Dynamic task allocation for efficient container handling at terminals

Wanne Wisse

June 2024

Abstract

This thesis addresses the problem of Automated Guided Vehicle (AGV) scheduling by employing a Multi-Agent pickup and delivery problem (MAPD) formulation, which we identified as a good approach compared to the other approaches considered in this paper for tackling the related routing and scheduling sub-problems inherent to this domain. We adjusted a simulation environment to experiment with different schedules and routes, addressing critical factors such as collisions, reserved crane parking, and the effects of randomness and variability. To optimize AGV operations, we utilized three key heuristics: idle time, collision time, and drive time. For routing, we implemented cooperative A* with deadlock prevention, ensuring robust, collision-free task completion.We introduced a new initialization method, the greedy method, which uses simulation results to generate a starting solution. Our approach explored the solution space from this starting solution using local search algorithms, specifically random, swap, and insertion operators combined with activity time blocks. We enhanced the search process with Iterated Local Search (ILS), employing iterated greedy neighbors (IG) to improve the solution space exploration, outperforming random neighbors (RN) by maintaining beneficial solution characteristics. Periodic rescheduling was implemented to manage the randomness of crane times, with experimental results confirming its necessity for improving solution robustness. Overall, our combined heuristics and search strategies effectively minimized the total container handling time at terminals.

1 Introduction

With the surge in container shipments in recent years, there has been considerable interest in optimizing the process of loading and unloading containers from ships. Container terminals, as dynamic complex systems, comprise several interdependent planning processes.

The terminal can be divided into distinct areas: the quayside, the yard, and the hinterland. The quayside is where ships dock and containers are transferred between ships and the yard. The yard serves as temporary container storage, where containers are stacked and stored. The hinterland refers to the area between the main land and the yard, where containers are transferred by train or truck (see Figure 1 for a visual representation).

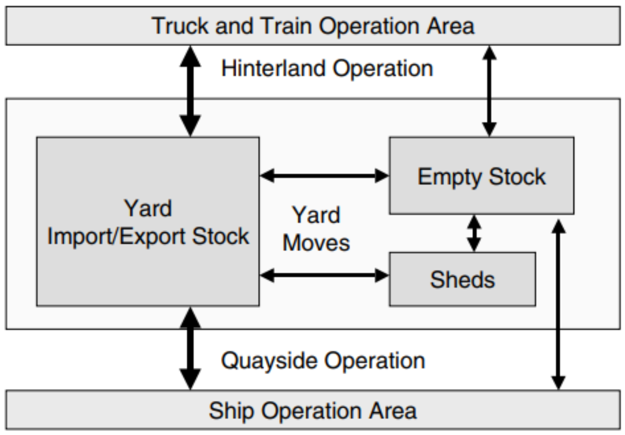


Figure 1: Schematic overview general terminal

Optimization challenges within a container terminal are distributed across these areas. Hinterland optimization involves planning truck and train movements from the mainland. The yard area focuses on efficient container storage and stacking. The quayside is concerned with ship planning, encompassing berth allocation, storage planning, and crane assignment. Berth allocation deals with the dock location of the ship, storage planning assigns containers to specific areas in the ship, and crane assignment determines which crane is assigned to which ship and how many cranes per ship.

In addition to ship planning, the quayside must address transportation challenges between the yard and the cranes. Often, this is done by manually driven trucks, but in some harbors, they use, Automated Guided Vehicles, AGVs to complete this challenge [Steenken et al. [2004]].

These area optimizations all primarily aim to minimize the berth time of ships. The longer a ship remains docked, the greater the time and costs incurred. This means all optimization problems are dependent on each other to successfully achieve the main goal. However, these optimization problems are highly complex, prompting a recent trend toward solving them independently.

This study investigates the transportation challenges of the quayside at different ports [Hudson [2023]]. At these ports, AGVs play a pivotal role: upon a ship's arrival with containers, these vehicles swiftly navigate to the ship's cranes. The cranes then loads containers onto the waiting AGVs, which subsequently transport them to designated ashore cranes for integration into container yards. This process operates bidirectionally, facilitating both loading and unloading activities.

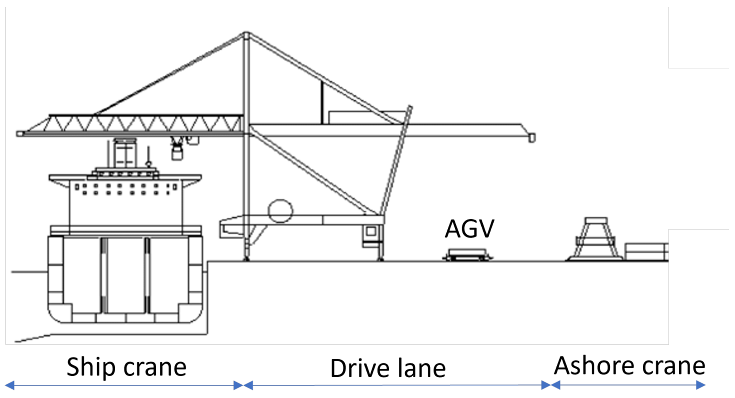


Figure 2: Schematic overview of a port

ICT-group assumes the responsibility of assigning Automated Guided Vehicles (AGVs) to containers task [ICT-group [2023]]. Their task encompasses strategic decision-making regarding which AGV is assigned to each container task and the sequence in which container tasks are performed.

The motivation of this paper is that the current implementation is going to be replaced by a new system. The reason for this is that the system is old and therefore hard to maintain. When building a new system, ICT group also wants to look at other directions compared to the current algorithm. The current approach relies on a Hungarian algorithm, as proposed by Kuhn Kuhn [1995], grounded in a distance heuristic to facilitate these assignments. However, when looking at the different activities in the terminal there are also other factors to consider, particularly in handling route conflicts due to AGVs being unable to access occupied positions on the drive lane, idle times when cranes are occupied by other AGVs, and the omission of variety in crane times for container handling.

This paper seeks to investigate the efficacy of heuristic approaches in addressing the task assignment and to assess the performance relative to the currently implemented Hungarian algorithm. The main research question which we try to answer in this paper is the following:

How can heuristics be used for assignment of tasks to AGVs to formulate a schedule that minimizes the overall time of the container handling at terminals, accounting for the inherent randomness associated with crane times?

The associated sub-questions with the main question are:

1. What are effective initialization methods recommended for initiating the search process?
2. Which local search operators, operators which try to converge to local optima, are considered effective for exploring the neighborhood of a solution?
3. What permutation operators, operators which try to jump out of local optima to nearby promising solution spaces, can be used for efficiently exploring new areas within the search space?
4. What meta heuristics can be integrated with local search and permutation operators for enhanced search performance?
5. How can random crane times be managed effectively, and what strategies perform well for rescheduling in response to them?

Aligned with this research question, several research objectives are identified:

1. **Integration of Time-Block Optimizations:** An AGV has to perform multiple actions before finishing its task. These actions cost time. The time which an AGV spends on a task can be divided in several blocks. Idle time, Current Position to s Routing Time, s to g Routing Time and Crane Time. These time blocks can be used as heuristics or objectives to develop a comprehensive and effective schedule.
2. **Determination of Rescheduling Timing:** Ascertain the appropriate timing for the execution of rescheduling operations. Recognizing that the initial schedule will have difficulties in addressing random crane times, adjustments become imperative as time elapses and deviations from the schedule intensify.
3. **Selection of Efficient Scheduling Algorithm:** Choose a scheduling algorithm that not only produces high-quality schedules but also operates with swiftness. This is crucial to align with the timing requirements of the rescheduling operation, ensuring adaptability to evolving conditions.

These research goals collectively aim to address the overarching question and contribute valuable insights to the optimization of task assignments and scheduling processes at terminals, particularly in the presence of unpredictable crane times.

2 Literature Review

The scheduling of Automated Guided Vehicles (AGVs) involves three interrelated sub-problems: task scheduling, path planning, and collision avoidance [Fazlollahtabar and Saidi-Mehrabad, 2015]. This study, however, focuses specifically on task scheduling.

Scheduling, a well-established domain, revolves around strategically allocating resources to tasks within specified time-frames, aiming to optimize one or multiple objectives [Renke et al., 2021]. In our scenario, AGVs constitute the scheduling resources, and the tasks entail container tasks. The time-frames are contingent on the task's path and crane time calculated by adding costs: the path from the current position of a_i to s_i of a task, crane time Δts, and the path from s_i to g_i. The objective to optimize would be the make-span, representing the time when the last task is completed [Renke et al., 2021].

2.1 Scheduling Methods

Various scheduling methodologies exist, including deterministic, stochastic, and dynamic paradigms.

Deterministic scheduling relies on prior knowledge of all time and task parameters.

Conversely, stochastic scheduling lacks a prior knowledge but can be modeled using probability distributions.

Dynamic scheduling operates in a fluctuating environment with unforeseeable real-time events, termed reactive events. These events are categorized as resource-related (e.g., machinery malfunctions, operator absenteeism) and job-related (e.g., urgent tasks, task cancellations) [Renke et al., 2021].

While crane times are stochastic, increased variance may transition them into dynamic events, hence our choice of a dynamic scheduling paradigm.

2.2 Handling Reactive Events

Addressing reactive events in a dynamic scheduling paradigm involves choosing between reactive scheduling, predictive-reactive scheduling, or robust pro-active scheduling. Reactive scheduling reacts to events using dispatch rules without proactive planning. Predictive-reactive scheduling adapts an initial schedule as events occur, and robust pro-active scheduling allocates extra time to accommodate unpredictable events [Renke et al., 2021].

