.18	0.34	0.84	1.46	1.57	1.64	2.00
301–350	989.22	23.38	2478.76	2.37	0.55	0.71	1.21	1.27	1.28	1.72
351–400	1326.53	24.01	2811.59	3.01	0.56	0.69	1.23	1.29	1.35	1.78
401–450	1438.58	26.00	3006.30	2.67	0.13	0.13	0.28	0.27	1.34	1.58
451–500	1581.54	27.49	3249.08	3.22	0.10	0.16	0.22	0.27	0.76	1.56
501–550	1694.52	27.80	3453.47	4.35	0.08	0.08	0.37	0.49	1.59	2.00
551–600	1989.84	28.30	3743.11	4.73	0.10	0.53	1.09	1.15	1.21	1.76
601–650	2149.81	35.85	4093.18	5.51	0.34	0.59	0.66	0.77	0.83	1.15
651–700	2316.35	40.05	4431.20	5.28	–	0.41	0.62	0.67	1.08	1.31
701–750	2660.47	48.19	4876.53	6.15	–	0.25	0.59	0.59	0.86	1.09
751–800	3019.32	50.29	5260.93	6.12	–	0.12	0.40	0.49	0.92	1.10
801–850	3154.56	55.00	5468.36	5.95	–	0.38	0.39	0.42	0.49	0.69
851–900	3290.80	59.76	5620.73	6.96	–	0.26	0.29	0.30	0.35	0.83
901–950	3497.78	66.16	6214.22	7.55	–	0.15	0.25	0.28	0.64	1.09
951–1000	3760.57	70.12	6551.99	8.88	–	0.29	0.42	0.47	0.90	1.37

Within each category the average over all results is shown. Whenever the specified time is not relevant or the optimizer failed to obtain a result within its scope. The distances shown are standardized

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).

Data availability This manuscript has no associated data or the data will not be deposited.

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.

