Synchronous parallel heuristics for solving the joint order batching and picker routing problem

Tran, Son; Almeida, Rui Jorge; Defryn, Christof & VAN NIEUWENHUYSE, Inneke (2024) Synchronous parallel heuristics for solving the joint order batching and picker routing problem. In: 2024 IEEE Congress on Evolutionary Computation (CEC), p. 1 -8.

DOI: 10.1109/CEC60901.2024.10612134
Handle: http://hdl.handle.net/1942/45632

Synchronous parallel heuristics for solving the joint order batching and picker routing problem

Son Thai Tran*, Rui Jorge Almeida*†, Christof Defryn‡, Inneke Van Nieuwenhuyse§
*Dept. of Quantitative Economics, School of Business and Economics, Maastricth University
Maastricht, The Netherlands. Email: t.tran@maastrichtuniversity.nl
†Dept. of Data Analytics and Digitization, School of Business and Economics, Maastricth University
Maastricht, The Netherlands. Email: rj.almeida@maastrichtuniversity.nl
‡Dept. of Engineering Management, Faculty of Business and Economics, University of Antwerp
Antwerp, Belgium. Email: christof.defryn@uantwerpen.be
§FlandersMake@UHasselt and Data Science Institute, Faculty of Sciences, Hasselt University
Hasselt, Belgium. Email: inneke.vannieuwenhuyse@uhasselt.be

2024 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-0836-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CEC60901.2024.10612134

***Abstract*—The joint order batching and picker routing problem is an important problem for improving warehouse efficiency. The goal is to minimize the total distance travel of picking customer orders. It is NP-hard, indicating that exact solutions are intractable for large instances. Many solution methods provided in the current literature use local search to solve the problem sequentially, often including a complex searching procedure for the order batching problem (OBP) followed by a simple heuristic for the picker routing problem (PRP). In this paper, we propose three heuristics that jointly solves the OBP and the PRP: two of these are combinations of variable neighborhood search, simulated annealing, and tabu search, while the third one uses guided local search. We discuss how the search procedure of the proposed algorithms can be parallelized, and benchmark them against state-of-the-art heuristics using well-studied instances. The results show a reduction of 3.06% to 7.63% in the geometric mean of the total travel distance across 64 instances. Increasing the number of threads in parallelization leads to diminishing returns in reducing running time and has no statistical effect on travel distance. In contrast, increasing the chunk size results in a linear increase in running time for all proposed algorithms, and improves the travel distance obtained when the batch size is 75 items.**

***Index Terms*—Joint order batching and picker routing, Heuristics, Parallelization**

I. Introduction

Order picking is one of the most critical processes in warehouse management [1], [2]. It is the process of selecting and retrieving the products from storage locations to fulfill customer orders [3]. One of the common objectives of order picking is to minimize the travel distance to pick all orders.

Research on the joint order batching and picker routing problem (JOBPRP) either solves the problem using an exact method or tackles both problems sequentially, focusing mainly on the order batching problem while using simple heuristics for the picker routing problem, in view of making the JOBPRP more manageable. This paper proposes algorithms to solve both problems jointly, using metaheuristics to address the order batching problem (OBP), such as variable neighborhood search (VNS) [4], [5], and one of the best heuristics, Lin-Kernighan (LK), to solve the picker routing problem (PRP).

While parallel computing has been applied for metaheuristics [6], its implementation in order picking research remains scarce, and it has never been used for solving the JOBPRP [7]. We propose algorithms that can be parallelized to achieve high computational efficiency and high solution quality, taking advantage of the computational capabilities of modern computers. By speeding up the search procedure via parallelization, we can employ more complex and powerful algorithms to solve the PRP, thus obtaining better solutions within a reasonable running time.

The resulting algorithms were implemented in the Rust programming language, which provides compile-time guarantees for memory and thread safety, enabling the detection of parallelization-related errors at compile-time. Rust's design also enables the program to achieve similar speed to languages like C/C++, while mitigating the memory hazards often associated with manual memory management [8]. Bugs involving parallelization are notoriously challenging to track down (e.g., unsafe mutation of an object via multi-threading, which can lead to data races) [8], making Rust a valuable programming language for parallelization in metaheuristics.

II. Related literature

The JOBPRP consists of two subproblems: the OBP and the PRP [9], [10]. Both problems are strongly connected [11], as one problem requires assumptions about the other problem and vice versa, i.e., the PRP can only be solved knowing the batches proposed by the OBP, while the OBP can only be solved knowing the cost of picking a batch of orders, which requires to solve the PRP.

A. Picker routing problem

Given a batch of orders that will be picked in a single pick tour, the PRP aims to minimize the total travel distance to pick all corresponding items by determining the optimal sequence to visit the pick locations [12]. Even in a rectangular warehouse with parallel aisles, the PRP is known to be NP-hard [12]. In [13], an exact algorithm is presented to solve the PRP in a single-block warehouse with parallel aisles; it was

extended to a two-block warehouse in [1]. For more than two blocks (and up to eight cross-aisles), a dynamic programming approach was proposed in [14]; yet, the complexity of this algorithm grows exponentially with the number of cross-aisles.

Heuristics for solving the PRP often consist of simple rules, e.g., the s-shape, largest gap, and combined routing heuristic. While these are intuitive and can be run in linear time, they do not guarantee optimal tours [3]. Furthermore, as the PRP can be formulated as a NP-hard Steiner traveling salesman problem (TSP) [15], state-of-the-art heuristics to solve the TSP can also be applied to the PRP. In [15], the Lin-Kernighan-Helsgaun algorithm, one of the best heuristics in TSP, is applied to solve the PRP, resulting in an average reduction in route distance of 47% compared to simple warehouse routing heuristics. In our study, we use the LK heuristic, a less performant but faster algorithm than the LKH heuristic. During our preliminary study, we achieved solutions close to optimal for the PRP with instance sizes of up to 75 items. Despite its good performance, the use of complex routing heuristics in the warehouse literature is limited [5].