Balancing speed and predictability is crucial when handling reactive events. Reactive scheduling aims for quick, on-the-fly local solutions, while predictive-reactive scheduling assumes minimal changes to the initial schedule, adjusting only for minor alterations. When significant deviations occur, prioritizing accuracy over speed becomes essential in schedule repair.

2.3 Rescheduling Strategies

Implementing predictive-reactive scheduling involves decisions on when and how to reschedule.

Timing for rescheduling can be periodic, event-driven, or a hybrid of both. Event-driven rescheduling is commonly preferred for its lower computational load and higher predictability.

Rescheduling strategies range from localized adjustments (schedule repair) to complete rescheduling, each affecting computational load, schedule stability, and production continuity [Ouelhadj and Petrovic, 2009].

AGV scheduling also necessitates decisions on when and how to reschedule. An option might be to reschedule when an AGV is loaded/unloaded, balancing speed and solution quality.

2.4 Dynamic Scheduling Algorithms

Beyond problem type considerations, attention shifts to algorithms for rescheduling. These algorithms can be organized into two strategies: centralized and decentralized.

Centralized solutions control all agents and tasks through a single governing entity, while decentralized approaches delegate decision-making to smaller agent groups. Decentralization minimizes scheduling delays and increases system robustness against individual machine failures but presents challenges in achieving a comprehensive global optimum [Renke et al., 2021].

2.4.1 Exact Methods

Exact solutions for small-scale AGV scheduling problems have been investigated in previous studies, as demonstrated by Bean et al. [1991] and Fazlollahtabar and Hassanli [2018]. However, their applicability is confined to small instances. Given our problem entails a minimum of 15 AGVs and over 100 container tasks, exact solutions become impractical. Furthermore, our research focuses exclusively on heuristic approaches rather than exact methods.

2.4.2 Heuristics

Heuristics, relying on problem-specific properties, provide sub-optimal yet practical solutions within reasonable time frames. Heuristics can be divided into heuristics to create an initial solution, to local search the neighborhood of the current solution and to perturbate the current solution.

The initial solution heuristics are used to create the solution to start from. Examples of this heuristics are: random initialisation, greedy construction and initialisation using dispatch rules [Ulaga et al. [2022]].

Local search heuristics are used to repair a solution or find the local optima in the neighborhood of the current solution. Examples of these operators are: job insertion, job swap, group swap, random removal, worst removal, and ”Shaw removal” [Ropke and Pisinger [2006] Xu et al. [2022]]. Additionally, a shortest distance heuristic, often combined with meta heuristics, constructs sub-optimal solutions for complete rescheduling [Chen et al. [2020], Queiroz et al. [2023]].

the perturbation heuristics are used to change the solution to escape local optima and search other promising solution spaces. Examples of perturbation heuristics are: Iterated greedy; remove certain number of jobs and place elsewhere and Random neighbor; insert a job in a other position or swap two jobs [Ropke and Pisinger [2006] Ulaga et al. [2022]].

The studies related to AGV scheduling lack exploration of heuristics addressing collisions or wait times factors we expect to significantly impact solutions in our context.

2.4.3 Meta-heuristics

As problem scales increase, numerous local solutions emerge. Several meta-heuristic solutions exist, including simulated annealing, two-stage ant colony algorithm and particle swarm optimization algorithm [Hamzeei et al. [2013], Li et al. [2019] and Qiuyun et al. [2021]]. However, these approaches strive for exceedingly optimal schedules under the assumption of infinite calculation time, which isn't practical in scenarios involving reactive events like fluctuations in crane times.

Considering our dynamic scheduling scenario with reactive events, speed becomes crucial. Research by Queiroz et al. [2023] employs Genetic Algorithm (GA) for task allocation among agents, exploring combinations of objectives using the Non-dominated Sorting Genetic Algorithm II (NSGA-II) in a dynamic setting with randomly added tasks.

A study by Jin [2016] combined AGV scheduling with quay crane scheduling. This study emphasises that AGV scheduling at container terminals is a dynamic problem. They also suggest that a scheduling horizon should be used. In the study they used a GA with a horizon of two containers per AGV.

Another study by Hu et al. [2023] adopts a hierarchical planning method, combining genetic algorithms and tabu search for a more efficient approach, taking advantage of speed without compromising on solution quality. They also adopt the shortest distance heuristic to create the initial population.

2.4.4 Artificial intelligence

Knowledge systems can utilize expert knowledge to derive practical conclusions. Notable examples include ISIS for job scheduling and OPIS for manufacturing production [Renke et al. [2021]].

Another prevalent methodology involves employing neural networks to predict schedules, necessitating the transformation of the combinatorial optimization problem into a classification problem. Weckman et al. achieved this by utilizing a genetic algorithm to solve various problems, using the obtained solutions as training data for the neural network. This method prioritizes task properties to predict ranges of priority. One limitation is the fixed schedule size, but the results demonstrate strong generalization capabilities and near-optimal solutions Weckman et al. [2008].

An alternative approach includes training a neural network with a knowledge base and using it to guide a genetic algorithm, exhibiting superior performance compared to a random initial population Noorul Haq et al. [2010].

Additionally, another strategy involves predicting a dispatch rule based on environmental characteristics such as processing time, queue size, and machine failures Mouelhi-Chibani and Pierreval [2010].

Neural networks excel in dynamic scheduling due to their rapid prediction process, yet they encounter challenges in accommodating variable input and output sizes when dealing with different amounts of AGVs or tasks for scheduling Weckman et al. [2008].

When the speed of a method becomes a concern, capturing the behavior of the algorithm in a neural network proves to be a viable approach.

Lately there has also been a shift to solving this problem using reinforcement learning. Zheng et al. applied a deep Q-network to the problem by using the amount of tasks, distances to task and current position as state. The output of the network is a combination of a schedule rule and an AGV. This paper also compared the results to a GA and for their specific simulation deep-q-networks had better performance for large instances [Zheng et al. [2022]].

In addition to deep Q-learning, Monte Carlo Tree Search (MCTS) is sometimes employed. MCTS is particularly suitable for sequential problems with large state spaces where conducting numerous simulations is feasible. The algorithm represents all possible states from the initial state as a tree, assigning a value to each state based on its desirability. During each iteration, the algorithm selects the state with the highest value and conducts a simulation from that state. The resulting value from the simulation is then propagated from the chosen state up to the root, updating the values of the states along the way. The objective is to prioritize exploration of states with higher values, thereby minimizing traversal of less desirable parts of the tree during simulation. In the context of scheduling, MCTS is employed by representing nodes as partial schedules, allowing simulations to evaluate the quality of these partial schedules using heuristics. This approach is compared with a GA, demonstrating significantly faster execution times for large problem instances [Li et al. [2021]].

3 Problem definition

The problem to be solved is defined as special instance of the Multi-agent pickup and delivery problem (MAPD) [Hu et al. [2023]]. This MAPD consists of a set of M AGVs $\{a_1, a_2, ..., a_M\}$, a set of N tasks $\{t_1, t_2, ..., t_N\}$ and the quayside which can be represented as a grid, $G(o, p)$, where o is the amount of vertices per row and p is the amount of vertices per column. An AGV a_m can be located on the grid at a position (i, j) where $i <= o$ and $j <= p$. A vertex can represent a road vertex or a crane vertex. The road vertices are vertices which are used by the AGV, a_m, to travel between crane vertices by the roads connected to the road vertices. The crane vertices can be further divided into vertices of ashore cranes and vertices of ship cranes. At these vertices AGVs can load or unload a container.

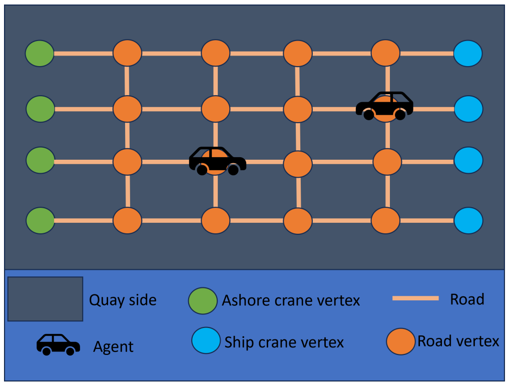


Figure 3: Problem representation

The Multi-Agent Pickup and Delivery (MAPD) problem consists of two sub-problems: the assignment problem and the routing problem. There are two approaches to address this problem: solving the sub-problems separately or integrating them into a single problem. Integrating the sub-problems into one problem expands the solution space and allows for the possibility of finding a global optimum. However, in real-world scenarios, it is crucial to find a solution within a reasonable time frame. Therefore, we have chosen to solve the sub-problems separately, with a primary focus on the assignment problem.

3.1 Goal

The goal is to assign every task, t_i, to a single AGV, a_j, in order to minimise the total time, ts, to complete all tasks.

3.2 Tasks

Table 1: Example set of tasks

TaskId	Ashore crane parking position	Ship crane parking position	Load status
1	(1,1)	(3,4)	"load"
2	(1,1)	(3,4)	"unload"
3	(2,1)	(5,4)	"unload"

Every task, t_i has a corresponding starting point, t_is and a goal point t_ig. These points represent crane vertices on the quayside, either Ashore or Ship crane vertices. A container can be loaded or unloaded on an AGV, a_i, at the crane vertices. Every t_i also has a load status, $L = \{load, unload\}$. A task, t_i, with load status, *load*, means that a container from ashore needs to be transferred to a ship. So the starting position t_is is an ashore crane vertex and the endpoint, t_ig is a ship crane vertex. When the load status was instead *unload*, The t_is and t_ig would have been swapped. The AGVs always start at a crane vertex.

