Formulating and Solving the Integrated Batching, Routing, and Picker Scheduling Problem in a Real-life Spare Parts Warehouse

Peer-reviewed author version

VAN GILS, Teun; CARIS, An; RAMAEKERS, Katrien & BRAEKERS, Kris (2019) Formulating and Solving the Integrated Batching, Routing, and Picker Scheduling Problem in a Real-life Spare Parts Warehouse. In: European journal of operational research, 277, p. 814-830..

DOI: 10.1016/j.ejor.2019.03.012
Handle: http://hdl.handle.net/1942/27968

Formulating and Solving the Integrated Batching, Routing, and Picker Scheduling Problem in a Real-life Spare Parts Warehouse

Teun van Gils, An Caris, Katrien Ramaekers, Kris Braekers

UHasselt, Research Group Logistics, Agoralaan Building D, 3590 Diepenbeek, Belgium

Abstract

New market developments increase the complexity of managing order picking operations. Integrating order picking planning problems enables warehouse managers to organize order picking operations more efficiently. This paper provides a decision support tool that integrates and solves the three main operational order picking planning problems (i.e., order batching, picker routing, and picker scheduling). Different from other studies, the objective is to increase order picking efficiency while ensuring a high customer service level. The new integrated planning problem accounts for order due times, a limited availability of order pickers, as well as a high-level storage locations, to ensure the applicability in practice. An iterated local search algorithm is introduced to solve the problem effectively and efficiently. Moreover, a real-life case shows the substantial performance benefits gained from integrating batching, routing, and picker scheduling.

Keywords: logistics, metaheuristic, order batching, order picking, routing

1. Introduction

Organizing efficient and flexible order picking operations has been identified as both an important and complex task for warehouse managers. Order picking operations account for a large part of the overall logistical costs, and they significantly impact the service level provided to customers (Marchet et al., 2015). The complexity of planning order picking operations results from the interdependencies among the wide range of planning problems (e.g., storage assignment, order batching, routing, picker scheduling). Warehouses can achieve significant efficiency benefits by considering these interdependencies (Van Gils et al., 2016). Serving e-commerce markets, globalisation and increased customer expectations further increase the complexity of managing order picking operations. Warehouses are forced to handle a larger number of small orders, while the time to pick orders has shortened (Wruck et al., 2017).

*Corresponding author
Email address: teun.vangils@uhasselt.be (Teun van Gils)

This paper provides an effective and efficient algorithm to integrate and solve the three main operational order picking planning problems (i.e., order batching, routing, and picker scheduling). As the time horizon of the resulting decisions is similar, order picking operations' efficiency can be improved by integrating these planning problems. The order batching problem is concerned with deciding on rules defining which orders to combine in a pick round. The routing decision defines the sequence of items in a pick round. The picker scheduling problem assigns batches to order pickers to ensure that all orders are picked before due time (Van Gils et al., 2018b). Traditionally, decisions are made sequentially: first orders are batched based on a distance or time related measure (De Koster et al., 1999; Henn and Wäscher, 2012; Pan et al., 2015), followed by routing each batch (Roodbergen and De Koster, 2001; Theys et al., 2010; Scholz et al., 2016) and finally assigning batches to the first available order picker (Henn, 2015). Although the efficiency of these planning problems has found to be strongly interdependent, the recent literature review of Van Gils et al. (2018b) shows that only a limited number of researchers examine multiple planning problems simultaneously.

The main contributions of this paper are as follows. First, a mathematical formulation for the new integrated batching, routing and picker scheduling problem is presented. Second, an efficient heuristic algorithm to solve the integrated problem is provided. Third, a real-life case demonstrates the benefits of optimizing the integrated batching, routing and picker scheduling problem compared to the current sequential solution of the warehouse. The case is based on an international warehouse located in Belgium that stores automotive spare parts to serve the B2B e-commerce vehicle market.

The remainder of the paper is organized as follows. Section 2 reviews publications which integrate different order picking planning problems. Section 3 introduces the mathematical programming model of the integrated problem. Next, a new, simple but effective iterated local search algorithm to solve the integrated problem is presented (Section 4) and thoroughly tested (Section 5). Section 6 provides the concluding remarks and future research directions.

2. Literature review

By simulating existing solution policies for the picker zoning, storage location assignment, order batching and picker routing, Van Gils et al. (2018a) show that decisions on which policy to apply for each planning problem are highly interdependent. In addition to the strong relation, the time horizon of the batching and routing decision is similar, making the integration of both planning problems highly relevant in terms of order picking efficiency. Instead of simulation existing solution methods, a new solution method is created in this study that integrates batching and routing decisions, as well as the picker scheduling problem. The

Table 1: Studies integrating order picking planning problems, based on Van Gils et al. (2018b).

	batching	routing	picker		scheduling
			1 picker		>1 picker
Won and Olafson (2005)	•	•			
Tsai et al. (2008)	•	•			
Ene and Öztürk (2012)	•	•			
Kulak et al. (2012)	•	•			
Henn and Schmid (2013)	•		•		
Matthews and Visagie (2013)		•			•
Matusiak et al. (2014)	•	•			
Chen et al. (2015)	•	•	•		
Cheng et al. (2015)	•	•			
Henn (2015)	•				•
Li et al. (2016)	•	•			
Lin et al. (2016)	•	•			
Matusiak et al. (2017)	•				•
Scholz et al. (2017)	•	•			•
Valle et al. (2017)	•	•			
Zhang et al. (2017)	•				•

integrated problem of batching and routing is extensively studied, as shown in Table 1. As both problems are NP-hard (Won and Olafson, 2005), metaheuristic algorithms are typically used to solve either batching, routing, or the integrated problem of batching and routing. These algorithms are able to find good solutions for the integrated batching and routing problem in small computation time, mainly for small warehouses of three to six picking aisles (Chen et al., 2015; Li et al., 2016; Ene and Öztürk, 2012), low-level storage locations (Chen et al., 2015; Scholz et al., 2017) and a single order picker (Li et al., 2016; Matusiak et al., 2014). To increase the practical relevance, solution methods that account for more real-life issues are needed (Van Gils et al., 2018b).

As accuracy in delivery times is an essential performance indicator for warehouses (Wruck et al., 2017), respecting due times is a critical issue when batching orders and routing pickers (Henn and Schmid, 2013; Chen et al., 2015). This initiates an additional planning problem: the picking sequence and completion time of all batches should be determined (Chen et al., 2015). Most studies aim at minimizing total tardiness of all customer orders (i.e., the positive difference between the order due time and the batch completion time to which the order is assigned) (Chen et al., 2015; Scholz et al., 2017). Solution algorithms often provide a solution in which one or more customer orders will be picked after the picking due time, resulting in orders that miss the shipping deadline (Henn and Schmid, 2013). In practice, such solutions may not be accepted by some warehouses, as this reduces the customer service level. Rather than accepting tardiness, the number of pickers will be increased (e.g., by shifting workers from other departments) to prevent orders being picked

after due time. For example, in the context of spare parts warehouses, service levels are considered as hard constraints (Kennedy et al., 2002): the objective is to increase order picking efficiency, while maintaining a high service level to customers. Tardiness is assumed to occur only as a result of unforeseen issues (e.g., technical defects and empty storage locations), which are not considered in this study.

Despite the importance of human resources in the labour-intensive environment of warehouses, few articles integrate workforce related planning problems in batching and routing problems. In a single order picking system, the batch sequencing decision simply determines the sequence of picking batches (Henn and Schmid, 2013; Chen et al., 2015). In case of multiple order pickers, the picker scheduling problem becomes more challenging. Batches need to be additionally assigned to order pickers prior to defining the sequence of picking batches (Henn, 2015; Scholz et al., 2017; Zhang et al., 2017).

Most studies consider travelling in two dimensions (i.e., low-level storage system), while many warehouses store products on high-level storage locations (i.e., each storage rack section consists of multiple levels, requiring the pick truck to lift to reach a location). High-level storage systems strongly increase the storage capacity for a given warehouse surface (Pan et al., 2014), and these systems are especially useful when products are large such as the vehicle spare parts of our real-life case. Solution algorithms are required that account for pick truck lifting. As lifting is typically very slow compared to travelling in horizontal direction, high-level storage locations and consequently lifting strongly influence the picking efficiency (Van Gils et al., 2018b).

This study goes beyond the current academic literature by integrating batching, routing and picker scheduling in a multiple order picker system. To the best of our knowledge, we are the first to optimize order picking efficiency by integrating order batching, routing, and picker scheduling while ensuring a high customer service level. Existing assumptions, such as a single order picker (Chen et al., 2015), low-level storage locations (Chen et al., 2015; Scholz et al., 2017), and minimizing tardiness (Chen et al., 2015; Scholz et al., 2017) are revised to increase the applicability of this study in practice. A suitable solution algorithm is provided that is able to cope with multiple pickers, high-level storage locations and avoiding tardiness. The benefits of integrating batching, routing and picker scheduling in practice are shown by a real-life case.

3. Integrated batching, routing and picker scheduling problem

The integrated problem of order batching, routing and scheduling of order pickers is introduced in this section. Section 3.1 describes the problem. The mathematical model is introduced in Section 3.2.

3.1. Problem description

The integrated batching, routing and picker scheduling problem (IBRSP) can be summarized as combining a predefined set of orders into batches (i.e., batching), for each batch defining the sequence of storage locations to visit in order to retrieve all orders assigned to the batch (i.e., routing), assigning the batches to the available order pickers and sequence the batches for each picker (i.e., picker scheduling). The aim of the integrated problem is to minimize the total order pick time. While most studies aim to minimize the total tardiness of all customer orders (Chen et al., 2015; Scholz et al., 2017), we include order due times as hard constraints in the model in order to guarantee a high service level to customers. Each order is assigned to a shipping truck. Order due times are defined by the schedule of shipping trucks. The assignment of orders to shipping trucks as well as the shipping schedule are assumed to be fixed at an operational decision level.

The objective is to increase order picking efficiency, while avoiding tardiness of orders. From a managerial point of view, the main order picking costs are defined by the number of pickers. At the decision level of IBRSP, the number of pickers is assumed to be constant. Batching, routing and picker scheduling decisions are usually made multiple times per **day** when a sufficient number of orders are available, while the number of pickers has been defined based on forecasts before a shift starts (Van Gils et al., 2017). Therefore, total order pick time is used as surrogate for order picking efficiency. **Although orders arrive continuously throughout the day, most warehouses release a large number of customer orders in single wave taking advantage of economies of scale in picking operations (Çeven and Gue, 2015). Assuming an order release mechanism,** a smaller total order pick time enables an earlier release of new orders resulting in more retrieved orders in a shift. Under the assumption of little idle capacity, the workload tends to be additionally balanced and the makespan tends to be small when minimizing total picking time and including order due times as hard constraints. As workload forecasts are used to determine the required number of pickers in practice, the alignment of number of pickers and workload (i.e., little idle capacity) is a reasonable assumption.

The total order picking time consists of following three elements: travel time, search and pick time, and batch setup time (Van Gils et al., 2018a). The travel time is assumed to be directly proportional to the travel distance, the search and pick time is assumed to be directly proportional to the number of order lines in a batch, and the setup time is the fixed amount of time consumed for administrative and setup tasks for a batch. Although travel velocity, search and pick time, and setup time may differ among order pickers, for simplicity we assume the time components to be constant in the model. However, human factors could be easily incorporated by assuming picker-dependent time components (Matusiak et al., 2017).

Batches are created by merging a particular number of orders on a pick list. Each order consists of a number of order lines representing an ordered stock keeping unit (SKU). Each SKU has a unique pick location in the warehouse. In accordance with previous research, the batch capacity is expressed in number of order lines (Valle et al., 2017), assuming that sorting activities should be performed afterwards. An order can be only assigned to a single batch (i.e., order integrity) (Van Gils et al., 2016). Each batch is assigned to a batch position of an order picker in order to define the sequence in which a picker should pick the batches assigned to him/her. Each batch can only be scheduled at one batch position and each batch position cannot consist of more than one batch of orders.

