Discrete Optimization

A matheuristic for AGV scheduling with battery constraints

Nitish Singh, Quang-Vinh Dang*, Alp Akcay, Ivo Adan, Tugce Martagan

Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, P.O. Box 513, Eindhoven, MB 5600, the Netherlands

ARTICLE INFO

Article history:
Received 24 August 2020
Accepted 7 August 2021
Available online 14 August 2021

Keywords:
Scheduling
Automated guided vehicles
Adaptive large neighborhood search
Mixed-integer linear programming

ABSTRACT

This paper considers the problem of scheduling automated guided vehicles (AGVs) with battery constraints. Each transport request involves a soft time window, and the AGV fleet used to service those requests is heterogeneous with a diverse set of capabilities and travel costs. In contrast to the existing literature, each transport request may require different AGV material handling capabilities (such as lift loads, tow loads, or handle loads with a mounted robot arm), and the AGV batteries can be recharged partially under consideration of a critical battery threshold. The problem is to assign the transport and charging requests to AGVs, sequence the requests, and determine their starting times and the recharging durations of the AGVs with the objective of minimizing a weighted sum of the tardiness costs of transport requests and travel costs of AGVs. A mixed-integer linear programming model is formulated. We also propose a new matheuristic that makes use of an adaptive large neighborhood search algorithm and a linear program to solve industry-size instances. We illustrate the efficacy of our approach with an industry case study using real-world data.

© 2021 The Author(s). Published by Elsevier B.V.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

1. Introduction

A long-term goal of smart industry is to have a lights-out factory, which requires the control of operations at the shop floor without human interference with the use of automated guided vehicles (AGVs). The AGVs are driverless vehicles that transport goods and materials throughout various areas, e.g., shipping and receiving areas, storage, and workstations (Confessore, Fabiano, & Liotta, 2013; Vivaldini, Rocha, Martarelli, Becker, & Moreira, 2016). The applications of AGVs have grown enormously because of their benefits, such as flexibility in processes, space utilization, product safety, and computer integration and control. More recently, the AGV technology has enhanced with new advances in their flexibility, where each vehicle will have specific capabilities (e.g., one can lift light or heavy loads, while another can tow loads) to perform heterogeneous tasks (De Ryck, Versteyhe, & Debrouwere, 2020; Riazii, Diding, Falkman, Bengtsson, & Lennartsson, 2019). This type of tasks is prevalent in the high-tech manufacturing sector, and it necessitates the handling of materials with specialized equipment, thus requiring a fleet of AGVs with varying capabilities.

An example of a factory requiring different material handling tasks is Brainport Industries Campus (BIC), a new high-tech campus constructed in Eindhoven, The Netherlands, which serves as a home front for far-reaching partnerships between suppliers, specialist companies, and innovative educational and knowledge institutions. The campus is a joint initiative of high-tech suppliers, also called tenants, to co-exist and collaborate on multiple fronts (Brainport Industries Campus, 2020). Since multiple tenants collectively make use of a pool of AGVs, practitioners are looking for a better understanding on how to manage large AGV fleets with heterogeneous capabilities that can cater to a more diverse set of requirements. Consequently, there is an increasing need for planning algorithms to manage such fleets. Industry feedback indicates that most companies rely on the prepackaged fleet management software that is provided by AGV manufacturers. However, such a software is only applicable for a homogeneous AGV fleet and does not make use of look-ahead information such as due dates and time windows. In addition, when considering long travel distances in BIC, the performance of the AGVs can be influenced by battery restrictions. Therefore, efficient management and control of such an AGV system become key factors in reducing material handling costs, work-in-process inventories, and overall operational costs.

In this paper, we address the problem of scheduling AGVs with battery constraints and heterogeneous capabilities. Our problem holds some similarities to the electric vehicle routing problem with time windows (EVRPTW) addressed by Keskin & Çatay (2016) and

* Corresponding author.

E-mail addresses: n.singh1@tue.nl (N. Singh), q.v.dang@tue.nl (Q.-V. Dang), a.e.akcay@tue.nl (A. Akcay), i.adan@tue.nl (I. Adan), t.g.martagan@tue.nl (T. Martagan).
The EVRPTW considers a fleet of electric vehicles (EVs) having limited driving range due to their battery capacities, which may need to visit charging stations while servicing customers in their routes. Our problem, however, distinguishes from the EVRPTW literature by the following features to comply with industrial needs. First, the fleet of AGVs considered in this study is heterogeneous in terms of not only travel speeds and costs, charging rates, and discharging rates but also, more importantly, capabilities to service different types of requests from various tenants. Second, partial charging for the AGVs is allowed under consideration of a critical battery threshold. Third, we consider both travel costs of AGVs and tardiness costs of transport requests, where different types of AGVs and requests have different unit travel costs and penalty charges, respectively. Motivated from our industry collaboration at BIC, the combination of these features constitutes the main novelty of the problem. We aim to make decisions on which AGVs the transport requests are assigned, the sequence in which the requests are processed, the times when AGVs should start servicing the requests, and the recharging duration of AGVs at charging stations so that a weighted average of the total travel and total tardiness costs is minimized.

The main contributions of this paper are as follows.

(i) We formulate the problem as a mixed-integer linear programming (MILP) model.

(ii) We propose a new matheuristic approach that surpasses the MILP and the dispatching policy currently used in practice (see Appendix D for details) given the same computational time limit. In our matheuristic, which combines adaptive large neighborhood search (ALNS) (Ropke & Pisinger, 2006) and linear programming (LP), we propose a new ALNS-based algorithm by introducing two pairs of destroy and repair operator sets to perform exploration and exploitation in distinct phases of the search. Here, we introduce problem-specific heuristics concentrating on the removal and insertion of charging tasks in which a possibility of partial charging for AGVs is allowable. We choose the ALNS framework to solve our problem since it provides flexibility to hybridize with other approaches to make more efficient implementations for optimization problems. Also, it has recently been demonstrated that ALNS-based matheuristics yield promising results on electric vehicle routing problems (Keskin, Laporte, & Catay, 2019; Keskin & Catay, 2018; Köc, Jabali, & Laporte, 2018) owing to the ability to find the best trade-off between the efficacy of exact approaches and the efficiency of metaheuristics (Guimares, Klajban, & Almada-Lobo, 2013).

(iii) We provide managerial insights by performing a sensitivity analysis on a real-world case study, which allows us to determine potential room for improvement in practice. In this analysis, we vary the scheduling horizon, tightness of time window, the number of requests, AGV fleet size, costs related to tardiness and AGV travel, battery thresholds, and computational time limit. Our computational results reveal that the proposed matheuristic allows finding solutions of high quality even for industry-size large instances and yields an average of 33% improvement with respect to the dispatching policy in practice.

(iv) We provide data for the real-world case study used for the experiments in this paper. The problem instances in the case study are generated from a fieldlab that has been built in the BIC. Additionally, the research is conducted in close collaboration with our industrial partner, Ijssel Technology, a company located in the Netherlands, which is the central fleet owner at the BIC. The outcomes of this research can be broadly applied to settings when there is a central operator of a heterogeneous AGV fleet servicing different types of transport requests.

The remainder of this paper is organized as follows. Relevant literature is discussed in Section 2. The problem is formally described in Section 3, and a mathematical model is formulated.
in Section 4. A matheuristic is proposed in Section 5 to solve industry-sized instances. Next, we describe the industry case study, conduct the parameter tuning, evaluate impacts of operators, and present computational results in Section 6. Finally, conclusions and future research directions are discussed in Section 7.

2. Literature review

The design of an AGV control and management system requires the best strategies to solve problems associated with common functions such as dispatching, routing, and scheduling (Qiu, Hsu, Huang, & Wang, 2002). Langvin, Lauzon, & Riopel (1996) define dispatching as the process of selecting and assigning tasks to vehicles, routing as the selection of the specific paths that each vehicle traverses to accomplish its transportation tasks, and scheduling as the determination of the arrival and departure times of vehicles at each station. The decisions related to these functions can be made either simultaneously or sequentially. The reviews of Le-Anh & De Koster (2006) and De Ryck et al. (2020) present guidelines for the control of AGVs and include often-neglected areas, such as idle-vehicle positioning and battery management.

Most dispatching rules in the literature are single-attribute ones, which dispatch vehicles based on only one parameter (Le-Anh, van der Meer et al., 2004; Li & Kuhl, 2017). Commonly used dispatching rules are first-come-first-serve (FCFS) and shortest-travel-distance-first (STTD). While single-attribute dispatching rules are common in practice, multi-attribute dispatching rules, which aggregate different attributes to select a task to be serviced by a vehicle, prove to be better in general (Le-Anh & De Koster, 2005). The methodology proposed in Ho, Liu, & Yih (2012) involves concurrently solving the pickup-dispatching problem along with the load-selection problem with a scoring function that makes use of the slack time of parts, waiting time of parts, and the distance to the load. Jeong & Randhawa (2001) define a scoring function governed by a trained neural net to calculate priorities of incoming jobs. This method is particularly useful if there is a lot of data available on jobs and no clearly defined score function to assess priorities.

Dispatching rules have proven to be good and easy to implement but have been criticized for having a myopic view. More advanced approaches have, therefore, been studied to improve system performance. Liu, Tan, Kurniawan, Zhang, & Sun (2018) develop a mixed-integer programming model that solves the dispatching problem of mobile robots, tasked with transporting materials among different places, in a periodic or event-driven or hybrid way (having a static schedule being modified with new events). Sabuncuoglu & Kızılçıldır (2003) study reactive scheduling problems in a flexible manufacturing environment, where the arrival of jobs is not known in advance. They develop a simulation-based scheduling approach and compare offline and online scheduling schemes. Ebben, van der Heijden, & van Harten (2005) present a serial scheduling method for a dynamic resource-constrained vehicle scheduling problem and analyze its dynamic behavior using discrete event simulation. Nevertheless, these studies do not consider time windows of transport requests and a heterogeneous fleet of AGVs with battery management.