An example can be found in table 1. A specific example from the table would be TaskId 1, t_1, first look at the load status, this is "load". This means in order to complete this task an AGV, a_i, first needs to get the container from ashore and then travel to the ship in order to "load" the ship. So the a_i first needs to travel to the ashore crane $(1, 1)$ which is his t_1s. When having loaded the container, the AGV needs to travel to the goal t_1g which is, in this case, a ship crane at $(3, 4)$. A schematic overview of this example can be found in figure 4.

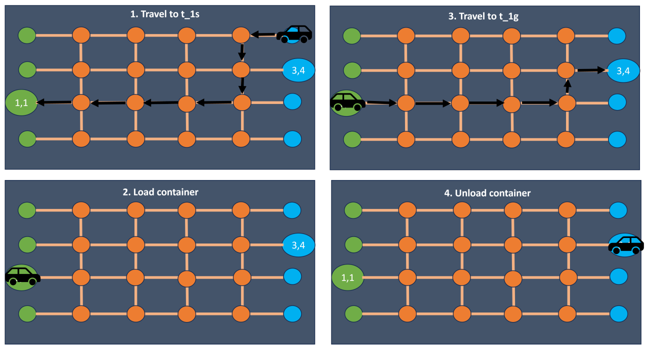


Figure 4: Schematic overview of TaskId 1 from table 1 processed by an AGV

When looking at TaskId 2, the load status is "unload", so the t_1s and t_1g are performed in opposite order.

3.3 Assignment

The goal is to assign tasks, t_i, to AGVs, a_i, to generate an assignment list, $a_it_1, a_jt_2, ..., a_kt_M$. An AGV can thus have multiple tasks but a task can only have one AGV. The AGVs need to execute their assigned tasks in the order which is specified in the list. We can look at table 2 for a specific example. AGV 1, first executes TaskId 1, when this task is done it performs TaskId 3.

3.4 Routing

When the AGVs are assigned to tasks, the AGVs need routing to the s or g of their assigned tasks, a_it_j. The routing consist of two steps: routing from current position of a_i to the s of the a_it_j, next the AGV needs to route from the s to the g of a_it_j.

Table 2: Example set of tasks assigned to AGVs

TaskId	Ashore crane parking position	Ship crane parking position	Load status	AGV
1	(1,1)	(3,4)	"load"	1
2	(1,1)	(3,4)	"unload"	2
3	(2,1)	(5,4)	"unload"	1

Routing should be done collision free. This means that two or more AGVs can not travel by the same road at the same time, ts. The AGVs also can not be at the same vertex at the same time, ts. See figure 10 for a schematic representation. This is also known as the Multi-agent Routing problem.

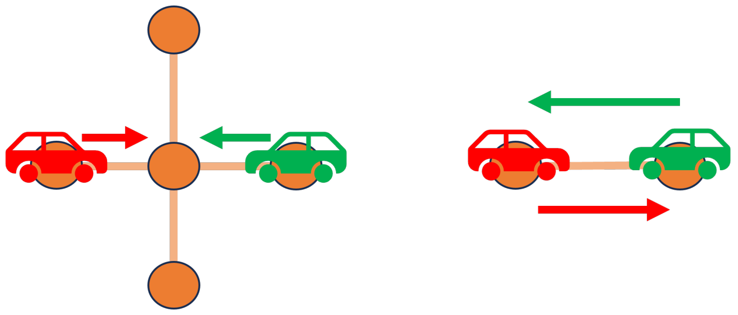


Figure 5: Collision problems: left represents multiple AGVs same vertex at same time and right represents multiple AGVs same edge at same time

4 Current Method

In the current method, ICT-group has chosen to use the Hungarian algorithm for the scheduling problem Kuhn [1995]. This method involves creating a cost matrix for all unassigned tasks, where the rows represent the tasks and the columns represent the Automated Guided Vehicles (AGVs). Each element of the cost matrix, corresponding to a combination of an AGV a_i and a task t_i, is populated with the Manhattan distance between the AGV's current position and the start position of task t_i. The AGV's position is determined by the goal position of its previously assigned task, $t_j g$.

Using the filled cost matrix, the Hungarian algorithm assigns each AGV a_i to the task t_i that minimizes the total cost, effectively trying to find the optimal task for each AGV. Once tasks are assigned, they are removed from the pool of remaining tasks, and this process is repeated until no tasks are left.

For routing, predefined routes are employed where each vehicle checks, at each time-step, to reserve the next segment of its route. If the segment is already reserved, the vehicle will wait at its current position.

5 Proposed Method

To develop a new method, we investigated the environment by consulting stakeholders, examining the old simulation, and reviewing the current method.

We characterized the environment by analyzing the activities of an AGV. The activities of an AGV are organized based on the time spent on each activity. There are multiple activities an AGV must perform to complete a single task, each with a specific time cost. An overview of these activities is provided in Figure 6. We further categorized these activities into three groups: Idle time, Drive time, and Crane time. To optimize the total time required to complete all tasks, it is crucial to minimize the time spent on each task. This involves optimizing the time spent in all three activity groups across all tasks, which can be achieved by assigning each task to the most suitable AGV at the right position in the sequence. The current method focuses solely on drive time, neglecting idle and crane time, which are also significant factors in scheduling. Since idle and crane times are challenging to predict without an actual roll out, we decided to create a simulation to evaluate these factors.

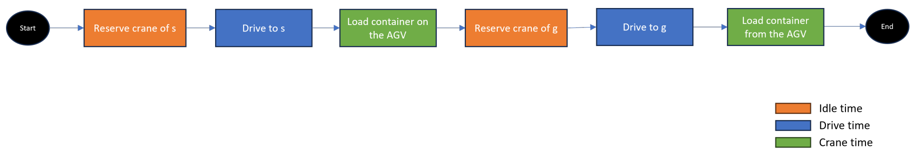


Figure 6: Overview of Activities

5.1 Algorithm overview

The processes of scheduling and routing AGVs are highly complex and can be approached from multiple perspectives. One popular method is to solve these challenges simultaneously. The advantage of this integrated approach is that it allows the algorithm to adapt routing and scheduling together, avoiding suboptimal solutions that could arise from treating them separately. However, the complexity of this combined approach can sometimes prevent the algorithm from finding a satisfactory solution for either process, potentially resulting in no solution.

Alternatively, the problem can be divided into two sub-problems: routing and scheduling, each solved separately. This approach simplifies each subproblem, increasing the likelihood of finding a solution. However, this can lead to sub-optimal overall solutions since the algorithms may not be well-coordinated. For instance, an optimal schedule might not align well with the routing policy.

For the ICT-group, robustness is important, especially to avoid AGV collisions. Thus, a deterministic and comprehensible routing policy is essential. To achieve this, we decided to address routing and scheduling separately, while maintaining some interdependence. The routing policy is deterministic and straightforward, while the scheduling process can be more complex and stochastic. This paper presents a routing policy based on the A* algorithm, which uses a pre-determined task schedule for each AGV, detailed in the Routing Policy section. The scheduling policy, based on Iterative Local Search (ILS), takes all tasks, AGVs, and the routing policy as inputs, and outputs the optimal task allocation according to the deterministic routing policy. This is discussed in section 8.

The stochastic nature of the environment, particularly variable crane times, presents another challenge. Instead of adjusting the routing and scheduling policies to handle these variations, we employ a rescheduling policy. This policy reschedules tasks using the overall policy at fixed intervals to adapt to varying crane times, as explained in section 9. An overview of the algorithms and their inter-dependencies is shown in Figure 7.

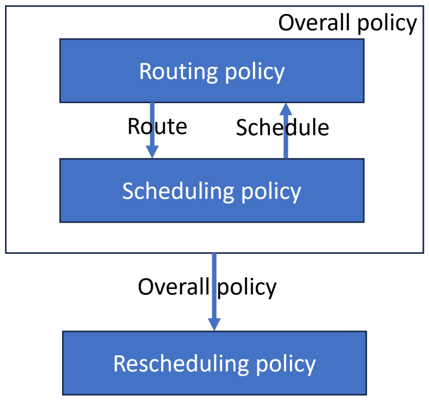
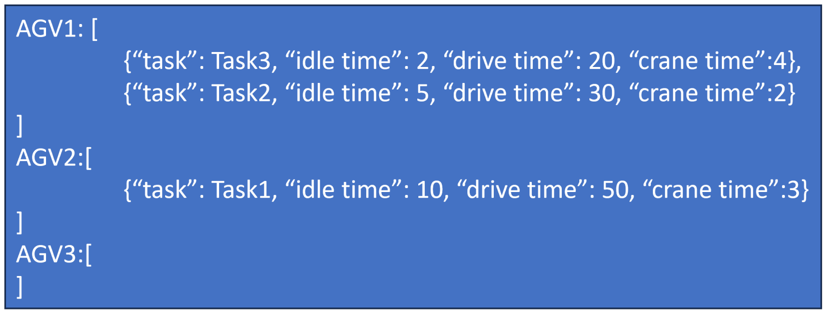


Figure 7: Overview of Algorithms

5.2 Solution Encoding

An important decision when addressing the Multi-AGV Pickup and Delivery (MAPD) problem is how to encode the solution. The chosen encoding impacts which algorithms can be used for AGV scheduling and routing problems. Given our interest in using evolutionary algorithms, we require a representation that allows easy modification of the sequence and placement of tasks. In this work, the schedule is encoded using a dictionary, where the keys are AGVs and the values are the corresponding tasks along with their performance on the three activity time blocks. An AGV's task list can be empty if it has no tasks to perform. The sequence of tasks in the list indicates the order in which the AGV will execute them. Completed tasks are deleted from the schedule.