B. Order batching problem

In the OBP, the routing strategy is considered known in advance. Given this strategy together with a maximum batch size, e.g. the maximum number of items that can be placed in the picking cart, and a list of orders to be picked, the OBP aims to minimize the total travel distance of picking customer orders by combining them into batches. The OBP is known to be NP-hard (it can only be solved in polynomial time when batches consist of only two orders [16]). Consequently, for realistic instances, OBP research mostly focuses on heuristics. A Mixed Integer Linear Programming (MILP) formulation was introduced in [17] to solve the OBP exactly, assuming that the routing heuristic is either s-shape, return, or midpoint.

Heuristics to solve OBP can be classified into five main types [18]: priority rule based algorithms, seed algorithms, savings algorithms, data mining, and metaheuristics. The first three types are constructive approaches, in which batches are created sequentially by assigning orders to a batch until the remaining capacity is insufficient to include even the smallest unassigned order [19]. A detailed review of the first four approaches is provided in [18].

Many metaheuristics to solve the OBP start from an initial solution created by a constructive approach, and aim to improve the incumbent solution by exploring its neighborhood [20]. Different types of metaheuristics have been proposed: iterated descent [16], genetic algorithm [21], attribute-based hill climber [18], tabu search [17], and multi-start variable neighborhood search (MS-VNS) [4].

C. Joint order batching and picker routing problem

JOBPRP aims to optimize batching and routing decisions *simultaneously*. In that way, higher distance savings can be obtained compared to optimizing each problem separately [12]. A detailed meta-review of the approaches used to solve the JOBPRP is provided in [7].

[9] develop a branch-and-cut algorithm to solve the JOBPRP for up to 20 orders. [22], [23] employ a branch-price-and-cut algorithm that significantly improves upon the exact approach proposed by [17], considering a single-block warehouse. To optimally solve the PRP, [23] use dynamic programming, exploiting the special characteristics of a single-block warehouse to obtain a solution in linear time.

Different metaheuristics have been proposed to solve the JOBPRP: genetic algorithm [24], a hybrid approach combining particle swarm optimization and ant colony optimization [25], iterated local search [10], a combination of discrete evolutionary particle swarm optimization, nearest neighbor, and two-opt [26], a heuristic based on column generation [27], and a two-level VNS (2-VNS) (see [5], who model the JOBPRP as a variant of the clustered vehicle routing problem).

D. Parallelization

There are four different approaches to parallelize the VNS in [28] and [29]. All four of these can be generalized and applied to any arbitrary metaheuristic that includes a local search phase. In this paper we apply one of these approaches to the local search phase of different metaheuristics.

The synchronous approach parallelizes the local search phase of a sequential metaheuristic by splitting the execution of the local search into several threads. The replicated approach explores a wider portion of the solution space, using a multi-start strategy and executing several metaheuristics in parallel. The replicated shaking approach follows a classical main–worker scheme, where the main thread executes the metaheuristics, and each worker executes the shake and local search methods. The cooperative approach also uses the main-worker scheme, yet it considers the cooperative exploration of different neighborhoods by different threads, while the main thread communicates the best solution found to the workers every time it is updated, allowing the workers to continue the search from the new best solution.

III. Parallel heuristics for JOBPRP

In this section, we propose three heuristics to solve the JOBPRP, all based on the concept of local search. We first present the neighborhood structure and the procedure to explore the neighborhood; next, we present the algorithms and discuss how to parallelize them.

A. Neighborhood structure

Let S be the set of feasible solutions of the JOBPRP. A solution $s' \in \mathcal{S}$ is called a neighbor of solution $s \in \mathcal{S}$ if it can be obtained by applying a single local transformation to s [18]. We define this single local transformation in the JOBPRP as either a `RELOCATE` or `SWAP` move. These two moves are constructed from a sequence of the two basic moves: `INSERT` and `REMOVE`. Specifically, given a solution s, $\texttt{INSERT}(O_i, B_k)$ adds order O_i to batch B_k while $\texttt{REMOVE}(O_j, B_l)$ removes order O_j from batch B_l in which $1 \leq i, j \leq n$ and $1 \leq k, l \leq m$. In this notation, n and m are the number of orders and batches, respectively.

A RELOCATE move of order O_i from batch B_k to batch B_l is equivalent to removing order O_i from batch B_k and inserting it to batch B_l:

$$\texttt{RELOCATE}(s, O_i, B_k, B_l) \coloneqq \texttt{REMOVE}(O_i, B_k) + \texttt{INSERT}(O_i, B_l)$$

We can define the neighborhood of a solution s with regard to the RELOCATE move as:

$$\mathcal{N}_{\texttt{RELOCATE}}(s) = \{s' \leftarrow \texttt{RELOCATE}(s, O_i, B_k, B_l)\} \quad (1)$$

As we only consider feasible moves, order O_i is assigned to batch B_l only if batch B_l has enough capacity (i.e., $q_{O_i} + q_{B_l} \leq Q$, where q_x refers to the quantity of items present in x), and the two batches are not the same, $k \neq l$.

A SWAP move of order O_i from batch B_k and order O_j from batch B_l is equivalent to removing the two orders from their current batches and inserting them to the other batch:

$$\begin{aligned}\texttt{SWAP}(s, O_i, B_k, O_j, B_l) \coloneqq \texttt{REMOVE}(O_i, B_k) + \texttt{REMOVE}(O_j, B_l) \\ + \texttt{INSERT}(O_j, B_k) + \texttt{INSERT}(O_i, B_l)\end{aligned}$$

We can define the neighborhood of a solution s with regard to SWAP move as:

$$\mathcal{N}_{\texttt{SWAP}}(s) = \{s' \leftarrow \texttt{SWAP}(s, O_i, B_k, O_j, B_l)\} \quad (2)$$

Similar to the RELOCATE move, the SWAP move is performed only if s' is a feasible solution, i.e., $(q_{B_k} - q_{O_i} + q_{O_j} \leq Q) \wedge (q_{B_l} - q_{O_j} + q_{O_i} \leq Q)$, with $(i \neq j) \wedge (k \neq l)$.