Another stream of relevant work concerns the electric vehicle routing problem (EVRP). It is a variant of the vehicle routing problem (VRP) in which a fleet of EVs is used, instead of internal combustion engine vehicles (ICEVs). Conrad & Figliozzi (2011) study the EVRP, where EVs may recharge fully or 80% of the battery capacity at certain customer locations during their services. The objectives are to minimize the number of vehicles and total costs related to travel distance, service time, and charging. They propose an iterative construction and improvement heuristic to solve the problem. Erdogan & Miller-Hooks (2012) then study the green VRP (GVRP) in which alternative fuel vehicles such as solar, electric, or biodiesel vehicles are fully refueled at stations en route within a constant amount of time. They propose two heuristics to solve the GVRP with the objective of minimizing the total travel distance. Schneider, Stenger, & Goeke (2014) introduce the EVRP-TW where the recharging may take place at any battery level with a linear charging function, and EVs always leave stations with a full battery. In their work, a hybrid heuristic of variable neighborhood search and tabu search (TS) is proposed for solving the EVRP-TW given the objective being to minimize the total travel distance by using the minimum number of vehicles.

Recently, many authors have relaxed the full charge restriction and allowed partial recharging (Bruglieri, Mancini, Pezzella, Pisacane, & Suraci, 2017; Bruglieri, Pezzella, Pisacane, & Suraci, 2015; Ding, Batta, & Kwon, 2015; Keskin & Çatay, 2016; Roberti & Wen, 2016; Schiffer & Walther, 2018b; Wen, Linde, Ropke, Mirchandani, & Larsen, 2016) with different charging technologies at stations (Felipe, Ortúñu, Righini, & Tirado, 2014; Keskin & Çatay, 2018). Several other extensions have also been studied. Some models take into account decisions on both vehicle routing and locating charging stations (Schiffer & Walther, 2017; 2018b). Other studies have considered the possibility of a heterogeneous fleet. Some of them analyze a fleet of EVs having different characteristics such as battery capacities and operating costs (Hiermann, Puchinger, Ropke, & Hartl, 2016; Zhao & Lu, 2019), while others deal with a mixed fleet of EVs and ICEVs (Goeke & Schneider, 2015; Koper & Vornhuisen, 2019; Sassi, Cherif-Khetfa, & Oulamara, 2015a; 2015b). Additionally, several works look at penalties for late arrivals (Grandinetti, Guerriero, Pezzella, & Pisacane, 2016; Keskin et al., 2019). Nevertheless, to the best of our knowledge, the problem in this paper, which concurrently considers transport requests with soft delivery times, heterogeneous AGVs with different material handling capabilities, and partial charging with a critical battery threshold, has not been addressed in the literature. We describe the considered problem in detail in the next section.

3. Problem description

The problem in this paper concerns a set of transport requests \( R \) that is serviced by a heterogeneous AGV fleet \( V \) with battery constraints. The source (pickup) and destination (delivery) nodes of request \( r \in R \) are denoted with \( s^r \) and \( d^r \), respectively. Each request \( r \in R \) requires a set of capabilities \( A^r \) from an AGV for transport, and it has a time window \([e^r, l^r] \), where \( e^r \) is the earliest pickup time and \( l^r \) is the latest delivery time of the request. An AGV is not allowed to reach the source of a request before its earliest pickup time. On the other hand, if AGV \( k \) arrives at the destination of request \( r \) after its latest delivery time, the tardiness occurs and is measured by \( \max(0, \text{arrival time of AGV } k \text{ at } d^r) - l^r \), with a penalty cost \( c_k^r \) incurred per time unit.

Each request needs to be performed by exactly one capable AGV in the heterogeneous fleet. The AGV \( k \in V \) has a set of capabilities \( A^k \) and is capable of servicing request \( r \in R \) if \( A^r \subseteq A^k \) holds, i.e., the capability requirements of a request are included in the capabilities of an AGV. For example, with a request classified as a heavy load that needs to be lifted on a pallet, only those AGVs that have the capability to lift heavy loads can carry out that request. Each AGV also has a travel cost per time unit \( c_k^t \). While an AGV is traveling, the battery charge level decreases proportionally with the traversed distance at a discharging rate of \( d_k^t \), and the AGV may need to visit a charging station before continuing operations. Here, the battery is recharged at a charging rate of \( c_k^r \). The charge level of an AGV (measured as a percentage) must take values between the minimum and maximum battery charge level denoted by \( b^l \) percent and \( b^u \) percent, respectively. The AGV must recharge its battery if the charge level drops below a critical threshold.
4.2. Mixed-integer linear programming model

The mathematical model of the described problem can be formulated as follows:

Objective:

\[
\min \quad \alpha \sum_{k \in V} \sum_{r \in T} c^{\alpha}_{r} T_{r} + (1 - \alpha) \sum_{k \in V} \sum_{r \in T} c^{\beta}_{r} \delta_{r} \tag{1}
\]

Subject to:

\[
\sum_{r \in T} x_{r} r_{k} = 1 \quad \forall r \in R \tag{2}
\]

\[
\sum_{r \in T} x_{r} r_{k} \leq 1 \quad \forall r \in B \tag{3}
\]

\[
x_{r} r_{k} = 0 \quad \forall r \in T, \forall k \in V \tag{4}
\]

\[
\sum_{r \in T} x_{r} r_{k} - \sum_{r' \in T} x_{r'} r_{k} = 0 \quad \forall r' \in T, \forall k \in V \tag{5}
\]

\[
y_{r}^{s} \geq e_{r}^{s} \quad \forall r \in T, \forall k \in V \tag{6}
\]

\[
y_{r}^{s} + h_{r}^{V} + t d_{r} d_{r}^{t} k \leq y_{r}^{s} \quad \forall r \in T, \forall k \in V \tag{7}
\]

\[
y_{r}^{s} + h_{r}^{V} + t d_{r} d_{r}^{t} k - M(1 - x_{r} r_{k}) \leq y_{r}^{s} \quad \forall r, r' \in T, r \neq r', \forall k \in V \tag{8}
\]

\[
T_{r} \geq y_{r}^{s} - l r^{c} \quad \forall r \in T, \forall k \in V \tag{9}
\]

\[
\delta_{r} r_{k} \geq \left(t d_{r} d_{r}^{t} k + t d_{r} d_{r}^{t} k\right) x_{r} r_{k} \quad \forall r, r' \in T, r \neq r', \forall k \in V \tag{10}
\]

\[
z_{r}^{p} r_{k} \geq z_{r}^{s} r_{k} + d_{r}^{t} t d_{r} d_{r}^{t} k \quad \forall r \in R, \forall k \in V \tag{11}
\]

\[
z_{r}^{p} r_{k} \geq z_{r}^{s} r_{k} + d_{r}^{t} t d_{r} d_{r}^{t} k - M(1 - x_{r} r_{k}) \quad \forall r, r' \in R, r \neq r', \forall k \in V \tag{12}
\]

\[
z_{r}^{p} r_{k} \geq z_{r}^{s} r_{k} - c_{r}^{t} \left(y_{r}^{s} - y_{r}^{s}\right) \quad \forall r \in B, \forall k \in V \tag{13}
\]

\[
b^{t} \leq z_{r}^{l} r_{k} \leq b^{t} - b^{l} \quad \forall r \in R, \forall k \in V \tag{14}
\]

\[
b^{t} \leq z_{r}^{l} r_{k} \leq b^{t} \quad \forall r \in R, \forall k \in V \tag{15}
\]

\[
b^{t} \leq z_{r}^{l} r_{k} \leq b^{t} \quad \forall r \in B, \forall k \in V \tag{16}
\]

\[
y_{r}^{s} r_{k}^{l} \delta_{r} r_{k}, T_{r} \geq 0 \quad \forall r \in T, \forall k \in V \tag{17}
\]

\[
x_{r} r_{k} \in \{0, 1\} \quad \forall r, r' \in T, \forall k \in V \tag{18}
\]

The objective function (1) minimizes the weighted sum of tardiness costs of requests and travel costs of AGVs, where \(\alpha\) is a weight coefficient to prioritize one objective over the other. Constraints...
(2) ensure that each request is attended by at most one AGV, while Constraints (3) define whether or not a charging request is performed. Constraints (4) ensure that the capabilities of the requests and AGVs match. Self-visits are avoided by Constraints (5). Constraints (6) conserve the incoming and outgoing arcs for each request done by an AGV. Constraints (7) ensure that an AGV arrives at the source node after the earliest pickup time. Constraints (8) and (9) compute the arrival times at destination and source nodes, respectively. Also, Constraints (9) prevent sub-tours with M defined as a large positive constant. Constraints (10) and Constraints (18) \((\tau_{P} \geq 0)\) define the tardiness of each request. Constraint (11) determines the travel time of an AGV to perform a request. Constraints (12) and (13) calculate the amount of battery discharge due to the travels between the source and destination of a request and between the destination and source of two requests, respectively. The charged amount due to a charging request is given by Constraints (14). Constraints (15–17) set the lower and upper limits \((b' = 0, b'' = 100)\) for an amount of battery discharge, while Constraints (18–19) guarantee valid domains for the remaining decision variables.

5. Proposed matheuristic

In this section, we propose a matheuristic based on the integration of ALNS and LP to solve the problem in Section 3. In our matheuristic, the ALNS is responsible for solving the assignments of AGVs and sequencing of requests, while the LP model is used to determine the schedule of requests, i.e., starting times and recharging duration. The ALNS involves multiple destroy and repair operators, which compete to modify a current solution. In each iteration, a destroy and a repair operator are chosen according to their past performance, and the resulting new solution is accepted if it satisfies some criteria defined by the ALNS (Demir, Belkta, & Laporte, 2012). Here, we integrate the LP into the ALNS framework to utilize the advantages of both methods to find high-quality solutions for large problem instances efficiently.