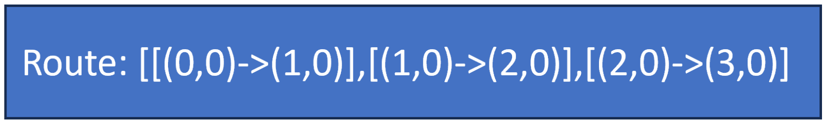
This encoding is efficient for simulation, allowing us to find the next task for an AGV in O(1) time. Additionally, it enables us to identify which tasks are eligible for local search, specifically those that are under-performing according to the three time blocks and should be switched. For the local search, tasks can be inserted or swapped. When inserting, we consider the task's position and place it before or after the target task in the schedule. If the insert is out of range of the AGV's task list, the task is placed at the start of another AGV's task list.



```
AGV1: [
        {"task": Task3, "idle time": 2, "drive time": 20, "crane time":4},
        {"task": Task2, "idle time": 5, "drive time": 30, "crane time":2}
]
AGV2:[
        {"task": Task1, "idle time": 10, "drive time": 50, "crane time":3}
]
AGV3:[
]
```

Figure 8: Example of a schedule

For routing, we do not determine all routes before the simulation starts, as the environment can change. Therefore, we have chosen to define a route policy. A policy is a set of rules executed to determine the path from the starting point of a task to its endpoint. Each time an AGV requires routing, it consults the policy for guidance. The policy provides a route in the form of a list of collision-free edges over time. We employ the A* search algorithm combined with a reservation table to handle the time dimension, known in the literature as "Cooperative A* search" Silver [2005]. Additionally, this policy includes logic for handling deadlock situations, where AGVs are unable to move.



```
Route: [[(0,0)->(1,0)],[(1,0)->(2,0)],[(2,0)->(3,0)]
```

Figure 9: Example of a route

6 Simulation

In order to verify the performance of the routing and scheduling policies a discrete-event simulation is created and used.

There are several rationales for assessing the efficacy of routing and scheduling algorithms through simulation. The current solution and other literature solely relies on shortest distances to ascertain the most efficient assignment list [Ropke and Pisinger [2006], Xu et al. [2022]]. However, our contention remains that AGVs must navigate without encountering collisions. Practically, this necessitates AGVs to search for alternative routes rather than the shortest one to reach a crane.

Another crucial aspect are the effects of occupied crane vertices. When a crane vertex is taken by AGV a_i, and another AGV, a_j, requires the same spot, it leads to a scenario where a_j has to wait a certain amount of time, we termed this the idle time, until a_i vacates the parking space.

Moreover, the influence of randomness in crane times needs to be covered, which can be validated by manipulating the variance of distributions within the simulation.

Finally, the complexity of terminal behavior, given the changes over time, underlines the need to adapt the simulation to mirror these developments.

6.1 Assumptions and Simplifications in Simulation Design

Numerous simplifications and assumptions underpin the creation of this simulation. Regarding tasks, we simplify the issue by establishing a fixed quantity of known tasks before the simulation commencement. Furthermore, we assume a lack of prescribed task order, acknowledging the complexity involved in task entry is beyond the scope of this research.

As for AGV simplifications, we limit their movement to one vertex per time step. This simplification aims to narrow the study's focus onto the task-assignment problem rather than focusing on path planning, despite the inherent connection between the two. Additionally, we presume each AGV to handle only a single container.

When considering crane times, we simplify their distribution to a normal distribution. This choice allows us to explore diverse variances, to see how this impacts the created algorithms,

6.2 Environment description

The environment is designed to emulate terminal operations, focusing on the dynamics of activities on the driving side. Within this environment, several entities interact, including cranes, Automated Guided Vehicles (AGVs), tasks, and segments categorized as non-driving or driving.

The visual representation of the simulation is presented in the following figure:

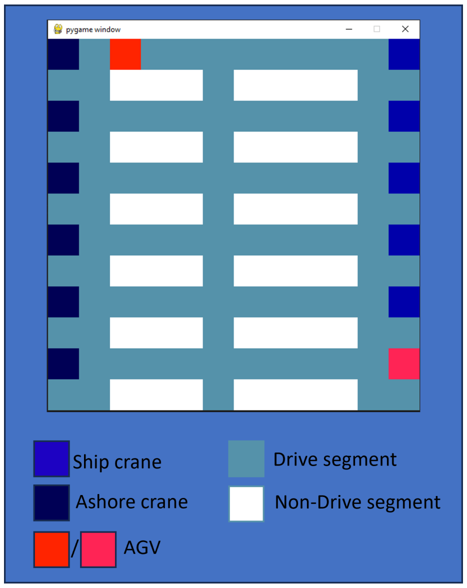


Figure 10: Visual representation

The simulation is adapted as a multi agent-based simulation within a reinforcement learning framework. This is based on a ”Gym environment” [Brockman et al. [2016]]. We have done this to make sure the simulation can be used with other optimisation frameworks such as reinforcement learning. This framework entails various functions: initialization, stepping through time, resetting, setting and getting schedules, and obtaining the current state of the environment. Let’s briefly explain each of these functions:

1. **Initialization**: This function establishes an instance of the environment, requiring parameters such as the number of AGVs and tasks, along with a routing policy.

2. **Step Function**: It progresses the simulation by one discrete time step, allowing AGVs to move along segments or facilitating the loading of containers onto AGVs when they’re positioned at cranes.

3. **Reset Function**: It restores the environment to its initial state.

4. **Set Schedule Function**: This function assigns tasks encoded in a specific format to the relevant AGVs within the environment.

5. **Get Schedule Function**: It retrieves the tasks designated for each AGV.

6. **Get State Function**: This function provides the current state of the environment, encompassing the AGVs’ locations, the passage of time, current routes, etc.

7. **Set State Function**: It updates the current state of the environment based on the given parameters.

Upon completion of all tasks, the step function returns a boolean value of "True," indicating the simulation's conclusion. Additionally, it furnishes statistical data on task performance categorized into three time groups: idle time, drive time, and crane time for each task performed by every AGV.

When using a simulation for optimization, it is crucial that the simulation runs relatively quickly, especially when employing algorithms that require numerous roll-outs, such as Monte Carlo Tree Search (MCTS).

Table 3 shows how the simulation performs as the solution scales up.

Amount AGVs	**Amount Tasks**	**Execution Time (s)**
10	60	0.003
10	80	0.008
10	100	0.010

Table 3: Execution Time for Different Amounts of Tasks with 10 AGVs

As observed, more tasks result in longer simulation times. This increase is mainly due to the routing required for all the AGVs. For every task, a route to the start and end crane must be determined. Therefore, we will gradually increase the complexity of the algorithms to ensure that they run within an acceptable amount of time.

7 Routing policy

As described earlier, a task has two points: a start crane, s, and a goal crane, g. To navigate to these locations, an AGV must follow a route that does not overlap with other AGVs' routes at the same moment in time. Most routing policies represent the environment as a grid with points and edges connecting these points. This problem is also represented as a grid, consistent with how AGVs navigate in the real world. A common challenge in standard routing algorithms is the time aspect, which is often ignored. For example, an edge occupied at time-step 1 may be free at time-step n. Without considering time, an edge could be incorrectly marked as permanently blocked.

To address the time aspect, the routing policy used in this study is based on David Silver's "Cooperative A* Search." This policy includes a structure that tracks the reservation of edges over time Silver [2005]. Specifically, there is a reservation table where each row corresponds to a discrete time step, and each column corresponds to an edge on the grid. The value of an entry in the table indicates which AGV has reserved a particular edge at a given time step.

Figure 11 illustrates Cooperative A*. In the figure, a blue car needs to move to the position at coordinates (2,2), while an orange car needs to move to the position at coordinates (1,1). Both cars reserve their routes in the reservation

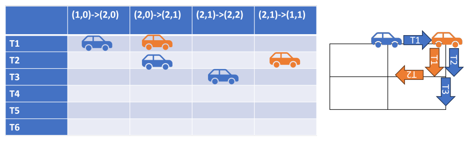


Figure 11: Cooperative A*

table, meaning they reserve the edges they traverse for each time step. This allows them to use the same edges at different times without conflict.

AGVs calculate routes and reserve edges sequentially to avoid overlapping reservations and race conditions. Using the reservation table, a standard routing algorithm can then find the best route. Routing algorithms typically use an adjacency list of points on the grid. In addition to the adjacency list, the reservation table is employed. To apply the reservation table, a time index is incorporated into the search. Each time a new edge is found using the routing algorithm, the time counter is incremented by one.

In this environment, it is crucial to balance finding routes efficiently and finding short routes that lead to optimal solutions. This study uses A* search with the Manhattan distance to the next crane as the heuristic. A* search is chosen because it finds a path relatively quickly compared to breadth-first search and finds shorter routes compared to greedy best-first search[Silver [2005]]. An algorithmic description can be found in the underneath algorithm.

