# Optimizing Workforce Allocation under Uncertain Activity Duration

Vincent Derkinderen[a,*], Jessa Bekker[b], Pieter Smet[c]

[a]*KU Leuven, Dept. of Computer Science, DTAI, B-3000 Leuven, Belgium*
[b]*KU Leuven, Dept. of Computer Science, DTAI, B-3000 Leuven, Belgium*
[c]*KU Leuven, Dept. of Computer Science, CoDeS, 9000 Gent, Belgium*

**Abstract**

Even though warehouses are becoming increasingly automated, humans remain their central and most important resource. Every day, various activities must be carried out by workers. The assignment of individual workers to specific tasks has a major impact on the overall efficiency of a warehouse. The problem of finding an efficient assignment is not trivial and is complicated by task durations being unknown in advance, operational constraints, and the fact that employee well-being must be taken into consideration to maintain employee satisfaction. The method proposed in this paper uses work profiles: ordered lists of task properties such as type and work zone. Each worker is assigned exactly one profile and tasks are dynamically allocated to workers based on their profile. Finding a good profile assignment is crucial, yet the profiles are usually assigned manually by shift supervisors. This paper proposes a framework for automating the assignment of profiles to employees under uncertain task durations. The proposed approach applies a metaheuristic algorithm together with discrete-event simulation to evaluate the quality of a solution. The simulation component is used to address uncertainty of the task durations either by considering multiple scenarios or by using mean task durations. Possible values for task durations originate from distributions which are assumed to be given or learned from historical data. The contributions of this paper are threefold: 1) we propose a profile-assignment framework that deals with uncertainty of the task durations, 2) we study the trade-offs between run time and accuracy within this framework, and 3) we analyze our main design decisions and demonstrate how our method outperforms reconstructed solutions produced by a human expert.

*Keywords:* warehouse scheduling, uncertainty, heuristics, simulation, decision tree

## 1. Introduction

Warehouses serve as intermediate storage facilities for goods on their way from producers to consumers, and constitute an essential part of the supply chain. Typical activities in warehouses include receiving and unloading incoming goods, storing those goods

*Corresponding author

*Email addresses:* vincent.derkinderen@kuleuven.be (Vincent Derkinderen), jessa.bekker@kuleuven.be (Jessa Bekker), pieter.smet@kuleuven.be (Pieter Smet)

at appropriate locations, order picking and preparing outbound goods for shipping. In non-automated warehouses, these activities are primarily carried out by employees, who must therefore be managed effectively in order to ensure high-quality service levels. Many academic studies have focused on optimizing specific warehouse processes such as order picking (De Koster et al., 2007). However, few studies have proposed decision support models for optimizing the deployment of available warehouse staff where the duration of their activities is uncertain. Tasks are usually assigned to employees from a centralized *warehouse management system* (WMS). When automated, this process is referred to as system-direct task dispatching. Whenever an employee becomes available, the WMS's dispatching algorithm assigns them a new task based on a number of parameters.

This paper considers a setting where tasks with unknown durations are dynamically assigned based on worker *profiles*: a description of the activities a particular worker may perform on that day as well as the priorities between each activity. For example, a profile may restrict an employee to only carry out activities which require a forklift in the inbound and storage areas of the warehouse. Assigning profiles to employees provides a more coarse, high-level form of control compared to directly assigning tasks. This makes it easier to steer priorities, while control over specific task assignments remains possible by defining more fine-grained profiles. Dealing with unforeseen events is also relatively simple. By way of example, consider a truck arriving earlier than planned. The tasks related to this truck can be prioritised, which would lead to a temporary re-assignment of profiles to employees. Another important benefit of using profiles is that they enable employees to know in advance the nature of the activities they will have to carry out. This certainty typically has a positive impact on employee job satisfaction.

Central to the profile-based dispatching approach is the assignment of profiles to employees. This assignment problem is essential to achieve a timely execution of all tasks, but it is nontrivial to solve. Furthermore, while we assume that the set of tasks is known in advance, it is very difficult to determine a priori precisely how long each task will take. This further complicates the assignment problem. While we investigate this problem in the context of warehouse operations, our contribution might also be applicable in other settings where profile-based dispatching is used.

To address the profile assignment problem, we propose a framework that combines a metaheuristic algorithm to explore the search space and discrete-event simulation to evaluate solution quality. The metaheuristic is responsible for searching the solution space of possible profile assignments, while the evaluation function simulates how well tasks can be scheduled given an assignment of profiles to employees. An additional feedback loop within the algorithm ensures that the metaheuristic is guided towards promising candidate solutions based on the simulation's outcome. To account for uncertain task durations, we investigate both a simulation using mean task durations as well as a simulation of several scenarios each with their own sampled task durations. A decision tree trained on historical data is used to select attribute-conditional distributions which accurately represent the task durations. The resulting methodology is empirically evaluated on data we obtained from our industry partner, Objective International: a leading European supply chain fulfillment provider to the manufacturing and process industry that offers manufacturing execution solutions and warehouse management solutions. The data used in our computational study has been made publicly available (Derkinderen et al., 2023). Our results demonstrate how our approach is capable of generating high-quality solutions. Moreover, the speed with which it produces these solutions indicates that it can be used in a real-world context.

This paper's contributions are threefold: 1) proposing a framework to automatically assign profiles while considering the task duration uncertainty, 2) a study of the trade-offs between run time and accuracy within this framework, and 3) an empirical evaluation that supports the algorithm's primary design decisions and demonstrates how the proposed algorithm finds solutions which outperform those manually constructed by human experts.

The remainder of this paper is organized as follows. Section 2 reviews related work in the academic literature. Section 3 outlines the context of the studied problem and introduces the required terminology. Section 4 more formally introduces the optimization problem addressed, while Section 5 details the proposed solution approach. Section 6 presents the results of a computational study which analyzes the impact of various algorithm parameters on the performance of our approach. Furthermore, a comparison is made to solutions constructed by human experts. Finally, Section 7 concludes the paper and identifies directions for future research.

## 2. Related work

Personnel scheduling has been extensively studied in the academic literature during the last decades (Ernst et al., 2004). Common applications of personnel scheduling include scheduling nurses in hospitals, scheduling agents in call centers and scheduling airline crews (Van den Bergh et al., 2013). In the context of warehouses, however, the topic has received limited attention, despite the fact that workers are considered one of the primary resources in warehouses (Davarzani and Norrman, 2015). Most academic studies related to warehouses have instead focused on the optimization of specific warehouse management aspects such as storage assignment, layout design or order batching. van Gils et al. (2018) provide an overview of common problems in warehousing, at both the tactical and operational levels of decision making. While most of these problems indirectly affect the activities of workers, there are a few which directly allocate workers to activities. We consider these to be the most relevant personnel-related problems and will further discuss them in this section. More specifically, the surveyed problems are:

- *Workforce level*: determine the required number of workers in either the warehouse as a whole or in different zones, in order to guarantee a desired service level.
- *Workforce allocation*: allocate the available workforce across warehouse zones in order to maximize throughput.
- *Job assignment and scheduling*: determine the sequence of order retrievals before assigning and scheduling the orders to workers so that due times and delivery schedules are respected as much as possible.

### *2.1. Workforce level*

The number of available workers in a warehouse should match the expected workload. If too few workers are available, service levels will deteriorate as a result of increased order lead times or delivery schedule delays. However, these workers are often the most costly resource in a warehouse and scheduling too many leads to unnecessarily high labor costs.

A well-designed methodology to determine the appropriate workforce size for a given day or shift is thus essential.

van Gils et al. (2017) propose the use of forecasting methods to predict next-day workload. Using time series models, they accurately and reliably predict the expected workload in different zones of a large warehouse. Supervisors can then use this information to derive the number of workers required per zone.

An adverse effect of scheduling too many workers is aisle congestion (Ruben and Jacobs, 1999). This congestion may lengthen the amount of time required to complete tasks due to workers blocking each other in the warehouse's aisles. Pan and Wu (2012) demonstrate how this may lead to routing policies that result in longer task waiting times. Chen et al. (2016) introduce an approach for routing multiple workers in a warehouse while considering congestion due to blocking effects. Another approach to mitigate the effects of blocking was proposed by Hong et al. (2012), who take into account possible congestion issues during the construction of order batches.

### *2.2. Workforce allocation*

In order to avoid bottlenecks in the warehouse and maximize order throughput, the available workforce must be carefully allocated to different zones in the warehouse. Van Nieuwenhuyse and de Koster (2009) use queuing models to evaluate the impact of different workforce allocations on picking and sorting operations. A series of simulations revealed that workforce allocation has a much larger impact on the system's performance than optimal batch sizing, a result which once again underscores the importance of optimized personnel scheduling.

De Koster et al. (2012) use mathematical programming to determine picking zone size and the allocation of workers to these zones. Small zones reduce the time required to pick an order since travel time is reduced. However, they also increase the time necessary to consolidate the partial orders from different picking zones. Larger zones reduce the likelihood of orders being split across many zones, but the travel time within a zone may increase. By comparing the outcome of different scenarios, the best performing setting is determined. A similar problem was addressed by Yu and De Koster (2008) with analytical methods. They used a queuing model to show how larger zones, and thus more workers per zone, increase picking time due to lengthier travel times. They determined the best configuration for a case study by comparing the outcome of different scenarios.

### *2.3. Job assignment and scheduling*

In systems with a single worker, job assignment simply involves sequencing the tasks which must be performed (Chen et al., 2015). When considering multiple workers, an assignment decision must also be made together with sequencing the tasks for each worker (Henn, 2015; Zhang et al., 2017). Common objectives in job assignment and scheduling are related to task completion time, for example minimizing the tardiness of customer orders (Scholz et al., 2016).

Traditionally, the process of job assignment and scheduling is the last of the three sequential steps in the operational planning of warehouses. These three steps are 1) order batching, 2) routing, and 3) assignment. Different approaches have been proposed which solve the assignment problem in combination with routing (Matthews and Visagie, 2013), order batching (Henn and Schmid, 2013), and both order batching and routing (van Gils et al., 2019). Due to the increase in size and complexity of these integrated problems, metaheuristics or exact methods which decompose the problem often represent the preferred solution methodologies.

Ladier et al. (2014) highlight the strong variability of workloads in warehouses and emphasize the complexity of weekly timetabling and daily rostering in this setting. Weekly timetabling determines the workforce size and assigns workers to shifts, while daily rostering assigns workers to individual tasks. Ladier et al. (2014) solve these problems sequentially using mathematical programming.

Rijal et al. (2021) simultaneously consider job assignment and scheduling in addition to shift scheduling. The resulting integrated problem involves decisions regarding the number of workers, shift start and end times, break times as well as the assignment and scheduling of worker activities. The authors present two solution approaches: a branch and price algorithm and a metaheuristic. By increasing flexibility in the break schedules, labor costs are reduced by up to 5%.

Ganbold et al. (2020) optimize the assignment of workers to activities using a heuristic search algorithm. Their problem is modeled as a generalized assignment problem such that each worker can be assigned to at most one activity. Several operational constraints are taken into account such as worker availability, qualifications/skills, and precedence constraints. Activity duration is modeled as a stochastic variable, while discrete-event simulation is used to compute the cost of assignments. A computational study demonstrates how the solutions generated by their algorithm outperform those manually produced by the organization.

### *2.4. Positioning of the paper*

The problem we study in this paper integrates workforce allocation and job assignment. Workers are allocated to zones situated throughout the warehouse and to specific activities taking place therein. A job assignment problem is solved to evaluate the quality of the allocation, with task durations modeled as stochastic variables. The previous study most closely related to our work is that of Ganbold et al. (2020), who evaluate the assignment of workers to activities with stochastic durations using simulation. However, there are two important differences compared to our problem. First, we consider activities to be defined as combinations of work zones and task types, such as picking and loading. We therefore generalize Ganbold et al. (2020)'s concept of activities, which only considers zones. Second, due to the restricted nature of the generalized assignment problem model used by Ganbold et al. (2020), workers can only be assigned to at most one activity. In our setting, workers can be allocated to multiple work zones and task types. As a consequence, evaluating the allocation becomes difficult as it not only involves simulating the flow of orders through the warehouse, but also the assignment of specific job types to workers.

## 3. Warehousing context

Before describing the optimization problem which is the focus of this paper, an overview of relevant terminology and concepts within warehousing is provided in Section 3.1. An explanation of the task assignment procedure employed in this paper is then provided in Section 3.2.

### *3.1. Warehouse activities*

Activities carried out within a warehouse are typically associated with one of three general zones: inbound, storage and outbound. The inbound zone serves as the entry point for new goods delivered by trucks. After receiving the goods and conducting quality control procedures, these goods are then transferred to their designated locations as determined by the storage plan. The storage zone comprises of racks in which the goods are kept until they are required to fulfill an order. Activities here include replenishing goods when picking locations become empty, periodic stock-taking and the relocation of goods if, for example, there is a change in the storage plan. Finally, a warehouse's outbound zone is where all necessary operations are conducted to prepare goods for shipping. After picking an order, additional value-adding operations may be performed such as kitting or labeling, before loading the goods into trucks and shipping them out. Figure 1 provides an overview of different warehouse activities and how they are grouped into these three zones.

In some facilities, the production of new goods is closely related to the warehousing activities. A designated area within the facility produces new goods, which can then be stored immediately in the warehouse without requiring any intermediary shipping. Similarly, goods used during the production process can be taken directly from the warehouse. This extension of regular warehousing operations is also illustrated in Figure 1 using gray boxes.

In addition to the three main zones depicted in Figure 1, the warehouse may be further subdivided into physical work areas. This decomposition is typically performed in accordance with the type of handling equipment required in an area. For example, some areas may require a forklift while others may be serviced using only a pallet truck.