3.2. Linear mixed integer programming model

A mixed integer linear programming (MIP) model is developed to formulate the problem. The efficient formulation of Valle et al. (2017), describing the integrated batching and routing problem, is used as start point for the new integrated batching, routing and picker scheduling problem. The formulation is adapted by including the assignment of batches to order pickers, evaluating the total order pick time of each batch and including order due times as hard constraints.

Sets

$\sigma = \{1, 2, ..., Q\}$	set of order pickers with index q.
$\pi = \{1, 2, ..., P\}$	set of batch positions of a picker with index p.
$\psi = \{0, 1, 2, ..., V\}$	set of vertices with index v (depot is 0).
$\Psi = \{\psi^1, \psi^2, ..., \psi^S\}$	set with all possible subsets of vertices $\psi^s \subset \psi \backslash 0 : \lvert\psi^s\rvert > 1$.
$\alpha = \{1, 2, ..., A\}$	set of arcs with index a connecting a start and end vertex $(v'; v'') : v', v'' \in \psi$.
$\alpha_{\psi^s} \subset \alpha$	subset of arcs with $a = (v'; v'') : v', v'' \in \psi^s$.
$\alpha_v^+ \subset \alpha$	subset of arcs ending in a vertex v.
$\alpha_v^- \subset \alpha$	subset of arcs starting in a vertex v.
$\kappa = \{1, 2, ..., K\}$	set of customer orders with index k.
$\psi_k \subset \psi$	subset of vertices that should be visited in customer order k.
$\mu = \{1, 2, ..., M\}$	set of pick aisles with index m.
$\epsilon = \{1, 2, ..., E\}$	set of cross-aisles with index e.
$\iota = \{1, 2, ..., J\}$	set of storage levels with index j.

Parameters

o_k	number of order lines of order k.
c	batch capacity (in number of order lines).
t_a	travel time when travelling across arc a (in seconds).
t_{setup}	batch setup time (in seconds).
t_{search}	search and pick time for visiting a storage location (in seconds).
t_k	due time of customer order k with respect to the start of the planning horizon ($t = 0$).

Decision variables

X_{qpa}	binary decision variable which is equal to 1 if and only if arc a is visited by order picker q at batch position p.
W_{qpv}	the outdegree of vertex v (i.e., number of arcs leaving v) by order picker q at batch position p.
Z_{qpv}	binary decision variable which is equal to 1 if and only if vertex v is visited by order picker q at batch position p.
R_{qpk}	binary decision variable which is equal to 1 if and only if order k is is completed by order picker q at position p.
T_{qp}	completion time of the batch completed by order picker q at position p.

The routing problem is formulated as a Steiner Travelling Salesman Problem (TSP) (Cornuéjols et al., 1985; Theys et al., 2010), as illustrated in Figure 1. White vertices, located at each intersection of a pick aisle and cross-aisle, represent artificial vertices to model the warehouse. Black vertices represent the pick locations. White vertices may be visited in a pick round, while black vertices should be visited in at least one pick round (Valle et al., 2017). Arcs are used to connect the vertices: each black vertex is connected to

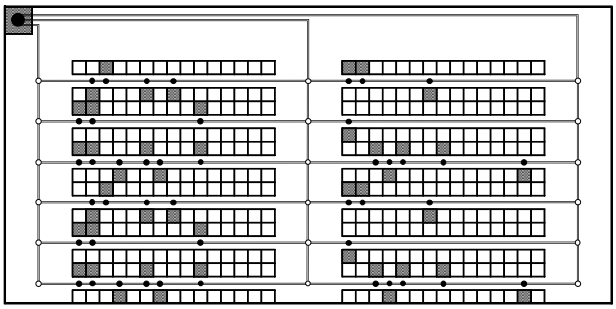

Figure 1: Directed graph of arcs and vertices representing the Steiner TSP.

the two neighbouring vertices within a pick aisle (either black or white), and arcs connect the neighbouring artificial vertices within a cross-aisle. Furthermore, for each cross-aisle, the closest artificial vertex with respect to the depot is connected to the depot. Compared to classical TSP formulations, the Steiner TSP has shown substantial computational improvements (Scholz et al., 2016).

To model the batching and picker scheduling, a set of pickers and batch positions is used. Batches are not explicitly modelled in the mathematical formulation. In this way, the number of sets is limited to four, which simplifies the notation and makes the model easier to read. As each order picker q is able to pick a single batch at each position p, each combination $(q;p)$ represents a batch in the mathematical formulation. The number of created batches (B) in the solution is equal to the number of depot visits:

$$B = \sum_{q \in \sigma} \sum_{p \in \pi} Z_{qp0} \tag{1}$$

In the discussion below, a batch refers to a combination of $(q;p)$. The linear MIP model can be stated as follows:

$$\min \sum_{q \in \sigma} T_{qP} \tag{2}$$

Subject to

$$\sum_{a \in \alpha_v^+} X_{qpa} = \sum_{a \in \alpha_v^-} X_{qpa} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall v \in \psi \tag{3}$$

$$\sum_{a \in \alpha_v^-} X_{qpa} = W_{qpv} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall v \in \psi \qquad (4)$$

$$\sum_{v' \in \psi^s} W_{qpv'} \geq Z_{qpv} + \sum_{a \in \alpha_{\psi^s}} X_{qpa} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall v \in \psi^s$$

$$\forall \psi^s \in \Psi \qquad (5)$$

$$X_{qpa} \leq Z_{qpv} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall v \in \psi$$

$$\forall a \in \alpha_v^- \qquad (6)$$

$$\sum_{a \in \alpha_0^+} X_{qpa} = \sum_{a \in \alpha_0^-} X_{qpa} = Z_{qp0} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall a \in \alpha \qquad (7)$$

$$X_{qpa} \leq Z_{qp0} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall a \in \alpha \qquad (8)$$

$$R_{qpk} \leq Z_{qp0} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall k \in \kappa \qquad (9)$$

$$\sum_{a \in \alpha_v^-} X_{qpa} \geq R_{qpk} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall k \in \kappa$$

$$\forall v \in \psi_k \qquad (10)$$

$$\sum_{k \in \kappa} R_{qpk} \geq Z_{qp0} \qquad \forall q \in \sigma$$

$$\forall p \in \pi \tag{11}$$

$$\sum_{k \in K} o_k R_{qpk} \leq c \qquad \forall q \in \sigma$$

$$\forall p \in \pi \tag{12}$$

$$\sum_{q \in \sigma} \sum_{p \in \pi} R_{qpk} = 1 \qquad \forall k \in \kappa \tag{13}$$

$$t_{setup} Z_{qp0} + t_{search} \sum_{k \in \kappa} o_k R_{qpk} + \sum_{a \in \alpha} t_a X_{qpa} = T_{qp} \qquad \forall q \in \sigma$$

$$p = 1 \tag{14}$$

$$T_{q(p-1)} + t_{setup} Z_{qp0} + t_{search} \sum_{k \in \kappa} o_k R_{qpk} + \sum_{a \in \alpha} t_a X_{qpa} = T_{qp} \qquad \forall q \in \sigma$$

$$\forall p \in \pi \backslash \{1\} \tag{15}$$

$$T_{qp} \leq t_k + \mathcal{M}(1 - R_{qpk}) \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall k \in \kappa \tag{16}$$

$$T_{qp} \leq \mathcal{M} \sum_{k \in \kappa} R_{qpk} \qquad \forall q \in \sigma$$

$$\forall p \in \pi \tag{17}$$

$$X_{qpa}, R_{qpk} \in \{0, 1\} \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall k \in \kappa$$

$$\forall a \in \alpha \tag{18}$$

$$Z_{qpv} \in [0; 1] \qquad \forall q \in \sigma$$

$$\forall p \in \pi$$

$$\forall v \in \psi \tag{19}$$

$$T_{qp} \geq 0 \qquad \forall q \in \sigma$$

$$\forall p \in \pi \tag{20}$$

The objective function (2) minimizes the total order pick time to retrieve all customer orders. Constraints (3) ensure that the number of arcs visiting a vertex v is equal to the number of arcs leaving the vertex in each batch. Constraints (4) define the outdegree of each vertex (i.e., the number of arcs leaving vertex v)

in each batch. Constraints (5) avoid the creation of sub–tours in a batch: the total outdegree of a subset of vertices should be greater than the number of vertices and arcs visited in the subset. These sub–tour elimination constraints are derived from the Vehicle Routing Problem (Laporte, 1992) and provide good results in an order picking context (Valle et al., 2017). Constraints (6) allow vertices to be visited in a batch only when an arc starts in the vertex. The number of depot visits (i.e., vertex 0) is defined by constraints (7): if the depot is visited, a batch should contain an incoming and outgoing arc from the depot. Furthermore, if the depot is included in a batch, at least one arc is used in a batch or at least one order is picked in the batch, as stated by constraints (8) and (9), respectively. Constraints (10) ensure that vertices of orders assigned to a batch are visited by enforcing at least one outgoing arc to be used in the batch. Constraints (11) make sure that at least one order is assigned to the batch if the depot is visited. Constraints (13) ensure that the number of order lines in each batch does not exceed the batch capacity and constraints (12) ensure order integrity (i.e., each order is assigned to a single batch). Constraints (14) and (15) incorporate the processing time of picking a batch for the first batch position and other batch positions, respectively. Additionally, constraints (15) prevent overlapping batches that are assigned to the same order picker. Constraints (16) guarantee that all orders are picked before due time, with $\mathcal{M}$ a sufficiently large positive number ($\mathcal{M} = \max\{t_k, \forall k \in K\}$). Constraints (17) ensure the calculation of a completion time for all scheduled batches. The domain constraints are provided by constraints (18)-(20). Note that the formulation forces the Z_{qpv} to be binary as well.

The number of sub–tour elimination constraints (i.e., Constraints 5) grows exponentially with the number of vertices in the problem. Therefore, initially these constraints are removed from the formulation and a branch-and-cut procedure is employed to check each integral candidate solution on sub–tours. For each sub–tour in the candidate solution, Constraints (5) are included with ψ' containing only vertices of the created sub–tour. To reduce the number of created subtours, and consequent number of cuts, Valle et al. (2017) introduce a series of optimality cuts and symmetry breaking constraints which are shown to substantially reduce the computation time to find the optimal total order pick time. Hence, we adapted these inequalities to our problem setting and include these as well. The applied optimality cuts and symmetry breaking constraints are provided in Appendix A. The reader is referred to Valle et al. (2017) for an comprehensive discussion on the optimality cuts.

4. Iterated local search algorithm for IBRSP

Due to the complex nature of IBRSP, solving instances of realistic size to optimality in a reasonable amount of computation time does not seem feasible. A metaheuristic algorithm, based on iterated local search, is proposed to approximate the global optimal solution. Iterated local search algorithms have proven to be efficient in optimizing order picking planning problems (Öncan, 2015; Scholz and Wäscher, 2017). The aim is to provide a simple but effective ILS algorithm to solve IBRSP.

The general principle of ILS is introduced by Lourenço et al. (2003). The main components of ILS include a procedure to generate an initial solution, a local search procedure, and a perturbation procedure. In addition to the general ILS principles, the diversification is increased by maintaining a set of six solutions S (instead of a single solution), as well as considering multiple operators during the local search procedure which is commonly applied metaheuristic algorithms. While multi-start ILS algorithms start from a randomly constructed new solution each iteration, the solution set allows starting from varying solutions in each iteration and each starting solution is a good solution (i.e., a local optimum). Moreover, multiple local search operators increase the quality of the local search, thereby improving the local optimum, compared to a single local search operator (Sörensen and Glover, 2013).

The ILS algorithm is described in Algorithm 1. First, an initial solution s^0 is created, followed by a local search on s^0 that results in a local optimum. All solutions in S are initialized by this local optimum. Next, four steps are performed iteratively: (1) selecting a solution s^* from $S(s^1; s^2; s^{r1}; s^{r2}; s^{r3}; s^{r4})$ with probability $\Phi(\phi_1; \phi_2; \frac{\phi_3}{4}; \frac{\phi_3}{4}; \frac{\phi_3}{4}; \frac{\phi_3}{4})$, respectively, and Φ a set of algorithm parameters; (2) perturbing s^*; (3) applying local search to reach a new local optimum; (4) updating S. If this procedure results in a solution with either a reduced tardiness or a reduced total order pick time without increasing tardiness, compared to the best (s^1) or second best (s^2) solution, the solution is accepted as new best or second best solution, respectively. Otherwise, the solution is saved as one of the four random solutions (i.e., s^{r1}, s^{r2}, s^{r3}, and s^{r4}). These steps are repeated until there are γ consecutive iterations with an improvement in total order pick time of the best solution $s^1_{picktime}$ of $\leq 0.005\%$ and a tardiness of zero in the best solution (with a maximum of $5,000$ iterations). The number of consecutive iterations without improvement also determines the intensity of the perturbation (see Algorithm 5).