Instead of searching $\mathcal{N}_{\texttt{RELOCATE}}(s)$ and $\mathcal{N}_{\texttt{SWAP}}(s)$ separately, we consider a single union neighborhood $\mathcal{N}_{\texttt{RELOCATE}} \cup \mathcal{N}_{\texttt{SWAP}}$. This approach allows us to explore both neighborhoods simultaneously, and eliminates the need to determine the order (or alternating order) in which to visit both neighborhoods.

As opposed to other heuristics in OBP and JOBPRP, we do not include the possibility to create new batches in our neighborhood structure. In many works on order picking, good solutions often have the minimal number of batches [5]. This becomes even more apparent when there is a cost associated with setting up or preparing a batch, which is always expected in real-life order picking. We do, however, allow for a reduction in the number of batches.

B. Exploring the neighborhood

The local search procedure in JOBPRP often starts from an initial solution, $s_0 \in \mathcal{S}$, generated by constructive heuristics such as seed algorithms [30]. The procedure then searches for a sequence of feasible solutions that have a better objective value than the previous one. If the problem is bounded, the sequence converges to a local optimum [18].

There are two classical strategies for exploring a neighborhood: best improvement and first improvement [4]. Under the best improvement strategy, the current neighborhood is completely explored using a fully deterministic procedure, and then the best move is made. In contrast, under the first improvement strategy, the first move that improves the current best solution is chosen to avoid exploring the entire neighborhood. Consequently, iterations performed in the first improvement strategy are generally faster [4]. When the neighborhood of the search is large, the best improvement strategy is expensive; yet, there is no general guarantee that it leads to better solutions than the first improvement strategy [4]. The downside of the first improvement is that decisions may be made too quickly, missing high quality solutions and prolonging the search time. Consequently, we decide to seize the best of both worlds by considering best improvement out of a chunk of the neighborhood (i.e., a random subset of all neighbouring solutions).

As we start the search procedure, the number of solutions that outperform the current solution is likely very high, making it more likely to encounter the move that improves our current best solution. If there is no improving solution within the chunk, a new chunk of the neighborhood will be considered. If all the chunks of the neighborhood are exhausted, the search strategy is equivalent to the best improvement.

If there is no improving solution when searching the entire neighborhood, our algorithm is stuck in a local minimum. To escape, one can employ a variety of strategies, e.g., accepting an inferior solution, prohibiting or discouraging the search from revisiting certain neighbors, destroying and repairing the solution [31]. The heuristics proposed in this paper employ the first two strategies to escape local minima.

C. Proposed algorithms

We propose three algorithms that consider a large neighborhood structure ($\mathcal{N}_{\texttt{RELOCATE}} \cup \mathcal{N}_{\texttt{SWAP}}$), and use delta-evaluation in their local search procedure. The delta-evaluation determines whether a neighbor solution is improving or not by calculating only the travel distance difference of the two modified batches [4], [5], [20]. The first two algorithms are a hybrid of VNS, simulated annealing (SA), and tabu search (TS). The third one is based on guided local search (GLS).

VNS, proposed by [32], explores the neighborhood of the current incumbent solution to find improving solutions. A local search method is applied repeatedly to get from solutions in the neighborhood to local optima. SA considers a neighbor solution of the current incumbent solution, and probabilistically decides whether to move to the neighbor solution or keep the incumbent solution. This randomness allows the heuristic to escape local optima, by accepting inferior solutions [31]. TS, introduced by [33], pursues local search, escaping local optima by allowing non-improving moves. Cycling back to recently visited solutions is prevented by the use of tabu lists that record several most recently visited solutions of the search. The main aim of GLS (proposed by [34]) is to guide the underlying local search procedures to efficiently and effectively explore the search space. This is done through the penalization of suboptimal solution features as part of the objective function.

VNS, SA, and TS are popular in JOBPRP as they are straightforward and easy to implement. GLS, however, has not yet been applied to JOBPRP, despite its success in solving the TSP [31], [34]. We propose the following hybrid algorithms:

a) *A hybrid of VNS, SA, and TS (VNS+SA+TS):* at the core, the algorithm uses a local search subroutine (VNS component), which performs delta-evaluation in parallel on a chunk of neighbors. If there is an improving neighbor within the chunk, the algorithm will carry out

the move. If none of the neighbors in the chunk yields an improvement, the algorithm can accept an inferior solution with a specified probability (SA component), while also keeping a list of tabu moves (TS component). For the SA component, we adopt the Metropolis algorithm as described in [31].

b) *A hybrid of VNS and SA (VNS+SA):* this algorithm is a simpler variant of the former, as it does not include a tabu list.

c) *GLS:* to escape local optima (i.e., when no improving solution can be found within the *entire neighborhood*), we keep a penalty table to promote exploration in novel areas of the search space. When a local optimum is reached, the algorithm assigns a penalty for the batches that have the highest utility value (i.e., the lowest average cost per item picked). In that way, it discourages moves to solutions that include these batches and promotes the exploration of other neighborhoods.

D. Parallelization of the proposed algorithms

By evaluating a chunk of neighbors in each iteration, we can parallelize the evaluations by distributing the delta-evaluations across multiple threads. This is a synchronous parallel approach as there is no communication between threads. When all threads finish the delta-evaluations, the results are gathered and compared to determine which neighbor yields the best improvement. In other words, we synchronize the results of the search. We implement the synchronous parallel strategy using Rayon, a framework in Rust to parallelize work. By default, Rayon uses work-stealing scheduling, meaning that each thread maintains its own local queue of work, and can steal work from other threads when this queue is empty. This is very efficient in our case, as in a typical computer, there is a very high chance that some cores are more powerful than others, such that the threads on these cores may finish their work earlier. By allowing them to steal work from the threads on less powerful cores, we can balance the workload.

The pseudo-code for the proposed parallelized algorithms are presented in Algorithms 1 and 2. Note that the cost is measured as travel distance. Removing the tabu list from Algorithm 1 provides the procedure for VNS+SA. The hyper-parameters presented in Algorithm 1 and 2 are the initial temperature of SA (t), the maximum number of consecutive iterations without improving solution (K), the size of tabu list (T), the chunk size (C), and the number of threads (R).