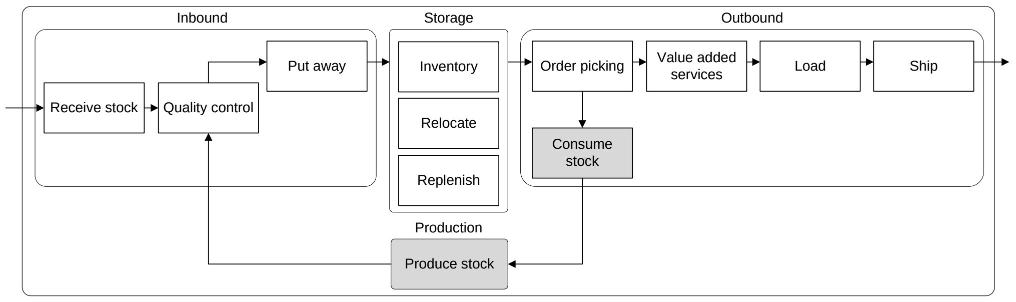


Figure 1: Overview of warehousing activities.

### *3.2. Task assignment to employees*

Activities are assigned to employees in the form of discrete tasks. Table 1 provides an overview of different task types. For the warehouses considered in this paper, tasks are assigned to employees using *system-directed dispatching*. In such environments, the WMS maintains a *task pool* containing tasks that are ready to be carried out. Tasks are released into the task pool according to the schedule which was constructed beforehand based on known incoming and outgoing shipments. Employees query the WMS through, for example, a handheld device in order to receive a new task from the pool whenever they are idle. Essentially, this is a dynamic scheduling problem which the WMS solves using simple dispatching rules based on task attributes such as priority and release time.

Table 1: Task types and the zone in which they typically occur.

| Warehouse zone | Task type | Description |
|---|---|---|
| Inbound | Put away | Transfer goods to picking locations in the storage area |
| Storage | Inventory | Periodic stock-taking |
| | Relocate | Move goods within the storage area |
| | Replenish | Refill picking locations with goods |
| Outbound | Pick | Retrieve goods from picking locations |
| | Marshal | Prepare goods for incoming trucks |
| | Load | Load goods into a truck for outbound transport |

For each shift, employees are restricted to certain task types and work areas. This means that at the beginning of each shift, workers have foreknowledge concerning the nature of the activities they will have to carry out. This ensures that employees are only assigned to tasks for which they are qualified. For example, some tasks may require a forklift license, a qualification not every employee will necessarily have. To enforce these restrictions, each employee is assigned a *profile* before their shift starts. This profile determines the tasks they may perform from the task pool, as well as the priorities between those tasks. Table 2 provides an example of a profile with four *profile rules*, each of which is defined as the combination of a task type and a work zone. In this example, once the employee becomes available the WMS's dispatching algorithm first checks if the current task pool contains any `put away` tasks in zone `010 - inbound area`. When no such tasks are available, the algorithm checks for `pick` tasks in zone `600 - pick locations`. If there are once again no such tasks available in the pool, the algorithm continues through the other profile rules until a matching task is found and assigned to the employee. The final profile rule typically serves as a catch-all rule for which there is, by design, always a (non-urgent) matching task present in the task pool.

Table 2: Example of a profile with four profile rules.

| Order | Task type | Work zone |
|---|---|---|
| 1 | Put away | 010 - inbound area |
| 2 | Pick | 600 - pick locations |
| 3 | Replenish | 600 - pick locations |
| 4 | Inventory | 600 - pick locations |

## 4. Problem definition

The goal of the profile assignment problem is to find an assignment of employees to profiles on a single day, such that the task execution follows the existing schedule for that day as closely as possible. In other words: a task's execution should begin as soon as possible once it is released into the task pool. The precise duration of each task is unknown in this setting. More formally:

**Given** (1) a set of tasks to be executed with their properties and their scheduled earliest start time, (2) the set of available employees and (3) a set of possible profiles. We also assume a probability distribution for each task's duration is given. In practice, this means sufficient data is available from which such a distribution can be learned. This assumption is reasonable for modern warehouses, where employing dynamic task dispatching already requires some form of automated tracking and registration of activities.

**Find** a solution $S$, which assigns each employee to a profile.

**Such that** all tasks can be executed and the expected total time spent by tasks waiting in the task pool is minimized.

A *profile* describes the types of task which can be carried out by an employee. It is an ordered set of profile rules, with each rule consisting of a task type and a work zone. Optionally, the task types or work zones can be wildcards, in which case they match with any task type or work zone, respectively. Wildcards are useful when only the task type or the work zone matter and can also be used to construct catch-all rules to prevent employees from spending extended periods idle. The order of the profile rules dictates the priority of the different combinations of task types and work zones.

Each *task* is characterized by its properties: type, work zone, priority, whether it is bulk or pallet, and whether it involves one or multiple items. The earliest possible start time of a task is the time at which it is released into the task pool. This time is given in a task schedule that is determined in advance, either computationally or manually by human planners.

The *cost* of a profile assignment solution $S$ is the total time spent by tasks in the task pool. This is the sum over all tasks, taking the difference between the time at which an employee actually started each task and its earliest possible start time:

$$\text{cost}(S, \mathbf{d}) = \sum_{t \in \text{tasks}} (\text{actual\_start\_time}_t(S, \mathbf{d}) - \text{earliest\_start\_time}_t) \tag{1}$$

Minimizing this cost ensures that tasks are started close to their release times. The time at which a task actually starts is deterministically decided by the WMS's dispatching algorithm based on the profile assignments $S$ and the duration of each task denoted as $\mathbf{d}$. When an employee becomes available — either at the beginning of their shift or after completing a task — a task is selected from the task pool based on their profile. If multiple tasks have the same highest priority according to the profile, task priority is used as a tie breaker. When the task priority is also equivalent then one of the tasks is arbitrarily chosen. Additionally, there may exist precedence constraints between tasks. In this case, a task cannot be selected from the task pool until its preceding tasks have been completed.

The individual task durations $\mathbf{d}$ are not known in advance given that they depend not only on known task properties such as task type, but also on unknown factors such as physical obstructions, unforeseen defects and variable employee performance. Therefore, task duration is modeled as a stochastic variable $\mathbf{d}$. As a result, the cost of a solution $S$, which partially depends on $\mathbf{d}$, is also stochastic. Since $\mathbf{d}$ is unknown, we will minimize the expected cost of a solution $S$:

$$\begin{aligned}&\mathop{\mathbb{E}}_{\mathbf{d}\sim\Pr(\mathbf{d})}[\text{cost}(S,\mathbf{d})] \\ &\qquad = \sum_{t\in\text{tasks}} \left( \mathop{\mathbb{E}}_{\mathbf{d}\sim\Pr(\mathbf{d})}[\text{actual\_start\_time}_t(S,\mathbf{d})] - \text{earliest\_start\_time}_t \right) \end{aligned} \tag{2}$$

For brevity purposes, we will abbreviate $\mathbb{E}_{\mathbf{d}\sim\Pr(\mathbf{d})}[\text{cost}(S,\mathbf{d})]$ to $\mathbb{E}[\text{cost}(S)]$ and $\text{cost}(S,\mathbf{d})$ to $\text{cost}(S)$.

Although the exact duration of each task is unpredictable, historical data concerning its duration is typically available. Given this data, it is possible to estimate a distribution of possible task durations as described in Section 5.3.

All tasks in the task pool must be executed. A profile assignment $S$ that does not cover one or more tasks should therefore be penalized by a very large cost. This is achieved by overwriting the actual start time for a task $t$ that does not get executed so that $\textit{actual\_start\_time}_t(S,\mathbf{d}) - \textit{earliest\_start\_time}_t = 24\ \textit{hours}$ in Equation (2).

The computation time needed to find a good solution is also important. Task schedules are typically constructed during the preceding workday, which imposes a hard time limit for finding a solution. Having an algorithm that can quickly produce a high-quality solution, say within five minutes, also has the additional benefit that it can accommodate last-minute changes. Therefore, an algorithm that finds good solutions fast is often more useful in practice than one that takes far longer to produce a slightly better solution.

## 5. Solution approach

To solve the profile assignment problem, we propose a sample-based approach that combines an Iterated Local Search (ILS) metaheuristic with discrete-event simulation. ILS is responsible for searching the solution space of all possible profile assignments $S$. The expected cost of a solution is computed by simulating the WMS system for one or more scenarios. In a scenario, tasks are scheduled and assigned to employees while respecting constraints imposed by the solution's profile assignments. A scenario is defined by the time it takes to complete each of its tasks, which is determined by the estimated task duration distributions. The interaction between these various components is outlined in Figure 2 and each will be discussed in the following subsections.

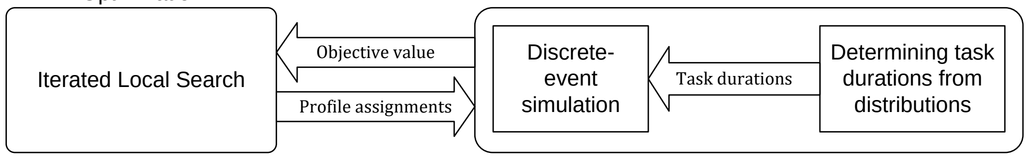


Figure 2: Overview of the components in the proposed framework.

## *5.1. Iterated local search*

ILS is a well-known metaheuristic which has been successfully applied to various optimization problems (Lourenço et al., 2003). Algorithm 1 outlines the proposed ILS approach. The algorithm begins by constructing an initial solution using the procedure `initialize_solution`(), which randomly selects one profile for each employee (line 1). Note that the set of possible profiles from which we randomly select is assumed to be given. These are either profiles that have been used before or newly constructed profiles that are deemed useful for the tasks that must be performed. The core of the algorithm is a local search loop which optimizes solution $S$ (lines 4-9). Every time an improved solution is found, it is compared against the best known solution $S^*$ that is then updated (line 9).

A straightforward representation of a solution $S$ is a mapping from employees to profiles. An alternative representation is a mapping from profiles to the number of employees that should be assigned to that profile, $S : P \mapsto \mathbb{N}$. The advantage of this second representation is that it exploits the interchangeability of employees, thereby breaking symmetry and thus reducing the size of the search space. For this reason, we employ the second representation.

Neighboring solutions are generated by the procedure `neighbor`($S$), which we define as swapping one profile in the current solution $S$ for another profile. In our representation $S : P \mapsto \mathbb{N}$, this corresponds to decrementing the count of profile $p^-$ and incrementing the count of another profile $p^+$. To decide what profiles $p^-$ and $p^+$ should be, we propose two approaches: a uniform and a weighted approach. The *uniform* approach is performance agnostic, deciding $p^-$ by randomly selecting an active profile (a profile assigned to at least one employee) and deciding $p^+$ by randomly selecting any profile other than $p^-$ (active or inactive). This approach considers a uniform distribution such that all profiles have an equal probability of being swapped in or out. By contrast, the *weighted* approach selects profiles using probabilities based on their effect on the expected cost of $S$. More specifically, a profile $x$ is weighted by the contribution to $\mathbb{E}[\mathrm{cost}(S)]$ of each task $t$ covered by $x$:

$$weight_x = \sum_{\substack{t \in tasks \\ x \text{ covers } t}} (\mathbb{E}[\mathrm{actual\_start\_time}_t(S)] - \mathrm{earliest\_start\_time}_t) \tag{3}$$

An estimate for $\mathbb{E}[\mathrm{actual\_start\_time}_t(S)]$ is a byproduct of the simulation-based evaluation[1], which will be discussed in Section 5.2. The probability $Prob^+_x$ for selecting profile $x$ is its normalized weight:

$$Prob^+_x = \frac{weight_x}{\sum\limits_{y \in \mathrm{profiles}} weight_y} \tag{4}$$

This ensures a higher probability for including profiles that are capable of reducing the expected cost the most. Similarly, the probability $Prob^-_x$ of swapping out a profile $x$ also

[1]The simulation-based evaluation of $S$ simulates one or more scenarios in which the tasks get executed at a specific time. The expected actual start time is the average actual start time of the task over all scenarios.

considers the contribution of each profile ($weight_x$):

$$Prob^-_x = \frac{(maxW - weight_x)\,[\![active_x]\!]}{\sum\limits_{y \in \text{profiles}} (maxW - weight_y)\,[\![active_y]\!]} \tag{5}$$

where $maxW = \max_{i \in profiles}(weight_i)$ and $[\![active_i]\!]$ is an Iverson bracket evaluating to 1 iff $i$ is an active profile, and 0 otherwise. Note that the weighted approach essentially behaves as a feedback loop, steering the neighborhood search in a more informed direction. We compare the performance of the uniform and weighted approaches in Section 6.3.3.

The ILS algorithm restarts from a new initial solution (line 10) when the restart criterion is met (line 4). This repeats until a termination criterion is reached (line 3), at which point the best-known solution $S^*$ is returned. As a restart criterion, we use a threshold on the number of non-improving consecutive iterations. As a termination criterion, we limit the number of restarts. Suitable values for these two parameters, which typically represent a trade-off between solution quality and run time, are determined in Section 6.3.3.