Algorithm 1 Cooperative A* algorithm

```
0: Inputs: AGVs, Grid, T
0: for each AGV in AGVs do
0:   search_T ← T
0:   route ← []
0:   if AGV has no route then
0:     while route not found do
0:       if A* finds edge on grid on search_T which is not in reservation table
   then
0:         add edge to route
0:         search_T += 1
0:       end if
0:     end while
0:   end if
0: end for=0
```

Another challenge with routing is the "deadlock problem," which is relevant to the environment studied. Deadlocks occur when AGVs attempt to reserve each other's cranes, forming a cycle that causes the AGVs to freeze in place. These cycles can range from involving just two AGVs to encompassing all AGVs.

A strategy to address this challenge is to use a time counter for each AGV, which tracks how many time steps an AGV has been waiting for a crane. When this counter exceeds a certain threshold, the AGV moves to another random crane to break the cycle. This is a simple yet effective approach. However, a downside is that the AGV moving to a random crane incurs additional drive time.

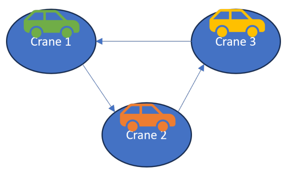


Figure 12: Deadlock Example

An example of the deadlock problem is shown in Figure 12. Here, the green AGV at crane 1 wants to reserve crane 2, which the orange AGV occupies. The orange AGV wants to reserve crane 3, which the yellow AGV occupies. The yellow AGV wants to reserve crane 1, which the green AGV occupies. Without a deadlock resolution algorithm, these AGVs will be stuck waiting for each other indefinitely.

8 Scheduling policies

Before we tackle the intricacies of dealing with randomly distributed crane times, it's essential to identify an algorithm that performs effectively in a static environment.

8.1 Selection criteria

In selecting an algorithm, two key aspects require consideration: the quality of the solution and the number of simulations required to find it. Given that each simulation takes approximately 0.003 seconds and we need to reschedule frequently in order to find the optimal schedule given the current situation, we need to focus on algorithms with a low simulation count.

To evaluate various algorithms, we established a standardized experimental setup. We tested with four different combinations of AGVs and tasks: 5x20, 5x40, and 5x60. In each scenario, we conducted 40 simulations where tasks were evenly distributed across the cranes, mirroring real-world conditions. The simulations varied in the s and g points of the tasks, ensuring a comprehensive assessment. Subsequently, we analyzed the minimum, median, and maximum results from these simulations.

8.2 Greedy

The approach to finding a solution with the fewest simulation runs involves employing a greedy search. We've employed two types of greedy strategies: the Hungarian strategy and a novel approach, which we will call the greedy strategy.

The Hungarian strategy is rooted in the original algorithm outlined in the Current Method section. It entails creating a table where each task-AGV combination is assigned a cost. This cost is calculated as the Manhattan drive time from the current position of the AGV to the start.

The new greedy strategy also involves selecting tasks one by one. Beginning with an empty schedule, a task is chosen randomly and placed as the last task for every AGV in the current schedule. Subsequently, a simulation is run for each new schedule, and the schedule with the lowest total time is chosen greedily. This process is repeated until all tasks are incorporated into the schedule. The difference with the current method lies in the fact that a simulation run looks at all time activities instead of only the drive time. However, this approach is, just as the Hungarian algorithm, sub-optimal as it selects tasks randomly in each iteration, potentially resulting in tasks being executed earlier or later without consideration of their impact on overall efficiency. A detailed description of the algorithm can be found in Algorithm: 2.

8.2.1 Experiment

For the experiment, We evaluated the two strategies according to the established experimental setup. Additionally, we tested random schedules to compare the efficacy of the strategies. The results are presented in Table 4.

Table 4: Experimental Results

Algorithm	**AGV x Task**	**Performance Measures**		
		Min	**Median**	**Max**
Hungarian	5x20	203	220	224
	5x40	278	290	302
	5x60	403	420	440
Greedy	5x20	145	153	155
	5x40	235	245	264
	5x60	365	374	382
Random	5x20	262	283	302
	5x40	342	360	376
	5x60	453	470	490

Based on the results, the new greedy strategy appears to outperform the old Hungarian algorithm, primarily because it considers idle times. However, the drawback of the greedy strategy is that it necessitates a higher number of simulations, equal to the product of the number of AGVs and tasks. Nevertheless, this quantity of simulations remains minimal when looking at the total time spend on searching for a solution.

8.3 Local Search

Having established that we can achieve better solutions with a minimal number of simulations, we can now explore the neighborhood of the solution to further improve. This can be accomplished through the use of local search operators.

In local search, task selection plays a crucial role, and it can be guided by considering the three activity time groups; idle time, drive time and crane time. An AGV-task combination with high idle and drive times indicates poor performance. By prioritizing the movement of these tasks, we can expedite the creation of better solutions faster. The heuristic used with the local search is $x1 * idletime + x2 * drivetime$ where $x1 + x2 = 1$. In the experiments we used 0.5 as value for both $x1$ and $x2$. Therefore idle and drive time have the same impact at how a solution performs. It is also possible to experiment with different values of x1 and x2. These local search operators have a prefix "heuristic".

Further we did some small sized experiments with first-improvement and best-improvement and found that first-improvement resulted in using less simulation runs and often finds the same optimum [Ochoa et al. [2010]]. Several local search operators were tested:

- Swap: Swaps the positions of two items.
- Insertion: Places a task at a different position in the schedule.
- Swap first-improvement: Continuously swaps one task with others until a better position is found. A detailed description of the algorithm can be found in Algorithm: 4.
- Insertion first-improvement: Continuously inserts one task at different positions until a better position is found. A detailed description of the algorithm can be found in Algorithm: 3.
- Heuristic Swap first-improvement: Prioritizes tasks based on idle and drive times, then continuously swaps them until a better position is found.
- Heuristic Insertion first-improvement: Prioritizes tasks based on idle and drive times, then continuously inserts them at different positions until a better position is found.
- Heuristic Swap Insertion first-improvement: Combines swapping and insertion operations, prioritizing tasks based on idle and drive times, until a better position is found.

8.3.1 Experiment

For the experiment, We tested these local search operators according to the established experimental setup. The results are presented in Table 5.

Table 5: Experimental Results

Algorithm	**AGV x Task**	**Performance Measures**		
		Min	**Med**	**Max**
Swap	5x20	143	145	150
	5x40	235	252	253
	5x60	360	371	376
Insertion	5x20	142	142	150
	5x40	236	250	253
	5x60	363	373	376
Swap first-improvement	5x20	132	136	140
	5x40	233	243	253
	5x60	361	375	376
Insertion first-improvement	5x20	140	141	145
	5x40	232	248	253
	5x60	**359**	374	376
Heuristic Swap first-improvement	5x20	132	135	140
	5x40	235	242	253
	5x60	370	373	376
Heuristic Insertion first-improvement	5x20	133	136	141
	5x40	235	249	253
	5x60	365	373	376
Heuristic Swap Insertion first-improvement	5x20	**127**	**129**	**131**
	5x40	235	243	245
	5x60	365	**372**	376
Swap Insertion first-improvement	5x20	129	134	138
	5x40	235	**238**	**250**
	5x60	365	373	376

From the table, it's evident that the first-improvement search operators outperform the random operators. Additionally, combining the swap and insertion neighborhoods leads to the discovery of new and improved solutions.

Once the local searcher has converged, we can stop the search to conserve simulation runs. To determine convergence, we have set a criterion: if the algorithm fails to find a new solution after a certain number of episodes, we terminate the local search and infer that it has reached a local optimum.

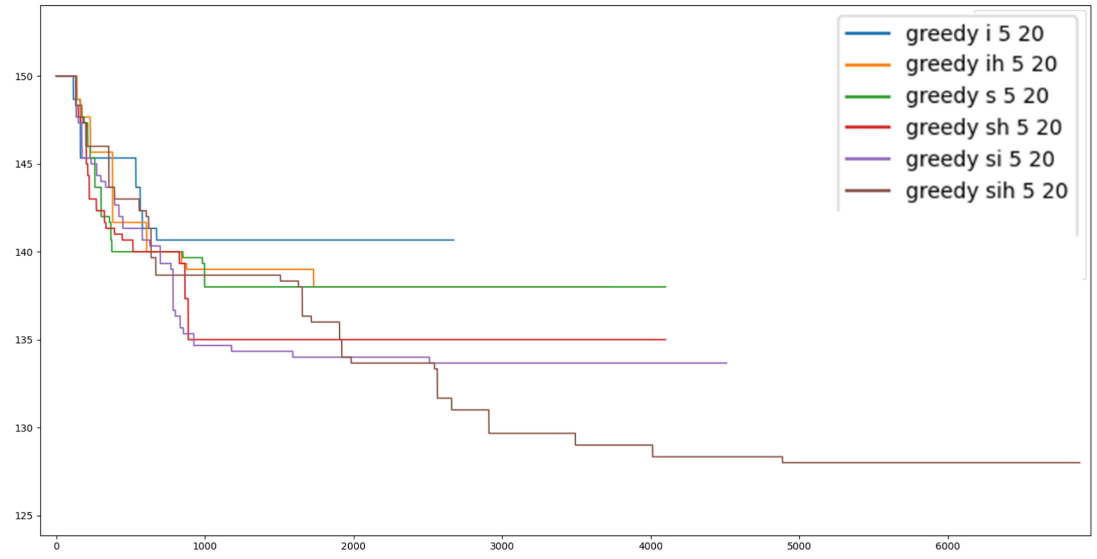


Figure 13: Results local search over time; the number of simulations required to converge to a local optimum for 5 AGVs and 20 tasks.