Algorithm 1: Pseudo-code of VNS+SA+TS algorithm

Input : The set of unassigned orders, $\mathcal{O}$;
Result: The set of batches, $\mathcal{B}_{\text{best}}$;
$\mathcal{B}_{\text{current}} \leftarrow \mathcal{B}_{\text{best}}$ using a seed algorithm [30]; $k \leftarrow 1$;
while $k \leq K$ **do**
  $\mathcal{N} \leftarrow \mathcal{N}_{\texttt{RELOCATE}(\mathcal{B}_{\text{current}})} \cup \mathcal{N}_{\texttt{SWAP}(\mathcal{B}_{\text{current}})}$, as Eq. 1, 2;
  accepted $\leftarrow$ False;
  while $\mathcal{N} \neq \emptyset \wedge$ *accepted is False* **do**
    Take and delta-evaluate C neighbors from $\mathcal{N}$ on R threads using LK heuristic;
    $\mathcal{B}_{\text{cand}} \leftarrow$ neighbor s' with lowest cost and not in the tabu list;
    if *Tabu list is full* **then**
      Remove the oldest solution from the list;
    end
    Add $\mathcal{B}_{\text{cand}}$ to the tabu list;
    if *cost of* $\mathcal{B}_{cand} <$ *cost of* $\mathcal{B}_{current}$ **then**
      $\mathcal{B}_{\text{current}} \leftarrow \mathcal{B}_{\text{cand}}$; accepted $\leftarrow$ True;
      if *cost of* $\mathcal{B}_{current} <$ *cost of* $\mathcal{B}_{best}$ **then**
        $\mathcal{B}_{\text{best}} \leftarrow \mathcal{B}_{\text{current}}$; $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
      end
    else
      $k \leftarrow k + 1$;
      if *SA accepted* **then**
        $\mathcal{B}_{\text{current}} \leftarrow \mathcal{B}_{\text{cand}}$; accepted $\leftarrow$ True;
      end
    end
  end
end

IV. Simulation Study

In [20], a set of 5,760 warehouse instances is proposed to benchmark and compare the results of studies on order picking problems. In [4], a subset of 64 instances — found to be representative — is used. Half of the set follows an ABC storage policy (divided into subsets abc1 and abc2), while the remaining half follows a random storage policy (divided into subsets ran1 and ran2). Similarly, in [5], 32 instances of ran1 and ran2 are used. Our results will be compared to the MS-VNS algorithm of [4] (which is considered the state-of-the-art for heuristics in JOBPRP), and the 2-VNS (non-time restricted HausAdap variation) of [5] (as it outperforms the results from heuristics proposed by [32] and [20], and even in some cases the algorithm by [4]). We use the subset of instances provided by [4] in our study to provide the benchmark and comparison.

In [20], the subsets ran1 and abc1 are "s-shape instances", while ran2 and abc2 are "largest gap instances". Evidently, this becomes irrelevant in our experiments, as the proposed algorithms build their own routes. Each subset contains instances differing in the number of orders (n) and the batch size (Q). We simulate each (n, Q)-combination ten times on a computer with CPU Ryzen 7 5800x3d (8 physical cores, 16 logical cores), and calculate the average over the ten runs.

All instances are defined in a single-block warehouse. The warehouse consists of 10 pick aisles, each having 45 pick locations on each side. In total, there are 900 pick locations. Each pick location has a width of 1m. It takes 5m to move from one pickaisle to the next and 1.5m to move from the depot to the first pick location. Under the ABC storage policy, SKUs with high order frequency are stored close to the depot. The general warehouse layout for an ABC storage policy can be visualized as in Figure 1. Under the random storage policy,

Algorithm 2: Pseudo-code of GLS algorithm

Input : The set of unassigned orders, $\mathcal{O}$;
Result: The set of batches, $\mathcal{B}_{\text{best}}$;
$\mathcal{B}_{\text{current}} \leftarrow \mathcal{B}_{\text{best}}$ using a seed algorithm [30]; $k \leftarrow 1$;
while $k \leq K$ **do**
  $\mathcal{N} \leftarrow \mathcal{N}_{\text{RELOCATE}(\mathcal{B}_{\text{current}})} \cup \mathcal{N}_{\text{SWAP}(\mathcal{B}_{\text{current}})}$, as Eq. 1, 2;
  accepted $\leftarrow$ False;
  while $\mathcal{N} \neq \emptyset \wedge$ *accepted is False* **do**
    Take and delta-evaluate C neighbors from $\mathcal{N}$ on R threads using LK heuristic;
    $\mathcal{B}_{\text{cand}} \leftarrow$ neighbor s' with lowest augmented cost;
    if *augmented cost of* $\mathcal{B}_{cand}$ *< augmented cost of* $\mathcal{B}_{current}$ **then**
      $\mathcal{B}_{\text{current}} \leftarrow \mathcal{B}_{\text{cand}}$; accepted $\leftarrow$ True;
      if $\mathcal{B}_{current}$ *< cost of* $\mathcal{B}_{best}$ **then**
        $\mathcal{B}_{\text{best}} \leftarrow \mathcal{B}_{\text{current}}$; $k \leftarrow 1$;
      else
        $k \leftarrow k + 1$;
      end
    else
      $k \leftarrow k + 1$;
      if $\mathcal{N} = \emptyset$ **then**
        Update the penalty table;
      end
    end
  end
end

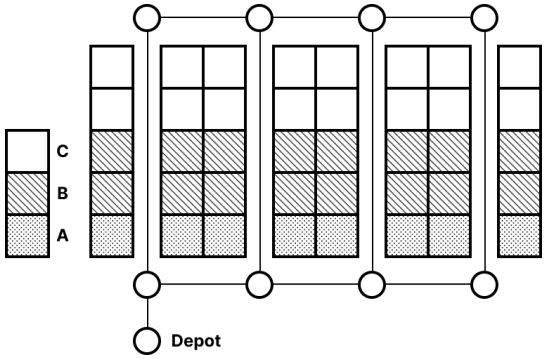

Fig. 1. Visual representation of a single block ABC warehouse layout (SKUs with high order frequency are stored close to the depot).