Let \(Q_0\) and \(S_0\) denote the initial sequence and initial schedule, respectively. Similarly, \(Q_{\text{current}}\) and \(S_{\text{current}}\) denote the current sequence and schedule, while \(Q_{\text{best}}\) and \(S_{\text{best}}\) denote the best sequence and schedule, respectively. In the proposed matheuristic, there are four sets of operators: request destroy, request repair, (charge) station destroy and station repair, which are denoted by \(\Omega^e, \Omega^r, \Omega^d\), and \(\Omega^s\), respectively. Let \(d_i \in \Omega^e\) and \(r_i \in \Omega^r\) denote a destroy and a repair operator for requests, respectively, while \(d_i \in \Omega^d\) and \(r_i \in \Omega^s\) denote a destroy and a repair operator for charging tasks, respectively. Also, the weights of operators in \(\Omega^e, \Omega^r, \Omega^d,\) and \(\Omega^s\), which reflect probabilities to choose an operator, are stored in vectors \(P_c, P_r, P_d,\) and \(P_s\), respectively. The pseudocode of the proposed matheuristic is presented in Algorithm 1.

The matheuristic starts with the initialization of a feasible sequence \(Q_0\) and weights of operators in \(P_c, P_r, P_d,\) and \(P_s\) (line 1). Initially, all operators have the same weight. The initial schedule \(S_0\) is then computed by an LP model with the input of \(Q_0\) (line 2). The matheuristic considers \(Q_0\) \((S_0)\) the current and best so far (lines 3–4) and sets the simulated annealing (SA) temperature \(\tau\) to its initial value \(\tau_0\) (line 5). Then, a loop is executed until a stopping criterion is met (lines 6–31). At the start of the loop, the matheuristic selects a pair of request destroy and repair operators \(d_i\) and \(r_i\) to modify the current sequence \(Q_{\text{current}}\) to generate a new sequence \(Q_{\text{new}},\) which is next used to compute \(S_{\text{new}}\) (lines 7–9). If a new solution is found, the matheuristic focuses on exploiting its neighborhood by applying station destroy and repair operators \(d_i\) and \(r_i\) until it reaches \(n_i\) iterations and does not obtain another new best solution (lines 10–25). Otherwise, a new solution is accepted if it is better than the current solution, or worsen but satisfies the SA acceptance criterion, where \(\mu^{(0.1)}\) is a random number generated according to the uniform distribution \(U(0, 1)\) (lines 26–27). The weights of selected request and station operators in \(P_c, P_r, P_d,\) and \(P_s\) are updated when being used, respectively (lines 23 and 29). Also, the SA temperature is the last to be updated in each iteration, where \(\varepsilon\) is the cooling rate (line 30). At the end of the matheuristic, the best schedule \(S_{\text{best}}\) is returned (line 32). We provide details on the algorithm in separate sections, as stated in the pseudocode.

5.1. Solution representation

The solution of our problem is a schedule in which tasks are carried out at specific time moments. In this section, we describe a solution representation by presenting the types, sequence, and schedule of tasks subsequently as follows.

5.1.1. Types of tasks

A schedule \(S\) consists of three types of tasks such as transport request (loaded travel) \(TR,\) unloaded travel \(UT,\) and charging \(CH,\) which are depicted in Fig. 1. Let \(g^T\) denote the type of task \(r.\) First, the task in Fig. 1(a) represents request \(r\) of type \(TR (g^T = TR),\) which has the time window \([e^T, l^T]\), source and destination nodes \((s^T, d^T)\), and set of capability requirements \(A^T.\) Second, Fig. 1(b) presents a task of type \(UT (g^T = UT),\) which indicates the travel of an AGV without any load to move from the destination of its
previous task \((s^R_i)\) to the source of its next task \((d^R_i)\). This travel occurs in between two tasks of type \(TR\) or types \(TR\) and \(CH\). Finally, Fig. 1(c) presents a task of type \(CH\) \((q^R_1 = CH)\), which has the source and destination nodes at the same charging station, i.e., \(s^R_i = d^R_i\), where \(s^R_i, d^R_i \in E\).

5.1.2. Sequence of tasks

Task sequences on AGVs are constructed from tasks of two types \(TR\) and \(CH\). We denote by \(Q\) the set of task sequences of all AGVs, i.e., \(Q = \{Q^R_k | V_k \in V\}\), where \(Q^R_k\) is the sequence of tasks on AGV \(k\). Additionally, let \(Q^R_k\) denote the \(th\) task in sequence \(Q^R_k\), and the attributes of the task are represented by \(e^R_k, l^R_k, s^R_k, A^R_k, d^R_k\). For example, Fig. 2 illustrates the set \(Q\) consisting of two AGVs, with their set of capabilities, and their respective task sequences \(Q^2\) and \(Q^2\) in which \(Q^2\) and \(Q^2\) are the third and second task of AGV 1 and 2, respectively. Here, the attribute values of \(Q^2\) are \((-,-, -,-, -,-, -,-)\), while those of \(Q^2\) are \((32, 55, 1, 12, (A, C), TR)\). Note that the symbol "-" indicates the absence of the corresponding attribute of a task.

5.1.3. Schedule of tasks

To compute the schedule \(S\) of all AGVs, each sequence \(Q^k\) is post-processed by adding tasks of type \(UT\) to result in sequence \(Q^k\) such that it is composed of transport requests, charging tasks, and unloaded travel tasks. Let \(S^k\) denote the schedule of AGV \(k\). Then \(S = \{S^k | V_k \in V\}\). We also denote by \(S^k\) the \(th\) task in schedule \(S^k\). Two additional attributes, namely \(y^R_k\) and \(y^U_k\), are included to indicate the arrival time of an AGV at the source and destination nodes of task \(r\), respectively. Note that these time moments are determined while computing the schedule \(S\) (see Section 5.6). In addition, for a task of type \(CH\), the difference between \(y^R_k\) and \(y^U_k\) (i.e., \(y^R_k - y^U_k\)) implies the recharging duration. Fig. 3 illustrates the set \(S = \{S^1, S^2\}\), where \(S^1\) and \(S^2\) are the schedule of tasks on AGV 1 and 2, respectively. In schedule \(S^1\), \(S^1\) indicates the second task (\(r = 2\)) with its attributes \((e^R_2, l^R_2, s^R_2, d^R_2, A^R_2, g^R_2, y^U_2, y^R_2) = (8, 55, 11, (A, B), TR, 8, 19)\).

After obtaining the schedule \(S\), its objective value is calculated according to Eq. (20), which is equal to the weighted sum of tardiness costs of requests and travel costs of AGVs. Here, \((l^R_i - l^R_i)^\ast\) gives the tardiness of each transport request \(F\),

\[
f(S) = \alpha \sum_{k \in V} \sum_{r \in S^k | g^R = TR} c^R_{k} (y^R_{i} - l^R_{i})^\ast + (1 - \alpha) \sum_{k \in V} \sum_{r \in S^k} c^R_{k} t^R_{i} d^R_{i} k \quad (20)
\]

5.2. Initialization

The quality of the initial sequence may have a crucial impact on the final outcome of the matheuristic. To generate a good initial sequence, we propose a constructive heuristic, which employs the Earliest Due Date (EDD) rule and the critical charge insertion method described in Section 5.5.2. The pseudocode of this constructive heuristic is given in Algorithm 2. It begins with sorting all transport requests \(r \in R\) in non-decreasing order of their latest delivery times to create the sorted set \(R\) (line 1). Then, we create the set of capable AGVs, denoted by \(V_c\), for servicing each sorted request (lines 2–3). If there exists only one candidate in \(V_c\), it is selected (lines 4–5). Otherwise, the AGV that has the minimum additional cost when inserting request \(r\) to its sequence is chosen (lines 6–10). The cost function \(\Delta f(r, k)\) of inserting request \(r\) to AGV \(k\) is determined in line 8, where the tentative arrival time \(y^d_{i}k\) of the AGV at the request destination is calculated given that the AGV should reach the source node as early as possible but not before the earliest pickup time. After servicing the request, if the charge level \(b^c_k\) of the chosen AGV \(v\) drops below the criti-
5.3. Adaptive weights for operators

At each iteration, the selection of a destroy and a repair operator is regulated by using the roulette wheel mechanism (Ropke & Pisinger, 2006). This gives bigger opportunities for suitable operators to be used during the search, and also allows poor performance operators to be selected with a low probability for diversification purposes. If the operator $d_i \in \Omega_i^-$ is associated with a weight $\rho_i^- \in P_i^-$, it is selected with probability $\rho_i^- / \sum_{d_i : \Omega_i^-} \rho_i^-$, and similarly for $r_i \in \Omega_i^+$, $d_i \in \Omega_i^+$, and $r_i \in \Omega_i^+$. The weight of each operator is set to 1 at the beginning. Then, the weights are adjusted automatically using statistics from earlier iterations and recent performance (Pisinger & Ropke, 2019). The performance of selected operators at each iteration is measured by the score value $\psi$. If a new best solution is found, their scores are rewarded by $\psi = \psi_1$. If the new solution is better than the current solution but not the best, the scores are rewarded by $\psi = \psi_2$. If the new solution is worse than the current solution but accepted according to the SA criterion, the scores are given by $\psi = \psi_3$. Here, a high score value corresponds to a successful operator. The weights of the selected destroy and repair operators are updated using the following equations:

$$\rho_i^- = \lambda \rho_i^- + (1 - \lambda) \psi$$

$$\rho_i^+ = \lambda \rho_i^+ + (1 - \lambda) \psi,$$

where $\lambda$ is the reaction factor that controls how sensitive the weights react to changes in the performance of the selected operators, and $x$ represents either $c$ (request) or $s$ (station). Note that the weights of the operators which are not used in the current iteration remain unchanged. The main aim of the adaptive weight adjustment is to encourage driving the search forward by selecting operators that work well for the instance being solved.

5.4. Destroy operators

In this section, we present nine destroy operators used in our matheuristic. The first five are classified as request destroy that removes transport requests, whereas the last two are classified as station destroy that removes charging tasks. These operators are either adapted or inspired by Ropke & Pisinger (2006), Keskin & Çatay (2016), and Shaw (1998). In general, the destroy phase mainly consists of removing a number of requests from the current sequence, adding them into a removal list, and returning a partially destroyed sequence.

5.4.1. Request destroy operators

We now describe the operators removing requests as follows.