**Algorithm 1:** Iterated local search

```
1  $S \leftarrow$ initialize_solution();
2  $S^* \leftarrow S$;
3  while termination criterion is not met do
4      while restart criterion is not met do
5          $S' \leftarrow$ neighbor($S$);
6          if expected_cost($S'$) < expected_cost($S$) then
7              $S \leftarrow S'$;
8              if expected_cost($S$) < expected_cost($S^*$) then
9                  $S^* \leftarrow S$;
10     $S \leftarrow$ initialize_solution();
11 return $S^*$;
```

## 5.2. Simulation-based evaluation

A candidate solution $S$ is evaluated based on the expected total time tasks spend in the task pool $\mathbb{E}[\text{cost}(S)]$. Algorithm 2 outlines how $\mathbb{E}[\text{cost}(S)]$ is computed for a solution $S$. Given a scenario where individual task durations are known, the total task pool time of that scenario can be calculated by simulating the WMS's system-directed task dispatching (line 5). This is implemented as a discrete-event simulation (Fishman, 2013) where the system state is only updated when an employee becomes available or when a task is released into the task pool. When an employee becomes available, a dispatching rule matches and assigns them to a task by comparing their profile against the newly available task or those currently in the task pool. If no match is found, the employee remains idle. Once all tasks have been assigned, the total time spent by tasks waiting in the task pool can be computed.

In our case, the individual task durations are unknown in advance and we instead assume an estimated distribution over the possible durations is available. We will describe two ways in which these duration distributions can be used for estimating the expected solution cost. An empirical comparison of both methods will be performed later in Section 6.

The first method for estimating the expected solution cost samples different scenarios and considers the average task pool time over those scenarios as the expected cost. Each scenario is sampled by sampling individual task durations (line 2). $N$ scenarios are sampled by executing Algorithm 2 $N$ times and the cost estimate is the average of the sampled scenarios: $\frac{1}{N}\sum_{i=1}^{N}\text{cost}_i$. While this converges to the expected cost, many samples may be required to do so.

Alternatively, the second method estimates the expected cost by considering only a single scenario in which each task has an expected (mean) duration (line 4) and then calculating this single scenario's cost. If the task pool time for a certain solution can be written as a weighted sum over the task durations, then this converges to the expected cost. However, it is unclear whether or not this condition holds, due to the complex interactions between employees and tasks encoded by the WMS's dispatching algorithm. Even if the condition does not hold exactly, a good approximation for the expected cost might still be calculated through such a sum. This approximation may be preferable over the sampling approach given how much quicker it is.

**Algorithm 2:** `cost`($S$): Cost calculation for a solution, for one scenario

```
Input: $S$: candidate solution
Output: cost(S) of a scenario
1 if Option 1: sample task durations then
2     $\mathbf{d} \leftarrow$ Sample task duration for each task in $T$ ;
3 if Option 2: mean task durations then
4     $\mathbf{d} \leftarrow$ Mean task duration for each task in $T$ ;
5 $s \leftarrow$ Simulate a scenario with $S$ and $\mathbf{d}$ (Discrete-event simulation) ;
6 cost $\leftarrow$ Calculate cost of scenario $s$ (Equation (1));
7 return cost
```

The run time complexity of our discrete-event simulation comprises of two parts: initialising the data structure that stores which profiles cover what tasks, and the simulation of a scenario itself. The first part can be reused across scenarios and simulations and therefore only needs to be executed once. Denoting the number of tasks and profiles with $T$ and $P$, respectively, this first part requires verifying for each task and profile whether the profile rules cover the task: $\mathcal{O}(T \times P \times max\_profile\_length)$ comparison operations. Assuming that most tasks have fewer precedence constraints ($|C|$) than there are tasks ($|C| << T$), the complexity of simulating a single scenario is primarily dictated by $\mathcal{O}(T \times P \times log_2(T))$. This originates from updating a data structure that tracks for each profile the released but still unassigned tasks that are covered by it. For a more detailed explanation of the run time complexity analysis we refer interested readers to Appendix A.

### *5.3. Task duration distributions*

The duration of each task, modeled as a stochastic variable, is estimated based on historical data. Rather than using a single distribution to approximate the duration of all tasks, we propose to exploit task attributes such as their type and work zone to obtain more informed distributions. For example, Figure 3 illustrates how tasks of different types exhibit different distributions.

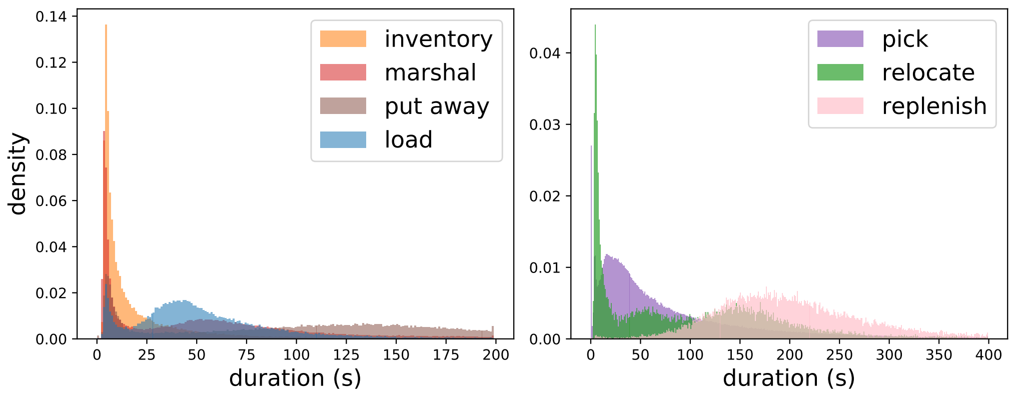


Figure 3: The number of tasks (y-axis) that had a certain duration (x-axis).

Partitioning tasks based on their attribute values decreases the number of tasks that can be used to approximate each individual distribution and can lead to overfitting. Consequently, it is important to carefully decide which attributes to consider and which to ignore. To do so, we use CaDET, a tool that performs conditional density estimation using decision trees (Cousins and Riondato, 2019). CaDET constructs a decision tree by iteratively splitting tasks based on their attribute values, forming conditional probability densities in the leaf nodes. To incentivize splits that improve predictive accuracy, CaDET employs empirical cross entropy as the impurity criterion for tree growth. This approach yields interpretable results and allows us to determine the conditional task duration distributions by traversing the learned decision tree once per task. Once the distributions are known, we can sample from them to perform simulations. For more details concerning the learning process and usage of CaDET, we refer to Cousins and Riondato (2019). If there is sufficient data available in each leaf node of the learned decision tree, then it also possible to sample directly from the leaf's histogram rather than from a fitted distribution. Alternatively, the sampling method can be replaced by using the mean task durations (Algorithm 2, line 4). In this case, a distribution is no longer required and simpler learning methods such as decision tree regression may be sufficient.

## 6. Computational study

Before assessing the quality of the proposed framework, the choices specific to its three main components will be analyzed. The outcome of this analysis will determine the final design decisions and hyperparameter values. For learning the task duration distributions, we study the importance of *(Q1)* distribution types and *(Q2)* conditioning on the attributes. For the simulation-based evaluation, which aims to estimate solution costs, we study the effect on the cost estimate quality of *(Q3)* the conditional duration distributions and *(Q4)* the number of samples drawn from the distributions. Furthermore, we analyze *(Q5)* whether employing mean durations yields sufficiently good estimates. For iterated local search, which aims to find the best solution, we evaluate the effect on the solution quality/computation time trade-off of *(Q6)* our proposed neighborhood operator, and *(Q7)* the restart and termination criteria. The complete method is evaluated by *(Q8)* an ablation study and *(Q9)* a comparison against the manually constructed solutions used in practice.

## *6.1. Data*

We learn the duration distributions and evaluate our approach using historical data obtained from our industry partner, Objective International. This data includes information of historical task executions: actual release time, start time, finish time, date of planned execution, task type, work zone, whether it was bulk or pallet, and whether it involved more than one item. We also obtained data concerning the different profiles used during the task execution period.

*Learning Task Duration Distributions.* When learning the duration distribution for each task, we used a representative time span with a reasonable number of tasks each day, filtering out tasks that were most likely registered incorrectly (performed unreasonably fast or slow[2]). This yielded a total of 559311 tasks spanning 16 months, involving 37 work zones and 7 task types. 5-fold cross validation of the data is used to properly evaluate the learning approach.

*Optimization.* When using this data set to evaluate our optimization approach, we ignored tasks that did not start on the same day as they were released or scheduled to be released. This ensures that possible registration errors in the data do not affect our experiments. The evaluation of our optimization approach focuses on ten problem instances, each corresponding to a single working day. Five of these instances are used as a validation set to tune the parameters, while the other five are used as a test set for the final method. For the final experiment, where we compare against solutions reconstructed from practice, we extended the test set to 100 problem instances to obtain a more representative view of the overall performance. Figure 4 illustrates the distribution of work zone and task type across the ten instances. Appendix B includes Figure B.12, which provides the distributions for each of the ten instances separately. Table 3 provides additional details.

The data set does not include the precedence constraints, with these instead being extracted from the task data set using the following logic. When a task $t$ is released into the task pool at the same time task $t'$ finishes, rather than being released at the time $t$ was actually scheduled to be released, we consider there to be a precedence constraint between $t$ and $t'$.

In addition to the tasks, the data set also describes the available profiles. There are 76 possible profiles from which a subset must be chosen. The number of profile rules within a single profile ranges from 1 to 48, with an average of six rules per profile. Figure 5 shows the frequency with which each task type and work zone features in a profile.

[2]After data analysis and a discussion with our industry partner, we decided to use data concerning tasks whose duration ranged from twenty seconds to one hour.

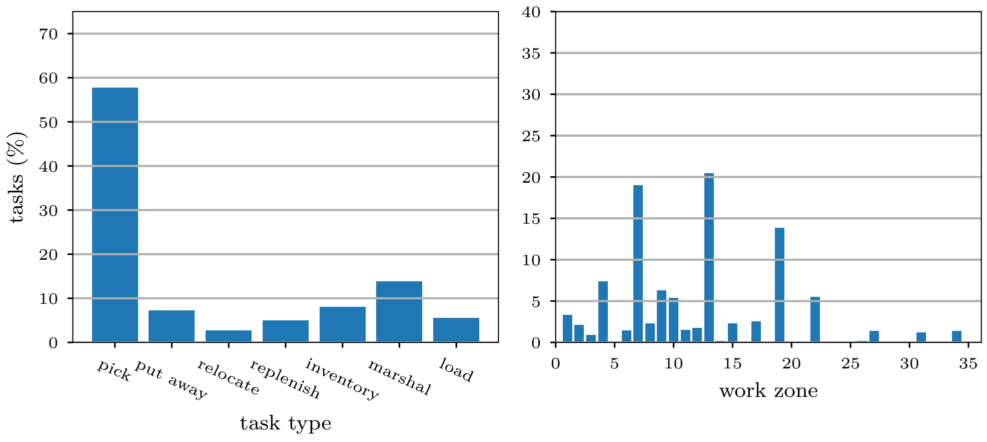


Figure 4: Distribution of task type and work zone across the ten instances.

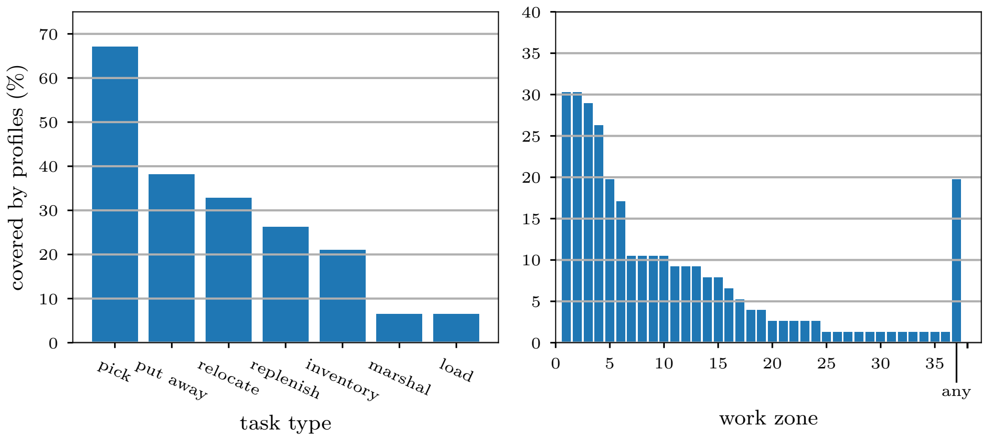


Figure 5: Task type and work zone coverage by the available profiles.

Table 3: Details of the validation set (instances 1 to 5) and test set (instances 6 to 10).

| Instance | Number of tasks | Number of employees | Number of task types | Number of work zones | Number of precedences |
|---|---|---|---|---|---|
| 1 | 1885 | 17 | 6 | 18 | 10 |
| 2 | 1969 | 20 | 7 | 18 | 49 |
| 3 | 2242 | 18 | 7 | 20 | 49 |
| 4 | 1601 | 22 | 7 | 21 | 18 |
| 5 | 1060 | 18 | 7 | 18 | 29 |
| 6 | 1306 | 16 | 7 | 17 | 37 |
| 7 | 1700 | 19 | 7 | 21 | 30 |
| 8 | 1032 | 20 | 7 | 21 | 37 |
| 9 | 1223 | 15 | 7 | 16 | 39 |
| 10 | 2141 | 19 | 7 | 18 | 49 |

Even when the tasks are covered well by the profiles, Figures 4 and 5 do not necessarily match since a profile covers multiple tasks through its profile rules and therefore may be assigned to multiple employees. Regardless, we can clearly see that picking tasks are the most common activity and are covered well, appearing in almost 70% of the profiles. Coverage of the various zones is less high, with the best-covered zones appearing in only 30% of the profiles. Note that the *any* zone appears in $\sim$20% of the profiles. This zone is typically used as part of a final catch-all profile rule to ensure employees do not become idle. Similarly, inventory tasks are also often deployed as a final catch-all in profiles. This explain why it represents less than 10% of the tasks while featuring in more than 20% of profiles.