The generation of an initial solution is described in Algorithm 2. Initially, each order is assigned to a separate batch. Orders are sorted with respect to the due time: the customer order that should be shipped most early (co_1) is assigned to the first batch position ($p = 1$) of the first order picker ($q = 1$). The next order on the sorted list of customer orders is assigned to the first batch position of the second order picker.

Algorithm 1 Iterated local search algorithm for IBRSP

create initial solution s^0 (Algorithm 2)
local search batching and picker scheduling on s^0 (Algorithm 3);
local search routing on s^0 (Algorithm 4);
initialize solution set $S(s^1; s^2; s^{r1}; s^{r2}; s^{r3}; s^{r4}) = (s^0; s^0; s^0; s^0; s^0; s^0)$;
repeat
  select solution s^* from S with probability $\Phi\big(\phi_1; \phi_2; \frac{\phi_3}{4}; \frac{\phi_3}{4}; \frac{\phi_3}{4}; \frac{\phi_3}{4}\big)$;
  perturbation on s^* (Algorithm 5);
  local search batching and picker scheduling on s^* (Algorithm 3);
  local search routing on s^* (Algorithm 4);
  if $\Big(s^*_{picktime} \leq s^1_{picktime}$ **and** $s^*_{tardiness} \leq s^1_{tardiness}\Big)$ **or** $s^*_{tardiness} < s^1_{tardiness}$ **then**
    new best solution: $s^1 = s^*$;
    count the number of non-improving iterations: $I^* = 0$;
  else if $\Big(s^*_{picktime} \leq s^2_{picktime}$ **and** $s^*_{tardiness} \leq s^2_{tardiness}\Big)$ **or** $s^*_{tardiness} < s^2_{tardiness}$ **then**
    new second best solution: $s^2 = s^*$;
    count the number of non-improving iterations: $I^* = \min\{10; I^* + 1\}$;
  else
    new random solution: $s^{r4} = s^{r3}$;
    new random solution: $s^{r3} = s^{r2}$;
    new random solution: $s^{r2} = s^{r1}$;
    new random solution: $s^{r1} = s^*$;
    count the number of non-improving iterations: $I^* = \min\{10; I^* + 1\}$;
  end if
until γ iterations with improvement $\leq 0.005\%$ **and** $s^1_{tardiness} = 0$;

Algorithm 2 Create initial solution

sort all customer orders with respect to due time;
initialize customer order ($k = 1$), position ($p = 1$) and picker ($q = 1$);
while $k \leq K$ **do**
  assign customer order co_k to position p of picker q;
  increase customer order: $k = k + 1$;
  increase picker: $q = q + 1$;
  if $q > Q$ **then**
    return to the first picker: $q = 1$;
    increase position: $p = p + 1$;
  end if
end while
local search routing on s^0 (Algorithm 4);

Once all picker's first positions are occupied, orders are assigned to the second batch positions ($p = 2$). These steps are repeated until all orders are assigned to a batch. Next, locations that should be visited to retrieve all items of a batch are sequenced by the routing algorithm, explained in Algorithm 4, to create initial routes.

The local search phase of the heuristic consists of a batching and order picker scheduling algorithm (Algorithm 3), and a routing algorithm (Algorithm 4). The batching and picker scheduling local search phase consists of four move types, applied in a fixed sequence: relocating a single customer order to another batch position and/or picker (i.e., *order shift*), relocating a batch to the same batch position of another picker (i.e., *batch shift*), exchanging two customer orders from different batches (i.e., *order swap*), and exchanging all customer orders from two different batches (i.e., *batch swap*).

Batch swaps and batch shifts are performed for each batch position of each picker. The neighbourhood of the batch moves consists of all positions and all pickers to which a move results in a new solution with reduced or equal total tardiness compared to the current solution. The total order picking time remains equal by shifting and swapping entire batches. In case of tardiness in the current solution, these move types are able to move quickly to a feasible solution (i.e., $s_{tardiness} = 0$): **efficient batches with urgent orders can be scheduled earlier to pick the urgent customer orders timely.** Therefore, a batch swap and batch shift are only performed when the solution is still infeasible with respect to tardiness.

Order shift and order swap moves that result in either a reduced tardiness or a reduced total order pick time (without increasing tardiness) are accepted as new solutions. Once a solution is feasible, a reduced total order pick time is the only binding constraint for accepting new solutions. A first improving move strategy is used to select a new solution. The order shift operator is efficient with respect to computational complexity ($\mathcal{O}(BK)$) and particularly effective to reduce the order pick time by reducing the number of batches very fast. The order shift aims to shift all orders (one-by-one) of a single batch before orders of another batch are considered. The order shift operator is the most efficient and effective operator. Therefore, this operator is positioned first in the local search algorithm. Whereas the effectiveness of the shift operator strongly decreases in case of fully loaded batches, the order swap operator can further decrease order pick times by switching two orders of different batches, at the cost of additional computational complexity ($\mathcal{O}(K^2)$). To prevent order shifts or order swaps that will probably be rejected because of tardiness, the completion time of a batch is compared to the order due times of the order(s) considered in the move before the move is performed. Parameter χ is defined as the maximum difference between the current batch completion time and the order due time for a move to be considered (i.e., $T_{qp^*} \leq t_k + \chi$). The order shift and order swap

Algorithm 3 Batching and picker scheduling

repeat
  repeat
    for all batches $(q;p)$ **do**
      for all customer orders $co_k \in (q;p)$ **do**
        for all batches $(q;p)^*$ **do**
          if $T_{qp^*} \leq t_k + \chi$ **or** $s^*_{tardiness} > 0$ **then**
            create temporary solution: $s^t = s^*$;
            shift customer order $co_k \in (q;p)$ to batch $(q;p)^*$ in s^t;
            insert each order line of co_k on the cheapest position of the route in s^t;
            if $(s^t_{picktime} \leq s^*_{picktime}$ **and** $s^t_{tardiness} \leq s^*_{tardiness})$ **or** $s^t_{tardiness} < s^*_{tardiness}$ **then**
              accept solution: $s^* = s^t$;
              break
            end if
          end if
        end for
      end for
    end for
  until no further improvement is possible;
  if $s^*_{tardiness} > 0$ **then**
    for all batches $(q;p)$ **do**
      for all batches $(q;p)^*$ **do**
        create temporary solution: $s^t = s^*$;
        shift batch $(q;p)$ to another picker q^* and/or another position p^* in s^t;
        if $s^t_{tardiness} \leq s^*_{tardiness}$ **then**
          accept solution: $s^* = s^t$;
          break
        end if
      end for
    end for
  end if
  repeat
    for all customer orders $co_k \in \kappa$ **do**
      for all customer orders $co_{k^*} \in \kappa$ **do**
        if $(T_{qp^*} \leq t_k + \chi$ **and** $T_{qp} \leq t_{k^*} + \chi)$ **or** $s^*_{tardiness} > 0$ **then**
          create temporary solution: $s^t = s^*$;
          swap customer order $co_k \in (q;p)$ and an order $co_{k^*} \in (q;p)^*$ in s^t;
          insert each order line of co_k on the cheapest position of the route of $(q;p)^*$;
          insert each order line of co_{k^*} on the cheapest position of the route of $(q;p)$;
          if $(s^t_{picktime} \leq s^*_{picktime}$ **and** $s^t_{tardiness} \leq s^*_{tardiness})$ **or** $s^t_{tardiness} < s^*_{tardiness}$ **then**
            accept solution: $s^* = s^t$;
            break
          end if
        end if
      end for
    end for
  until no further improvement is possible;
  if $s^*_{tardiness} > 0$ **then**
    for all batches $(q;p)$ **do**
      for all batches $(q;p)^*$ **do**
        create temporary solution: $s^t = s^*$;
        swap batch $(q;p)$ and batch $(q;p)^*$ in temporary solution s^t;
        if $s^t_{tardiness} \leq s^*_{tardiness}$ **then**
          accept solution: $s^* = s^t$;
          break
        end if
      end for
    end for
  end if
until no further improvement is possible;

moves are repeated until no further improvement is possible. Note that there is no explicit repair method in the move operators for solutions with tardiness: the move operators create highly efficient batches with respect to travelling and batches are filled to capacity. In this way travel time and setup time are small, reducing the probability of tardiness.

Algorithm 4 Routing

```
for all pickers q ∈ σ do
  for all positions p ∈ π do
    if number of locations to visit in batch (q; p) ≤ 8 then
      calculate exact route length of batch (q; p);
    else
      LKH-routing of batch (q; p);
    end if
  end for
end for
```

The routing algorithm minimizes the order picker travel distance by sequencing items in a batch. Only for a small number of locations to be visited, an optimal route can be calculated in reasonable computing times. The Lin–Kernighan–Helsgaun (LKH) heuristic (Helsgaun, 2000) for the TSP is used as alternative to approximate the optimal route length. The LKH heuristic has shown to provide excellent results, both in a general TSP context, and in the context of routing order pickers in a warehouse (Theys et al., 2010). Pretests of our algorithm showed that calculating the optimal route length by enumerating all feasible solutions is faster compared to executing the LKH-routing heuristic if the number of storage locations to visit in a batch is smaller than or equal to eight locations, including the depot. For all other batches, the routing problem is solved by the LKH heuristic. The same settings for the LKH heuristic as in Theys et al. (2010) are used. **Despite switching off n-opt moves for $n > 3$ and the multi-start procedure, as well as limiting route calculations to batches that have been changed during the local search batching and picker scheduling (Algorithm 3), the large majority of computation effort is dedicated to the local search routing (Algorithm 4)**.

Applying Algorithms 3 and 4 results in a local optimum. To escape from this local optimum, a large change (i.e., perturbation) is performed to a solution included in solution set Φ. The perturbation of the ILS algorithm consists of splitting I batches: in each of the I perturbation iterations a random number of orders from an existing batch are assigned to a new batch, created at a random position of a random picker. After the creation of a new batch, the local search routing algorithm is performed to sequence the locations in the initial batch as well as the new batch. A perturbation iteration is repeated (for at most 50 times), starting from the current solution, if the tardiness of the perturbed solution is larger than the tardiness of the current solution. The perturbation intensity (i.e., the number of split batches) depends on the last

Algorithm 5 Perturbation

for $it = 1$ **to** I **do**
  Initialize count variable: $a = 0$;
  repeat
    $s^t = s^*$;
    choose a random batch $(q; p)$ in s^t;
    choose a random number of orders $k^* \in \left[1; \sum_{k \in \kappa} R_{qpk}\right]$ to shift;
    shift k^* orders from $(q; p)$ to a new batch $(q; p)^*$ in s^t;
    local search routing of $(q; p)$ on s^t (Algorithm 4);
    local search routing of $(q; p)^*$ on s^t (Algorithm 4);
    count perturbation attempts: $a = a + 1$
  until $s^t_{tardiness} \leq s^*_{tardiness}$ **or** $a > 50$
  if $a \leq 50$ **then**
    accept solution: $s^* = s^t$;
  end if
  increase iterator: $it = it + 1$;
end for

found best solution and is defined as $I = \lceil \theta \times B \times I^* \rceil$, with θ a parameter and I^* calculated in Algorithm 1. **Note that there is a risk of continuing to the local search without perturbing the solution if all perturbation attempts a in all I perturbation iterations fail because of tardiness. Although this risk cannot be eliminated, this scenario never happens when testing the algorithm.**

5. Computational results

To assess the performance of the proposed ILS algorithm, a series of numerical experiments is performed. All algorithms are implemented in C++. To solve the MIP formulation, ILOG Cplex 12.7 is used with a runtime limit of 4 h. In accordance with Valle et al. (2017), branching priority is given to R_{qpk}. Other parameter settings are left as default as these parameters have minor impact. Cplex and ILS are run on an Intel Xeon Processor E5-2680 at 2.8 gigahertz, using a single thread, provided by the Flemish Supercomputer Center.