SKUs with high order frequency are scattered around the warehouse randomly.

We do not extensively tune the hyper-parameters of our algorithms but instead conduct a simple manual search to find a balance between the travel distance obtained and the running time of the algorithms. For all studies, the following hyper-parameters are fixed: $t = 100$, $K = 150$, $T = 32$. In the comparison of travel distance and running time across all subsets of instances, $C = 128$ and $R = 16$. In the study of the effect of the number of threads, $C = 64$ and $R \in \{1, 2, 4, 8, 16\}$. In the study of the effect of chunk size, $R = 16$ and $C \in \{16, 32, 64, 128, 256\}$.

TABLE I
GEOMETRIC MEAN TRAVEL DISTANCE OBTAINED BY THE VNS+SA+TS ALGORITHM (IN METERS), AND DIFFERENCES (IN %) OBSERVED FOR THE COMPETING ALGORITHMS (MS-VNS [4] AND 2-VNS [5]). LOWER IS IMPROVEMENT.

		Geo. mean	Difference (%)			
Subset	Q	VNS+SA+TS	VNS+SA	GLS	MS-VNS	2-VNS
abc1	30	10802	0.32	0.14	6.92	-
	45	7428	0.38	0.65	4.83	-
	60	6015	0.47	0.35	3.52	-
	75	4710	0.28	0.32	3.06	-
abc2	30	10678	0.12	-0.01	6.92	-
	45	7549	0.17	0.56	4.64	-
	60	5891	0.25	0.37	4.31	-
	75	4711	0.08	0.53	3.38	-
ran1	30	14533	0.14	0.11	7.47	5.84
	45	10001	0.63	0.17	4.95	6.04
	60	8017	0.07	0.19	4.27	5.61
	75	6304	0.19	0.16	3.41	5.09
ran2	30	14419	0.25	0.08	7.63	5.94
	45	10096	0.55	0.65	5.67	6.28
	60	7915	0.28	-0.03	4.83	6.23
	75	6242	0.34	-0.02	3.20	4.90

V. RESULTS

A. Travel distance comparison

Table I reports the average geometric mean of the travel distance obtained with different algorithms for different batch sizes Q. Each row in Table I thus combines the results of all (n, Q) settings that have the same batch size. We opt for this approach as the batch size affects the running time of the routing methods and, thus, significantly impacts the running time of the algorithms. We report the geometric mean of travel distance to mitigate the effect of different numbers of orders.

Overall, the results of the VNS+SA+TS outperform the results of the MS-VNS from [4] by 3.06% to 7.63%, and outperforms the results of the 2-VNS by [5] by 4.90% to 6.28%. The most substantial improvement over MS-VNS is obtained for the smallest batch sizes (30 items); it decreases as the batch size increases. This may be caused by the fact that, for larger batch sizes, the benefit of a good heuristic for PRP reduces: as the warehouse has one single block and there are many locations to visit, the difference compared to simple heuristics becomes smaller. In warehouses with a more complex layout or more blocks, we expect the improvements to be more significant. There is no clear trend compared to the 2-VNS but the improvement when the batch size is 75 items is less than when the batch size is either 30, 45, or 60 items.

Considering only the three proposed algorithms, we see that the VNS+SA+TS performs better than the VNS+SA and GLS by a small amount and there is no clear difference between VNS+SA and GLS. When performing paired t-tests on the travel distance between VNS+SA+TS and VNS+SA, between VNS+SA+TS and GLS, and between VNS+SA and GLS, the first two pairs are statistically different (p-values are 0.00 and

0.00, respectively) while the last pair is not (p-value is 0.32), at a significance level of $\alpha = 0.05$. Additionally, Table II reports the coefficient of variation, which is the ratio of the standard deviation to the mean over ten runs for each combination of instance, for our algorithms. The maximum variation coefficient of travel distance for all proposed algorithms is 0.01, indicating the consistent quality of obtained solutions.

B. Running time comparison

Table III reports the average geometric mean of the running time for our three algorithms, for different batch sizes (we cannot fairly compare with the running times reported for the other algorithms, as these were obtained with different machine settings). In general, the running times are lowest for VNS+SA, followed by GLS, while VNS+SA+TS has the longest running times. As the increase in solution quality obtained by VNS+SA+TS is less than 1.00%, opting for VNS+SA or GLS is likely attractive. From Table III, we can observe that VNS+SA yields an improvement in running time over VNS+SA+TS of up to 30.00%, while the improvements of GLS are less consistent (it sometimes even has a longer running time). When performing paired t-tests on the running time between VNS+SA+TS and VNS+SA, between VNS+SA+TS and GLS, and between VNS+SA and GLS, we observe that the first two pairs are statistically different (p-values are 0.00 and 0.00 respectively), while the last pair is not (p-value is 0.29) at a significance level of $\alpha = 0.05$. Furthermore, the coefficients of variation for the running time of the proposed algorithms in Table II, ranging from 0.18 to 0.58, indicate significant variations in the running time between the ten runs of the same instance.

Flamegraphs are used to visualize where time is spent in the program. The main functions of the program are positioned close to the bottom on the y-axis while the called functions are stacked on top. The width of each box on the x-axis represents the proportion of the total CPU time that the given function utilizes. The wider the box, the more CPU time the program spends executing the corresponding function. The box color is added for visualization purposes only.

Analyzing the flamegraph reveals that the program allocates approximately 75% of its running time to solving the PRP (using the LK) and around 25% to synchronization, generating new moves, and the overhead of the parallelization heuristics. This highlights the benefit of implementing parallelization to speed up the search, enabling the use of better routing methods.

C. Performance when varying the number of threads

In this section, we report the average geometric mean of the running time and the travel distance obtained, when varying the number of threads. We choose to provide a detailed analysis for batch sizes of 30 and 75 items (subset ran1), as these represent either the easiest or the most difficult instances to solve.