Shaw destroy (SHD): This operator is designed to remove requests which are related to each other in respect of several aspects and thus are easy to reshuffle them around. In our algorithm, the relatedness between two requests $r$ and $r'$ is defined by $S(r, r')$ and is computed by Eq. (23):

$$S(r, r') = \phi_1 (d_{r^e} + d_{r'^e}) + \phi_2 (|r^c - r'^c| + |l^r - l'^r|)$$

$$+ \phi_3 \left(1 - \frac{|K| \cap K'}{\min(|K|, |K'|)}\right)$$

where $\phi_1$, $\phi_2$, and $\phi_3$ are weights (Shaw parameters) corresponding to the distance, time window, and capability requirement terms, respectively, and $K_r$ represents the set of AGVs that are capable of servicing request $r$. A lower $S(r, r')$ indicates a higher degree of relatedness between two requests $r$ and $r'$. It is assumed that $d_{r^e}$, $e_{r^c}$, and $l_{r^r}$ are normalized by scaling such that they only take the values between $[0, 1]$, where $x$ and $y$ respectively represent the first and second nodes in the distance term, while $x$ in the time-window term indicates either request $r$ or $r'$. The SHD first chooses randomly a transport request $r$ to be removed from a sequence (set) $Q$. Then, it sorts an array of non-removed requests $L$ in increasing order of their relatedness values with respect to request $r$. Next, it selects the request placed in the position defined by $|y^p| |L|$, where $y \in [0, 1)$ is a random number, and $p$ is a deterministic power introducing some variation in the selection of related requests, making related requests more likely to be selected. [-] is rounded to the nearest integer. The procedure iterates in a similar fashion until $q$ requests are removed (Keskin & Çatay, 2016; Ropke & Pisinger, 2006). The value of $q$ will be determined in the parameter tuning section (Section 6.2).

We also use three special cases of the SHD operator referred to as distance-based Shaw destroy (SHD-D), time-window-based Shaw destroy (SHD-T), and capability-based Shaw destroy (SHD-C), where $\phi_1 = 1$, $\phi_2 = 1$, and $\phi_3 = 1$, respectively, and the other parameters in Eq. (23) are set to 0.

Worst request destroy (WRD): This operator attempts to remove $q$ requests with a high cost from a sequence $Q$. We define the cost of a request $r$ in a schedule $S$ as $\Delta f(r, S) = (y^c - l^r)^+$. The operator reuses some of the ideas from the SHD. At each iteration, it sorts an array of non-removed requests $L$ in decreasing order of their costs and removes the request listed in position $|y^p| |L|$, where $y$ and $p$ are defined as in the SHD. This removal process is repeated for $q$ iterations.

Random request destroy (RRD): The operator simply removes $\gamma$ requests randomly chosen from the sequence $Q$ to diversify the search mechanism.

5.4.2. Station destroy operators

Charging tasks at stations are crucial components of our problem. Hence, removing or repositioning them in an AGV sequence may improve the solution. We here use the following station destroy operators.
Worst charge destroy (WCD): This operator chooses an AGV \( k \) randomly and calculates the amount of charge that it accumulates at each particular charging station. Then, the WCD selects the charging task that leads to the least increase in charge and removes it from the sequence \( Q^k \).

Random charge destroy (RCD): The operator simply selects an AGV randomly and removes a charging task which is also chosen randomly from the sequence of that AGV.

All charge destroy (ACD): The operator simply selects an AGV randomly and removes all charging tasks from the sequence of that AGV.

5.5. Repair operators

To repair a partially destroyed sequence, six insertion operators are employed in our algorithm. These operators are divided into two groups, namely request repair and station repair. While the operators in the first group attempt to reinsert transport requests removed by destroy methods, those in the second group insert charging tasks into partial sequences thus affecting the visits to the charging stations of AGVs along their route. The insertion operators return a feasible sequence \( Q \) that serves as the input for the LP model, presented in Section 5.6, to compute a feasible schedule \( S \).

5.5.1. Request repair operators

We adapt the greedy and regret-\( \kappa \) insertions from Ropke & Pisinger (2006) and propose a new method called EDD-based greedy insertion to reinsert the transport requests. These three operators are described as follows.

**Greedy insertion (GRI):** This operator repeatedly inserts a request into its best position overall that is called the minimum cost position. Let \( Q^k_{\Delta} \) denote the resulting sequence when inserting request \( r \) at position \( i \) in sequence \( Q^k \) and \( f(r, k, i) \) denote the cost of the whole partial sequence \( Q \) after the insertion. The pseudocode of the greedy insertion is presented in Algorithm 3. For each request \( r \) in the removal list \( D \), the operator collects a set of capable AGVs \( V_C \) for servicing that request (line 3) and determines the cost \( f(r, k, i) \) (line 5), where \( y^k_r \) is the tentative arrival time at the request destination as defined in Section 5.2. The process is repeated for all requests in the set \( D \), and the request that has the minimum cost \( f(r, k, i) \) is inserted at its designated position (lines 8–9). The process continues until all destroyed requests are reinserted.

**Regret-\( \kappa \) insertion (RRI):** One potential downside of the GRI is that it often postpones the insertion of requests that have high insertion costs. When those requests need to be reinserted, there may not be many good positions available. Hence, the regret-\( \kappa \) heuristic is used to overcome this shortcoming. Let \( f(r, k, i) \) be defined as in the GRI and \( f^*(r, k) \) denote the minimum cost of inserting request \( r \) into the sequence of AGV \( k \). In addition, let \( f_{\text{regt-}\kappa}(r) \) denote the regret value that is the difference in the cost of inserting request \( r \) in its best sequence and its \( \kappa \)-best sequence. The pseudocode of the RRI is given in Algorithm 4. In each iteration, the operator chooses the request that has the maximum regret value overall (line 12), where each regret value \( f_{\text{regt-}\kappa}(r) \) is determined in line 10. Then, the chosen request is inserted into its minimum cost position. The process is repeated until all removed requests are reinserted. We use small values of \( \kappa \), i.e., the regret-2 and regret-3 heuristics are employed in our matheuristic.

**EDD-based greedy insertion (EGI):** Although the greedy and regret-\( \kappa \) insertions are often the preferred heuristics in the ALNS literature (Demir et al., 2012; Grangier, Gendreau, Lehd, & Rousseau, 2016; Gullhav, Cordeau, Hvattum, & Nygreen, 2017; Ke-skin & Çatay, 2016; Žulj, Kramer, & Schneider, 2018), they are computationally expensive since the insertion cost has to be calculated for each available position in a sequence \( Q \). We hence introduce a faster heuristic called the EDD-based greedy insertion. Here, the difference between the EGI and the GRI is that the EGI only considers to reinsert a destroyed request \( r \) in the sequence of a capable AGV \( k \) at the first position \( i \), where \( L^k_r \) is the difference in cost w.r.t. the idea lies in the fact that requests with earlier delivery times (earlier due dates) should be serviced before the ones with later delivery times.

It is notable that a request insertion operator is feasible with respect to capability requirements but may not meet the critical battery threshold. In this case, the critical charge insertion presented in Section 5.5.2 is applied to make repaired solutions feasible.

### Algorithm 4 Regret-\( \kappa \) insertion.

**Require:** level \( \kappa \), set of destroyed requests \( D \), set of AGVs \( V \), partial sequence \( Q \).

1: **while** \( |D| > 0 \) **do**
2: **for** each \( r \) \in \( D \) **do**
3: \( V_C \leftarrow \{ k \in V \mid A^k_r \subseteq A^k_r \} \)
4: **for** each \( k \in V_C \) **do**
5: \( f^*(r, k) \leftarrow \min f(r, k, i) \mid \forall i \in \{1, \ldots, |Q^k|\} \)
6: **for** each \( k \in V_C \) **do**
7: \( f^*(r, k) \leftarrow \sum f^*(r, k, i) \)
8: **end** for
9: \( f^*(r) \leftarrow \arg \min \{ f^*(r, k) \mid \forall k \in V \} \)
10: **end** for
11: Insert \( r^\star \) at position \( i^\star \) in sequence \( Q^k \)
12: **end** while

5.5.2. Station repair operators

After removing charging tasks, a partially destroyed sequence may become infeasible to the battery charge requirement of AGVs. To repair the sequence, we propose two operators inserting charging tasks, namely critical charge and non-critical charge insertions. Since AGVs can charge as many times as needed, any charging task can be inserted throughout the station repair operators, and thus it may not be necessary to reinsert all the charging tasks which are removed by the station destroy heuristics. Note that any AGV should perform a charging task at a charging station closest to the location of its previously serviced request.

**Critical charge insertion (CCI):** This operator estimates the battery charge level after servicing each request in a partial sequence.
of AGV $k$ and inserts a charging task whenever its charge level falls below the critical battery threshold $b_k^c$.

Non-critical charge insertion (NCI): In this operator, we introduce the non-critical battery threshold for AGV $k$, denoted by $b_k^{c'}$, to represent an upper threshold for the AGV charging. The main difference between $b_k^c$ and $b_k^{c'}$ is that AGV $k$ may visit a charging station if the charge level is below $b_k^c$, but it is not necessary to do so if the visit results in a missed deadline (whereas the AGV must visit a station if its charge drops below $b_k^{c'}$). In other words, this operator determines any request in a partial sequence, after which the battery state of charge is under the non-critical threshold and inserts a charging task if there is enough idle time between that request and the next request. The idea is to obtain possibly better solutions if charging tasks are added in advance, and not just when the battery level falls below the critical threshold as in the CCI.

All charge insert (ACI): The operator simply selects an AGV randomly and inserts a charging task after every request in the sequence of that AGV.

5.6. LP scheduling

The destroy and repair operators of ALNS presented thus far are employed to solve the assignment and sequencing of requests on AGVs. In this section, we propose an LP model to optimize starting times and recharging duration of each AGV $k \in V$, where $V$ represents the set of AGVs whose sequence $Q_k$, and thus $\hat{Q}_k$, is modified by ALNS in each iteration of the matheuristic.