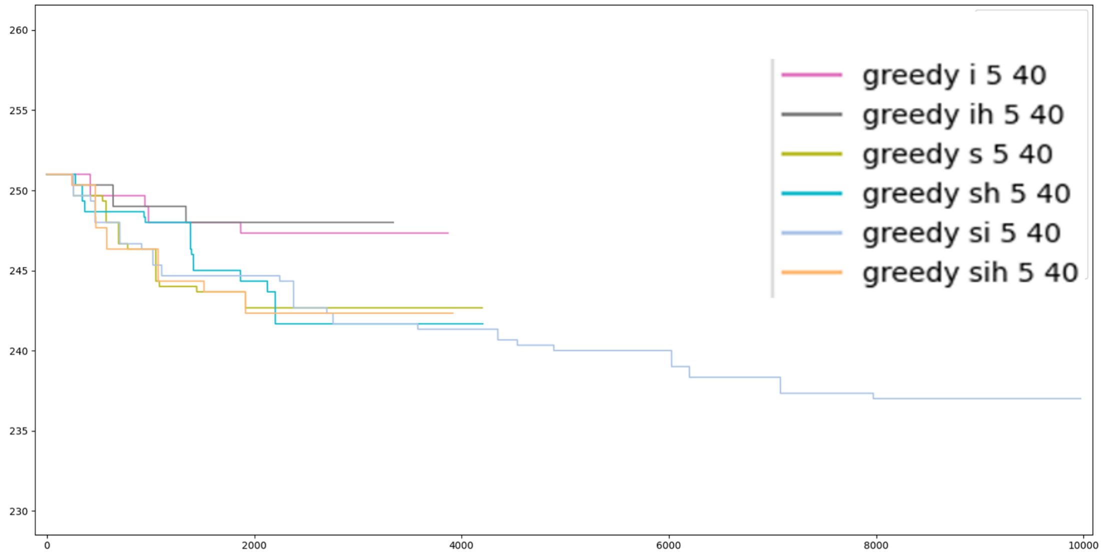


Figure 14: Results local search over time; the number of simulations required to converge to a local optimum for 5 AGVs and 40 tasks.

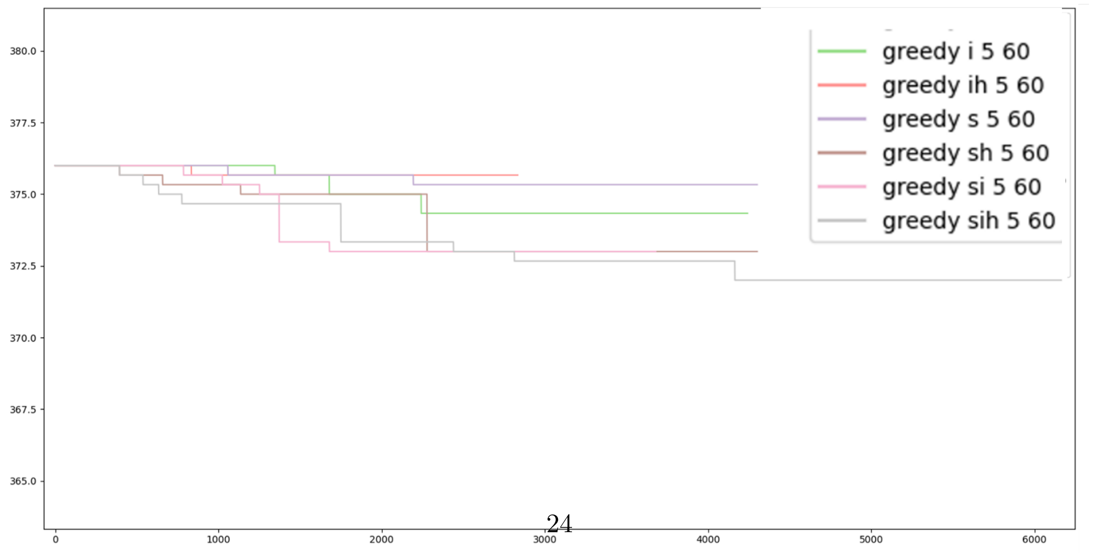


Figure 15: Results local search over time; the number of simulations required to converge to a local optimum for 5 AGVs and 60 tasks.

Figure 13, 14, 15 showcases the number of simulations needed to achieve the local optimum. In these figures the x-axis represent the amount of simulation runs, the y-axis represents the ts of the simulation. In the legend the s represents swap, i represents insertion and h represents heuristic, Notably, local searching in larger instances, such as the 5x40 scenario, can already require up to 10,000 simulation runs.

8.4 ILS

Having explored the immediate neighborhood of the greedy solution, we can now venture into nearby neighborhoods to potentially obtain better results. This can be accomplished using ILS (Iterated Local Search), an algorithm that combines local search with perturbations—small changes to the solution to reach superior neighborhoods. The concept involves iterative improving upon a given solution. To execute ILS, a perturbation or mutation operator is essential to modify the current solution and transition to another neighborhood [Ulaga et al. [2022]]. A detailed description of the algorithm can be found in Algorithm: 7.

In this study, we examine two perturbation operators:

- IG (Iterative Greedy): Selects x tasks from the current schedule and greedily places them back into the schedule. A detailed description of the algorithm can be found in Algorithm: 5.

- RN (Random Neighbor): Selects x tasks from the current schedule and randomly places them back into the schedule. A detailed description of the algorithm can be found in Algorithm: 6.

8.4.1 Experiment

For the experiment, We tested the heuristic swap insertion local searcher in combination with the two perturbation operators according to the established experimental setup. The results are presented in Table 6.

Table 6: Experimental Results

Algorithm	AGV x Task	Performance Measures		
		Min	Med	Max
IG Heuristic Swap Insertion best-first 1	5x20	**124**	129	131
	5x40	234	243	**235**
	5x60	363	370	376
IG Heuristic Swap Insertion best-first 3	5x20	126	**127**	**129**
	5x40	**220**	**229**	**235**
	5x60	**355**	**360**	**370**
IG Heuristic Swap Insertion best-first 5	5x20	125	**127**	128
	5x40	220	**229**	**235**
	5x60	363	372	374
RN Heuristic Swap Insertion best-first 1	5x20	127	128	129
	5x40	230	241	245
	5x60	364	370	376
RN Heuristic Swap Insertion best-first 3	5x20	127	129	131
	5x40	235	243	245
	5x60	362	372	376
RN Heuristic Swap Insertion best-first 5	5x20	127	129	131
	5x40	235	243	245
	5x60	362	372	376

From the results, it's evident that IG outperforms RN. One possible explanation for this could be that IG generates solutions that are still in the nearby neighborhood of the current solution. This suggests that even small adjustments to the schedule can significantly impact its performance, highlighting the complexity of the environment's behavior.

Looking at the IG algorithms, we observe that switching three tasks yields the best results across all instances. It might be worthwhile to explore combining different jump factors, starting with a small number of tasks and adjusting based on performance to optimize the search process.

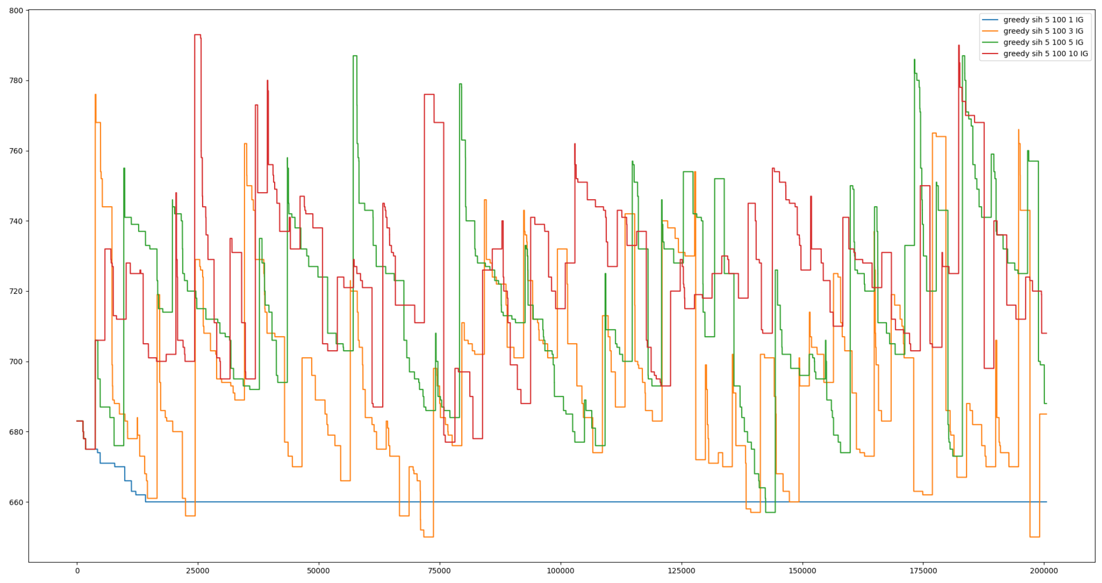


Figure 16: Results ILS over time; the number of simulations required to converge to a local optimum. The blue line represents IG with 1 mutated task, the orange line represents IG with 3 mutated tasks, the green line represents IG with 5 mutated tasks and the red line represents IG with 10 mutated tasks

In figure 16 we tested an even larger instance with 100 tasks. In this figure the x-axis represent the amount of simulation runs, the y-axis represents the ts of the simulation. Every time the results get worse means a jump in the solution space, after this jump it tries to local search again, converging in sometimes even better solutions, again we can observe the best results by switching three tasks.