## *6.2. Experimental Setup*

Unless explicitly mentioned otherwise, the following settings are used for all experiments. The ILS restart parameter is set to 200 non-improving consecutive iterations and a maximum of 17 restarts is used as the termination criterion. The simulation-based evaluation uses the task duration distributions that were learned using CADET version 1.1, with the minimum number of tasks in each leaf node set to 1000 and inverse gamma as the distribution class.

The quality of profile assignments is quantified by their *average relative regret.* The *regret* of a solution $S$ is defined as the difference between the $\mathbb{E}[\text{cost}(S)]$ of that solution and the $\mathbb{E}[\text{cost}(S^*)]$ of the optimal solution $S^*$. To average over multiple problem instances, we consider the *relative regret*: $\frac{|\mathbb{E}[\text{cost}(S)]-\mathbb{E}[\text{cost}(S^*)]|}{\mathbb{E}[\text{cost}(S^*)]}$, the regret divided by the cost of the optimal solution. Given that we do not know the optimal solution for an instance, we instead use the best solution found for it using any of the parameter values. Due to the stochastic nature of our approach, caused by for example the `neighbor`$(S)$ operator, we evaluate each combination of parameter values five times and average the results to obtain the average relative regret for a problem instance. To calculate the expected cost of a solution, we use our proposed evaluation module and sample 10000 scenarios so that the cost likely converges to its true value. In addition to solution quality, the time it took to find the solution is also measured.

Different metrics are used to evaluate the individual components of our approach. The goal of task duration distribution learning is to find distributions that fit the data as well as possible. This is quantified by the *likelihood* of those distributions generating the data $\mathcal{L}(model|data) = \Pr_{model}(data)$. The goal of the evaluation module is to estimate the expected cost $\mathbb{E}[\text{cost}(S)]$. The quality of the estimate $\mathbb{E}[\widehat{\text{cost}(S)}]$ is measured as its *relative error* with respect to the true expected cost: $\frac{\mathbb{E}[\widehat{\text{cost}(S)}]-\mathbb{E}[\text{cost}(S)]}{\mathbb{E}[\text{cost}(S)]}$. The true expected cost is again calculated using our evaluation module by sampling 10000 scenarios. The goal of the ILS component is also evaluated using average relative regret and total computation time.

Each experiment involving the ILS algorithm was repeated five times with different seed values for the random number generator. The reported values are the averages of these five runs. The only exception is the final experiment, which evaluates the performance of our approach by comparing the optimized solutions against those reconstructed from practice. This experiment was repeated 100 times to provide more accurate final conclusions concerning the algorithm's performance.

All experiments were conducted on Ubuntu 20.04.1 LTS using an Intel Xeon E5-2670 processor with 128GB RAM. The algorithms were implemented in Java 11.

### *6.3. Study of Design Decisions and Parameter Tuning*

In this section, we study design decisions and parameter tuning for each of the components of our approach illustrated in Figure 2.

#### *6.3.1. Task Duration Distribution Learning*

CaDET version 1.1 is used to learn task duration distributions and considers the following four task attributes: type, work zone, whether the task is bulk or pallet, and whether the task concerns a single item or multiple. In a CaDET decision tree, each leaf node is associated with tasks that share one or more attributes, for example the same work zone. The distribution associated with each leaf node is fitted over the durations of all its tasks.

*Q1: What distribution class best fits the historical data?.* The class of this distribution is a parameter that must be chosen before learning the tree. We compared several distribution families available in CaDET: inverse gamma, gamma, exponential, inverse Gaussian, log Gaussian, and Gaussian. The resulting scores are shown in Figure 6. The decision tree with inverse gamma distributions yielded the best results and is therefore used in later experiments to determine the duration distribution for each task.

*Q2: Does conditioning on the task attribute values result in better duration estimates?.* Rather than using a decision tree to select attribute-conditional distributions, a single distribution could have been fitted to the data and used for all tasks. However, this would yield a less informed distribution, as confirmed by the results shown in Figure 6 where each conditional approach always scores better than its unconditional single distribution counterpart. Figure 7 depicts two leaf nodes of the learned CaDET decision tree, showing their inverse gamma distribution (conditional), the inverse gamma distribution learned over all data (unconditional), and a histogram of the actual data in that leaf node. This figure demonstrates that using a conditional distribution learned

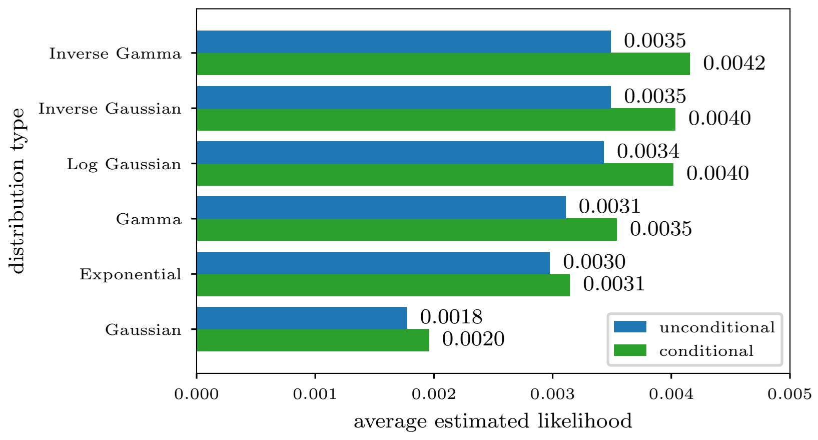


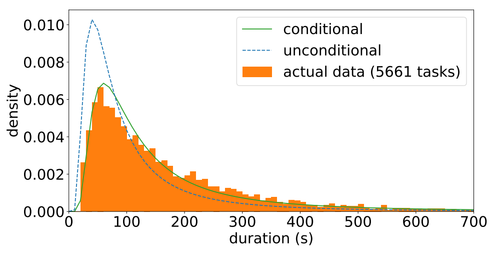
Figure 6: The score of each distribution class, fitted over all tasks (unconditional) or using CaDET (conditional). The reported score is the likelihood of the learned distribution generating the data, averaged over all instances and estimated using all five folds. Higher is better.

with CaDET provides better estimates than an unconditional distribution. This is expected as each conditional distribution is fitted to the actual data, whereas the single distribution fits over all data and is consequently less informed.

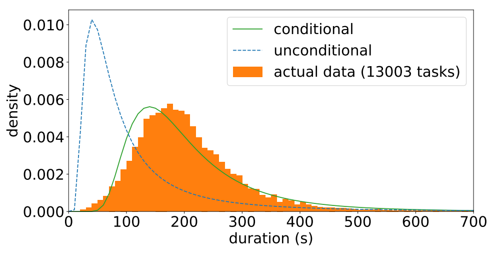
*Conclusion.* Conditional distributions using inverse gamma distributions best capture the task durations.

### *6.3.2. Simulation-Based Evaluation*

We generated 5000 random solutions and estimated their expected cost on the validation set in two ways: 1) with a single scenario using the mean task durations of their distribution and 2) with $N$ sampled scenarios using task durations sampled from their distributions, where the number of scenarios is varied: $N \in \{1, 50, 100, 150, 200, 250\}$.



(a) Picking tasks in work zone 33702



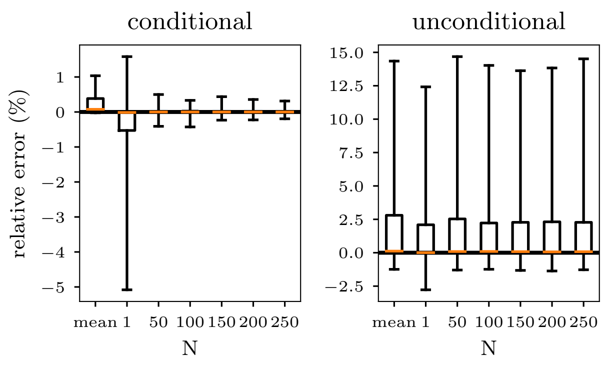
(b) Replenish tasks in work zones other than work zones 118616 and 118669

Figure 7: The density of task durations for two attribute-conditional task groups according to the actual data, the attribute-conditional approach (CaDET) and the unconditional approach. The more similar the learned distribution is to the histogram of actual data, the better.

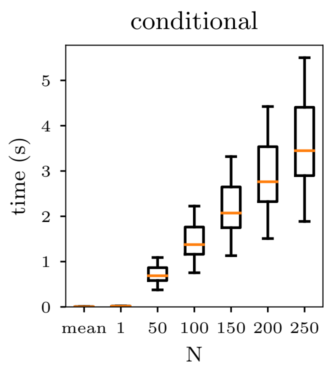
Each of these settings is executed twice: once using the unconditional task duration distributions and once using the conditional task duration distributions.

*Q3: Does using the conditional distribution result in better cost estimates?.* Figure 8a shows that expected cost estimates produced using conditional distributions generally have a smaller relative error. They are therefore more accurate than estimates produced using unconditional distributions, both when the mean or samples are used.

*Q4: How does the estimate improve in function of the number of samples?.* When sampling from the unconditional distributions, using more samples does not yield more accurate results. Indeed, Figure 8a shows it has little effect. This is because the unconditional distributions from which we sample do not sufficiently match the conditional distributions used to approximate the true distributions. By contrast, and as one would expect, using more samples from the conditional distributions does reduce the variance in relative error, thereby yielding estimates closer to the true expected cost. However, as Figure 8b shows, this also significantly lengthens the simulation run time.



(a) Relative error while varying $N$



(b) Simulation time while varying $N$

Figure 8: Results of evaluating 5000 random solutions, using either the mean duration or various values of $N$. The whiskers denote 5% to 95% of the data. The simulation time using unconditional distributions is similar to using conditional distributions and therefore we only show results of the conditional.

*Q5: Does mean task duration yield a sufficient estimate of the solution cost?.* Increasing the number of samples $N$ does not improve cost estimates when those samples originate from the unconditional distributions. We will therefore only consider samples from the conditional distributions in order to address this research question. Figure 8a shows that using 250 samples often yields better estimates than using mean values. However, the estimates produced using the mean values do have an acceptable relative error. For all but one of the 5000 data points the relative error was within the range $[-1\%, 1\%]$, while for 97.4% of the data points the error was within the range $[-0.5\%, 0.5\%]$. As the ablation study in Section 6.4.1 will later show, estimating the solution cost using mean task durations provides a sufficient signal to steer ILS in the right direction. Furthermore, there is also a time trade-off when using more samples because 250 samples means executing 250 scenarios. This is reflected in the algorithm's run time, which is 4.3ms on average when using the mean values and 3628.7ms when using 250 samples, an increase of almost three orders of magnitude. This increase is not $\times 250$ due the cost of sampling from an inverse gamma distribution. However, we did observe the expected increase of approximately $\times 250$ when comparing to the use of a single sample rather than the mean values. Given that the available computation time is typically limited and that the relative error is small, we propose to use the mean values to evaluate a solution rather than simulating $N$ scenarios.

*Conclusion.* The simulation-based evaluation should use conditional task duration distributions and a single scenario where the task durations are the mean of their distributions. Sampling scenarios can provide more accurate estimates of $\mathbb{E}[\text{cost}(S)]$, however this gain in accuracy is limited compared to the substantial increase in computational run time.

### *6.3.3. Iterated Local Search*

The goal of ILS is to quickly find a high-quality solution. Several design decisions influence its performance: 1) the used neighborhood operator, 2) the restart criterion, and 3) the termination criterion. These three choices influence both the final solution quality as well as the time taken to find it. For the restart and termination criterion there is a natural trade-off in effect, as solution quality typically improves if one spends more time searching.

*Q6: Which neighborhood operator performs best?.* To compare the performance of the neighborhood operators, ILS was ran twice for each instance in the validation set: once using the uniform operator and once using the weighted operator, without restarts and beginning from the same initial solution. This experiment was repeated five times, tracking the relative regret over time for each run. The relative regret at any point in time is based on the expected cost of the best solution found up until that point in the run and the optimal expected cost for the problem instance. This optimal expected cost is based on the best solution eventually found across all five attempts while using either operator. Both the expected cost and the optimal expected cost were estimated using a simulation with mean-conditional based task durations. The results from the first validation instance are shown in Figure 9, while the results for the other instances are included in Appendix C (Figure C.17). The results indicate that while both operators yield similar results given enough time, the weighted operator converges to better solutions much quicker than the uniform operator does. The uniform operator also exhibits more variance between the five runs than the weighted operator. This is explained by the stochastic nature of the uniform distribution used by the uniform operator, whereas with the weighted operator the distribution is more focused on changes that improve the solution.