To present the model, we define decision variables $\tau_r$, $y_i^r$, and $y_i^r$ as the tardiness, arrival time at the source, and arrival time at the destination of task $r$, respectively. We also introduce the variable $\beta_r$ to keep track of the battery charge level of the input AGV at the start of task $r$. The initial battery charge level, $\beta_0$, is set to $b_k^c$. The proposed LP model is as follows:

$$\begin{align*}
\min \quad & \sum_{r \in Q_k | g^r = TR} c_i^r \tau_r \\
\text{Subject to:} & \\
& y_i^r + h_i^r + t_{i;i}^r d_j^r \leq y_{i'}^r, \quad r \in \hat{Q}_k \\
& y_{i'}^r = y_{i'+1}^r, \quad r \in \hat{Q}_k \\
& y_i^r \geq c_i^r, \quad r \in \hat{Q}_k, g_{i'}^r = TR \\
& \tau_r \geq y_i^r - t_i^r, \quad r \in \hat{Q}_k, g_{i'}^r = TR \\
& \beta_0 = b_k^{c'} \\
& \beta_{(r+1)} \leq \beta_r + (y_i^r - y_{i'}^r - t_{i;i}^r d_j^r) c_i^r - (t_{i;i}^r d_j^r) d_i^r, \quad r \in \hat{Q}_k, g_{i'}^r = CH \\
& \beta_{(r+1)} = \beta_r - t_{i;i}^r d_j^r d_i^r, \quad r \in \hat{Q}_k, g_{i'}^r \neq CH \\
& b_i' \leq \beta_r \leq b_i^u \\
& b_i' \leq \beta_r \leq b_i^u \\
& y_i^r, y_{i'}^r, \tau_r \geq 0, \quad \forall r \in \hat{Q}_k
\end{align*}$$

Objective (24) minimizes the total tardiness cost. Constraints (25) ensure that there is sufficient time to move from the source to the destination of each request. Since the source of the subsequent request is the destination of the current request, Constraints (26) ensure that the time abides the same. A request cannot be serviced before the earliest pickup time as imposed by Constraints (27). The tardiness of each request is defined by Constraints (28) and Constraints (34). $T_r \geq 0$. Constraints (29) initialize the battery charge of the input AGV $k$ at a level of $b_k^{c'}$.

The charging of AGV is given by Constraints (30), while the discharging due to loaded and unloaded travel is given by Constraints (31). Constraints (32) and (33) define the lower and upper limits ($b_i' = 0$ and $b_i^u = 100$) of the given AGV’s charge level, while Constraints (34) impose restrictions on the other decision variables.

5.7. Acceptance and stopping criterion

After we obtain a schedule $S$ from the proposed LP model, it is necessary to determine whether or not the schedule can be used as the basis for the next iterations of our matheuristic. To this end, we apply the SA acceptance criterion, which is most commonly used in the ALNS framework, to avoid getting trapped in local optima (Santini, Ropke, & Hvattum, 2018). That is, a new improving solution is always accepted, i.e., $f(S_{new}) < f(S_{current})$ or $f(S_{new}) < f(S_{best})$, or a deteriorating solution is accepted with probability $e^{-\left( f(S_{new}) - f(S_{current}) \right) / T}$, where $T$ is the current temperature, and $f(S_b)$ is determined according to Eq. (20). We initialize $\tau$ at $T_0$ such that a solution which is $w\%$ worse than the current solution is accepted with probability $0.5$. Therefore, we have $T_0 = -W \cdot f(S_b)/\ln(0.5)$, where $W$ denotes the initial temperature control parameter (Ropke & Pisinger, 2006; Zhao & Lu, 2019). Then, the current temperature decreases gradually by a cooling rate $\epsilon$ at every iteration $(\tau \leftarrow \tau + 1)$, where $0 < \epsilon < 1$. The matheuristic stops when its running time has passed a given time limit $T_{max}$.

6. Computational experiments

This section presents the experimentation process for assessing the performance of the MILP model and the proposed matheuristic (MH). First, we describe the generation of problem instances based on the floor layout of a fieldlab in the BIC in Section 6.1. Next, Section 6.2 discusses the parameter tuning for MH, and Section 6.3 quantifies the impact of the destroy and repair operators. In Section 6.4, we describe the current dispatching rule used by our industry partner at BIC as a benchmark to the MILP and the proposed MH. We then compare the performance of the three methods to validate the performance of the proposed MH with small-sized problem instances. Finally, we carry out the sensitivity analysis with larger-sized problem instances to further demonstrate the MH’s effectiveness and generate managerial insights for the industrial case study in Section 6.5.

6.1. Case study design

The problem instances in this research are generated after discussions with industry practitioners from BIC. We generate problem instances by using the distance matrix based on the real world layout at the BIC, shown in Fig. 7 (see Appendix A). In practice, requests arrive over the layout and are made known to the scheduler at the start of a scheduling horizon $H$ that can typically be of length 1800 or 3600 seconds. The number of requests $|R|$ within this time horizon can range between 20 and 60 depending on the workload. Additionally, requests contain information about their priority. High priority requests are usually production critical requests, which, if delivered late, can decrease the productivity of
downstream processes on the shop floor, and therefore incur high penalty costs. Low priority requests have little effect on being delivered late and therefore incur lower penalty costs. Low priority, medium priority, and high priority requests incur a penalty of $0.01, $0.02, and $0.03 per second, respectively, on being delivered after the latest delivery time. The AGV fleet consists of three types of AGVs with different capabilities. For example, one particular type of AGV has capabilities $A$, $B$, $C$ or $[A, C, D]$ or $E$ in which $A$ represents the capability to lift a load, $B$ to handle heavy loads, $C$ to handle light loads, $D$ to tow loads, and $E$ to handle loads with a robotic arm mounted on the AGV. Therefore, when a request arrives with capability requirements $[A, C]$, i.e., a light load that needs to be lifted, the scheduler must assign it to an AGV that contains these capabilities. Further, AGVs incur travel costs of $0.01$, $0.02$, and $0.03$ per second. These costs are representative of operational and maintenance costs and have been altered to protect confidentiality. The representative data from the actual AGV fleet used in practice is shown in Table 9 (see Appendix B).

The instance generation procedure is presented in Algorithm 5 (see Appendix C). This procedure takes an input parameter set including the scheduling horizon $H$, the probability of tight time windows $TW$, and the number of requests $|R|$ and outputs an instance set containing $|R|$ requests with a random arrival time moment within $H$ seconds and variable time window lengths. The tightness of a time window is measured by the gap between the earliest pickup time and the latest delivery time, and a value of $TW$, e.g., equal to 0.5, indicates that transport requests have a 50% chance of having a tight time window, where $TW \in [0,1]$. For each parameter set\(^1\) (scenario), we generate ten instances and run each instance ten times, making a total of 100 runs. We have observed that ten runs for each instance are sufficient to obtain a stable width around the mean with 95% confidence interval (Wang et al., 2013). The mean and coefficient of variation (CV) of the costs for a parameter set are calculated from costs obtained in these 100 runs. The weight coefficient in the objective function, $\alpha$, is set to 0.5 for all instances.

All experiments have been run with an execution time limit of 15s (i.e., $T_{max} = 15$s), unless stated otherwise, on a computer with Intel Core i7-8750H CPU @ 2.20GHz CPU and 16GB RAM with Windows 10 operating system. The MH and MILP are programmed in Python v3.6 and Gurobi Python package, respectively, while the dispatching rule is implemented in the simulation software Anylogic v8.3.1.

6.2. Parameter tuning

The parameter tuning methodology is adopted from Ropke & Pisinger (2006). For each parameter shown in Table 1 (see columns 1–2), we consider five scenarios, generate ten instances per scenario by Algorithm 5, and perform ten runs per instance for each of five discrete possible values. The considered scenarios and parameter values are reported in Table 10 and Table 11, respectively (see Appendix E). Then, for each parameter value, the tuning procedure calculates the average percent deviation from the average of the best achieved solutions and selects the parameter value that results in the least average percent deviation. This procedure is repeated until all parameter values are tuned. The detailed results of the parameter tuning are given in Table 11, and the parameters’ best values are summarized in Table 1.

Moreover, to determine parameter $q$ (the number of requests removed via Shaw destroy operators), we carry out a separate tuning experiment across various instance sizes that depend on the number of requests ($|R|$) and the number of AGVs ($|V|$). The result shows that the value of $q$ can be determined by the piece-wise function as given in Eq. (35). It can be seen that the value of $q$ is nonincreasing in the instance size since a larger $q$-value results in a longer running time (due to an increase in computations) per iteration, which slows down the exploration rate of the matheuristic given the computational time limit. While a successful ALNS algorithm relies on the availability of several fast-to-explore operators, having a large $q$ for large instances goes against this principle. For small instances, however, that is not the case, which is also reflected in our findings.

$$q(|R|, |V|) = \begin{cases} 
3 & \text{if } |R| \leq 20 \text{ and } |V| \leq 9 \\
2 & \text{if } 20 \leq |R| < 40 \text{ and } |V| \leq 3 \\
1 & \text{otherwise}
\end{cases} \quad (35)$$

6.3. Impact of destroy and repair operators on the performance of ALNS

The ALNS framework consists of different destroy and repair operators. To gain some insights into the operators, we analyze their impact on the performance of the MH by reporting the number of times they are selected (‘Used’), contribute to an improvement (‘Imp.’), and lead to a new best solution (‘Best’) in Table 2. All the instances generated in the parameter tuning section are used to test the performance of the operators. The results show that the operators in all groups are selected with approximately equal proportions. They also contribute equally to improvements in solution quality, except for ‘Station repair’ group, where CCI dominates NCI and ACI. The operators in ‘Request destroy’ and ‘Request repair’ groups contribute equally to finding new best solutions. On the other hand, operators in ‘Station destroy’ and ‘Station repair’ groups have noticeable disparity in performance, i.e., RCD, ACD, NCI, and ACI operators contribute relatively less to finding the best solution.

| Parameter | Description | Value |
|-----------|-------------|-------|
| $\psi_1$  | heuristic reward for a best solution | 0.9   |
| $\psi_2$  | heuristic reward for a better solution | 0.7   |
| $\psi_3$  | heuristic reward for an accepted worse solution | 0.2   |
| $\lambda$ | reaction factor | 0.9   |
| $n_t$     | number of times station methods are invoked | 5     |
| $p$       | degree of randomness in choosing related tasks | 10    |
| $\phi_k$  | weight for relatedness in terms of distance | 3     |
| $\phi_k'$ | weight for relatedness in terms of time window | 8     |
| $\phi_1$  | weight for relatedness in terms of AGV capability | 6     |
| $\gamma$  | number of random tasks to remove | 1     |
| $\kappa$  | level in the class of $\kappa$-regret heuristic | 3     |
| $\varepsilon$ | cooling rate in simulated annealing | 0.8   |
| $w$       | initial temperature parameter in simulated annealing | 0.1   |