However, as ILS requires a rapid increase in simulation runs, and considering the objective of this study is to develop an algorithm capable of finding solutions quickly for rescheduling purposes, we have chosen not to delve further into investigating more complex algorithms such as Genetic algorithms.

9 Rescheduling Policy

In this section, we introduce variability into the scenario and use a rescheduling policy in combination with the routing and scheduling policy, initially designed for a static environment, to adapt to these changes.

Randomness is introduced by modeling crane times as a probability distribution based on empirical data. According to the literature, crane times can be seen as reactive events, which are job or task-related. Consequently, we can model this as a stochastic scheduling problem. However, the real environment also includes unpredictable events, such as crane operator breaks or delayed ships, which are not yet simulated. Therefore, we also handle the crane-times within the dynamic scheduling paradigm.

To address reactive events, we employ a predictive-reactive scheduling approach. This method involves making incremental adjustments to the initial schedule to accommodate changes in the real environment. We selected this approach because we assume that not every reactive event necessitates a complete rescheduling. Additionally, this approach minimizes the number of simulation runs, maintaining algorithm efficiency, whereas starting from scratch would require significantly more simulations.

We reschedule periodically because the timing of reactive events, such as a crane operator's break, is unpredictable. For instance, we reschedule every ts time steps.

An example illustrating the necessity of rescheduling is as follows: consider two AGVs, AGV1 and AGV2. ILS initially distributes tasks equally among the AGVs, considering idle time, crane time, and drive time, resulting in a schedule like {AGV1: [Task1, Task2, Task3], AGV2: [Task4, Task5, Task6]}. If AGV1 encounters an issue with its vehicle or experiences a high crane time while performing Task1, it will pause. Meanwhile, AGV2 completes Task4, Task5, and Task6. Without rescheduling, AGV2 will idle, wasting time, while it could have taken over Task2 or Task3 from AGV1. Periodic rescheduling would redistribute tasks, resulting in a more efficient schedule.

The procedure is as follows: there are two environments—the real and the simulation environments. The real environment is where AGVs execute tasks and crane times are unpredictable. Initially, before the real environment begins, the simulation environment, using the positions of AGVs from the real environment, applies Iterative Local Search (ILS) to find a starting solution. In the simulation environment, crane times are set to their mean values. Once a solution is found, the real environment operates for t time steps. After every t time steps, ILS is applied again in the simulation environment, using the current state of the real environment, to find a new solution until all tasks are finished.

9.1 Experiment

In this experiment, we maintain the same setup as in the static environment, but introduce a rescheduling interval of 5 time steps, which is based on experiments on smaller instances. Further exploration may involve testing with longer intervals to gauge their impact.

Table 7: Experimental Results

Algorithm	**AGV x Task**	**Performance Measures**		
		Min	**Med**	**Max**
no rescheduling	5x20	140	150	155
	5x40	260	273	284
	5x60	450	487	502
rescheduling	5x20	**133**	**142**	**150**
	5x40	**240**	**243**	**270**
	5x60	**396**	**413**	**449**

From the results in table 7 we can see that rescheduling helps in dealing with randomness. Especially in larger instances we can see that without rescheduling the amount of time-steps will increase overtime. With the rescheduling procedure we can see that this deviation can be kept to a minimum.

10 Conclusion

From the original textual problem representation, we concluded that the Multi-Agent Decision Process (MADP) formulation can be used for addressing the given problem. This conclusion is primarily based on the routing and scheduling sub-problems, which are fundamental to AGV scheduling.

We identified three main activity time blocks essential for optimizing the total time required for AGVs to complete all container tasks: idle time, collision time, and drive time. To experiment with different schedules and routes, we adjusted a simulation environment. This decision was driven by the need to handle collisions, reserved crane parking places, and to examine the effects of randomness and variability in the problem domain.

For the routing policy, we employed cooperative A* routing with deadlock prevention to route AGVs based on a given schedule. This policy is robust, ensuring that collisions are avoided and container tasks are always completed given any schedule.

To determine a schedule, we initially search for a feasible solution. Previous work only considered drive time from the activity time blocks as a heuristic. However, with our simulation, we could evaluate how other activity time blocks perform with the given solution. We introduced a new initialization method, the greedy method, which leverages the simulation to generate a starting solution.

Using this initial solution, we explored its neighbourhood by employing various local search algorithms such as random, swap, and insertion operators in combination with the activity time blocks. Our experiments indicated that a combination of swap and insertion operator neighbourhoods with the activity time blocks yielded the best performance.

We further enhanced our search process by employing Iterated Local Search (ILS), using the local search operators to broaden the search region and identify other optima. We experimented with random neighbors (RN) and iterated greedy neighbors (IG) as mutation operators. IG outperformed RN as it altered the solution sufficiently to explore new neighborhoods without excessive deviation from the current solution. Our experiments demonstrated that minor modifications to the solution could lead to significantly different solution spaces.

To address the randomness of crane times, we implemented a periodic rescheduling strategy. Our experiments confirmed that rescheduling is essential for managing randomness, as results with rescheduling were superior. Overall, we conclude that various heuristics, such as activity time blocks combined with local and global search strategies, can be effectively utilized for task assignment. This approach results in a schedule that minimizes the overall time for container handling at terminals, outperforming the current Hungarian algorithm. The rescheduling strategy also effectively mitigates the inherent randomness of crane times.

11 Discussion

In this research, we have demonstrated that combining heuristics with a rescheduling procedure can be effective for task assignment in automated guided vehicle (AGV) scheduling. We implemented three initialization methods, with the greedy method performing the best. In each iteration, the algorithm randomly selects a task and places it in the optimal position within the current schedule. The random selection of tasks also determines the sequence of task execution, whether early or late. It would be beneficial to develop a method for predicting in advance which tasks should be scheduled early and which should be scheduled later.

We also experimented with different local search operators, such as swap and insertion, and combined them with meta-heuristics using Iterative Greedy and Random Neighbor mutation strategies. These methods proved effective in finding solutions with low completion times for this problem. However, evaluating the true effectiveness of the solutions remains challenging due to the lack of knowledge about the global minimum time for this problem. Our approach begins with an initial solution and explores the neighboring solution space. This method is advantageous because it yields better results than Hungarian scheduling and can find a sub-optimal solution within a reasonable number of simulation runs. However, this strategy may lead to local optima, which can significantly differ in quality from the global optimum.

To address this, we conducted some tests using Multi-start Local Search (MLS) from random solutions to obtain a broader overview of the search space. These tests did not yield optima of significantly better quality. This underscores the inherent trade-off in search problems between exploration and exploitation. Given more simulation runs and extended research time, it would be beneficial to explore the search space more extensively.

Simulation design should accurately represent the real-world environment, particularly in complex settings such as a harbour, where factors such as AGV availability, route maintenance, and changes in storage locations can vary. We consulted stakeholders and reviewed related literature to incorporate key aspects, but there are also some simplifications made, especially in routing and timing. For example, we assumed that AGVs travel exactly one unit to the right, left, up, or down in one time step, which does not account for variations in AGV speed or turning degrees. Despite these limitations, simulation testing proved valuable for understanding both the algorithms and the environment. Simulations provide a visual representation of system dynamics, which aids in stakeholder discussions and in analysing algorithm behaviour to devise new solutions.

Our algorithms were designed with the goal of computing solutions quickly to facilitate more frequent rescheduling. Given that simulations consume the majority of computation time, the number of simulations is crucial for designing a fast algorithm. Currently, the simulation is implemented in Python, a relatively slow language unless optimized with C++ libraries. Reducing simulation time would enable more simulation runs, allowing us to employ algorithms that require more simulations or explore more of the search space.

During our experiments, we tested various parameters but kept some constants that could be varied to potentially yield new solutions. For Iterated Local Search (ILS), we experimented with the parameter of the number of tasks to mutate. In the experiments we used constant values. However a dynamic approach could start with a small number of tasks and increase it if no optimum is found after several iterations, then decrease it when necessary to avoid overshooting the optimum.

The AGV scheduling and routing problems were defined as multi-agent decision processes (MADP) solved by a central agent. An alternative formulation would treat routing and scheduling as a single problem, potentially facilitating the discovery of a global optimum due to the interdependence of the subproblems. Instead of a central agent scheduling all tasks, each AGV could also act as an agent choosing its tasks based on the environment. This decentralized approach eliminates the need for a complete upfront schedule, continuous rescheduling and routing strategies.

12 Appendix

In this section, we detail some of the algorithms that performed well during the experiments.