As can be seen from Figures 2 and 3, there is a slight variation in the obtained travel distance as the number of

TABLE II
COEFFICIENT OF VARIATION OF TRAVEL DISTANCE AND RUNNING TIME ACROSS TEN RUNS USING PROPOSED ALGORITHMS

			VNS+SA+TS		VNS+SA		GLS	
Subset	Q	n	Dist.	Time	Dist.	Time	Dist.	Time
abc1	30	40	0.00	0.26	0.01	0.04	0.00	0.11
		60	0.00	0.30	0.00	0.26	0.00	0.13
		80	0.00	0.23	0.00	0.09	0.00	0.32
		100	0.00	0.36	0.00	0.28	0.00	0.39
	45	40	0.01	0.35	0.01	0.30	0.00	0.23
		60	0.00	0.27	0.01	0.27	0.00	0.32
		80	0.00	0.44	0.00	0.38	0.00	0.44
		100	0.01	0.38	0.01	0.37	0.00	0.34
	60	40	0.01	0.48	0.01	0.38	0.00	0.27
		60	0.01	0.14	0.01	0.38	0.01	0.41
		80	0.01	0.26	0.01	0.26	0.01	0.23
		100	0.01	0.23	0.01	0.35	0.01	0.38
	75	40	0.00	0.45	0.01	0.15	0.00	0.39
		60	0.01	0.28	0.01	0.40	0.01	0.31
		80	0.01	0.34	0.01	0.42	0.00	0.21
		100	0.01	0.33	0.01	0.37	0.01	0.54
abc2	30	40	0.00	0.16	0.00	0.32	0.00	0.08
		60	0.00	0.29	0.00	0.17	0.00	0.33
		80	0.00	0.47	0.00	0.27	0.00	0.28
		100	0.00	0.39	0.00	0.23	0.00	0.52
	45	40	0.00	0.29	0.01	0.29	0.01	0.44
		60	0.01	0.27	0.01	0.27	0.01	0.44
		80	0.01	0.26	0.00	0.29	0.01	0.34
		100	0.01	0.25	0.01	0.38	0.01	0.38
	60	40	0.01	0.31	0.00	0.31	0.00	0.21
		60	0.01	0.31	0.01	0.30	0.01	0.35
		80	0.01	0.40	0.01	0.26	0.01	0.39
		100	0.01	0.41	0.01	0.42	0.00	0.22
	75	40	0.01	0.48	0.01	0.32	0.00	0.28
		60	0.01	0.38	0.01	0.31	0.01	0.27
		80	0.01	0.33	0.01	0.20	0.01	0.34
		100	0.01	0.27	0.01	0.39	0.01	0.32
ran1	30	40	0.00	0.35	0.00	0.05	0.00	0.17
		60	0.00	0.25	0.00	0.03	0.00	0.04
		80	0.00	0.18	0.00	0.27	0.00	0.17
		100	0.00	0.30	0.00	0.40	0.00	0.27
	45	40	0.00	0.33	0.01	0.21	0.00	0.37
		60	0.01	0.25	0.01	0.40	0.01	0.34
		80	0.01	0.39	0.01	0.40	0.00	0.38
		100	0.00	0.35	0.00	0.31	0.00	0.45
	60	40	0.00	0.28	0.00	0.28	0.00	0.36
		60	0.01	0.41	0.01	0.44	0.01	0.15
		80	0.00	0.23	0.01	0.23	0.01	0.58
		100	0.00	0.28	0.00	0.36	0.01	0.44
	75	40	0.01	0.30	0.00	0.40	0.01	0.32
		60	0.01	0.42	0.01	0.51	0.01	0.41
		80	0.00	0.48	0.00	0.31	0.01	0.33
		100	0.01	0.44	0.01	0.34	0.02	0.48
ran2	30	40	0.00	0.41	0.00	0.17	0.00	0.07
		60	0.00	0.29	0.00	0.17	0.00	0.09
		80	0.00	0.37	0.00	0.22	0.00	0.34
		100	0.00	0.28	0.00	0.36	0.00	0.34
	45	40	0.00	0.33	0.02	0.22	0.00	0.39
		60	0.00	0.26	0.00	0.35	0.00	0.25
		80	0.01	0.39	0.01	0.26	0.01	0.38
		100	0.01	0.33	0.01	0.45	0.01	0.28
	60	40	0.00	0.31	0.01	0.25	0.00	0.28
		60	0.01	0.50	0.01	0.41	0.00	0.41
		80	0.01	0.44	0.01	0.59	0.00	0.49
		100	0.00	0.30	0.00	0.30	0.00	0.33
	75	40	0.01	0.27	0.01	0.29	0.01	0.26
		60	0.01	0.40	0.01	0.43	0.01	0.37
		80	0.01	0.45	0.01	0.31	0.01	0.29
		100	0.01	0.25	0.00	0.33	0.01	0.39

TABLE III
GEOMETRIC MEAN RUNNING TIME (IN SECONDS) FOR THE VNS+SA+TS, AND % DIFFERENCE OBSERVED FOR THE VNS+SA AND GLS. LOWER IS IMPROVEMENT.

		Geo. mean	Diff. (%)				Geo. mean	Diff. (%)	
S.set	Q	VNS+ SA+TS	VNS +SA	GLS	S.set	Q	VNS+ SA+TS	VNS +SA	GLS
abc1	30	22	-27.27	-18.18	**ran1**	30	20	-30.00	-25.00
	45	88	-19.32	-19.32		45	77	-15.58	0.00
	60	149	-12.75	2.01		60	137	-9.49	0.73
	75	227	-8.81	-11.89		75	198	-19.19	-9.60
abc2	30	22	-18.18	-4.55	**ran2**	30	19	-26.32	-15.79
	45	87	-18.39	-17.24		45	85	-21.18	-27.06
	60	159	-20.13	-8.81		60	134	0.75	21.64
	75	239	-13.39	-21.34		75	189	-16.93	4.23

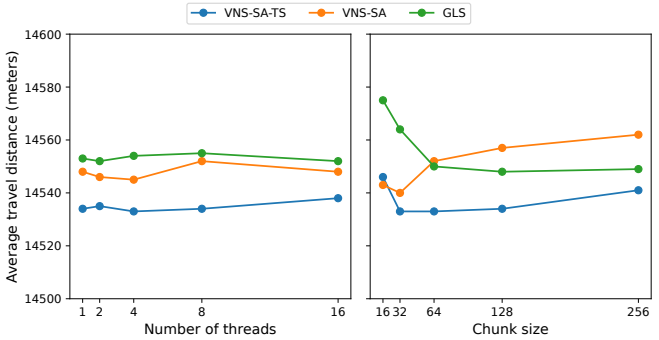

Fig. 2. Travel distance obtained vs number of threads and chunk size, for a batch size of 30 items.