*Q7: Which restart and termination criteria gives the best quality-time trade-off?.* We tested several combinations of restart and termination criterion values, each time recording the total ILS run time and the estimated expected cost of the final solution. We performed this five times for each of the five validation instances. This resulted in 25 data points for each combination of parameter values, which we evaluated using the relative regret metric. The optimal cost used to compute this metric was based on the

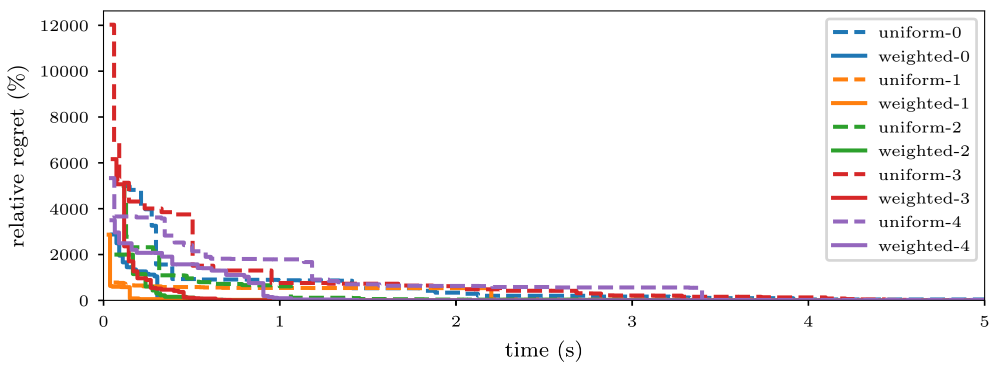


Figure 9: The relative regret over time during ILS for the first five seconds for both the uniform and weighted `neighbor` operator, for five different starting solutions.

best solution found for each instance across all attempts. The heat map in Figure 10 shows the relative regret averaged out over all instances and all five attempts for each parameter value combination. A heat map of each individual instance is included in Appendix C (Figure C.13 - C.15). When tuning the parameter values, it is important to also consider the run time. On average, ILS run time scales linearly with the number of restarts because the restart execution times are independent of each another. It is therefore sufficient to only vary the restart criterion value when analyzing run time. Figure 11 shows the results for the first validation instance, using $5 \times 25$ data points for each restart criterion value. The run time behavior of the other instances is similar and is documented in Appendix C (Figure C.16). Increasing the maximum number of consecutive non-improving iterations beyond 300 did not significantly improve the results. Furthermore, when multiple restarts were used, 100 iterations was sufficient to produce results with an average relative regret under 1%. If we consider all instances separately (Appendix C), this became 200 iterations. When using 200 iterations, at least 15 restarts are required to obtain a relative regret under 1%. To prevent overfitting, we opted for 17 restarts. This yielded an average relative error of at most 0.5%.

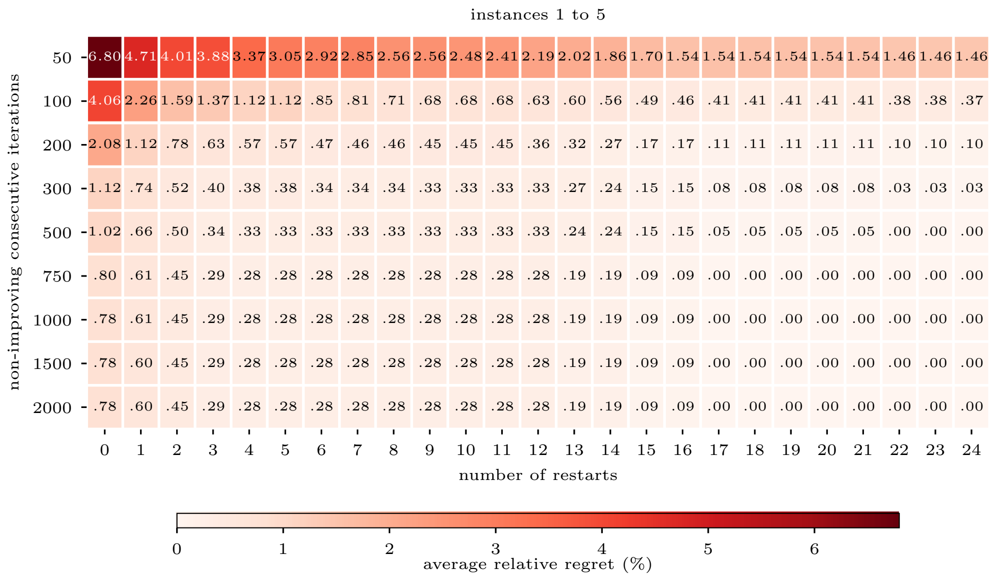
*Conclusion.* ILS should use the weighted neighborhood operator to find good solutions faster. The optimal restart and termination criteria depend on available time for running the simulation. If there is no real time limit, we propose a restart criterion value of 200 consecutive non-improving iterations and 17 restarts, as searching for a longer time and with more restarts is unlikely to produce significantly better solutions.

## *6.4. Evaluation of the Proposed Framework*

To evaluate our proposed framework as a whole, an ablation study is performed to study the contribution of each design decision. Thereafter, a comparison against the manually constructed solutions that were used in practice is conducted.

### *6.4.1. Ablation study*

*Q8: How much do each of the proposed design decisions contribute to the quality of the final solution and the total run time?.* To study the contribution of each design decision,



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 6.80 | 4.71 | 4.01 | 3.88 | 3.37 | 3.05 | 2.92 | 2.85 | 2.56 | 2.56 | 2.48 | 2.41 | 2.19 | 2.02 | 1.86 | 1.70 | 1.54 | 1.54 | 1.54 | 1.54 | 1.54 | 1.54 | 1.46 | 1.46 | 1.46 |
| 100 | 4.06 | 2.26 | 1.59 | 1.37 | 1.12 | 1.12 | .85 | .81 | .71 | .68 | .68 | .68 | .63 | .60 | .56 | .49 | .46 | .41 | .41 | .41 | .41 | .41 | .38 | .38 | .37 |
| 200 | 2.08 | 1.12 | .78 | .63 | .57 | .57 | .47 | .46 | .46 | .45 | .45 | .45 | .36 | .32 | .27 | .17 | .17 | .11 | .11 | .11 | .11 | .11 | .10 | .10 | .10 |
| 300 | 1.12 | .74 | .52 | .40 | .38 | .38 | .34 | .34 | .34 | .33 | .33 | .33 | .33 | .27 | .24 | .15 | .15 | .08 | .08 | .08 | .08 | .08 | .03 | .03 | .03 |
| 500 | 1.02 | .66 | .50 | .34 | .33 | .33 | .33 | .33 | .33 | .33 | .33 | .33 | .33 | .24 | .24 | .15 | .15 | .05 | .05 | .05 | .05 | .05 | .00 | .00 | .00 |
| 750 | .80 | .61 | .45 | .29 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .19 | .19 | .09 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1000 | .78 | .61 | .45 | .29 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .19 | .19 | .09 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1500 | .78 | .60 | .45 | .29 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .19 | .19 | .09 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 2000 | .78 | .60 | .45 | .29 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .28 | .19 | .19 | .09 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |

Figure 10: The relative regret of solutions found using ILS for different restart and termination criterion values. The relative regret was averaged out over all five validation instances, over all five attempts. This answers the question: “if we had used a certain restart and stop criterion, how much worse (%) on average would the found solution be compared to the best solution?”. Higher values are worse.

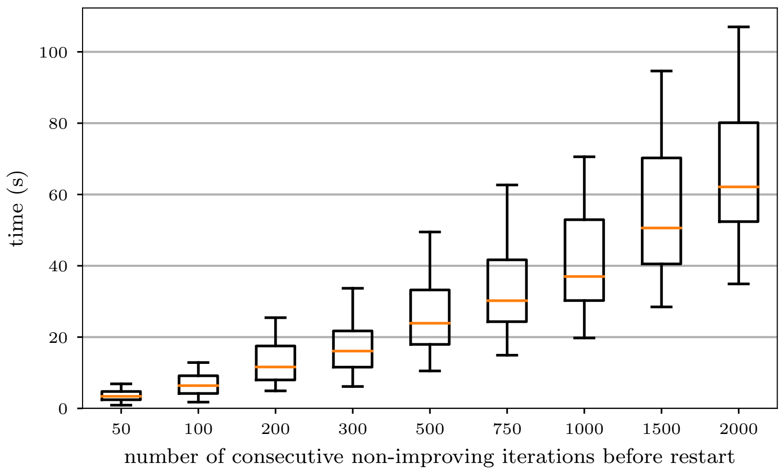


Figure 11: The execution time of a single ILS restart on the first validation instance for various restart criteria values. The whiskers denote 5% to 95% of the data.

we ran the proposed framework with an alternative decision in order to observe the deterioration in solution quality. We ran each configuration five times for all test set instances and report the resulting average relative regret and the average run time in Table 4. These results show that the proposed method's approach of using a conditional distribution as a source for the task durations is preferable to using a non-conditional distribution. The difference with respect to average relative regret is especially significant for problem instances 9 and 10. The results also clearly show that sampling does not guarantee much better results, but does require significantly more run time. The uniform neighbor operator yields solutions of similar quality compared to the proposed weighted operator, but requires slightly more time to find those solutions.

Table 4: Ablation study: average relative regret and run time of the proposed method, compared against the following alternative versions of the proposed method: 1) Using non-conditional distributions instead of conditional ones for the task durations, 2) sampling 150 scenarios, 3) sampling 250 scenarios, and 4) using the uniform neighborhood operator instead of the weighted one.

| Instance | Proposed Method | Non-conditional | Sampling-150 | Sampling-250 | Uniform operator |
|---|---|---|---|---|---|
| | Average Relative Regret (%) | | | | |
| 6 | 2.31 | 7.28 | 2.25 | 2.46 | **1.81** |
| 7 | **0.24** | 0.63 | 0.33 | 0.32 | **0.24** |
| 8 | 0.26 | 1.09 | 0.36 | **0.22** | 0.56 |
| 9 | **0.09** | 15.45 | 0.26 | 0.37 | 0.10 |
| 10 | 2.89 | 27.22 | 3.54 | 3.73 | **2.86** |
| | Average Run time (s) | | | | |
| 6 | 145 | **97** | 25174 | 42759 | 184 |
| 7 | 213 | **167** | 46608 | 75225 | 244 |
| 8 | 68 | **61** | 20276 | 32760 | 82 |
| 9 | 71 | **60** | 18180 | 30903 | 90 |
| 10 | 255 | **213** | 42168 | 79609 | 330 |

### *6.4.2. Comparison with practice*

*Q9: Does the approach proposed in this paper find better profile assignments than those used in practice?.* The historical data provided to us by our industry partner does not include the profile used by each employee at every point in time. The data only contains the set of used profiles, not when they were used. Therefore, in order to compare the profile assignments produced by our approach against those used in practice, we reconstructed the profile assignments for all test set instances. This reconstruction process extracts the profile assignment possibly used on a specific day by considering all task assignments of each employee on that day. In practice, profile assignments are often updated throughout the day and, as a result, no single matching profile could be found by our reconstructive procedure in some cases. When this happened, an existing profile in the set was extended with a single profile rule which covered the unmatched task. Table 5 contains the percentage of profiles that needed to be updated in this manner for each of the days we evaluated. For 59 of the 100 days, all profiles needed to be updated. This mismatch can be the result of unknown precedence constraints, profile assignment changes during operations or data inconsistencies. Note that while the reconstructed profile assignments may contain adapted profiles, our approach only used the initially provided profiles and excluded the adapted ones.

The profile assignments, both those derived from historical data and the framework's solutions, were evaluated using the simulation procedure described in Section 5.2. During this simulation the actual duration of each task as recorded in the historical data was used, allowing us to determine whether our solution would have outperformed the profile assignment that was used in practice. In this way, only a single scenario needed to be solved to obtain the cost. Naturally, our approach did not use the actual task duration to find a solution, as these durations are unknown in advance. Instead we used the mean of the conditional distributions previously learned in Section 6.3.1.

Since our approach uses mean values rather than sampling, and is therefore relatively fast, we extended the test set with 95 randomly selected instances in which at least 1000 tasks were executed (excluding instances from the validation set). This brought the test set size to 100 unique instances, thereby providing a better indication of our approach's performance. We ran our approach 100 times on each of the instances to account for randomness in the ILS algorithm.

Table 5 reports the results and compares the solutions obtained by our framework against those reconstructed from historical data. For 92 of the 100 instances, the average relative regret of the profile assignment found our approach was better than that of the assignment reconstructed from practice. In most cases this difference was very substantial. Furthermore, the computation time required was relatively minimal and in line with previous experiments. The longest average run time was only 404.4 seconds. This means that our approach is not only suitable for producing profile assignments a day in advance, but it can also be applied dynamically in response to changing circumstances during the day.

*Conclusion.* The approach proposed in this paper is able to quickly find profile assignments that are generally significantly better than those reconstructed from practice.

Table 5: Comparison against solutions used in practice for ten randomly selected instances. A comparison of all 100 instances (instances 6 − 105) can be found in Appendix C (Table C.6). RR is the relative regret evaluation metric, with the optimal cost being the best cost found across all runs of the algorithm or the solution used in practice. The lower the RR, the better. The average RR and run time of all 100 algorithm runs are reported, as well as the standard deviation. The column "new profiles" indicates the percentage of employees whose profile required adaptation in order to correctly simulate practice.

| Instance | Proposed approach | | Manual solutions | |
|---|---|---|---|---|
| | avg. RR (%) | avg. Time (s) | RR (%) | New profiles (%) |
| 14 | **0.4** (± 0.3) | 64.1 (± 5.9) | 18.1 | 100 |
| 21 | **0.0** (± 0.0) | 76.6 (± 6.6) | 39.5 | 100 |
| 23 | **0.3** (± 0.3) | 239.6 (± 27.9) | 100.8 | 86 |
| 38 | **1.5** (± 1.4) | 314.9 (± 36.6) | 41.1 | 100 |
| 63 | **2.1** (± 1.7) | 114.1 (± 11.4) | 27.7 | 100 |
| 66 | **0.0** (± 0.0) | 24.0 (± 2.5) | 143.2 | 83 |
| 69 | **2.4** (± 1.2) | 50.9 (± 6.2) | 67.4 | 100 |
| 78 | **0.7** (± 0.3) | 30.8 (± 3.1) | 39.1 | 68 |
| 89 | **0.8** (± 0.6) | 58.1 (± 6.0) | 37.8 | 100 |
| 103 | **0.5** (± 0.7) | 184.5 (± 21.3) | 18.2 | 74 |

## 7. Conclusions

This paper considered the problem of assigning profiles to warehouse workers, which represents a task faced by shift supervisors on a daily basis. This is a nontrivial yet impactful decision, as profile assignment greatly influences the efficiency of tasks being executed in the warehouse. The decision is further complicated by unknown task durations. Our proposed framework involves a metaheuristic algorithm which uses discrete-event simulation to evaluate the quality of solutions. To accommodate for unknown task durations, historical data was used to estimate task duration distributions, conditional on task attributes. Two approaches were considered: a more precise one that takes the entire distribution into account and a faster one that only considers the expected duration.