References

1. 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.
2. Applegate D, Cook W, Dash S, Rohe A. Solution of a min-max vehicle routing problem. INFORMS J Comput. 2002;14:132–43.
3. Applegate DL, Bixby RE, Chvatal V, Cook WJ. The traveling salesman problem: a computational study. Princeton: Princeton University Press; 2006.
4. Azadnia AH, Taheri S, Ghadimi P, Samanm MZM, Wong KY. Order batching in warehouses by minimizing total tardiness: a hybrid approach of weighted association rule mining and genetic algorithms. Sci World J. 2013. https://doi.org/10.1155/2013/246578.
5. Bozer YA, Kile JW. Order batching in walk-and-pick order picking systems. Int J Prod Res. 2008;46(7):1887–909.
6. Briant O, Cambazard H, Cattaruzza D, Catusse N, Ladier A-L, Ogier M. An efficient and general approach for the joint order batching and picker routing problem. Eur J Oper Res. 2020;285(2):497–512. https://doi.org/10.1016/j.ejor.2020.01.059.
7. Bué M, Cattaruzza D, Ogier M, Semet F. A two-phase approach for an integrated order batching and picker routing problem. In: Dell'Amico M, Gaudioso M, Stecca G, editors. A view of operations research applications in Italy, 2018. AIRO springer series, vol 2. Cham: Springer; 2019. p. 3–18. https://doi.org/10.1007/978-3-030-25842-9_1.
8. Cergibozan Ç, Tasan A. Genetic algorithm based approaches to solve the order batching problem and a case study in a distribution center. J Intell Manuf. 2020. https://doi.org/10.1007/s10845-020-01653-3.
9. Chen M-C, Wu H-P. An association-based clustering approach to order batching considering customer demand patterns. Omega. 2005;33(4):333–43. https://doi.org/10.1016/j.omega.2004.05.003.
10. Cordeau J-F, Laporte G, Savelsbergh M, Vigo D. Vehicle Routing. In: Transportation, handbooks in operations research and management science vol. 14. 2007. p. 195–224.
11. Defryn C, Sörensen K. A fast two-level variable neighborhood search for the clustered vehicle routing problem. Comput Oper Res. 2017;83:78–94. https://doi.org/10.1016/j.cor.2017.02.007.
12. Esposito C, Castiglione A, Choo K-KR. Challenges in delivering software in the cloud as microservices. IEEE Cloud Comput. 2016;3(5):10–4. https://doi.org/10.1109/MCC.2016.105.
13. Gademann AJRM (noud), Van Den Berg JP, Van Der Hoff HH. An order batching algorithm for wave picking in a parallel-aisle warehouse. IIE Transactions. 2001;33(5):385–98.
14. Henn S. Algorithms for on-line order batching in an order picking warehouse. Comput Oper Res. 2012;39(11):2549–63.
15. Henn S, Koch S, Doerner KF, Strauss C, Wäscher G. Metaheuristics for the order batching problem in manual order picking systems. Bus Res. 2010;3(1):82–105.
16. 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.
17. Ho Y-C, Su T-S, Shi Z-B. Order-batching methods for an order-picking warehouse with two cross aisles. Comput Ind Eng. 2008;55(2):321–47.
18. Jiang X, Zhou Y, Zhang Y, Sun L, Hu X. Order batching and sequencing problem under the pick-and-sort strategy in online supermarkets. Proc Comput Sci. 2018;126:1985–93.
19. Kruk S. Practical python AI projects: mathematical models of optimization problems with google OR-tools. New york: Apress; 2018.
20. Kulak O, Sahin Y, Taner ME. Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. Flex Serv Manuf J. 2012;24(1):52–80. https://doi.org/10.1007/s10696-011-9101-8.
21. Li J, Huang R, Dai JB. Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. Int J Prod Res. 2017;55(2):447–61. https://doi.org/10.1080/00207543.2016.1187313.
22. Mackay DJC. Introduction to Monte Carlo Methods. In: Jordan MI, editor. Learning in graphical models. NATO ASI Series, vol 89. Dordrecht: Springer; 1998. https://doi.org/10.1007/978-94-011-5014-9_7
23. Masae M, Glock CH, Grosse EH. Order picker routing in warehouses: a systematic literature review. Int J Prod Econ. 2020;224:107564.
24. Naumenko T, Petrenko A. Analysis of problems of storage and processing of data in serverless technologies. Technol Audit Prod Reserves. 2021;2(2):58.
25. Oxenstierna J, Malec J, Krueger V. Layout-agnostic order-batching optimization. In: Mes M, Lalla-Ruiz E, Voß S, editors. Computational logistics. ICCL 2021. Lecture notes in computer science, vol 13004. Cham: Springer; 2021. https://doi.org/10.1007/978-3-030-87672-2_8
26. Oxenstierna J, Malec J, Krueger V. Analysis of computational efficiency in iterative order batching optimization. In: Proceedings of the 11th international conference on operations research and enterprise systems–ICORES. 2022. p. 345–53. https://doi.org/10.5220/0010837700003117
27. Rajasekaran S, Reif JH. Nested annealing: a provable improvement to simulated annealing. Theoret Comput Sci. 1992;99(1):157–76. https://doi.org/10.1016/0304-3975(92)90177-H.

28. Ratliff H, Rosenthal A. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Oper Res. 1983;31:507–21.
29. van Rensburg LJ. Artificial intelligence for warehouse picking optimization—An NP-hard problem [Master's Thesis]. Uppsala University; 2019.
30. Roodbergen KJ, Koster R. Routing methods for warehouses with multiple cross aisles. Int J Prod Res. 2001;39(9):1865–83.
31. Scholz A, Schubert D, Wäscher G. Order picking with multiple pickers and due dates—simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. Eur J Oper Res. 2017;263(2):461–78. https://doi.org/10.1016/j.ejor.2017.04.038.
32. Sharp GP, Gibson DR. Order batching procedures. Eur J Oper Res. 1992;58:57–67.
33. Tak H, Meng X-L, van Dyk DA. A repelling-attracting metropolis algorithm for multimodality. J Comput Graph Stat. 2018;27(3):479–90. https://doi.org/10.1080/10618600.2017.1415911.
34. Valle CA, Beasley BA. Order batching using an approximation for the distance travelled by pickers. Eur J Oper Res. 2019;284:460–84.
35. Valle CA, Beasley JE, da Cunha AS. Optimally solving the joint order batching and picker routing problem. Eur J Oper Res. 2017;262(3):817–34.
36. Wales DJ, Doye JPK. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. J Phys Chem A. 1997;101:5111–6.
37. Yu VF, Maulidin A, Redi AP, Lin SW, Yang CL. Simulated annealing with restart strategy for the path cover problem with time windows. Mathematics. 2021;9(14):1625. https://doi.org/10.3390/math9141625.