1 Problem instances are available from https://github.com/nitman118/qlns-instances

| Operator group | Operator | Used [%] | Imp. [%] | Best [%] |
|----------------|---------|----------|----------|----------|
| Request destroy | SHD     | 16.90 | 22.77 | 18.23 |
|                 | SHD-T   | 17.24 | 17.22 | 20.01 |
|                 | SHD-C   | 16.74 | 14.44 | 19.33 |
|                 | SHD-D   | 16.69 | 19.44 | 16.36 |
|                 | WRD     | 16.01 | 9.44  | 11.45 |
| Request repair  | GRI     | 33.48 | 36.66 | 36.98 |
|                 | RKI     | 33.57 | 33.33 | 39.44 |
|                 | EGI     | 32.95 | 30.01 | 23.57 |
| Station destroy | WCD     | 35.75 | 36.94 | 65.91 |
|                 | RCD     | 33.61 | 39.40 | 28.66 |
|                 | ACD     | 30.63 | 23.65 | 5.41  |
| Station repair  | CCI     | 38.85 | 80.57 | 99.77 |
|                 | NCI     | 30.59 | 9.55  | 0.23  |
|                 | ACI     | 30.55 | 9.87  | 0.00  |
To better understand the benefit of each operator to the proposed method, we conduct a statistical analysis (see Appendix F), inspired by the work of Schiffer & Walther (2018a). This analysis shows that removing the SHD-D, ACD, and ACI leads to the best increase in solution quality. Hence, we removed these operators afterwards. The remaining operators will then be used for all subsequent experiments.

6.4. Comparisons of MILP, the proposed MH, and a practical scheduling policy

In this section, we study the performance of the MILP and the proposed MH. In addition, the results of our MH are assessed with respect to the current practice of scheduling AGVs at IJssel Technology, our industry partner at BIC. The current practice makes use of a bidding-based dispatching rule. In this rule, a transport request is sent to the scheduler when the material to be transported is ready at its pickup location. This rule makes use of EDDs to prioritize requests for assignments on eligible idle AGVs. An AGV is considered idle when it has sufficient charge \( b_k^c \geq b_k^c \) and is not assigned to any request. The scheduler selects the idle AGV that is closest to the source node of the request, i.e., distance-based bidding. Note that after servicing a request, an AGV is sent to the nearest charging station if its charge level drops below the critical threshold (the CCI heuristic). A detailed explanation of this dispatching rule, referred to as EDDBID, is provided in Appendix D.

The three approaches’ performance is investigated for the problem instances in which the number of requests \(|R|\) ranges from 3 to 30 with \(H = 1800, TW = 0.5, \) and \(|V| = 6\). To ensure a fair comparison, we run the instances on one processor core and limit the computational time to 15s (i.e., \(T_{\text{max}} = 15s\)) for all approaches. We also extend the time limit to 3600s on the MILP to check whether the optimal value is reached and further validate the solutions of the MH. Ten iterations are run per instance of a parameter set to obtain the mean and CV of the objective value. A percentage gap is calculated from the mean objective values found by two respective approaches, either the MH and MILP or the MH and EDDBID. A negative gap indicates an improvement in the solution quality of the MH over the other approaches.

Table 3 shows that the MILP model can solve small instances to optimality, but with an increase in the number of requests, it fails to reach the optimal value within the considered time limits and returns only the best solution found so far. Namely, the MILP running in 3600s (MILP-3600) starts to give only the best feasible solutions when there are more than 5 requests, and not all iterations find a solution beyond 20 requests. Additionally, the MILP-3600 is not able to find a feasible solution for the instances with more than 27 requests. Similar trends can be observed for the MILP running in 15s (MILP-15) and are marked by distinguishing symbols in Table 3.

On the other hand, the MH is able to find solutions for all instances and provides a better solution than the MILP-15 in all cases and MILP-3600 in a fourth of the cases. For the remaining cases, although the objective values found by the MH are greater than those of the MILP-3600, the difference is always smaller than 2%. In addition, the MH consistently finds a better solution beyond 20 requests. We also observe that the MH can obtain the first solution within a second, which is not the case with the MILP for even small instances. Finally, the MH outperforms EDDBID in all instances, with an improvement of 27% on average.

6.5. Sensitivity analysis

This section studies the performance of the MH and EDDBID and produces managerial insights by performing a sensitivity analysis on larger instances that are typical in practice. Both approaches are run with \(T_{\text{max}}\) of 15s, except in Section 6.5.4, where the computational time is varied. The MILP is excluded from the analysis since it was not able to find a feasible solution for large-sized instances, as shown in Table 3. Each scenario, considered in this analysis, is represented as \(H - TW - |R| - |V|\) and consists of ten instances in which \(H\) is varied in the set \([1800, 3600]\), \(TW\) in \([0.2, 0.5, 0.8], |R|\) in \([20, 40, 60]\), and \(|V|\) in \([3, 6, 9]\). Again, each instance is run ten times. The data of AGV fleet size \(V\) is also in Table 9 (see Appendix B), and the fleet size \(|V|\) of, e.g., three indicates that the first three AGVs in this table are employed.

We analyze the effect of the parameters \(H, TW, |R|, \) and \(|V|\) in Section 6.5.1, while the effects of cost parameters, battery thresholds, and time limit are investigated in Sections 6.5.2, 6.5.3, and 6.5.4, respectively.

6.5.1. Impact of scheduling horizon, time-window tightness, number of requests, and fleet size

The mentioned values of the parameters \(H, TW, |R|, \) and \(|V|\) are combined to obtain 54 scenarios, thus 540 instances in total. The average costs (\(\mu\)) obtained by MH and EDDBID are presented in Table 4. The column ‘Gap’ reports the percentage gap between the costs of the two approaches. Several selected trends across the varied parameters are also shown in Fig. 4.

Fig. 4 (a), under a constant AGV fleet size, e.g., \(|V| = 6\), shows the effect of varying \(TW, |R|, \) and \(H\) on the average cost. It can be observed that there is a lower gap in the performance of MH and EDDBID when the number of requests is less compared to when the number of requests is high. The cost of EDDBID increases at a
higher rate than MH when increasing from 20 to 60 requests. On the other hand, the cost decreases with an increase in the scheduling horizon in both MH and EDDBDID. However, the average improvement for all scenarios with respect to 1800s and 3600s remains the same, i.e., about 33%, which implies the robustness of MH in different scheduling horizons.

Moreover, it can be seen in Fig. 4(b) that MH performs better than EDDBDID under tight time windows, and the improvement is constant. In addition, Fig. 4(c) shows that the cost goes down with an increase in fleet size. The reduction in cost is higher when moving from 3 to 6 AGVs than when moving from 6 to 9 AGVs. Additionally, Fig. 4(d) shows the increase in cost due to a higher number of requests. It is observable that when the number of requests is low ($|R| = 20$) or the available fleet size is large ($|V| = 9$), the difference in costs is smaller when there are more requests to schedule on a limited fleet. This trend is also evident from the results in Table 4.

In general, the cost gains are substantial across a wide number of scenarios with the mean gap equal to 33% while the CV ranges between 0.7% and 9%. Additionally, the cost savings are highest when working in an extremely constrained situation, i.e., when the number of requests to schedule is high, and the available fleet size is small. MH makes use of look-ahead information to feasibly plan and dispatch AGVs, resulting in large cost savings, and in scenarios where the fleet size is large, the effects of making use of look-ahead information diminish.

6.5.2. Impact of penalty cost for tardiness and AGV travel cost

The experiments in Section 6.5.1 contain requests with varying penalty costs for tardiness and AGVs with varying travel costs.

We report, in this section, our observations drawn from these aspects. We use three types of requests with penalty costs 1, 2, and 3 in the experiments. Table 5 shows the breakdown in tardiness values among requests with different costs when scheduled with EDDBDID and MH. It can be observed that requests with a higher cost constitute a lower share of overall tardiness value in MH, whereas these requests are found to constitute a larger proportion in EDDBDID. In other words, the schedule obtained via MH reduces the tardiness and, if tardiness is unavoidable due to extremely tight time windows, MH prioritizes requests with a higher cost. EDDBDID, on the contrary, is indifferent to request penalties and can lead to a high total cost.

Similarly, we use three types of AGVs with travel costs 1, 2, and 3 for the experiments in Section 6.5.1. A larger travel cost is often accorded to AGVs that have high operational costs. Table 5 shows that MH shares the travel times of AGVs inversely proportional to their travel costs. Namely, the AGV with a higher travel cost has a lower proportion of the total travel time, while the AGV with the least travel cost constitutes the largest share. On the other hand, EDDBDID has approximately an equal share of travel time, since it does not account for travel costs. It is notable that travel time also depends on capability requirements of a request ($A^2_r$) and capability set of AGVs ($A^2_k$), i.e., the matching between a request and AGV should satisfy the feasibility condition ($A^2_r \subseteq A^2_k$), and preference is considered when AGVs with different travel costs satisfy the capability requirements of a particular request. It is for this reason that the AGV with $c^2_k = 3$ has the same proportion when scheduled by EDDBDID and MH in Table 5.