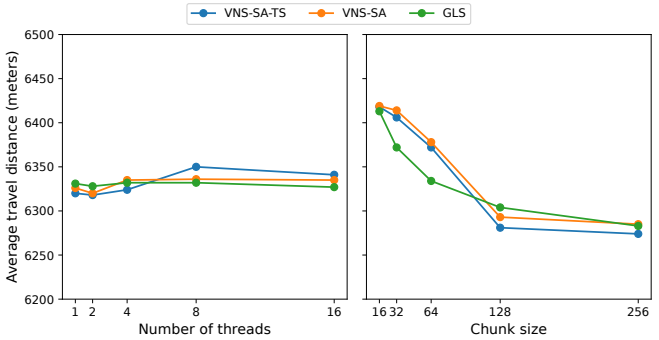

Fig. 3. Travel distance obtained vs number of threads and chunk size, for a batch size of 75 items.

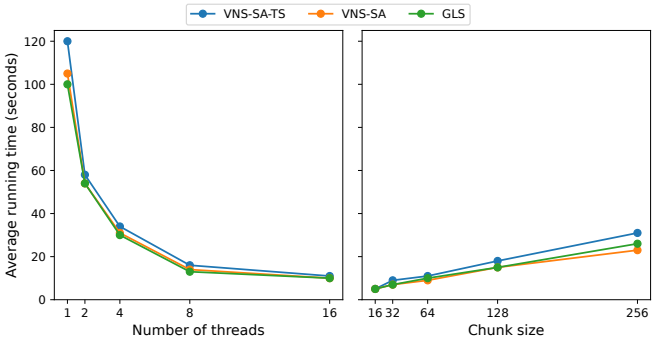

Fig. 4. Running time obtained vs number of threads and chunk size, for a batch size of 30 items.

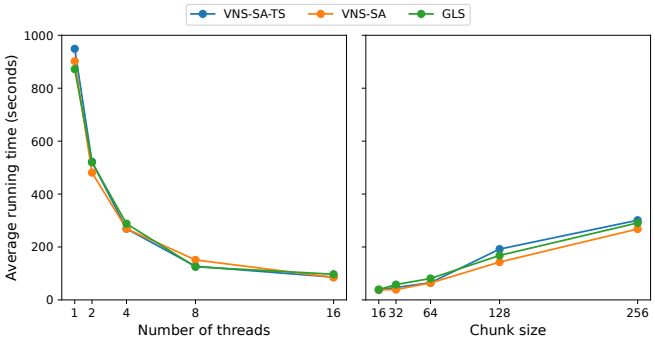

Fig. 5. Running time obtained vs number of threads and chunk size, for a batch size of 75 items.

threads varies, yet it is not statistically significant. This is as expected, since the number of threads only affects the number of delta-evaluations that we can simultaneously perform.

The running time of the algorithms, by contrast, clearly decreases as we increase the number of threads (see Figures 4 and 5). We observe, though, that the biggest improvement in running time is obtained when using up to 8 threads; a further increase to 16 threads only yields a marginal reduction. The reasons are twofold. First, the computer has only 16 logical cores, so at least 1 core is needed for administrative tasks. Allocating 16 threads to 15 cores leads to at least 1 thread waiting for its turn to be scheduled on a core. Second, we use a chunk size of 64 in these simulations. As a result, each thread on a core only has to execute 4 delta-evaluations. This under-utilizes the core capability: when the execution time is low, the algorithm will more often spend time synchronizing the results than evaluating the neighboring solutions.

D. Performance when varying the chunk size

As evident from Figure 2, there is no clear impact of the chunk size on the optimal travel distance obtained for all proposed algorithms when batch size is 30 items. However, when batch size is 75 items, the solution quality of all algorithms improve as the chunk size increases (see Figure 3). An explanation is that the instances for which the batch size is 75 items are more difficult than those for which the batch size is 30 items. Thus, by allowing a larger chunk, with the same maximum number of consecutive iterations with a non-improving solution (K), more explorations are executed compared to smaller chunks, leading to better solution quality.

When considering the running time in Figure 4 and 5, we see that for both batch sizes, an increase in the chunk size leads to a linear increase in the running time (for all algorithms).

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose three heuristics for solving the JOBPRP, using a simple search procedure for the order batching problem and a more complex heuristic for the picker rout-

ing problem. The reason is that we expect the routing solution to have a big impact on the quality of the final solution. Contrary to what is common in the literature, we *jointly* solve the OBP and the PRP, instead of using a sequential approach. Our proposed algorithms rely on common metaheuristics (VNS, SA, TS, and GLS) that allow the computations for multiple delta-evaluations in the local search phase to easily be parallelized; this speeds up the search procedure and enables the use of more complex routing heuristics to solve the PRP.

The algorithm that combines VNS+SA+TS improves the state-of-the-art JOBPRP heuristics by up to 7.63%, highlighting the potential of more complex routing algorithms when solving JOBPRP. The three proposed algorithms differ less than 1.00% regarding solution quality, yet VNS+SA has a significantly lower running time than VNS+SA+TS. As the number of threads increases, the running time of the algorithms decreases but the solution quality is unaffected. Concerning the chunk size, the solution quality of the proposed algorithms improves as the chunk size increases when the batch size is 75 items, but not when the batch size is 30 items. The running time increases with the chunk size, irrespective of the batch size.

In future work, we aim to explore the parallelization of cooperative search algorithms to further improve the exploration capabilities of our algorithms.

REFERENCES

[1] K. J. Roodbergen and R. De Koster, "Routing methods for warehouses with multiple cross aisles," *International Journal of Production Research*, vol. 39, no. 9, pp. 1865–1883, 2001.