The properties of the problem instances are introduced in Section 5.1 and algorithm parameters are tuned in Section 5.2. First, the ILS algorithm is tested on small problem instances. Results are compared with the optimal solutions of the MIP formulation (Section 5.3). In a second experimental design (Section 5.4), the ILS algorithm is performed on a set of large problem instances to demonstrate its applicability in practice and analyze the effects of different warehouse parameters. Finally, a real-life case is used in Section 5.5 to show the real-life benefits of optimizing IBRSP.

5.1. Problem instances

The problem parameters from Van Gils et al. (2016) are adopted in this paper. Table 2 summarizes the warehouse layout parameters and the time components of the picking operation. Picking aisles are two-sided

and wide enough for two-way travel: the effect of picker blocking is assumed to be negligibly small.

Table 2: Warehouse parameter values

Warehouse parameter		**Parameter value**	
		Small instances	Large instances
B	number of warehouse blocks	2 blocks	2 blocks
J	number of levels	1 level	1 level
l_{length}	storage location length	1.3 m	1.3 m
l_{width}	storage location width	0.9 m	0.9 m
m_{width}	pick aisle width	3.0 m	3.0 m
e_{width}	cross-aisle width	6.0 m	6.0 m
v	picker travel velocity	$\frac{1}{3}$ m/s	1 m/s
t_a	travel time for arc a	$\frac{d_a}{v}$ s	$\frac{d_a}{v}$ s
t_{setup}	setup time	540 s	180 s
t_{search}	search and pick time	30 s	10 s
$t_{picking}$	planning period	4 h	4 h

The heuristic algorithm is tested for a wide range of warehouse parameters. Three layouts, three storage location assignment policies, three batch capacity levels, three different order structures, as well as a varying distribution of due times among orders are included in the experimental design. The five factors and their associated factor levels are summarized in Table 3. All problem instances are available from `XXX` *(instances will be made available after paper acceptance).*

Table 3: Experimental factor setting

Factor	**Factor levels**	
	Small instances	Large instances
Layout	(1) 6×60 locations	6×60 locations
	(2) 12×120 locations	12×120 locations
	(3) 18×180 locations	18×180 locations
Storage policy	(1) random (Ran)	random (Ran)
	(2) within-aisle (WA)	within-aisle (WA)
	(3) across-aisle (AA)	across-aisle (AA)
Batch capacity	(1) 4 order lines	15 order lines
	(2) 8 order lines	30 order lines
	(3) 12 order lines	45 order lines
Order struct.[a]	(1) 18 orders ($\beta = \frac{4}{3}$)	300 orders ($\beta = \frac{8}{3}$)
	(2) 12 orders ($\beta = 2$)	200 orders ($\beta = 4$)
	(3) 6 orders ($\beta = 4$)	100 orders ($\beta = 8$)
Due time distr.[b]	(1) uniform (Uni)	uniform (Uni)
	(2) progressive ($Prog$)	progressive ($Prog$)
	(3) degressive (Deg)	degressive (Deg)

[a] the number of order lines for each order is generated using following formula: $\min(c; \lfloor Exp(\beta) + 0.5 \rfloor)$, with $Exp(\beta)$ an exponential distribution with mean β.
[b] the uniform due time distribution corresponds to U(1.0; $t_{picking}$), progressive and degressive due time distributions are approximated by triangular distributions as follows: TRIA(1.0; 3.0; $t_{picking}$) and TRIA(1.0; 1.5; $t_{picking}$), respectively.

The two-block warehouse layout differs in number of aisles, as well as number of storage location per aisle. The layout varies between 360 (6 aisles $\times$ 60 locations per aisle $\times$ 1 level) and 3,240 (18 aisles $\times$ 180 locations per aisle $\times$ 1 level) storage locations. An example of the smallest order picking layout is illustrated in Figure

1. Other layouts are equivalent. Note that the MIP model and optimality cuts provided in Appendix A are only valid if following assumption is fulfilled: $t_{a_1} = t_{a_2} + t_{a_3}$ with $a_1 = (v_1; v_3), a_2 = (v_1; v_2), a_3 = (v_2; v_3)$. This is only true in case of a linear distance approximation function (e.g., rectilinear distance metric), which is the case for low-level storage systems. In case of high-level storage systems, the Chebychev distance metric includes vertical travel as follows: the travel time between two vertices equals the maximum of the horizontal travel time and lifting time (Clark and Meller, 2013). Consequently, the number of arcs increases tremendously compared to the general Steiner TSP formulation as all vertices within a pick aisle need to be connected by arcs, making the MIP model too hard to solve even for very small instances. Therefore, the performance of the ILS algorithm is compared with the MIP model for a low-level storage system ($J = 1$) in the experimental design. In the real-life case, high-level storage locations are taken into account.

Besides randomly assigning SKUs to storage locations, a within-aisle as well as an across-aisle storage location assignment policy are tested. SKUs are grouped into classes in such a way that class A contains $\frac{1}{6}$ of the SKUs stored in the warehouse. These SKUs account for 60% of the picking activity. Class B and class C contain $\frac{1}{3}$ and $\frac{1}{2}$ of the storage locations and account for 30% and 10% of the order frequency, respectively. From the problem formulation, the complexity of the integrated batching, routing and picker scheduling problem seems to be independent of the layout and storage policy. Therefore, small and larger instances are tested on the same factor levels with regard to layout and storage policy.

Batch capacity and order structure impact the number of created batches and consequently the complexity of the planning problem, as shown in the formulation. Different factor levels for small and large instances are considered during the analysis, as shown in Table 3. Finally, the due time distribution factor describes the distribution of due times of customer orders. The complexity of the planning problem seems to be independent from this factor. Besides a uniform distribution over the planning period $t_{picking}$, a progressive and a degressive due time distribution are considered. For the progressive distribution most orders are picked at the end of the planning period. In a degressive situation, most orders have a due time in the first time intervals. **Note that a planning horizon of 4 h is assumed consisting of 100-300 customer orders, at least for the large instances. Without loss of generality, this planning horizon can be easily reduced or enlarged. In practice, the length of the planning horizon is a trade-off: a large number of orders (large $t_{picking}$) typically result in more efficient batches, while a short time between order entry and deadline forces the use of a small $t_{picking}$ in practice as all orders in the algorithm are assumed to be known at the beginning of the planning period.**

This factorial setting results in a $3 \times 3 \times 3 \times 3 \times 3$ full factorial design. Among the 243 possible factor com-

binations, thirty large instances (i.e., test instances) are randomly selected to derive the relation between the required number of order pickers and the properties of the warehouse. **In practice,** the required number of order pickers may be predicted based on demand forecasts (Van Gils et al., 2017). Consequently, the number of order pickers is included as warehouse parameter and assumed to be fixed and known when orders are being batched. **To experiment in this paper with a reasonable number of pickers as input,** the input parameter is derived from following regression equation: $Q^* = \lceil 1.20\,(0.254M + 0.006O - 0.072c + 1.383Deg) \rceil$ with O the number of order lines. The regression analysis is performed based on results of our algorithm with the test instances. The 30 test instances are used to derive the regression coefficients as follows: each test instance is solved using the heuristic with $Q = 10$ order pickers, next the instance is resolved with $Q = 9$ pickers, and so on. The procedure stops when the heuristic provides a solution with tardiness and the required number of pickers for a test instance is defined as $Q' + 1$, with Q' the last value of Q. Using a regression analysis on these results, layout, batch capacity, order structure and due time distribution are proven to be statistically significantly related with the required number of order pickers ($R^2_{adjusted} = 0.987$). **In this artificial context, productivity of pickers can be nearly perfectly predicted as randomness is limited to the random component of order due times, order size and assignment of SKUs to storage locations. The regression equation is only used to experiment with a reasonable number of pickers as input given the warehouse characteristics of the instance (without intending to support decisions in practice).** Note that the number of pickers for each instance in the experiments is increased with 20 % to ensure that the number of pickers is large enough to prevent tardiness in all large instances.

Without loss of generality, the number of pickers is fixed at 2 for the small benchmark instances. Moreover, for running the MIP model, the parameter P, describing the number of batch positions, should be defined. For simplicity, P is set large enough by fixing it at $\left\lceil \frac{K}{Q} \right\rceil$. A more complex upper bound for parameter P could slightly improve the computational efficiency. However, as the MIP model is only used as benchmark, this upper bound provides acceptable solutions to evaluate the solution quality of the ILS heuristic. Note that with respect to the ILS algorithm, only parameter Q is relevant as batch positions could easily be created and removed during computation.

5.2. Parameter tuning

Tuning algorithm parameters may result in significant performance benefits of the tested algorithm (Pellegrini and Birattari, 2011). With respect to the ILS algorithm, parameter tuning is performed on the set of thirty randomly selected test instances. Table 4 introduces the experimental design that is used to

tune the three algorithm parameters: γ (i.e., parameter defining the algorithm stop criterion), $(\phi_1; \phi_2; \phi_3)$ (i.e., parameters defining which solution is selected in each iteration), and θ (i.e., parameter defining the intensity of the perturbation). Pretests of the ILS algorithm were performed to select these factors and fix the factor levels. Note that χ (i.e., parameter limiting the moves in the local search) is not included in the experiments as χ is not related to other algorithm parameters. Based on pretests, the parameter value is fixed at $1\,h$. This value is large enough in order not to exclude promising moves and small enough to prevent a large number of non-promising moves, probably resulting in tardiness.

Table 4: Experimental factor setting to tune the ILS algorithm

Factor	**Factor levels**
γ (Algorithm 1)	(1) 100
	(2) 200
	(3) 300
	(4) 400
$(\phi_1; \phi_2; \phi_3)$ (Algorithm 1)	(1) $\left(1; 0; 0\right)$
	(2) $\left(\frac{1}{2}; \frac{1}{6}; \frac{1}{3}\right)$
	(3) $\left(\frac{1}{3}; \frac{1}{3}; \frac{1}{3}\right)$
	(4) $\left(\frac{1}{3}; \frac{1}{6}; \frac{1}{2}\right)$
θ (Algorithm 5)	(1) 0.000
	(2) 0.005
	(3) 0.010
	(4) 0.015
	(5) 0.020
	(6) 0.025
	(7) 0.030

Each factor level combination is tested on all thirty test instances. Five replications per factor level combination are performed. Consequently, the $4 \times 4 \times 7$ factorial design results in $16,800$ observations. Figure 2 shows the results of the parameter tuning procedure. Both the average total order picking time and the average CPU time for each factor level combination are illustrated on the graph. Due to the bad performance of $\theta = 0$ (i.e., no perturbation), this factor level is removed from the graph for visibility reasons.