A detailed computational study was conducted to evaluate the various components of our proposed method. Predicting task durations based on the task attributes is shown to have a large impact on the quality of the generated profile assignments. Studying the trade-off between more precise and faster duration distribution incorporation led to the conclusion that the fast method, which uses the expected task duration, does not suffer from significantly lower quality and is therefore the preferred method. A comparison with the profile assignments reconstructed from practice revealed that the assignments produced by our method were superior.

In principle, our proposed method is not just restricted to warehouse operations but is also applicable to other domains where profile-based dispatching can be used. We plan to investigate this direction in future work. Another valuable direction for future research concerns the problem's stochastic elements. The problem addressed in this paper considered task duration as the single source of uncertainty. A natural extension would be to also include uncertainty concerning the earliest possible start times of the tasks, as this also represents a source of uncertainty in practice. Finally, as our proposed approach is heuristic in nature, we have no guarantee concerning solution quality. An exact algorithm that manages to find optimal solutions would allows us to better evaluate the performance of our method.

## Author Contributions

**V. Derkinderen**: Conceptualization, Methodology, Software, Data curation, Visualization, Writing. **P. Smet**: Conceptualization, Methodology, Software, Visualization, Writing. **J. Bekker**: Methodology, Supervision, Writing.

## Acknowledgments

This research was supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, and by the Research Foundation Flanders (FWO) under grant S007318N and 1SA5520N. We are grateful for the insightful discussions with Tias Guns, Luc De Raedt and Greet Vanden Berghe. Editorial consultation provided by Luke Connolly (KU Leuven).

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix A. Simulation run time complexity

The run time complexity of the implemented discrete-event simulation comprises of two parts: initialising the data structure that stores which profiles cover what tasks, and the simulation of a scenario itself. The first part is re-usable across simulations and therefore only needs to be performed once. Denoting the number of tasks as $T$ and profiles as $P$, the initialisation requires verifying for each task and profile whether the profile's rules cover the task: $\mathcal{O}(T \times P \times max\_profile\_length)$ comparison operations. The complexity of the simulation itself is less straightforward. A scenario simulation has one *release event* and one *finished event* per task. The insertion of these two events into an ordered timeline data structure happens once per task ($\times T$), with each insertion requiring $\mathcal{O}(log_2(T))$ comparisons. In order to efficiently perform a simulation we also track each profile's list of idle employees as well as the released yet still unassigned tasks covered by that profile. The former data structure is a queue with insertion complexity $\mathcal{O}(1)$ while the latter also requires efficient removal of specific elements and guarantees an insertion and removal complexity of $\mathcal{O}(log_2(T))$ comparisons.

Because each release and finish event occurs once per task, the following complexities must be multiplied by $T$. The release of task $i$ first requires checking its precedence constraints. This involves verifying for each of its precedence tasks whether they are finished: $\mathcal{O}(|C_i|)$ Boolean checks with $|C_i|$ the number of precedence constraints of task $i$. If these constraints are all satisfied, the task is assigned to a suitable idle employee if one is available. Since we track for each profile all idle employees with that profile, this simply means iterating over (relevant) profiles and verifying whether there is any idle employee: $\mathcal{O}(P)$ Boolean checks. If no suitable employee is available, we add the task to the aforementioned data structure that tracks for each profile the released yet still unassigned tasks. This insertion has complexity $\mathcal{O}(P \times log_2(T))$. If a precedence constraint is violated, the task is not released but is instead put on hold. Given that a finished task $i$ can cause the release of another task on hold, we re-verify the precedence constraints of all tasks waiting for $i$. This requires $\mathcal{O}(\sum_{j \text{ waits on } i} |C_j|)$ Boolean checks. A finished task also causes an employee to become available who might match with a released but unassigned task. Removing the newly assigned task from the relevant data structures requires $\mathcal{O}(P \times log_2(T))$ comparisons.

Assuming that tasks have few precedence constraints, $|C_i| << T$, the simulation's complexity is primarily dictated by $\mathcal{O}(T \times P \times log_2(T))$.

## Appendix B. Additional instance analysis

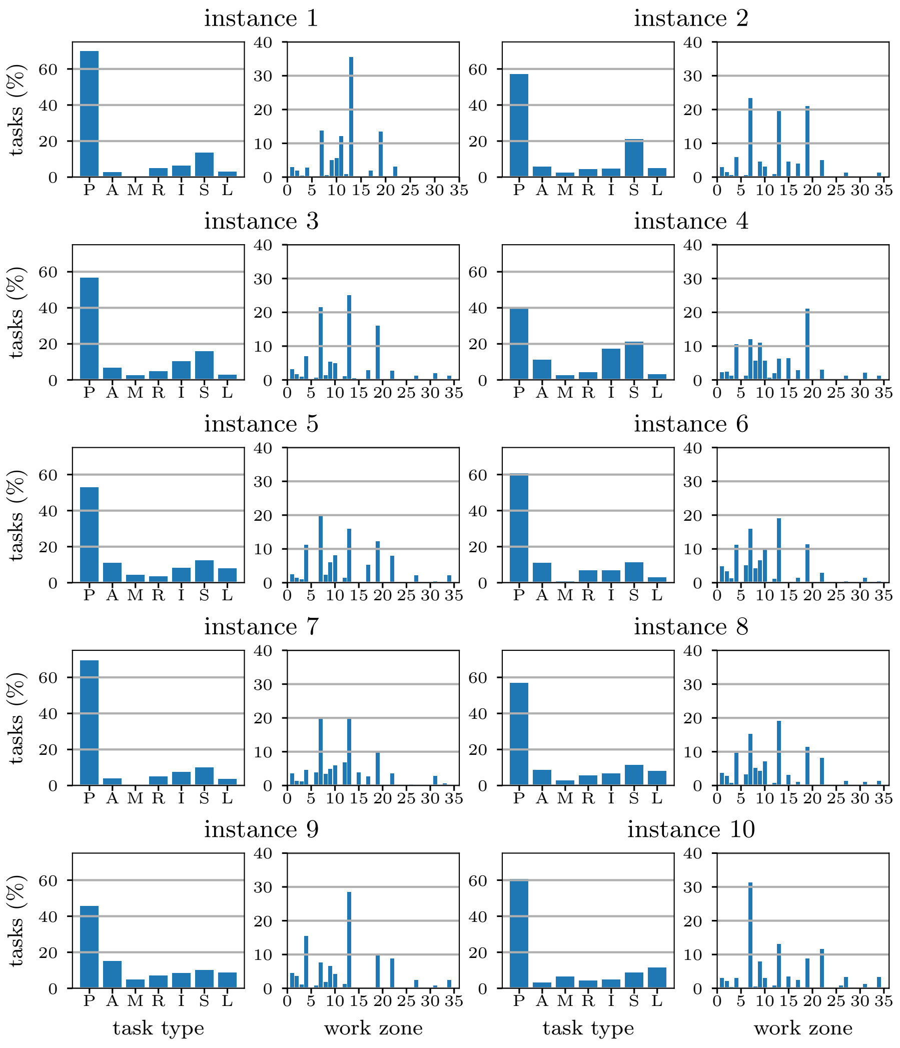


Figure B.12: Distribution of task type and work zone for the tasks in each of the ten problem instances (days). Task types: **p**ick, put **a**way, **m**ove, **r**eplenish, **i**nventory, mar**s**hal, **l**oad.

## Appendix C. Additional experimental analysis

Table C.6: A comparison of the profile assignments used in practice and those of the proposed approach. RR is the relative regret evaluation metric (Section 6.2). The average RR and run time of all hundred algorithm runs are reported, as well as the standard deviation. The column "new profiles" indicates the percentage of employees whose profile required adaptation in order to correctly simulate practice.

| Instance | Proposed approach | | Manual solutions | |
|---|---|---|---|---|
| | avg. RR (%) | avg. Time (s) | RR (%) | New profiles (%) |
| 6 | **2.2** (± 1.1) | 145.0 (± 15.6) | 117.0 | 100 |
| 7 | 1.8 (± 0.4) | 196.2 (± 19.6) | **0.0** | 74 |
| 8 | **0.0** (± 0.0) | 60.7 (± 5.6) | 20.8 | 60 |
| 9 | **0.5** (± 0.2) | 67.3 (± 6.5) | 13.6 | 100 |
| 10 | **0.1** (± 0.2) | 257.1 (± 28.9) | 21.2 | 100 |
| 11 | **1.6** (± 1.9) | 128.2 (± 16.4) | 37.3 | 90 |
| 12 | **1.1** (± 2.4) | 34.7 (± 2.9) | 45.2 | 95 |
| 13 | **0.4** (± 0.2) | 148.0 (± 15.4) | 45.2 | 100 |
| 14 | **0.4** (± 0.3) | 64.1 (± 5.9) | 18.1 | 100 |
| 15 | **0.8** (± 0.8) | 139.7 (± 11.5) | 29.7 | 90 |
| 16 | **6.7** (± 4.2) | 119.3 (± 13.1) | 25.8 | 100 |
| 17 | **0.6** (± 0.3) | 65.8 (± 6.4) | 39.1 | 100 |
| 18 | **1.3** (± 0.2) | 53.3 (± 5.5) | 75.0 | 52 |
| 19 | 3.6 (± 0.9) | 109.3 (± 13.7) | **0.0** | 70 |
| 20 | **3.0** (± 1.1) | 154.6 (± 14.2) | 72.3 | 100 |
| 21 | **0.0** (± 0.0) | 76.6 (± 6.6) | 39.5 | 100 |
| 22 | **0.1** (± 0.1) | 266.4 (± 24.1) | 53.3 | 100 |
| 23 | **0.3** (± 0.3) | 239.6 (± 27.9) | 100.8 | 86 |
| 24 | **0.0** (± 0.0) | 321.5 (± 31.4) | 109.7 | 90 |
| 25 | **0.3** (± 0.1) | 87.1 (± 9.5) | 5.9 | 75 |
| 26 | **0.1** (± 0.1) | 63.7 (± 6.5) | 4.8 | 100 |
| 27 | **1.2** (± 0.6) | 139.0 (± 13.0) | 5.4 | 100 |
| 28 | **0.2** (± 0.1) | 203.3 (± 22.8) | 39.0 | 100 |
| 29 | **6.3** (± 1.9) | 311.7 (± 29.9) | 92.1 | 100 |
| 30 | **0.2** (± 0.1) | 190.7 (± 23.1) | 154.4 | 100 |
| 31 | **5.7** (± 2.3) | 49.2 (± 4.6) | 21.1 | 100 |
| 32 | 1.6 (± 0.9) | 121.4 (± 11.1) | **1.1** | 100 |
| 33 | **0.0** (± 0.0) | 388.4 (± 37.1) | 78.6 | 100 |
| 34 | **0.1** (± 0.0) | 147.5 (± 12.6) | 42.9 | 94 |
| 35 | **3.1** (± 2.5) | 207.1 (± 20.9) | 98.7 | 100 |
| 36 | **1.4** (± 0.4) | 120.2 (± 12.8) | 57.5 | 82 |
| 37 | **0.1** (± 0.0) | 138.7 (± 16.8) | 13.6 | 100 |
| 38 | **1.5** (± 1.4) | 314.9 (± 36.6) | 41.1 | 100 |
| 39 | **0.0** (± 0.0) | 307.2 (± 26.2) | 81.6 | 100 |
| 40 | **0.1** (± 0.0) | 404.4 (± 37.2) | 9.8 | 100 |
| 41 | **0.7** (± 0.3) | 251.0 (± 22.3) | 80.4 | 100 |