### Table 4

All results of performance of MH in comparison to EDDBDID for 54 generated problem instances when varying parameters $H, T, W, |R|$, $|V|$.

| # | Scenario | EDDBDID | MH | Gap [%] | # | Scenario | EDDBDID | MH | Gap [%] |
|---|---------|---------|----|--------|---|---------|---------|----|--------|
| 1 | 1800-0.2-20-3 | 12441.95 | 7655.34 | -38.75% | 28 | 3600-0.2-20-3 | 9216.94 | 6502.21 | -29.45% |
| 2 | 1800-0.2-20-6 | 8432.76 | 5961.15 | -29.31% | 29 | 3600-0.2-20-6 | 7853.68 | 6011.24 | -23.46% |
| 3 | 1800-0.2-20-9 | 8048.77 | 5685.56 | -29.36% | 30 | 3600-0.2-20-9 | 7451.34 | 5689.13 | -23.48% |
| 4 | 1800-0.4-20-6 | 5959.73 | 3647.96 | -38.00% | 31 | 3600-0.4-20-3 | 47060.27 | 24482.52 | -47.98% |
| 5 | 1800-0.4-20-6 | 18231.13 | 12071.04 | -33.79% | 32 | 3600-0.4-20-6 | 15826.06 | 10860.30 | -31.38% |
| 6 | 1800-0.4-20-9 | 14994.71 | 10219.90 | -31.84% | 33 | 3600-0.4-20-9 | 15083.99 | 10259.79 | -31.98% |

### Table 5

Tardiness proportion for requests and travel time proportion for AGVs.

| Penalty cost per time unit ($c^2_r$) | Tardiness proportion [%] | Travel cost per time unit ($c^2_k$) | Travel time proportion [%] |
|---|---------|---|---|---|
| EDDBDID | MH | EDDBDID | MH |
| 1 | 34.77 | 40.93 | 1 | 39.02 | 45.54 |
| 2 | 32.57 | 31.87 | 2 | 35.42 | 28.47 |
| 3 | 32.65 | 27.17 | 3 | 25.56 | 25.98 |
6.5.3. Impact of battery thresholds

In this experiment, we vary critical and non-critical battery thresholds of AGVs (i.e., $b_{i,k}^c$ and $b_{i,k}^n$). While the former threshold defines a must condition of visiting a charging station of an AGV, the latter only presents a possibility of the visit. We select three parameter sets, representing a small, medium, and large setting, where $b_{i,k}^c$ is tested with five values from 10 to 50 (percent of the full charge) and $b_{i,k}^n$ is varied with three levels 60, 80, and 100. The results are shown in Table 6 and Fig. 5.

For the medium and large settings, the lowest cost is obtained when $b_{i,k}^c$ is around 30%, whereas, in a small setting, the best performing critical thresholds are higher, which implies that having a small critical threshold is not necessarily profitable. On the other hand, there is not an evident trend when varying the non-critical threshold, which may indicate that MH is able to schedule critical charge duration in order to maximize the feasibility of serving all requests.

6.5.4. Impact of computational time

We analyze the effect of the computational time on the solution quality of the MH by increasing the time limit to 15, 30, 45, and 60s. This experiment is also conducted with three problem settings (scenarios). The average total costs of each scenario obtained in 15s, 30s, and 45s are compared to that found in 60s. The
\begin{table}[h]
\centering
\begin{tabular}{|c|cccc|cccc|cccc|}
\hline
Scenario & 60s & 15s & 30s & 45s & 60s & 15s & 30s & 45s & 60s & 15s & 30s & 45s \\
\hline
1800-0.2-20-3 & 7624.32 & 1.21 & 0.65 & 0.30 & 13539.82 & 2.47 & 1.11 & 0.36 & 26817.25 & 6.37 & 2.02 & 0.51 \\
1800-0.5-40-6 & 7660.83 & 1.52 & 0.85 & 0.45 & 13644.43 & 2.53 & 1.14 & 0.38 & 27531.64 & 6.48 & 2.04 & 0.52 \\
1800-0.8-60-9 & 7686.15 & 1.80 & 0.95 & 0.48 & 13843.71 & 2.70 & 1.17 & 0.40 & 28046.47 & 6.68 & 2.07 & 0.53 \\
\hline
\end{tabular}
\caption{Performance of MH when varying computational time.}
\end{table}

To further investigate the computational efficiency of the MH, we report in Table 8 the percentage of the time spent in each of the algorithm’s components relative to the total time available (T_{max}). We here consider the scenario with $|V|=9$, $H=1800$, $TW=0.8$, and $T_{max}=15s$. It can be seen that the maximum time is spent in the ‘Request destroy/repair’ component. When increasing the number of requests, the number of iterations decreases due to expensive computations of operators that aim to destroy requests from their worst locations and reinsert them to their best ones, and these candidate locations increase with an increase in the number of requests. As a result of the reduced number of iterations, the percentage gap gradually diminishes between MH and EDDBID. However, we observe at our industrial partner that the instances of 9 AGVs and 60 requests are the busiest scenario in practice. Even if with 100 requests, the MH still produces a better solution than the EDDBID.

To summarize the sensitivity analyses, the proposed MH is able to guarantee better results than the current way of scheduling results in Table 7 and Fig. 6 show that the gap relative to the cost obtained in 60s decreases with the increase in the computational time. However, the gap is only at a maximum of about 2%, 3%, and 7% for a small, medium, and large setting, respectively. Additionally, it can be seen in Fig. 6 that the objective values of the three scenarios converge as the computational time increases, and the rate of convergence, for the large setting, is higher than it does for the smaller ones.
within a computational time acceptable in practice. We also compute the CV values for all instances to examine the volatility in the obtained results. We observe an average CV of 4% that signifies the stability of the proposed MH leading to reliable solutions in different scenarios.

7. Conclusions

This paper studies the problem of scheduling single-load AGVs. The main novelty of the problem lies in simultaneously addressing the following characteristics that often arise in practice: transport requests with soft due dates and different penalty charges, a heterogeneous fleet of AGVs with different travel costs and capabilities, and partial charging of AGVs with a critical battery threshold. An MILP model was formulated with the objective of minimizing a weighted average of tardiness costs of requests and AGV travel costs. Nevertheless, the MILP model could only be applied to small-scale problems, i.e., up to 22 requests. Therefore, we proposed an ALNS-based matheuristic (MH) algorithm that can account for the heterogeneity in requests and AGVs and use various strategies to handle charging tasks in order to find good-quality solutions within a reasonable computational time in practice.

The computational experiments on a large number of scenarios generated from real-world data reveal that the solutions of the proposed MH are better than those of the MILP and the current way of scheduling employed in practice under the same time restriction. A sensitivity analysis on the variation of different parameters indicates an average of 33% improvement in current practice by using our solution approach. This improvement also results in better utilization of the available AGV fleet in addition to monetary savings.

Interesting future works may include the extension of our approach to multiple pickups and deliveries along the same route (multi-load AGVs) and the inclusion of path planning in the scheduling process.

Acknowledgments

This research work has been funded by a grant provided by Provincie Noord-Brabant in the Netherlands for the project “Advanced Manufacturing Logistics”. We also thank Koen Herps from KMWE and Roel Nieuwenhuizen from IJssel Technologies for their support in this study.

Appendix A. Layout of case study

Fig. 7. Layout used for experiments. Each box in the layout is a pickup and delivery node. The areas marked with ‘AGV parking/charging’ contain charging nodes. In total, there are 41 pickup/delivery nodes and 10 charging nodes.
Appendix B. Real data of AGV fleet

| AGV k | \( A_k^u \) | \( v_k^u \) | \( b_k^u \) | \( d_k^u \) | \( c_k^u \) | \( s_k^u \) | \( d_k^u' \) |
|-------|--------|--------|--------|--------|--------|--------|--------|
| 0     | A,B,C  | 1.0    | 40     | 0.01   | 0.02   | 2      | 30     |
| 1     | A,C,D  | 1.5    | 30     | 0.01   | 0.02   | 1      | 30     |
| 2     | E      | 1.0    | 40     | 0.01   | 0.02   | 3      | 30     |
| 3     | A,B,C  | 1.0    | 40     | 0.01   | 0.02   | 2      | 30     |
| 4     | A,C,D  | 1.5    | 40     | 0.01   | 0.02   | 1      | 30     |
| 5     | E      | 1.0    | 40     | 0.01   | 0.02   | 3      | 30     |
| 6     | A,B,C  | 1.0    | 40     | 0.01   | 0.02   | 2      | 30     |
| 7     | A,C,D  | 1.5    | 40     | 0.01   | 0.02   | 1      | 30     |
| 8     | E      | 1.0    | 40     | 0.01   | 0.02   | 3      | 30     |

Appendix C. Generation of instances

For a given interval \((a, b)\), let \( y \sim (a, b) \) and \( z \sim (a, b) \) denote a random number and an integer, respectively. The pseudocode for the instance generation is given by Algorithm 5. For the experiments, the capability requirement \( A_k^z \) of request \( r \) is sampled from the set \( A_k^z = \{ [A,B], [A,C], [C,D], [E] \} \) such that \( A_k^z \in A_k^z \). Here, a random element from set \( A_k^z \), e.g., \([A,B]\) or \([A,C]\), is denoted by \( y_k^A \).

Algorithm 5 Test instance generation procedure.

**Input:** maximum distance in layout \( d_{\text{max}} \), minimum speed of AGV \( \nu_{\text{min}} \), a constant factor_of_inflation to inflate \( d_{\text{max}} \), number of pickup and delivery nodes \( N_{\text{nodes}} \), set of request capabilities \( A_k^z \), number of requests to create \( N \), tight time-window probability \( TW \), scheduling horizon \( H \).

1. Adjust \( d_{\text{max}} \leftarrow d_{\text{max}} \times \text{factor\_of\_inflation} \)
2. \( R \leftarrow \phi \)
3. repeat
   4. if \( y \sim (0,1) < TW \) then
   5. \( e_k^z \leftarrow z \sim (0,H) \)
   6. \( L_k^z \leftarrow d_{\text{max}} + (d_{\text{max}} / \nu_{\text{min}}) (0.25 + 0.25 y(0,1)) \)
   7. else
   8. \( e_k^z \leftarrow z \sim (0,H) \)
   9. \( L_k^z \leftarrow d_{\text{max}} + (d_{\text{max}} / \nu_{\text{min}}) (1 + y(0,1)) \)
10. end if
11. \( A_k^z \leftarrow Y_k^A \)
12. \( s_k^z \leftarrow z \sim (0, N_{\text{nodes}}) \)
13. \( d_k^z \leftarrow z \sim (0, N_{\text{nodes}}) \)
14. add \( r \) to set \( R \)
15. until \( N \) requests have been created

the capability requirement \( A_k^z \) of request \( r \) is sampled from the set \( A_k^z = \{ [A,B], [A,C], [C,D], [E] \} \) such that \( A_k^z \in A_k^z \). Here, a random element from set \( A_k^z \), e.g., \([A,B]\) or \([A,C]\), is denoted by \( y_k^A \).