Computation time increases about linearly with increasing values of γ. Total order pick time is strongly reduced as γ is increased from 100 to 200. Further increasing γ has a much weaker effect on pick time. Therefore, 200 non-improving iterations as stop criterion seems a good compromize between computation time and solution quality. CPU time increases when intensifying the perturbation, while the total order pick time turns out to be minimal with medium values of θ. Therefore, θ is set at 0.015. Finally, $(\phi_1; \phi_2; \phi_3)$ seem to have little effect on both solution quality and CPU time, except for the first factor level that shows an increased order pick time, demonstrating the positive effect of maintaining a solution pool. As the number of iterations in the algorithm is large, the impact of the probability values for choosing a solution from the

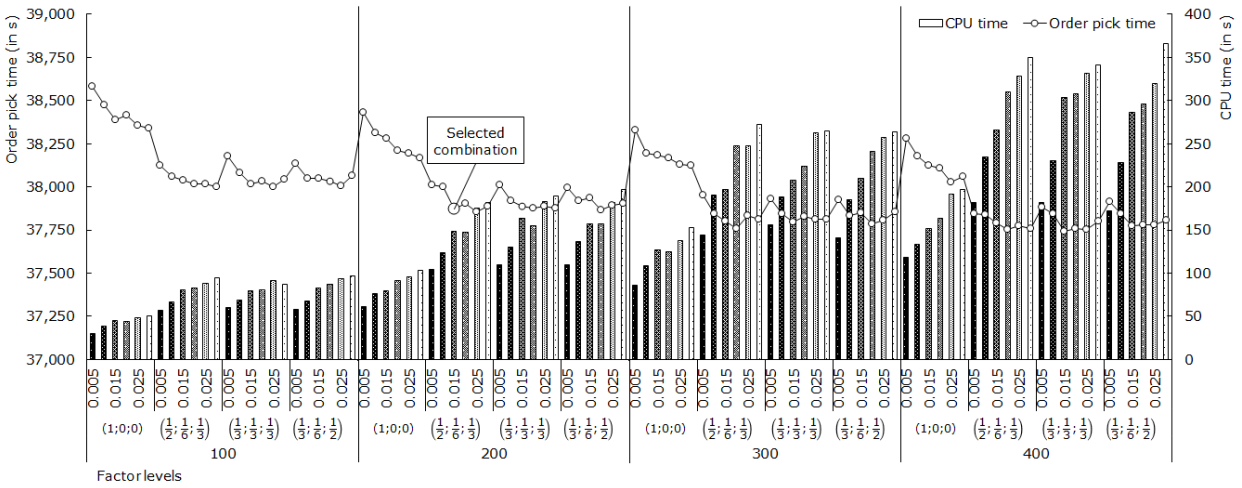

Figure 2: Comparison of average total order pick time and CPU time per factor level combination.

solution pool is negligible. The values of $(\phi_1; \phi_2; \phi_3)$ are fixed at the second factor level: $\left(\frac{1}{2}; \frac{1}{6}; \frac{1}{3}\right)$.

5.3. Comparison between exact algorithm and ILS algorithm

To assess the performance of the proposed algorithm, its results are compared with the optimal solutions obtained by solving the MIP model with Cplex. Due to the complex nature of the integrated problem, Cplex is only able to solve small instances, i.e., a small batch capacity and a limited number of customer orders, in reasonable computing times. Ten order lists are generated for each factor level combination of the factorial design (see Table 3) in order to reduce the stochastic effect of order generation. This setting results in $2,430$ small instances.

Table 5 shows the results of the MIP model. For each factor level, the number of observations that have not been solved to optimality by Cplex within the run time limit of 4 h is given. In total, 40.6 % instances (987 out of $2,430$) have not been solved to optimality. Among these, for 84 instances no feasible integer solution has been formed. The right-hand side of the table presents the minimum, mean and maximum optimality gap of the non-optimal instances for which a feasible integer solution was found (i.e., 903 instances). Layout, storage policy, and due time distribution have a limited effect on the number of non-optimal solutions. Non-optimal solutions are strongly concentrated in the two smallest batch capacity levels and the largest order structure level. These levels result in a large number of batches and increase the number of feasible solutions. Overall, the mean optimality gap of the instances (i.e., 16.2 %) is rather high, even for these small problem sizes. This demonstrates the complexity of the problem.

To assess the ILS performance, the total order pick time of ILS is compared to the optimal solution. The

Table 5: Optimality gap after solving the MIP model

	Instances		**Optimality gap (in %)**		
	#	%	Min.	Mean	Max.
Layout					
6×60	289	35.7	0.2	13.7	38.9
12×120	325	40.1	0.9	17.5	50.7
18×180	373	46.0	0.6	17.2	48.7
Storage policy					
Ran	352	43.5	0.2	16.4	45.2
WA	335	41.4	0.6	16.8	50.7
AA	300	37.0	0.9	15.4	48.7
Batch capacity					
4	487	60.1	0.2	19.8	50.7
8	328	40.5	0.6	14.9	40.2
12	172	21.2	0.8	9.5	25.8
Order struct.					
18	685	84.6	0.9	18.6	50.7
12	302	37.3	0.2	11.5	38.4
6	0	0.0	-	-	-
Due time distr.					
Uni	328	40.5	0.6	16.5	50.7
Prog	337	41.6	0.9	16.5	48.7
Deg	322	39.8	0.2	15.4	43.0
Total	**987**	**40.6**	**0.2**	**16.2**	**50.7**

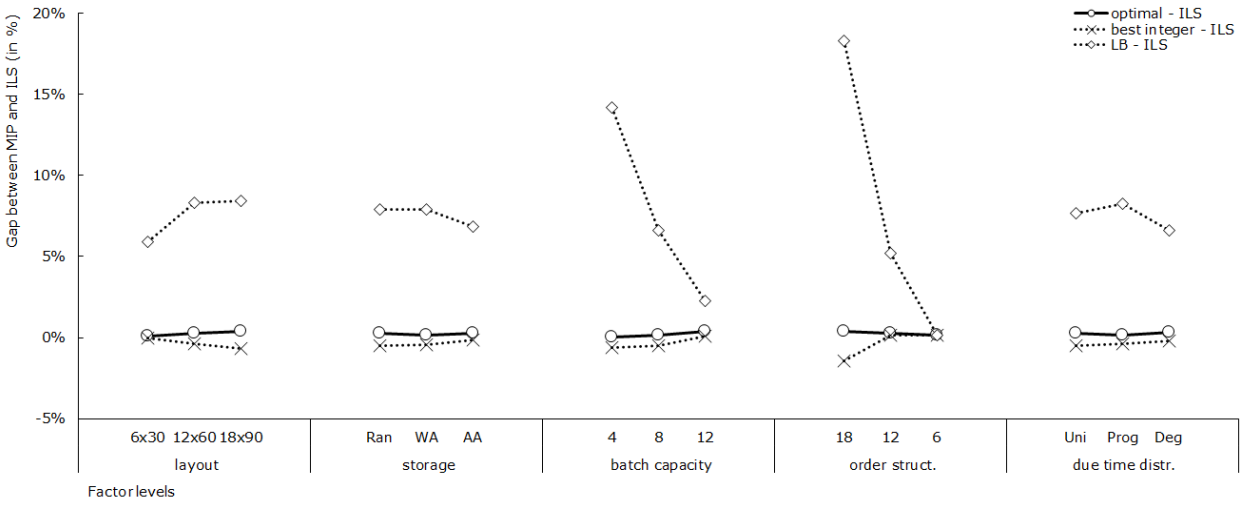

Figure 3: Percentage gap in order picking time between ILS and MIP for small problems.

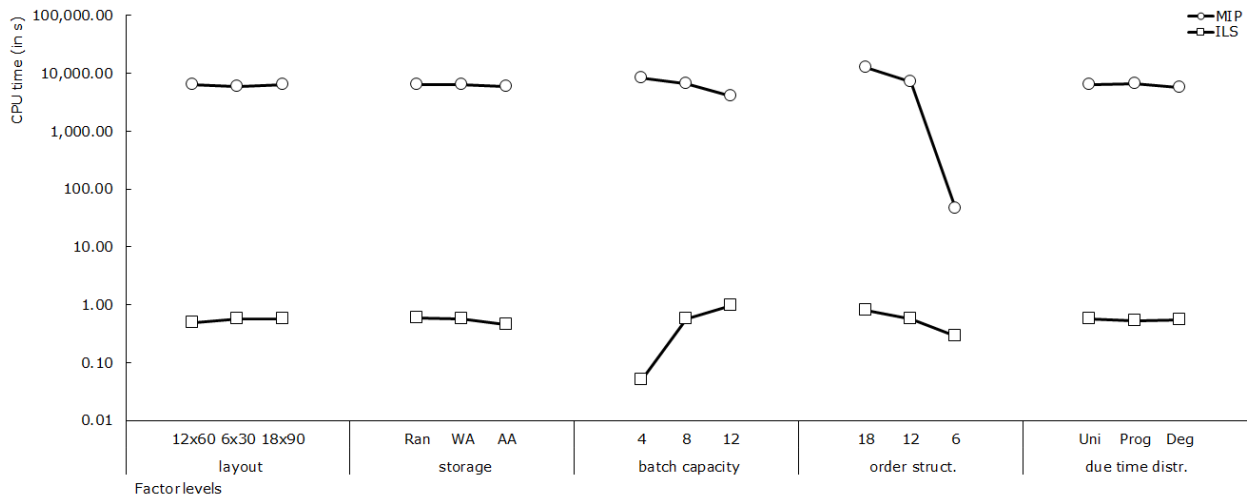

Figure 4: CPU time (in s) of MIP and ILS for small problems.

instances for which no feasible integer solution could be obtained by Cplex have been excluded from the analysis. From the 1,443 instances that could be solved to optimality using Cplex, the ILS algorithm is able to provide this optimal solution for 86.9 % of the instances in a single run per instance. The remaining 189 instances yield a mean gap between the ILS solution and the optimal order pick time of only 1.74 %. Figure 3 provides an overview of the performance of the ILS algorithm with respect to the total order picking time. The solid line on the graph illustrates the average gap between the optimal solution and the ILS objective function value for 1,443 instances solved to optimality by Cplex, while the other two lines compare the ILS solution to the lower bound and best MIP integer solution for all 2,346 instances for which Cplex finds a feasible solution within the run time limit. The size of the optimality gaps is rather equally distributed across the factor levels. With respect to the lower bound, gaps are substantial, at least for the factor levels with a high number of non-optimal instances (i.e., small batch capacity and a large number of orders). This can be explained by the large gaps between Cplex' best integer solution and corresponding lower bound. In general, the ILS algorithm is providing equal or even smaller order pick times compared to Cplex' best integer solution. To conclude, this analysis indicates that the ILS algorithm is able to effectively solve the integrated batching, routing and picker scheduling problem, at least for small problem sizes.

In order to evaluate the efficiency of the ILS algorithm, the computation times of the ILS algorithm are compared with the computation times for solving the MIP model with Cplex (Figure 4). Computation times decrease substantially when the problem is solved by the ILS algorithm. Furthermore, computation times of both approaches are rather insensitive to the order picking layout, storage policy and due time

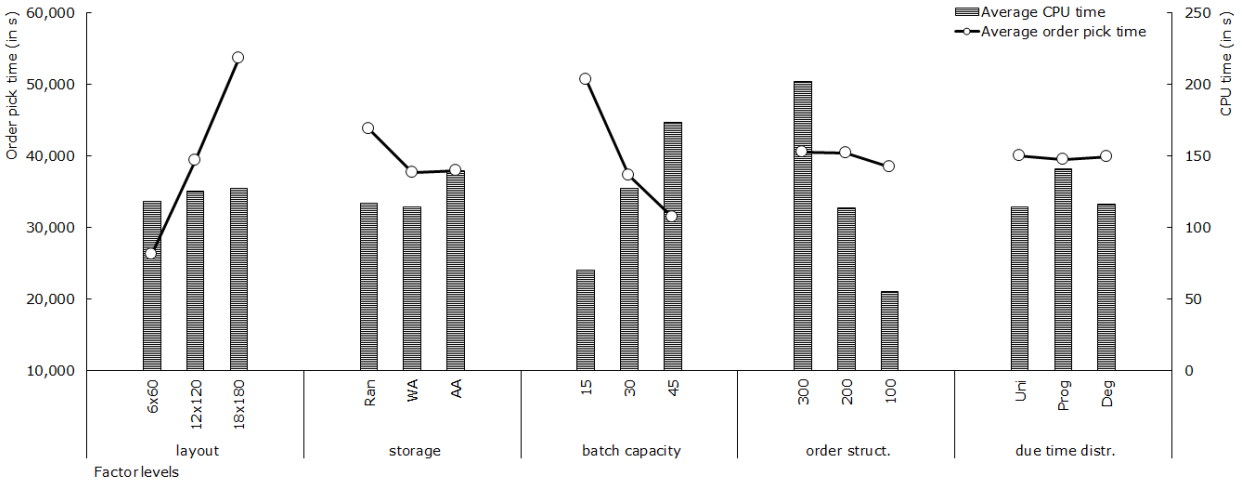

Figure 5: Order pick time and CPU time (in s) of ILS algorithm for large problems.

distribution of orders. With respect to the order structure, computation times strongly increase as the number of orders increases. Contradicting effects can be observed for the MIP model and the ILS algorithm with respect to the batch capacity factor. Computation times of the heuristic algorithm are mainly defined by the complexity of the routing problem. An increasing batch capacity results in more complex TSPs and thus increasing CPU times, whereas computation times of solving the MIP model are mainly defined by the number of created batches. Results show that the ILS algorithm is an efficient tool for solving the integrated batching, routing and picker scheduling problem, at least for small instances.