*Continued on next page*

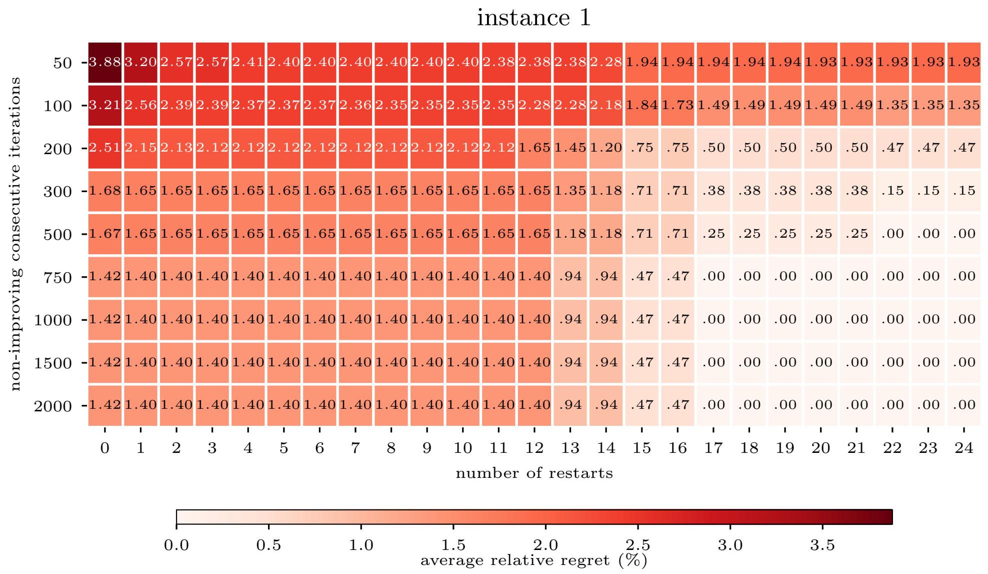
Table C.6 – *Continued from previous page*

| Instance | Proposed approach | | Manual solutions | |
|---|---|---|---|---|
| | avg. RR (%) | avg. Time (s) | RR (%) | New profiles (%) |
| 42 | **1.3** (± 1.2) | 195.3 (± 19.9) | 44.9 | 100 |
| 43 | **0.8** (± 0.8) | 258.2 (± 26.2) | 40.5 | 100 |
| 44 | **0.2** (± 0.1) | 42.5 (± 4.2) | 4.0 | 100 |
| 45 | **0.2** (± 0.2) | 153.3 (± 13.0) | 39.2 | 100 |
| 46 | 12.7 (± 0.8) | 95.4 (± 9.0) | **0.0** | 100 |
| 47 | **6.0** (± 1.1) | 148.7 (± 15.5) | 81.9 | 100 |
| 48 | **1.7** (± 1.0) | 248.7 (± 28.0) | 26.0 | 100 |
| 49 | **0.3** (± 0.5) | 190.6 (± 18.9) | 52.2 | 76 |
| 50 | **0.9** (± 0.4) | 264.5 (± 26.2) | 24.7 | 100 |
| 51 | **0.5** (± 0.2) | 96.3 (± 10.1) | 24.3 | 100 |
| 52 | 23.0 (± 1.4) | 91.4 (± 6.9) | **0.0** | 100 |
| 53 | **2.9** (± 1.8) | 76.7 (± 6.8) | 31.6 | 100 |
| 54 | 3.7 (± 2.3) | 127.2 (± 12.6) | **1.1** | 82 |
| 55 | **4.0** (± 1.4) | 51.3 (± 5.9) | 15.5 | 100 |
| 56 | **3.2** (± 5.0) | 155.6 (± 19.4) | 41.7 | 100 |
| 57 | **0.8** (± 0.5) | 85.3 (± 9.4) | 24.5 | 100 |
| 58 | **0.1** (± 0.1) | 106.4 (± 10.7) | 220.1 | 100 |
| 59 | **0.0** (± 0.0) | 127.1 (± 13.9) | 42.4 | 100 |
| 60 | **4.5** (± 3.1) | 74.9 (± 7.6) | 26.9 | 88 |
| 61 | **1.2** (± 0.4) | 76.5 (± 8.2) | 86.4 | 47 |
| 62 | **0.7** (± 0.3) | 106.9 (± 9.8) | 74.9 | 91 |
| 63 | **2.1** (± 1.7) | 114.1 (± 11.4) | 27.7 | 100 |
| 64 | **5.6** (± 3.7) | 42.5 (± 4.3) | 90.0 | 73 |
| 65 | **0.6** (± 0.3) | 31.6 (± 3.0) | 277.4 | 100 |
| 66 | **0.0** (± 0.0) | 24.0 (± 2.5) | 143.2 | 83 |
| 67 | **1.4** (± 1.3) | 74.2 (± 5.7) | 27.0 | 68 |
| 68 | **8.5** (± 1.3) | 44.4 (± 4.6) | 56.0 | 67 |
| 69 | **2.4** (± 1.2) | 50.9 (± 6.2) | 67.4 | 100 |
| 70 | **5.6** (± 1.0) | 41.8 (± 3.3) | 124.3 | 100 |
| 71 | **2.0** (± 0.6) | 63.6 (± 7.6) | 44.4 | 100 |
| 72 | **2.3** (± 1.5) | 39.7 (± 3.3) | 81.3 | 100 |
| 73 | **0.3** (± 0.3) | 16.8 (± 1.6) | 193.6 | 68 |
| 74 | **0.8** (± 0.1) | 35.7 (± 3.7) | 166.8 | 86 |
| 75 | **0.3** (± 0.3) | 24.0 (± 2.1) | 135.1 | 100 |
| 76 | **0.1** (± 0.0) | 31.6 (± 3.0) | 19.8 | 78 |
| 77 | **0.6** (± 0.3) | 47.8 (± 4.8) | 45.5 | 100 |
| 78 | **0.7** (± 0.3) | 30.8 (± 3.1) | 39.1 | 68 |
| 79 | **2.3** (± 1.4) | 44.6 (± 3.9) | 29.2 | 100 |
| 80 | **5.2** (± 1.7) | 82.4 (± 10.7) | 34.1 | 74 |
| 81 | **2.7** (± 1.2) | 114.7 (± 13.7) | 30.3 | 100 |
| 82 | **0.7** (± 1.3) | 111.6 (± 10.7) | 53.7 | 90 |

*Continued on next page*

Table C.6 – *Continued from previous page*

| Instance | Proposed approach | | Manual solutions | |
|---|---|---|---|---|
| | avg. RR (%) | avg. Time (s) | RR (%) | New profiles (%) |
| 83 | 5.8 (± 2.2) | 49.4 (± 4.7) | **0.0** | 82 |
| 84 | **0.1** (± 0.0) | 72.5 (± 6.6) | 143.1 | 71 |
| 85 | **0.4** (± 0.3) | 84.7 (± 8.4) | 80.0 | 65 |
| 86 | **0.9** (± 0.3) | 67.4 (± 7.5) | 13.6 | 86 |
| 87 | **2.2** (± 1.1) | 257.7 (± 31.8) | 23.6 | 100 |
| 88 | **0.5** (± 0.3) | 238.9 (± 25.6) | 23.6 | 100 |
| 89 | **0.8** (± 0.6) | 58.1 (± 6.0) | 37.8 | 100 |
| 90 | **3.4** (± 1.3) | 23.4 (± 2.6) | 19.4 | 85 |
| 91 | **0.1** (± 0.1) | 22.1 (± 1.9) | 28.8 | 100 |
| 92 | **0.7** (± 0.3) | 68.7 (± 7.2) | 29.2 | 68 |
| 93 | **1.7** (± 0.9) | 67.6 (± 6.7) | 6.7 | 100 |
| 94 | **4.2** (± 1.6) | 116.4 (± 11.4) | 34.9 | 100 |
| 95 | **3.8** (± 1.4) | 55.9 (± 5.5) | 17.2 | 100 |
| 96 | **0.2** (± 0.1) | 109.6 (± 12.7) | 19.4 | 100 |
| 97 | **0.4** (± 0.1) | 241.5 (± 17.1) | 126.5 | 94 |
| 98 | **5.7** (± 1.0) | 109.4 (± 10.8) | 22.9 | 100 |
| 99 | **0.0** (± 0.0) | 310.3 (± 32.2) | 96.7 | 52 |
| 100 | **4.1** (± 2.3) | 50.1 (± 6.2) | 59.2 | 75 |
| 101 | 35.0 (± 0.2) | 62.0 (± 6.0) | **0.0** | 58 |
| 102 | **9.4** (± 2.5) | 230.5 (± 31.5) | 80.8 | 84 |
| 103 | **0.5** (± 0.7) | 184.5 (± 21.3) | 18.2 | 74 |
| 104 | **0.9** (± 0.4) | 52.1 (± 5.6) | 13.4 | 81 |
| 105 | **1.9** (± 0.6) | 66.9 (± 5.9) | 26.0 | 95 |



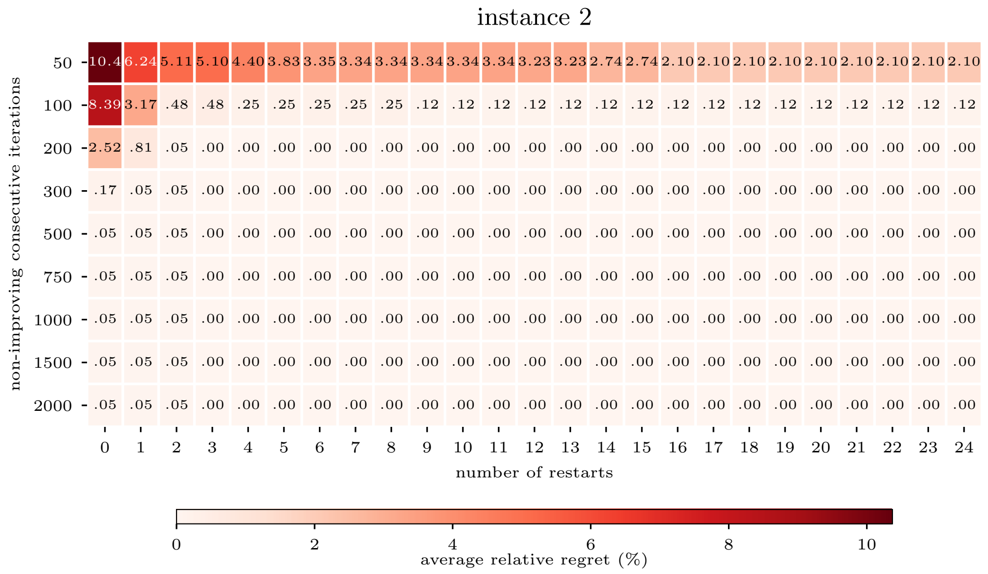
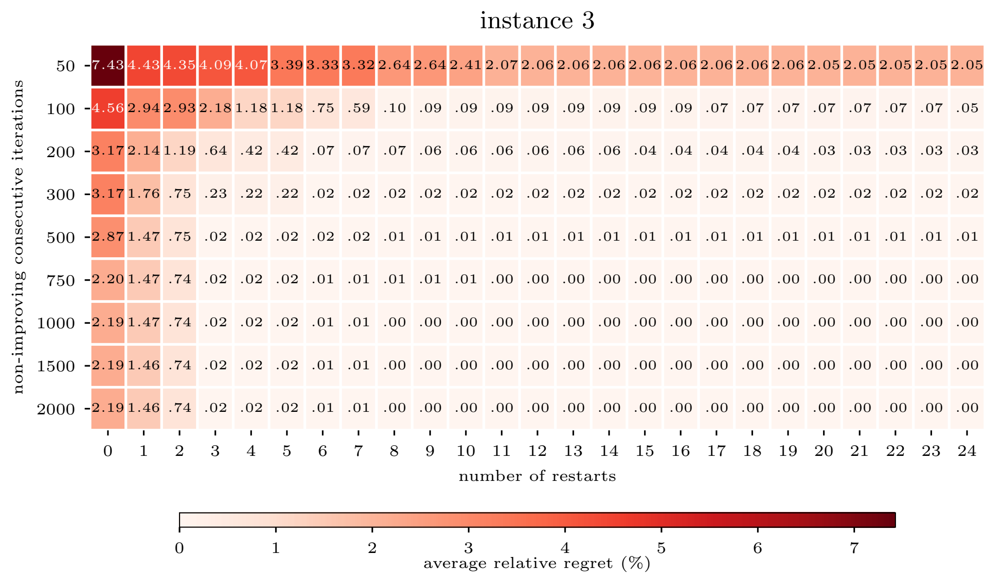
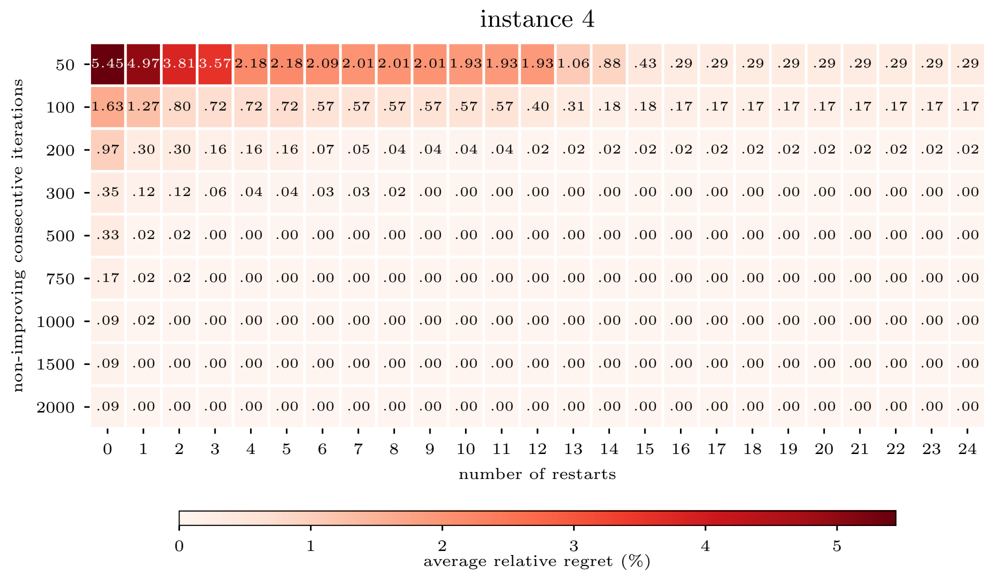


Figure C.13: The relative regret of solutions found using ILS for different restart and termination criterion values. The relative regret was averaged out over all five attempts. Higher values are worse. Note the different relative regret scale used for each instance.



| non-improving consecutive iterations | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 7.43 | 4.43 | 4.35 | 4.09 | 4.07 | 3.39 | 3.33 | 3.32 | 2.64 | 2.64 | 2.41 | 2.07 | 2.06 | 2.06 | 2.06 | 2.06 | 2.06 | 2.06 | 2.06 | 2.06 | 2.05 | 2.05 | 2.05 | 2.05 | 2.05 |
| 100 | 4.56 | 2.94 | 2.93 | 2.18 | 1.18 | 1.18 | .75 | .59 | .10 | .09 | .09 | .09 | .09 | .09 | .09 | .09 | .09 | .07 | .07 | .07 | .07 | .07 | .07 | .07 | .05 |
| 200 | 3.17 | 2.14 | 1.19 | .64 | .42 | .42 | .07 | .07 | .07 | .06 | .06 | .06 | .06 | .06 | .06 | .04 | .04 | .04 | .04 | .04 | .03 | .03 | .03 | .03 | .03 |
| 300 | 3.17 | 1.76 | .75 | .23 | .22 | .22 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 |
| 500 | 2.87 | 1.47 | .75 | .02 | .02 | .02 | .02 | .02 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 | .01 |
| 750 | 2.20 | 1.47 | .74 | .02 | .02 | .02 | .01 | .01 | .01 | .01 | .01 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1000 | 2.19 | 1.47 | .74 | .02 | .02 | .02 | .01 | .01 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1500 | 2.19 | 1.46 | .74 | .02 | .02 | .02 | .01 | .01 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 2000 | 2.19 | 1.46 | .74 | .02 | .02 | .02 | .01 | .01 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |



| non-improving consecutive iterations | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 5.45 | 4.97 | 3.81 | 3.57 | 2.18 | 2.18 | 2.09 | 2.01 | 2.01 | 2.01 | 1.93 | 1.93 | 1.93 | 1.06 | .88 | .43 | .29 | .29 | .29 | .29 | .29 | .29 | .29 | .29 | .29 |
| 100 | 1.63 | 1.27 | .80 | .72 | .72 | .72 | .57 | .57 | .57 | .57 | .57 | .57 | .40 | .31 | .18 | .18 | .17 | .17 | .17 | .17 | .17 | .17 | .17 | .17 | .17 |
| 200 | .97 | .30 | .30 | .16 | .16 | .16 | .07 | .05 | .04 | .04 | .04 | .04 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 | .02 |
| 300 | .35 | .12 | .12 | .06 | .04 | .04 | .03 | .03 | .02 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 500 | .33 | .02 | .02 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 750 | .17 | .02 | .02 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1000 | .09 | .02 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 1500 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |
| 2000 | .09 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 | .00 |

Figure C.14: The relative regret of solutions found using ILS for different restart and termination criterion values. The relative regret was averaged out over all five attempts. Higher values are worse. Note the different relative regret scale used for each instance.