Appendix D. Current practice

This section presents the current practice of scheduling AGVs at IJssel Technology that makes use of a bidding-based dispatching rule (EDDBID), as illustrated in Fig. 8. In this rule, a transport request is sent to the scheduler when the material to be transported is ready at its pickup location. A new request is stored in a database. The requests in the database are sorted in increasing order of their latest delivery times. This step ensures that requests which have early deadlines are processed first according to the EDB rule. The scheduler sends a call for bids for requests in the database to idle and capable AGVs. An AGV is considered idle when it has sufficient charge \( \left( b_k^1 > b_k^1 \right) \) and is not assigned to any task. On receiving this call from the scheduler, each candidate sends a bid that is simply the distance from the AGV location to the source of a request. The scheduler then receives bids from the competing AGVs and awards the request to the best bidder, in this case, the nearest capable AGV. Note that after servicing a request, an AGV will be sent to the nearest charging station if its charge level drops below the critical threshold (the CCI heuristic).

Appendix E. Parameter tuning

The scenarios considered for parameter tuning are given in Table 10, and the complete results are reported in Table 11, where we tune the parameters in the sequence from top to bottom. Five different values for each parameter are considered, and for each value, the tuning procedure runs ten times per instance. An average deviation is obtained by averaging the standard deviations for all instances in a given scenario, and the deviations across all scenarios are averaged to get “Dev%”. The parameter value with the smallest “Dev%” is selected. This procedure is repeated until all parameters have been tuned. The selected values of the parameters are indicated in bold in Table 11.

Table 10 Parameter tuning scenarios.
| Parameter | Values tested |
|-----------|---------------|
| \( H \) | 1800 0.5 60 | 6 | |
| \( H \) | 1800 0.5 40 | 3 | |
| \( H \) | 1800 0.5 40 | 6 | |
| \( H \) | 1800 0.5 40 | 9 | |
| \( H \) | 1800 0.5 20 | 3 | |

Table 11 Parameter tuning results.

| Parameter | Values |
|-----------|--------|
| \( \psi_1 \) | Value 0.1 0.3 0.5 0.7 0.9 |
| Dev\% | 0.083 0.113 0.082 0.072 0.064 |
| \( \psi_2 \) | Value 0.1 0.3 0.5 0.7 0.9 |
| Dev\% | 0.070 0.071 0.071 0.067 0.072 |
| \( \psi_3 \) | Value 0.2 0.4 0.6 0.8 1.0 |
| Dev\% | 0.055 0.058 0.059 0.074 0.065 |
| \( \lambda \) | Value 0.1 0.3 0.5 0.7 0.9 |
| Dev\% | 0.066 0.068 0.060 0.067 0.055 |
| \( n_k \) | Value 1 3 5 7 9 |
| Dev\% | 0.076 0.070 0.063 0.085 0.069 |
| \( r \) | Value 2 4 6 8 10 |
| Dev\% | 0.068 0.059 0.059 0.067 0.054 |
| \( \phi_1 \) | Value 1 3 5 7 9 |
| Dev\% | 0.057 0.051 0.065 0.071 0.079 |
| \( \phi_2 \) | Value 2 4 6 8 10 |
| Dev\% | 0.092 0.066 0.072 0.051 0.087 |
| \( \phi_3 \) | Value 2 4 6 8 10 |
| Dev\% | 0.076 0.075 0.054 0.054 0.070 |
| \( y \) | Value 1 3 5 7 9 |
| Dev\% | 0.052 0.060 0.056 0.062 0.087 |
| \( w \) | Value 0.1 0.3 0.5 0.7 0.9 |
| Dev\% | 0.055 0.078 0.059 0.089 0.076 |
| \( \epsilon \) | Value 0.2 0.4 0.6 0.8 1.0 |
| Dev\% | 0.055 0.074 0.059 0.058 0.063 |

Appendix F. ALNS operator analysis

To better understand the benefit of each operator to the proposed method, we conduct a statistical analysis based on the work of Schiffer & Walther (2018a). The results of this analysis are presented in Table 12. Within this analysis, we remove each operator one at a time, and a percentage gap is calculated from the objective value obtained when none of the operators are removed. A negative gap indicates an improvement in the solution quality of the MH when the corresponding operator is excluded from the original set of operators. If the exclusion of more than one operator leads to improvement in solution quality, then we investigate all possible combinations of removing multiple operators at once to...
make sure that there are no adverse effects of removing them. Finally, we select the operator configuration which provides the best result.

Table 12 shows that the solution quality increased without using the SHD-D, RKI, RCD, ACD, and ACI operators. We then investigated the configurations without all these five, four out of the five, three out of the five, and two out of these five operators in more detail. Since removing the SHD-D, ACD, and ACI leads to the best result (improvement of 2%), we removed these operators after the analysis.
Table 12
Statistical analysis of operators.

| SHD | SHD-T | SHD-C | SHD-D | RRD | WRD | GRI | KKI | EGI | RCD | ACD | WCD | NCI | CCI | ACI | Gap [%] |
|-----|-------|-------|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|
| 1.00% | 0.55% | 0.39% | -0.61% | 0.04% | 0.24% | 0.13% | -0.48% | 0.03% | -0.30% | -0.59% | 0.25% | 0.21% | 0.05% | 0.53% |
| 0.00% | -1.49% | -1.46% | -1.10% | -0.99% | -0.89% | -0.82% | -0.75% | -0.06% | 0.96% | -2.00% | -1.32% | -1.25% | -1.21% | 0.00% |
| 0.00% | -0.80% | -0.76% | -0.16% | 0.03% | 0.18% | 0.22% | -1.17% | 0.05% | 0.61% | 0.03% |

(*) operator included in analysis

References

Brainport Industries Campus (2020). The evolution of the high-tech manufacturing industry. Accessed 23 August 2020 https://www.brainportindustriescampus.com/en/innovating/factory-of-the-future.

Bruglieri, M., Mancini, S., Pezzella, F., Pisacane, O., & Suraci, S. (2017). A three-phase matheuristic for the time-effective electric vehicle routing problem with partial recharges. Electronic Notes in Discrete Mathematics, 58, 95–102.

Bruglieri, M., Pezzella, F., Pisacane, O., & Suraci, S. (2015). A variable neighborhood search branching for the electric vehicle routing problem with time windows. Electronic Notes in Discrete Mathematics, 47, 221–228.

Confessore, G., Fabiano, M., & Liotta, G. (2013). A network flow based heuristic approach for optimising AGV movements. Journal of Intelligent Manufacturing, 24(2), 405–412.

Conrad, R. C., & Figliozzi, M. A. (2011). The recharging vehicle routing problem. In T. Doolen, & E. Van Akers (Eds.), Proceedings of the industrial engineering research conference. Norcross, GA: ASME.

De Ryck, M., Verstrynge, M., & Debrouwere, F. (2020). Automated guided vehicle systems, state-of-the-art control algorithms and techniques. Journal of Manufacturing Systems, 54, 152–173.

Dennis, E., Bekta, T., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. European Journal of Operational Research, 223(2), 346–359.

Ding, N., Batta, R., & Kwon, C. (2015). Conflict-free electric vehicle routing problem with capacitated charging stations and partial recharge. In Technical report. Available at: http://www.acsu.buffalo.edu/~batta/Nan%20Ding.pdf.

Ebberen, M. J., van der Heijden, M. C., & van Harten, A. (2005). Dynamic transport scheduling under multiple resource constraints. European Journal of Operational Research, 167(2), 320–335.

Erdano, S., & Miller-Hooks, E. (2012). A green vehicle routing problem. Transportation Research Part E: Logistics and Transportation Review, 48(1), 100–114.

Felipe, A., Ortulino, M. T., Righini, G., & Tirado, G. (2014). A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. Transportation Research Part E: Logistics and Transportation Review, 71, 111–128.

Goeke, D., & Schneider, M. (2015). Routing a mixed fleet of electric and conventional vehicles. European Journal of Operational Research, 245(1), 81–92.

Grandinetti, L., Guerriero, F., Pezzella, F., & Pisacane, O. (2016). A pick-up and delivery problem with time windows by electric vehicles. International Journal of Productivity and Quality Management, 15(1–2), 251–423.

Grangier, P., Gendreau, M., Lelud, F., & Rousseau, L-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. European Journal of Operational Research, 254(1), 80–91.

Guimares, L., Klabjan, D., & Almada-Lobo, B. (2013). Pricing, relaxing and fixing under lot sizing and scheduling. European Journal of Operational Research, 230(2), 399–411.

Guilbaud, A., N., Cordeau, J-F., Hourtoum, L. M., & Nygren, B. (2017). Adaptive large neighborhood search heuristics for multi-tier service deployment problems in clouds. European Journal of Operational Research, 259(3), 829–846.

Hiermann, G., Puchinger, J., Ropke, S., & Hartl, R. F. (2016). The electric fleet size and mix vehicle routing problem with time windows and recharging stations. European Journal of Operational Research, 252(3), 995–1018.

Ho, Y-C., Liu, H-C., & Yih, Y. (2012). A multiple-attribute method for concurrently solving the pickup-dispatching problem and the load-selection problem of multiple-load AGVs. Journal of Manufacturing Systems, 31(3), 288–300.

Jeong, B. H., & Randhawa, S. U. (2001). A multi-attribute dispatching rule for automated guided vehicle systems. International Journal of Production Research, 39(13), 2817–2832.

Keskin, M., &  Çatay, B. (2016). Partial recharge strategies for the electric vehicle routing problem with time windows. Transportation Research Part C: Emerging Technologies, 65, 111–127.

Keskin, M., Laporte, G., & Çatay, B. (2019). Electric vehicle routing problem with time-dependent waiting times at recharging stations. Computers & Operations Research, 107, 77–94.