5.4. Analysis of the ILS algorithm for large problems

This section shows the performance of the ILS algorithm with respect to practically relevant problem sizes. Thirty order lists are generated for each factor level combination (see Table 3). A single ILS run is performed on each of the $7,290$ resulting instances. Additionally, a full factorial ANOVA is presented to analyze the effect of the experimental factors on the order pick time and CPU time of the ILS algorithm. Appendix B shows the statistical significance of the different factors on total order pick time as well as CPU time. The graph of Figure 5 illustrates the average order pick time and mean CPU time for each factor level.

With respect to the order picking layout, the order pick time increases linearly with increasing number of aisles and storage locations as the travel distance of order pickers rises. The computation time for running the ILS rises slightly when enlarging the order picking area. As more storage locations (and more SKUs) are included, while the number of order lines remains equal, the similarity of orders decreases (i.e., the

probability of equal locations in multiple orders decreases), resulting in increasing computation times.

Given the position of the depot and the location of the pick aisles, the within-aisle and across-aisle storage policies yield the smallest average order pick time. When designing order picking systems, the choice of the storage location assignment policy may yield significant performance benefits. Even in case of optimal order batching, routing and picker scheduling, order picking efficiency can be statistically significantly increased by choosing the right storage policy. On average, a reduction of 14 % (i.e., 1.7 hours in the four hour planning period) can be achieved by within-aisle storage classes, compared to random storage. The effect on CPU time is only minor, except for the slightly increased computation time in case of across-aisle storage location assignment.

A strong and statistically significant negative relation can be observed between batch capacity and order pick time. Batching more order lines in a single pick round significantly reduces travelling and setup time, resulting in a substantially lower order pick time. On the other hand, increasing batch capacity leads to a larger number of storage locations to be visited in each pick round. This complicates the routing problems, increasing CPU time of the ILS algorithm.

Figure 5 indicates a small statistically significant effect of the order structure level on average total order pick time. As more orders should be picked, average pick time increases slightly. However, the order structure does substantially influence the CPU time of the algorithm. A larger number of small orders increases the complexity as the neighbourhood size of the local search increases. Small orders facilitate shifting and swapping of orders, because of a decreasing probability of violating the batch capacity. Within each local search iteration, a larger number of order shifts and order swaps are tested, resulting in a strongly increased CPU time.

Finally, both average order pick time and CPU time are slightly depending on the due time distribution of orders. This small effect can be explained by the large number of orders that is included in the experiments, which facilitates combining similar orders in terms of SKUs. So, even with tight due times (i.e., degressive), the ILS algorithm is able to organize order picking operations efficiently. This means that the ILS algorithm can easily handle the arrival of new orders. As computation times are small enough, even if due times are tight, the initial schedule can be revised in case of the arrival of a significant number of new orders during the planned period, which allows to use the ILS in a dynamic setting as well.

In summary, the findings show that the proposed heuristic is able to find good solutions in reasonable computation times for problems of realistic size. The mean CPU time is less than four minutes (124 s). The proposed algorithm yields good performances for a wide range of realistic warehouse factors.

5.5. Analysis of the ILS algorithm for a real-life case

In order to show the benefits of integrating batching, routing, and picker scheduling in a real-life situation, the IBRSP is solved for a real-life case. Real-life data of a warehouse storing automotive spare parts are used to compare the performance of the ILS algorithm to the current operation of the warehouse (i.e., earliest due time (EDT) batching, return routing, batch assignment to the first available picker).

The experiments in this section focus on the order pick zone that stores the automotive spare parts that are ordered on-line. Order picking operations are performed 24 hours a day, divided into three 8 h shifts. As time windows for picking e-commerce orders are tight, orders are released multiple times during the day by supervisors. To simulate this order release mechanism, we assume a planning period of 4 h, **meaning that during each release, the set of orders whose due time is within the next four hours is released.** We simulate a high demand during each release, consisting of 200 orders. Due times of orders are approximated by an empirical distribution based on historical data of two weeks. The historical data are used to set the other warehouse parameters, as summarized in Table 6.

Table 6: Warehouse parameter values of the real-life case

Warehouse parameter	**Parameter value**
B	3 warehouse blocks
M	11 aisles
L	140 locations per level per aisle
J	7 levels
l_{length}	0.9 m
l_{width}	0.9 m
l_{height}	1.0 m
m_{width}	1.5 m
e_{width}	6.0 m
v	1.0 m/s
v_{lift}	0.2 m/s
t_a	$\max\left\{\frac{d_a}{v}; \frac{jl_{height}}{v_{lift}}\right\}$ s
t_{setup}	187 s
t_{search}	33 s
$t_{picking}$	4 h
Q	6 to 8 order pickers
c	13 order lines
K	200 orders
β	4 order lines

The layout of the order pick zone is shown in Figure 6. Arrows on the figure indicate the direction that order pickers should follow due to safety reasons. The high-level storage system consists of three warehouse blocks, two cross-aisles. Each storage rack consists of seven levels. Consequently, travelling in vertical direction is taken into account when creating order picker routes by using the Chebychev distance metric. SKUs are assigned to storage locations based on the across-aisle storage policy.

Thirty order lists, each consisting of 200 orders, are generated and evaluated using the ILS algorithm.

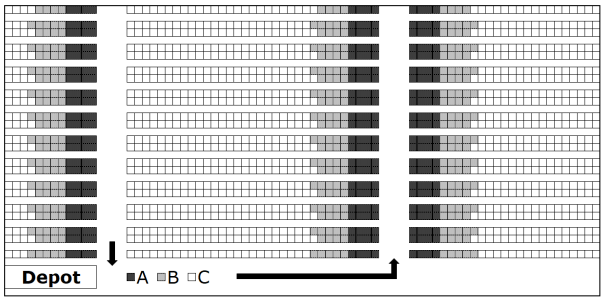

Figure 6: Layout of the order picking area.

The warehouse solves the problem sequentially: batches are created using EDT and assigned to the first available order picker. Order pickers follow a return routing policy. The currently applied policies in the warehouse are used as benchmark to evaluate the performance of the ILS algorithm. To illustrate that efficiency improvements are not possible by only optimizing routes, the LKH heuristic is applied to the batches created by EDT and compared to the integrated solution.

Figure 7 illustrates the average order picking time as well as the average CPU time for the real-life case. The EDT batching and return routing results in an average order pick time of 20.5 h (73,669 s), thereby employing 8 order pickers to prevent infeasible solutions due to tardiness. Optimizing order pick routes using the LKH heuristic results in a decline of 4.1 %, while the warehouse under consideration can reduce total order pick time with 16.9 % on average by integrating batching, routing and picker scheduling. The ILS algorithm provides an average order pick time of 17.5 h (62,995 s) with 8 pickers. This means that the effect of optimizing routes is small compared to the efficiency benefits of solving the IBRSP. Notice that the CPU time of the current policy combination is negligibly small. The ILS algorithm requires 79 s of computation time to find the integrated solution with 8 order pickers. This is acceptable in practice, given the strongly reduced order pick time. **At the short term, this reduced order picking time enables an earlier release of a new set of orders. This not only results in more retrieved orders, but also reduces the risk of tardiness due to unforeseen issues as the buffer between order retrieval and deadline is larger.**

The reduction of 10,674 s (73,669 s−62,995 s) by solving the IBRSP using the ILS algorithm, could eventually reduce the number of pickers **as the productivity of pickers increases. The workload forecast and mean productivity defines the daily required number of pickers. Therefore, a productivity increase reduces the required number of pickers to retrieve the forecast workload.** This effect is tested by simulating the experiments with a reduced number of order pickers. Reducing the

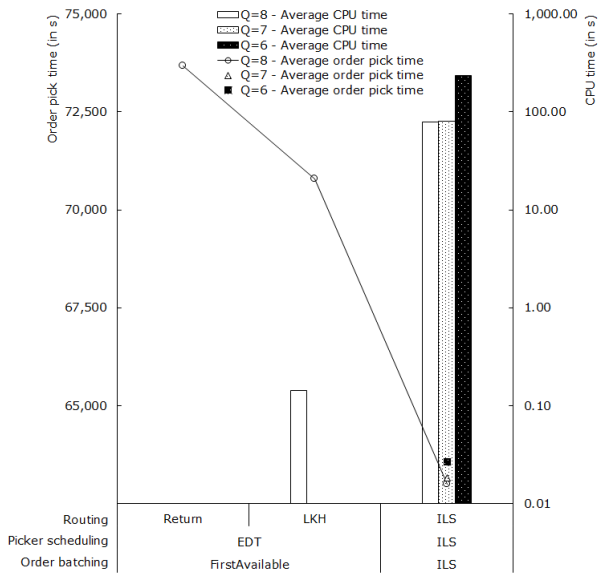

Figure 7: Order pick time and CPU time (in s) of the real-life case.

available number of pickers from 8 to 7 provides some infeasible instances (i.e., 13.3 %) with respect to the tardiness constraint if the benchmark policies are applied. When only routes are optimized, tardiness occurs in 10 % of the instances. This means that 8 pickers are required to prevent tardiness with respect to the benchmark. Due to the infeasible solutions, these results are not shown in Figure 7. When solving the IBRSP for all instances, even 6 order pickers are enough to pick all orders before the due time. Figure 7 shows that the experiments with 6 pickers result in a slightly higher mean order pick time and a substantial increase in computation time (i.e., from 79 s to 235 s) due to the tight **solution space: a large number of moves is tested and rejected during the local search because of tardiness. However, results of Figure 7 show that even with a tight solution space, similar picking efficiency benefits are possible compared to the scenario with** 8 **pickers.** Given the strongly increased productivity causing a substantially reduced number of pickers, the mean computation time of 235 s is acceptable in practice.

In summary, the ILS algorithm shows significant performance benefits compared to the current operation of the warehouse. For picking the same orders, the spare parts warehouse can significantly reduce the order pick time, without reducing the service level. Solving the IBRSP using the ILS algorithm increases the productivity of order pickers, requiring a smaller number of pickers. This reduces the required number of pickers by 25 % in this particular order picking zone of the spare parts warehouse. The ILS algorithm is able to solve the integrated batching, routing and picker scheduling efficiently even in case of high-level storage locations, in addition to the low-level storage systems that have been tested in previous sections.

Consequently, the ILS algorithm can be easily transferred to other order picking zones as well, either low-level or high-level storage locations.

6. Conclusions

Serving e-commerce markets forces warehouses to handle a larger number of orders in shorter time windows. This paper considers the integrated batching, routing and picker scheduling problem, ensuring a high customer service level. The proposed ILS algorithm accounts for order due times, a limited availability of order pickers as well as high-level storage locations to increase the applicability of the algorithm in practice. Results show that the proposed ILS algorithm is able to solve practically relevant problems in reasonable computation times.

Since batching, routing and picker scheduling are operational order picking planning problems, the new heuristic algorithm is rather easy to implement. Furthermore, solving the integrated problem results in substantial performance benefits of 16.9 % on average for the real-life spare parts warehouse of our case study. This makes the ILS algorithm an excellent decision support tool for managers to organize order picking operations and face the new market developments.

As order picking operations are labour-intense activities, future research may focus on integrating human factors while planning batching, routing and picker scheduling. Individual employee skills and capabilities may significantly impact the order pick time. This research opportunity is highly relevant to practice as considering these human factors can reduce the risk of tardiness due to unforeseen issues: assigning the most critical batches to the best performing pickers reduces the risk of orders that are picked too late, and in this way improves customer service. Furthermore, besides order due times, a limited availability of order pickers and high-level storage locations, respecting precedence constraints while creating order picker routes due to weight or fragility restrictions, or considering multiple locations of a single SKU (i.e., scattered storage) may further increase order picking efficiency and practical applicability. Considering these real-life characteristics may be highly relevant to practice and is largely unexplored in literature integrating order picking planning problems.

Acknowledgments

This work is supported by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office (research project COMEX, Combinatorial Optimization: Metaheuristics & Exact Methods).