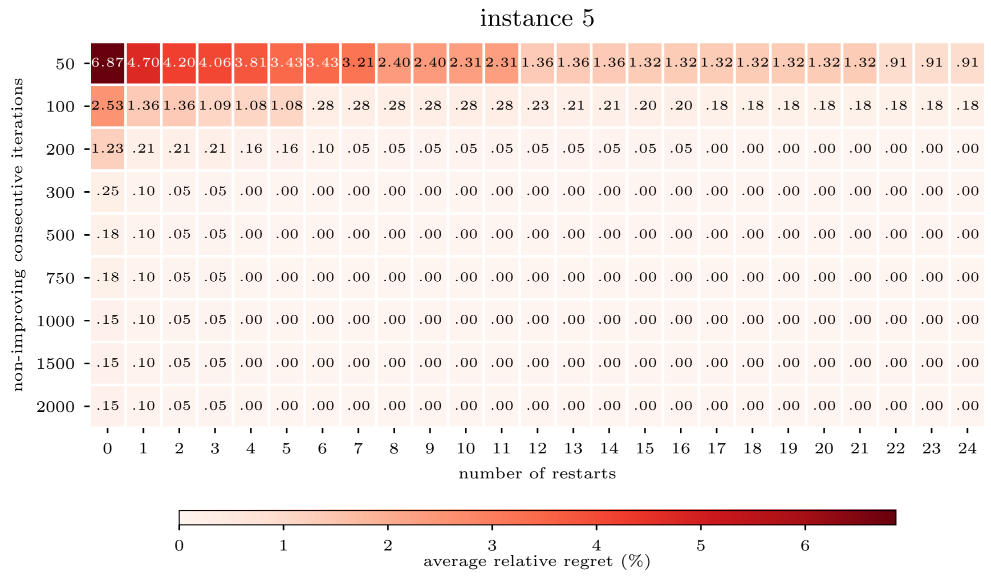


Figure C.15: The relative regret of solutions found using ILS for different restart and termination criterion values. The relative regret was averaged out over all five attempts. Higher values are worse.

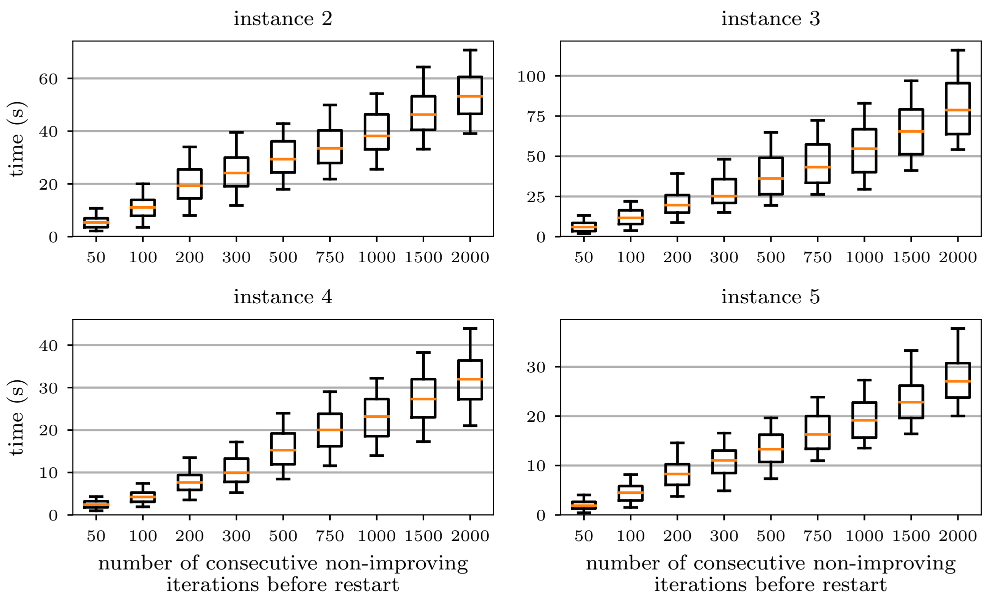


Figure C.16: The execution time of one ILS restart on validation instances 2 to 5, for various restart criteria thresholds. The whiskers denote 5% to 95% of the data.

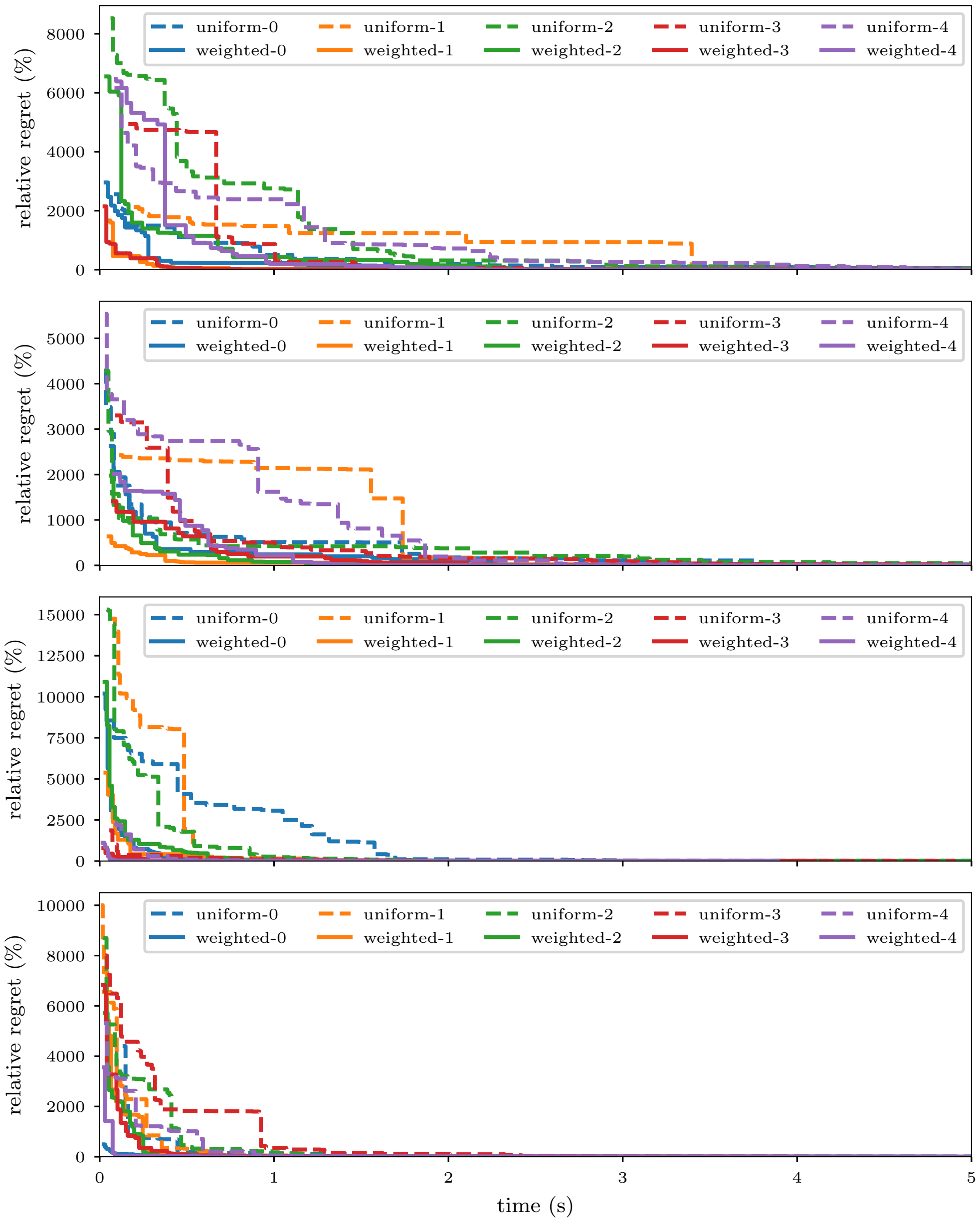


Figure C.17: Relative regret during the first five seconds of ILS using either the uniform or weighted **neighbor** operator, for instances 2 to 5 (top to bottom).

## References

F. Chen, H. Wang, Y. Xie, and C. Qi. An aco-based online routing method for multiple order pickers with congestion consideration in warehouse. *Journal of Intelligent Manufacturing*, 27(2):389–408, 2016.

T.-L. Chen, C.-Y. Cheng, Y.-Y. Chen, and L.-K. Chan. An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics*, 159:158–167, 2015.

C. Cousins and M. Riondato. Cadet: interpretable parametric conditional density estimation with decision trees and forests. *Mach. Learn.*, 108(8-9):1613–1634, 2019.

H. Davarzani and A. Norrman. Toward a relevant agenda for warehousing research: literature review and practitioners' input. *Logistics Research*, 8(1):1–18, 2015.

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

R. B. De Koster, T. Le-Duc, and N. Zaerpour. Determining the number of zones in a pick-and-sort order picking system. *International Journal of Production Research*, 50(3):757–771, 2012.

V. Derkinderen, J. Bekker, and P. Smet. Replication Data for: Optimizing Workforce Allocation under Uncertain Activity Duration, 2023. URL `https://doi.org/10.48804/YHMU7R`.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

G. S. Fishman. *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.

O. Ganbold, K. Kundu, H. Li, and W. Zhang. A simulation-based optimization method for warehouse worker assignment. *Algorithms*, 13(12):326, 2020.

S. Henn. Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. *Flexible Services and Manufacturing Journal*, 27(1):86–114, 2015.

S. Henn and V. Schmid. Metaheuristics for order batching and sequencing in manual order picking systems. *Computers & Industrial Engineering*, 66(2):338–351, 2013.

S. Hong, A. L. Johnson, and B. A. Peters. Batch picking in narrow-aisle order picking systems with consideration for picker blocking. *European Journal of Operational Research*, 221(3):557–570, 2012.

A.-L. Ladier, G. Alpan, and B. Penz. Joint employee weekly timetabling and daily rostering: A decision-support tool for a logistics platform. *European Journal of Operational Research*, 234(1):278–291, 2014.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

J. Matthews and S. Visagie. Order sequencing on a unidirectional cyclical picking line. *European Journal of Operational Research*, 231(1):79–87, 2013.

J. C.-H. Pan and M.-H. Wu. Throughput analysis for order picking system with multiple pickers and aisle congestion considerations. *Computers & Operations Research*, 39(7):1661–1672, 2012.

A. Rijal, M. Bijvank, A. Goel, and R. de Koster. Workforce scheduling with order-picking assignments in distribution facilities. *Transportation Science*, 2021.

R. A. Ruben and F. R. Jacobs. Batch construction heuristics and storage assignment strategies for walk/ride and pick systems. *Management Science*, 45(4):575–596, 1999.

A. Scholz, S. Henn, M. Stuhlmann, and G. Wäscher. A new mathematical programming formulation for the single-picker routing problem. *European Journal of Operational Research*, 253(1):68–84, 2016.

J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European journal of operational research*, 226(3):367–385, 2013.

T. van Gils, K. Ramaekers, A. Caris, and M. Cools. The use of time series forecasting in zone order picking systems to predict order pickers' workload. *International Journal of Production Research*, 55 (21):6380–6393, 2017.

T. van Gils, K. Ramaekers, A. Caris, and R. B. de Koster. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15, 2018.

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

I. Van Nieuwenhuyse and R. B. de Koster. Evaluating order throughput time in 2-block warehouses with time window batching. *International Journal of Production Economics*, 121(2):654–664, 2009.

M. Yu and R. De Koster. Performance approximation and design of pick-and-pass order picking systems. *IIE Transactions*, 40(11):1054–1069, 2008.

J. Zhang, X. Wang, F. T. Chan, and J. Ruan. On-line order batching and sequencing problem with multiple pickers: A hybrid rule-based algorithm. *Applied Mathematical Modelling*, 45:271–284, 2017.