[2] F. H. Staudt, G. Alpan, M. Di Mascolo, and C. M. T. Rodriguez, "Warehouse performance measurement: a literature review," *International Journal of Production Research*, vol. 53, no. 18, pp. 5524–5544, 2015.

[3] R. De Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European journal of operational research*, vol. 182, no. 2, pp. 481–501, 2007.

[4] B. Menéndez, E. G. Pardo, A. Alonso-Ayuso, E. Molina, and A. Duarte, "Variable neighborhood search strategies for the order batching problem," *Computers & operations research*, vol. 78, pp. 500–512, 2017.

[5] B. Aerts, T. Cornelissens, and K. Sörensen, "The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem," *Computers & Operations Research*, vol. 129, p. 105168, 2021.

[6] A. Abdelhafez, G. Luque, and E. Alba, "Parallel execution combinatorics with metaheuristics: Comparative study," *Swarm and Evolutionary Computation*, vol. 55, p. 100692, 2020.

[7] E. G. Pardo, S. Gil-Borrás, A. Alonso-Ayuso, and A. Duarte, "Order batching problems: taxonomy and literature review," *European Journal of Operational Research*, 2023.

[8] J. Wurth, H. Stegherr, M. Heider, L. Luley, and J. Hähner, "Fast, flexible, and fearless: A rust framework for the modular construction of metaheuristics," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 1900–1909.

[9] C. A. Valle, J. E. Beasley, and A. S. Da Cunha, "Optimally solving the joint order batching and picker routing problem," *European Journal of Operational Research*, vol. 262, no. 3, pp. 817–834, 2017.

[10] A. Scholz and G. Wäscher, "Order batching and picker routing in manual order picking systems: the benefits of integrated routing," *Central European journal of operations research*, vol. 25, no. 2, pp. 491–520, 2017.

[11] T. Van Gils, K. Ramaekers, K. Braekers, B. Depaire, and A. Caris, "Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions," *International Journal of Production Economics*, vol. 197, pp. 243–261, 2018.

[12] J. Won and S. Olafsson, "Joint order batching and order picking in warehouse operations," *International journal of production research*, vol. 43, no. 7, pp. 1427–1442, 2005.

[13] H. D. Ratliff and A. S. Rosenthal, "Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem," *Operations Research*, vol. 31, no. 3, pp. 507–521, 1983.

[14] H. Cambazard and N. Catusse, "Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane," *European Journal of Operational Research*, vol. 270, no. 2, pp. 419–429, 2018.

[15] C. Theys, O. Bräysy, W. Dullaert, and B. Raa, "Using a tsp heuristic for routing order pickers in warehouses," *European Journal of Operational Research*, vol. 200, no. 3, pp. 755–763, 2010.

[16] N. Gademann and S. Velde, "Order batching to minimize total travel time in a parallel-aisle warehouse," *IIE transactions*, vol. 37, no. 1, pp. 63–75, 2005.

[17] T. Öncan, "Milp formulations and an iterated local search algorithm with tabu thresholding for the order batching problem," *European journal of operational research*, vol. 243, no. 1, pp. 142–155, 2015.

[18] S. Henn and V. Schmid, "Metaheuristics for order batching and sequencing in manual order picking systems," *Computers & Industrial Engineering*, vol. 66, no. 2, pp. 338–351, 2013.

[19] G. Wäscher, *Order picking: a survey of planning problems and methods*. Springer, 2004.

[20] S. Henn and G. Wäscher, "Tabu search heuristics for the order batching problem in manual order picking systems," *European Journal of Operational Research*, vol. 222, no. 3, pp. 484–494, 2012.

[21] C.-M. Hsu, K.-Y. Chen, and M.-C. Chen, "Batching orders in warehouses by minimizing travel distance with genetic algorithms," *Computers in industry*, vol. 56, no. 2, pp. 169–178, 2005.

[22] K. Heßler and S. Irnich, "Modeling and exact solution of picker routing and order batching problems," Technical Report LM-2022-03, Chair of Logistics Management, Gutenberg School ..., Tech. Rep., 2022.

[23] J. Wahlen and T. Gschwind, "Branch-price-and-cut-based solution of order batching problems," *Transportation Science*, 2023.

[24] C.-Y. Tsai, J. J. Liou, and T.-M. Huang, "Using a multiple-ga method to solve the batch picking problem: considering travel distance and order due time," *International journal of production research*, vol. 46, no. 22, pp. 6533–6555, 2008.

[25] C.-Y. Cheng, Y.-Y. Chen, T.-L. Chen, and J. J.-W. Yoo, "Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem," *International Journal of Production Economics*, vol. 170, pp. 805–814, 2015.

[26] P. Kübler, C. H. Glock, and T. Bauernhansl, "A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses," *Computers & Industrial Engineering*, vol. 147, p. 106645, 2020.

[27] O. Briant, H. Cambazard, D. Cattaruzza, N. Catusse, A.-L. Ladier, and M. Ogier, "An efficient and general approach for the joint order batching and picker routing problem," *European Journal of Operational Research*, vol. 285, no. 2, pp. 497–512, 2020.

[28] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega, "The parallel variable neighborhood search for the p-median problem," *Journal of Heuristics*, vol. 8, pp. 375–388, 2002.

[29] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović, "Cooperative parallel variable neighborhood search for the p-median," *Journal of heuristics*, vol. 10, pp. 293–314, 2004.

[30] S. T. Tran, R. J. Almeida, and C. Defryn, "Order picking: Exploring the properties of the greedy seed-based batching algorithm," in *2023 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2023, pp. 1–8.

[31] M. Gendreau, J.-Y. Potvin *et al.*, *Handbook of metaheuristics*. Springer, 2019, vol. 3.

[32] M. Albareda-Sambola, A. Alonso-Ayuso, E. Molina, and C. S. De Blas, "Variable neighborhood search for order batching in a warehouse," *Asia-Pacific Journal of Operational Research*, vol. 26, no. 05, pp. 655–683, 2009.

[33] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[34] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European journal of operational research*, vol. 113, no. 2, pp. 469–499, 1999.