The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government – department EWI.

References

Çeven E, Gue KR. Optimal Wave Release Times for Order Fulfillment Systems with Deadlines. Transportation Science 2015;51(1):52–66. doi:10.1287/trsc.2015.0642.

Chen TL, Cheng CY, Chen YY, Chan LK. An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. International Journal of Production Economics 2015;159:158–67. doi:10.1016/j.ijpe.2014.09.029.

Cheng CY, Chen YY, Chen TL, Jung-Woon Yoo J. 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 2015;170, Part C:805–14. doi:10.1016/j.ijpe.2015.03.021.

Clark KA, Meller RD. Incorporating vertical travel into non-traditional cross aisles for unit-load warehouse designs. IIE Transactions 2013;45(12):1322–31. doi:10.1080/0740817X.2012.724188.

Cornuéjols G, Fonlupt J, Naddef D. The traveling salesman problem on a graph and some related integer polyhedra. Mathematical Programming 1985;33(1):1–27. doi:10.1007/BF01582008.

De Koster RBM, Poort ESVD, Wolters M. Efficient orderbatching methods in warehouses. International Journal of Production Research 1999;37(7):1479–504. doi:10.1080/002075499191094.

Ene S, Öztürk N. Storage location assignment and order picking optimization in the automotive industry. The International Journal of Advanced Manufacturing Technology 2012;60(5-8):787–97. doi:10.1007/s00170-011-3593-y.

Helsgaun K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research 2000;126(1):106–30. doi:10.1016/S0377-2217(99)00284-2.

Henn S. Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. Flexible Services and Manufacturing Journal 2015;27(1):86–114. doi:10.1007/s10696-012-9164-1.

Henn S, Schmid V. Metaheuristics for order batching and sequencing in manual order picking systems. Computers & Industrial Engineering 2013;66(2):338–51. doi:10.1016/j.cie.2013.07.003.

Henn S, Wäscher G. Tabu search heuristics for the order batching problem in manual order picking systems. European Journal of Operational Research 2012;222(3):484–94. doi:10.1016/j.ejor.2012.05.049.

Kennedy WJ, Wayne Patterson J, Fredendall LD. An overview of recent literature on spare parts inventories. International Journal of Production Economics 2002;76(2):201–15. doi:10.1016/S0925-5273(01)00174-8.

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. Flexible Services and Manufacturing Journal 2012;24(1):52–80. doi:10.1007/s10696-011-9101-8.

Laporte G. The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research 1992;59(3):345–58. doi:10.1016/0377-2217(92)90192-C.

Li J, Huang R, Dai JB. Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. International Journal of Production Research 2016;55(2):447–61. doi:10.1080/00207543.2016.1187313.

Lin CC, Kang JR, Hou CC, Cheng CY. Joint order batching and picker Manhattan routing problem. Computers & Industrial Engineering 2016;95:164–74. doi:10.1016/j.cie.2016.03.009.

Lourenço HR, Martin OC, Stützle T. Iterated Local Search. In: Glover F, Kochenberger GA, editors. Handbook of Metaheuristics. Boston, MA: Springer US; International Series in Operations Research & Management Science; 2003. p. 320–53.

Marchet G, Melacini M, Perotti S. Investigating order picking system adoption: a case-study-based approach. International Journal of Logistics Research and Applications 2015;18(1):82–98. doi:10.1080/13675567.2014.945400.

Matthews J, Visagie S. Order sequencing on a unidirectional cyclical picking line. European Journal of Operational Research 2013;231(1):79–87. doi:10.1016/j.ejor.2013.05.011.

Matusiak M, De Koster RBM, Kroon L, Saarinen J. A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. European Journal of Operational Research 2014;236(3):968–77. doi:10.1016/j.ejor.2013.06.001.

Matusiak M, De Koster RBM, Saarinen J. Utilizing individual picker skills to improve order batching in a warehouse. European Journal of Operational Research 2017;263(3):888–99. doi:10.1016/j.ejor.2017.05.002.

Öncan T. MILP formulations and an Iterated Local Search Algorithm with Tabu Thresholding for the Order Batching Problem. European Journal of Operational Research 2015;243(1):142–55. doi:10.1016/j.ejor.2014.11.025.

Pan JCH, Shih PH, Wu MH. Order batching in a pick-and-pass warehousing system with group genetic algorithm. Omega 2015;57:238–48. doi:doi:10.1016/j.omega.2015.05.004.

Pan JCH, Wu MH, Chang WL. A travel time estimation model for a high-level picker-to-part system with class-based storage policies. European Journal of Operational Research 2014;237(3):1054–66. doi:10.1016/j.ejor.2014.02.037.

Pellegrini P, Birattari M. Out-of-the-Box and Custom Implementation of Metaheuristics. A Case Study: The Vehicle Routing Problem with Stochastic Demand. In: Intelligent Computational Optimization in Engineering. Springer, Berlin, Heidelberg; Studies in Computational Intelligence; 2011. p. 273–95. doi:10.1007/978-3-642-21705-0_10.

Roodbergen KJ, De Koster RBM. Routing order pickers in a warehouse with a middle aisle. European Journal of Operational Research 2001;133(1):32–43. doi:10.1016/S0377-2217(00)00177-6.

Scholz A, Henn S, Stuhlmann M, Wäscher G. A New Mathematical Programming Formulation for the Single-Picker Routing Problem. European Journal of Operational Research 2016;253(1):68–84. doi:10.1016/j.ejor.2016.02.018.

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. European Journal of Operational Research 2017;263(2):461–78. doi:10.1016/j.ejor.2017.04.038.

Scholz A, Wäscher G. Order Batching and Picker Routing in manual order picking systems: the benefits of integrated routing. Central European Journal of Operations Research 2017;:1–30doi:10.1007/s10100-017-0467-x.

Sörensen K, Glover FW. Metaheuristics. In: Encyclopedia of operations research and management science. Springer; 2013. p. 960–70.

Theys C, Bräysy O, Dullaert W, Raa B. Using a TSP heuristic for routing order pickers in warehouses. European Journal of Operational Research 2010;200(3):755–63. doi:10.1016/j.ejor.2009.01.036.

Tsai CY, Liou JJH, Huang TM. Using a multiple-GA method to solve the batch picking problem: considering travel distance and order due time. International Journal of Production Research 2008;46(22):6533–55. doi:10.1080/00207540701441947.

Valle CA, Beasley JE, Da Cunha AS. Optimally solving the joint order batching and picker routing problem. European Journal of Operational Research 2017;262(3):817–34. doi:10.1016/j.ejor.2017.03.069.

Van Gils T, Braekers K, Ramaekers K, Depaire B, Caris A. Improving Order Picking Efficiency by Analyzing Combinations of Storage, Batching, Zoning, and Routing Policies. In: Paias A, Ruthmair M, Voß S, editors. Lecture Notes in Computational Logistics. Springer International Publishing; number 9855 in Lecture Notes in Computer Science; 2016. p. 427–42. doi:10.1007/978-3-319-44896-1_28.

Van Gils T, Ramaekers K, Braekers K, Depaire B, Caris A. Increasing Order Picking Efficiency by Integrating Storage, Batching, Zone Picking, and Routing Policy Decisions. International Journal of Production Economics 2018a;197(Part C):243–61. doi:10.1016/j.ijpe.2017.11.021.

Van Gils T, Ramaekers K, Caris A, Cools M. The use of time series forecasting in zone order picking systems to predict order pickers' workload. International Journal of Production Research 2017;55(21):6380–93. doi:10.1080/00207543.2016.1216659.

Van Gils T, Ramaekers K, Caris A, De Koster RBM. Designing Efficient Order Picking Systems by Combining Planning Problems: State-of-the-art Classification and Review. European Journal of Operational Research 2018b;267(1):1–15. doi:10.1016/j.ejor.2017.09.002.

Won J, Olafson S. Joint order batching and order picking in warehouse operations. International Journal of Production Research 2005;43(7):1427–42. doi:10.1080/00207540410001733896.

Wruck S, Vis IFA, Boter J. Risk control for staff planning in e-commerce warehouses. International Journal of Production Research 2017;55(21):6453–69. doi:10.1080/00207543.2016.1207816.

Zhang J, Wang X, Chan FTS, Ruan J. On-line order batching and sequencing problem with multiple pickers: A hybrid rule-based algorithm. Applied Mathematical Modelling 2017;45:271–84. doi:10.1016/j.apm.2016.12.012.

Appendix A. Optimality cuts

This appendix outlines the optimality cuts used to strengthen the formulation. For a detailed discussion on the optimality cuts, the reader is referred to Valle et al. (2017). To describe the optimality cuts, each arc a is defined by its starting and ending vertex $(v'; v'')$. Let I_e^m be the number of vertices between the subaisle defined by cross-aisle e and cross-aisle $e+1$ in pick aisle m: a subaisle is defined as the part of a pick aisle between two cross-aisles. Each vertex v can be additionally expressed with respect to the location of intersection between the pick aisle, the cross-aisle neighbouring to the subaisle and most closely located to the depot (i.e., the cross-aisle to the left of the pick location in Figure 1) and the other vertices in the subaisle: let $v_e^m\{i\}$ be the ith vertex located in pick aisle m, with e the cross-aisle on the left-hand side of the pick location and i the position of an ordered set of vertices within subaisle between cross-aisle e and $e+1$ in pick aisle m. For artificial vertices, $i = 0$ (i.e., the intersection of a pick aisle and cross-aisle) and i is dropped in the notation. Let $v_0^0\{0\}$ be the vertex located at the depot, or simply v_0.

Order picking performance is assumed to be independent of the individual order picker. Therefore, formulation (3)-(20) may be subject to symmetry issues (i.e., swapping all orders assigned to two pickers yields an equivalent solution). This symmetry may increase computation times (Valle et al., 2017). Symmetry

breaking constraints (A.1) are added to the formulation to overcome this issue by forcing the first order to be assigned to the first order picker, the second order to the first or second picker, and so on.

$$R_{qpk} = 0 \qquad \forall k \in \kappa \qquad \forall q \in \sigma, q > k \qquad \forall p \in \pi \tag{A.1}$$

As distance is a symmetric function, incoming and outgoing arcs from the depot may be subject to symmetry issues (i.e., travelling is equal when performing a pick round clockwise or counter clockwise). Therefore, constraints (A.2) break symmetry by enforcing that the arc from the depot to a cross-aisle should be closer to the depot compared to the arc from a cross-aisle to the depot.

$$\sum_{\substack{e' \in \epsilon \\ e' \geq e}} X_{qp(v^1_{e'};v_0)} \geq \sum_{\substack{e' \in \epsilon \\ e' \geq e}} X_{qp(v_0;v^1_{e'})} \qquad \forall q \in \sigma \qquad \forall p \in \pi \qquad \forall e \in \epsilon \backslash \{1\} \tag{A.2}$$

In addition to symmetry breaking constraints, the feasible region can be reduced by including cuts that should be fulfilled in case of optimality. Let $\kappa_e \subset \kappa$ be a subset of orders for which other subaisles than the first pick aisle between cross-aisle e and $e+1$ should be visited to retrieve all order lines. This implies that the route should visit other pick aisles before returning to the depot as stated by constraints (A.3)-(A.5). For each vertex connected to the depot (i.e., for each cross-aisle), the constraint should be included.

$$X_{qp(v^1_1\{I^1_1\};v^1_2)} + X_{qp(v^1_1;v^2_1)} \geq X_{qp(v_0;v^1_1)} - (1 - R_{qpk}) \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall k \in \kappa_1 \tag{A.3}$$

$$X_{qp(v^1_e\{I^1_e\};v^1_{e+1})} + X_{qp(v^1_e;v^2_e)} + X_{qp(v^1_{e-1}\{1\};v^1_{e-1})} \geq X_{qp(v_0;v^1_e)} - (1 - R_{qpk}) \quad \forall q \in \sigma \quad \forall p \in \pi \quad \forall e \in \epsilon \backslash \{1; E\} \quad \forall k \in \kappa_e \tag{A.4}$$

$$X_{qp(v^1_E;v^2_E)} + X_{qp(v^1_{E-1}\{1\};v^1_{E-1})} \geq X_{qp(v_0;v^1_E)} - (1 - R_{qpk}) \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall k \in \kappa_E \tag{A.5}$$

Furthermore, cross-aisle and pick aisle cuts can be included. Each pick aisle (cross-aisle) cut separates the warehouse in two horizontal (vertical) parts, of which one part contains the depot (i.e., depot part). If a pick location of an order is not located in the depot part, at least one arc crossing the separation line

from the depot part to the other warehouse part should be used. In addition, at least one arc crossing the separation line in the other direction should be used. Optimality cuts (A.6)-(A.9) provide the cross-aisle cuts, equations (A.10)-(A.11) illustrate pick aisle cuts. Let $\kappa' \subset \kappa$ be the subset of orders containing at least one pick location not located in the depot part.

$$\sum_{m\in\mu} X_{qp(v_e^m;v_e^m\{1\})} \geq R_{qpk} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall e \in \epsilon\backslash\{E\} \quad \forall k \in \kappa' \tag{A.6}$$

$$\sum_{m\in\mu} X_{qp(v_e^m;v_e^m\{1\})} = \sum_{m\in\mu} X_{qp(v_e^m\{1\};v_e^m)} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall e \in \epsilon\backslash\{E\} \quad \forall k \in \kappa' \tag{A.7}$$

$$\sum_{m\in\mu} X_{qp(v_{e-1}^m\{I_{e-1}^m\};v_e^m)} \geq R_{qpk} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall e \in \epsilon\backslash\{1\} \quad \forall k \in \kappa' \tag{A.8}$$

$$\sum_{m\in\mu} X_{qp(v_{e-1}^m\{I_{e-1}^m\};v_e^m)} = \sum_{m\in\mu} X_{qp(v_e^m;v_{e-1}^m\{I_{e-1}^m\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall e \in \epsilon\backslash\{1\} \quad \forall k \in \kappa' \tag{A.9}$$

$$\sum_{e\in\epsilon} X_{qp(v_e^{m-1};v_e^m)} \geq R_{qpk} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu\backslash\{1\} \quad \forall k \in \kappa' \tag{A.10}$$

$$\sum_{e\in\epsilon} X_{qp(v_e^{m-1};v_e^m)} = \sum_{e\in\epsilon} X_{qp(v_e^m;v_e^{m-1})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu\backslash\{1\} \quad \forall k \in \kappa' \tag{A.11}$$

In addition to pick aisle and cross-aisle cuts, computation time is improved by including subaisle cuts (Constraints (A.12)-(A.18)). Let X^1_{qpv} be the minimum X_{qpa} value over all arcs a in the unique path from the left cross-aisle artificial vertex to a vertex v in the subaisle associated with the cross-aisle artificial vertex. X^2_{qpv} is similarly defined from the right cross-aisle artificial vertex to the left. Furthermore, let $\kappa_v \subset \kappa$ be the subset of orders containing at least one pick location at vertex v.

$$X^1_{qpv_e^m\{1\}} = X_{qp(v_e^m;v_e^m\{1\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \quad \forall e \in \epsilon\backslash\{E\} \tag{A.12}$$

$$X^1_{qpv_e^m\{i\}} \leq X_{qp(v_e^m\{i-1\};v_e^m\{i\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \quad \forall e \in \epsilon\backslash\{E\} \quad \forall i \in [1; I_e^m]\} \tag{A.13}$$

$$X^1_{qpv_e^m\{i\}} \leq X^1_{qpv_e^m\{i-1\}} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \quad \forall e \in \epsilon\backslash\{E\} \quad \forall i \in [1; I_e^m]\} \tag{A.14}$$

$$X^2_{qpv_e^m\{I_e^m\}} = X_{qp(v_{e+1}^m;v_e^m\{I_e^m\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \quad \forall e \in \epsilon\backslash\{E\} \tag{A.15}$$

$$X^2_{qpv_e^m\{i-1\}} \leq X_{qp(v_e^m\{i\};v_e^m\{i-1\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \quad \forall e \in \epsilon\backslash\{E\} \quad \forall i \in [1; I_e^m]\} \tag{A.16}$$

$$X^2_{qpv^m_e\{i-1\}} \leq X^2_{qpv^m_e\{i\}} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \qquad \forall e \in \epsilon \backslash \{E\} \quad \forall i \in [1; I^m_e]\} \tag{A.17}$$

$$X^1_{qpv^m_e\{i\}} + X^2_{qpv^m_e\{i\}} \geq R_{qpk} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall v^m_e\{i\} \in \psi, i > 0 \quad \forall k \in \kappa_{v^m_e\{i\}} \tag{A.18}$$

Computation time is further reduced by including optimality cuts that prevent routes to return in an artificial vertex. Constraints (A.19) prevent reversals between two artificial vertices, while Constraints (A.20)-(A.23) deal with reversals between a pick location vertex and artificial vertex.

$$\sum_{\substack{(v'';v^*) \in \alpha^-_{v''} \\ v^* \neq v'}} X_{qp(v'';v^*)} \geq X_{qp(v';v'')} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall v', v'' \in \bigcup_{m \in \mu} \bigcup_{e \in \epsilon} \{v^m_e\} : (v'; v'') \in \alpha^+_{v''} \tag{A.19}$$

$$\sum_{\substack{(v^m_e;v^-) \in \alpha^-_{v^m_e} \\ v^- \neq v^m_e\{1\}}} X_{qp(v^m_e;v^-)} \geq X_{qp(v^m_e\{1\};v^m_e)} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \qquad \forall e \in \epsilon \backslash \{E\} \tag{A.20}$$

$$\sum_{\substack{(v^-;v^m_e) \in \alpha^+_{v^m_e} \\ v^- \neq v^m_e\{1\}}} X_{qp(v^-;v^m_e)} \geq X_{qp(v^m_e;v^m_e\{1\})} \qquad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \qquad \forall e \in \epsilon \backslash \{E\} \tag{A.21}$$

$$\sum_{\substack{(v^m_e;v^-) \in \alpha^-_{v^m_e} \\ v^- \neq v^m_{e-1}\{I^m_{e-1}\}}} X_{qp(v^m_e;v^-)} \geq X_{qp(v^m_{e-1}\{I^m_{e-1}\};v^m_e)} \quad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \qquad \forall e \in \epsilon \backslash \{1\} \tag{A.22}$$

$$\sum_{\substack{(v^-;v^m_e) \in \alpha^+_{v^m_e} \\ v^- \neq v^m_{e-1}\{I^m_{e-1}\}}} X_{qp(v^-;v^m_e)} \geq X_{qp(v^m_e;v^m_{e-1}\{I^m_{e-1}\})} \quad \forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu \qquad \forall e \in \epsilon \backslash \{1\} \tag{A.23}$$

Finally, pass through optimality cuts prevent reversals at pick location vertices where no picking occurs. Let $\kappa_{v^m_e\{i\}} \subset \kappa$ be the set of orders that have a pick location at vertex $v^m_e\{i\}$. Then, constraints (A.24)-(A.27) ensure that the route passes through the vertices where no picking occurs, instead of returning.

$$\sum_{k \in \kappa_{v_e^m\{i\}}} -R_{qpk} \leq X_{qp(v_e^m\{i-1\};v_e^m\{i\})} - X_{qp(v_e^m\{i\};v_e^m\{i+1\})} \leq \sum_{k \in \kappa_{v_e^m\{i\}}} R_{qpk}$$

$$\forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu$$

$$\forall e \in \epsilon \quad \forall i \in [1; I_e^m - 1]\} \tag{A.24}$$

$$\sum_{k \in \kappa_{v_e^m\{I_e^m\}}} -R_{qpk} \leq X_{qp(v_e^m\{I_e^m-1\};v_e^m\{I_e^m\})} - X_{qp(v_e^m\{I_e^m\};v_{e+1}^m)} \leq \sum_{k \in \kappa_{v_e^m\{I_e^m\}}} R_{qpk}$$

$$\forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu$$

$$\forall e \in \epsilon \tag{A.25}$$

$$\sum_{k \in \kappa_{v_e^m\{i\}}} -R_{qpk} \leq X_{qp(v_e^m\{i+1\};v_e^m\{i\})} - X_{qp(v_e^m\{i\};v_e^m\{i-1\})} \leq \sum_{k \in \kappa_{v_e^m\{i\}}} R_{qpk}$$

$$\forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu$$

$$\forall e \in \epsilon \quad \forall i \in [1; I_e^m]\} \tag{A.26}$$

$$\sum_{k \in \kappa_{v_e^m\{I_e^m\}}} -R_{qpk} \leq X_{qp(v_{e+1}^m;v_e^m\{I_e^m\})} - X_{qp(v_e^m\{I_e^m\};v_e^m\{I_e^m-1\})} \leq \sum_{k \in \kappa_{v_e^m\{I_e^m\}}} R_{qpk}$$

$$\forall q \in \sigma \quad \forall p \in \pi \quad \forall m \in \mu$$

$$\forall e \in \epsilon \tag{A.27}$$

Appendix B. ANOVA results on large problems

Tables B.7 and B.8 present results of a $3 \times 3 \times 3 \times 3 \times 3$ full factorial ANOVA on average order pick time and mean CPU time, respectively. Results show the statistical significance of the main effects and the two-way interactions among the warehouse factors.

Table B.7: $3 \times 3 \times 3 \times 3 \times 3$ full factorial ANOVA on average order pick time

	Sum of squares	**df**	**Mean square**	F	p**-value**
Main effects					
Layout	912,819,618,087	2	456,409,809,043	43,780.31	0.000
Storage policy	58,916,943,244	2	29,458,471,622	2,825.75	0.000
Batch capacity	475,507,803,914	2	237,753,901,957	22,806.13	0.000
Order struct.	6,243,561,363	2	3,117,280,681	299.02	0.000
Due time distr.	332,652,634	2	166,326,317	15,95	0.000
Two-way interaction					
Layout × Storage policy	15,022,966,698	4	3,755,741,674	360.26	0.000
Layout × Batch capacity	21,362,500,968	4	5,340,625,242	512.29	0.000
Layout × Order struct.	640,314,732	4	160,078,683	15.36	0.000
Layout × Due time distr.	215,727,537	4	53,931,884	5.17	0.000
Storage policy × Batch capacity	426,029,184	4	106,507,296	10.22	0.000
Storage policy × Order struct.	253,013,230	4	63,253,308	6.07	0.000
Storage policy × Due time distr.	21,315,277	4	5,328,819	0.51	0.728
Batch capacity × Order struct.	25,553,283,352	4	6,388,320,838	612.79	0.000
Batch capacity × Due time distr.	41,810,928	4	10,452,732	1.00	0.405
Order struct. × Due time distr.	38,378,577	4	9,594,644	0.92	0.451
Residuals					
Between subjects	75,466,590.370	7,239	10,425,002		
Total	1,592,862,510,095	7,289			

Table B.8: $3 \times 3 \times 3 \times 3 \times 3$ full factorial ANOVA on CPU time

	Sum of squares	**df**	**Mean square**	F	p**-value**
Main effects					
Layout	103,129	2	51,565	13.01	0.000
Storage policy	928,521	2	464,261	117.12	0.000
Batch capacity	13,049,663	2	6,524,832	1,646.08	0.000
Order struct.	26,607,029	2	13,303,514	3,356.20	0.000
Due time distr.	1,059,850	2	529,925	133.69	0.000
Two-way interaction					
Layout × Storage policy	224,468	4	56,117	14.16	0.000
Layout × Batch capacity	267,300	4	66,825	16.86	0.000
Layout × Order struct.	236,227	4	59,057	14.90	0.000
Layout × Due time distr.	55,270	4	13,818	3.49	0.008
Storage policy × Batch capacity	275,010	4	68,753	17.34	0.000
Storage policy × Order struct.	444,193	4	111,048	28.02	0.000
Storage policy × Due time distr.	58,342	4	14,585	3.68	0.005
Batch capacity × Order struct.	2,147,753	4	536,938	135.46	0.000
Batch capacity × Due time distr.	84,043	4	21,011	5.30	0.000
Order struct. × Due time distr.	446,724	4	111,681	28.17	0.000
Residuals					
Between subjects	28,694,378	7,239	3,964		
Total	74,701,900	7,289			