12.1 Initialisation algorithm

Algorithm 2 Greedy Scheduling

```
0: $initial\_solution \leftarrow$ {AGV : [] for AGV in AGVs}
0: $tasks\_to\_schedule \leftarrow$ [Tasks]
0: $AGVs \leftarrow$ [AGVs]
0: $sim \leftarrow$ Simulation
0: while len($tasks\_to\_schedule$) $\neq 0$ do
0:   $task\_to\_schedule \leftarrow$ random.choice($tasks\_to\_schedule$)
0:   $best\_AGV \leftarrow$ None
0:   $best\_time \leftarrow \infty$
0:   for each $AGV$ in $AGVs$ do
0:     $new\_schedule \leftarrow$ copy($initial\_solution$)
0:     $new\_schedule[AGV].append(task\_to\_schedule)$
0:     $sim\_result \leftarrow sim.run(new\_schedule)$
0:     if $sim\_result.time < best\_time$ then
0:       $best\_time \leftarrow sim\_result.time$
0:       $best\_AGV \leftarrow AGV$
0:     end if
0:   end for
0:   $initial\_solution[best\_AGV].append(task\_to\_schedule)$
0:   $tasks\_to\_schedule.remove(task\_to\_schedule)$
0: end while=0
```

12.2 Local search algorithms

Algorithm 3 Local Search Insertion First Improvement

```
0: function LOCAL SEARCH_INSERTION_FIRST_IMPROVEMENT(current_schedule,
   sim, iterations)
0:   $(AGV\_index, task\_index) \leftarrow$ GET_RANDOM_TASK_TO_MOVE(current_schedule)
0:   $task \leftarrow current\_schedule[AGV\_index][task\_index]$
0:   $current\_time \leftarrow$ sim.run(current_schedule)
0:   $tried\_positions \leftarrow []$
0:   TRIED_POSITIONS.APPEND($(AGV\_index, task\_index)$)
0:   while iterations $> 0$ do
0:     $(rand\_AGV\_index, rand\_task\_index) \leftarrow$
0:         GET_RANDOM_POSITION_FROM_SCHEDULE(current_schedule,
   tried_positions)
0:     TRIED_POSITIONS.APPEND($(rand\_AGV\_index, rand\_task\_index)$)
0:     current_schedule[AGV_index].DELETE(task)
0:     current_schedule[rand_AGV_index].INSERT(task, rand_task_index)
0:     $new\_time \leftarrow$ sim.run(current_schedule)
0:     if $new\_time < current\_time$ then
0:       return current_schedule
0:     else
0:       current_schedule[rand_AGV_index].DELETE(task)
0:       current_schedule[AGV_index].INSERT(task, task_index)
0:     end if
0:     iterations $\leftarrow$ iterations - 1
0:   end while
0: end function=0
```

Algorithm 4 Local Search Swap First Improvement

```
0: function LOCAL SEARCH_SWAP_FIRST_IMPROVEMENT(current_schedule, sim, iterations)
0:   $(AGV\_index, task\_index) \leftarrow$ GET_RANDOM_TASK_TO_MOVE(current_schedule)
0:   $task \leftarrow current\_schedule[AGV\_index][task\_index]$
0:   $current\_time \leftarrow$ sim.run(current_schedule)
0:   $tried\_positions \leftarrow []$
0:   TRIED_POSITIONS.APPEND($(AGV\_index, task\_index)$)
0:   while $iterations > 0$ do
0:     $(rand\_AGV\_index, rand\_task\_index) \leftarrow$ GET_RANDOM_POSITION_FROM_SCHEDULE(current_schedule, tried_positions)
0:     $rand\_task \leftarrow current\_schedule[rand\_AGV\_index][rand\_task\_index]$
0:     $current\_schedule[AGV\_index][task\_index] \leftarrow rand\_task$
0:     $current\_schedule[rand\_AGV\_index][rand\_task\_index] \leftarrow task$
0:     TRIED_POSITIONS.APPEND($(rand\_AGV\_index, rand\_task\_index)$)
0:     $new\_time \leftarrow$ sim.run(current_schedule)
0:     if $new\_time < current\_time$ then
0:       return current_schedule
0:     else
0:       $current\_schedule[AGV\_index][task\_index] \leftarrow task$
0:       $current\_schedule[rand\_AGV\_index][rand\_task\_index] \leftarrow rand\_task$
0:     end if
0:     $iterations \leftarrow iterations - 1$
0:   end while
0: end function=0
```

12.3 Mutation algorithms

Algorithm 5 Iterative Greedy

```
0: function ITERATIVE_GREEDY(current_schedule, sim, n)
0:   $(AGV\_indices, task\_indices) \leftarrow$ GET_N_RANDOM_TASKS(current_schedule,
   sim, n)
0:   $tasks\_to\_reinsert \leftarrow []$
0:   for $i \leftarrow 0$ to $|\text{AGV\_indices}| - 1$ do
0:     $AGV\_index \leftarrow AGV\_indices[i]$
0:     $task\_index \leftarrow task\_indices[i]$
0:     $task \leftarrow current\_schedule[AGV\_index][task\_index]$
0:     CURRENT_SCHEDULE[AGV_INDEX].DELETE(task)
0:     $tasks\_to\_reinsert.append(task)$
0:   end for
0:   for $task$ in $tasks\_to\_reinsert$ do
0:     $best\_time \leftarrow \infty$
0:     $best\_AGV \leftarrow$ None
0:     $best\_task\_position \leftarrow$ None
0:     for $AGV$ in CURRENT_SCHEDULE.KEYS do
0:       for $position \leftarrow 0$ to $|\text{current\_schedule[AGV]}|$ do
0:         CURRENT_SCHEDULE[AGV].INSERT(task, position)
0:         $new\_time \leftarrow$ sim.run(current_schedule)
0:         if $new\_time < best\_time$ then
0:           $best\_time \leftarrow new\_time$
0:           $best\_AGV \leftarrow AGV$
0:           $best\_task\_position \leftarrow position$
0:         end if
0:         CURRENT_SCHEDULE[AGV].DELETE(task)
0:       end for
0:     end for
0:     CURRENT_SCHEDULE[BEST_AGV].INSERT(task, best_task_position)
0:   end for
0:   return current_schedule
0: end function=0
```

Algorithm 6 Random Neighbor

```
function RANDOM_NEIGHBOR(current_schedule, sim, n)
  $(AGV\_indices, task\_indices) \leftarrow$ GET_N_RANDOM_TASKS(current_schedule, sim, n)
  $tasks\_to\_reinsert \leftarrow []$
  for $i \leftarrow 0$ to |AGV_indices| $- 1$ do
    $AGV\_index \leftarrow AGV\_indices[i]$
    $task\_index \leftarrow task\_indices[i]$
    $task \leftarrow current\_schedule[AGV\_index][task\_index]$
    CURRENT_SCHEDULE[AGV_INDEX].DELETE(task)
    $tasks\_to\_reinsert.append(task)$
  end for
  for $task$ in $tasks\_to\_reinsert$ do
    $(random\_AGV\_index, random\_task\_index) \leftarrow$ GET_RANDOM_POSITION_FROM_SCHEDULE(current_schedule)
    CURRENT_SCHEDULE[RANDOM_AGV_INDEX].INSERT(task, random_task_index)
  end for
  return current_schedule
end function=0
```

12.4 Iterative local Search, ILS

Algorithm 7 Iterative Local Search

```
function ITERATIVE_LOCAL_SEARCH($sim, amount\_mutated\_tasks, iterations$)
  $global\_schedule, global\_time, iterations \leftarrow$
    GREEDY_INIT($sim, iterations$) or
    RANDOM_INIT( $iterations$) or
    HUNGARIAN_INIT($iterations$)
  while $iterations > 0$ do
    $mutated\_schedule, iterations \leftarrow$
      ITERATIVE_GREEDY($global\_schedule, sim, amount\_mutated\_tasks$)or
      RANDOM_NEIGHBOR($global\_schedule, sim, amount\_mutated\_tasks$)
    $local\_schedule, local\_time, iterations \leftarrow$
      LOCALSEARCH_SWAP_FIRST_IMPROVEMENT($mutated\_schedule, sim, iterations$)or/and
      LOCALSEARCH_INSERTION_FIRST_IMPROVEMENT($mutated\_schedule, sim, iterations$)or/and
      HEURSITIC_LOCALSEARCH_INSERTION_FIRST_IMPROVEMENT($mutated\_schedule, sim, iterations$)or/
      HEURSITC_LOCALSEARCH_SWAP_FIRST_IMPROVEMENT($mutated\_schedule, sim, iterations$)
    if $local\_time < global\_time$ then
      $global\_time \leftarrow local\_time$
      $global\_schedule \leftarrow local\_schedule$
    end if
  end while
  return $global\_schedule$
end function=0
```

References

James C Bean, John R Birge, John Mittenthal, and Charles E Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016.

Guang Chen, Jing Hou, Jinhu Dong, Zhijun Li, Shangding Gu, Bo Zhang, Junwei Yu, and Alois Knoll. Multiobjective scheduling strategy with genetic algorithm and time-enhanced A* planning for autonomous parking robotics in high-density unmanned parking lots. *IEEE/ASME Transactions on Mechatronics*, 26(3):1547–1557, 2020.

Hamed Fazlollahtabar and Samaneh Hassanli. Hybrid cost and time path planning for multiple autonomous guided vehicles. *Applied Intelligence*, 48:482–498, 2018.

Hamed Fazlollahtabar and Mohammad Saidi-Mehrabad. Methodologies to optimize automated guided vehicle scheduling and routing problems: A review study. *Journal of Intelligent & Robotic Systems*, 77:525–545, 2015.

Mahdi Hamzeei, Reza Zanjirani Farahani, and Hannaneh Rashidi-Bejgan. An exact and a simulated annealing algorithm for simultaneously determining flow path and the location of P/D stations in bidirectional path. *Journal of Manufacturing Systems*, 32(4):648–654, 2013.

Enze Hu, Jianjun He, and Shuai Shen. A dynamic integrated scheduling method based on hierarchical planning for heterogeneous AGV fleets in warehouses. *Frontiers in Neurorobotics*, 16:1053067, 2023.

Hudson. ECT-Delta. *https://www.ect.nl/nl/terminals/hutchison-ports-ect-delta*, 2023.

ICT-group. Ict-group. *https://www.ict.eu/en*, 2023.

Jian Jin. Multi-AGV scheduling problem in a GV scheduling problem in automated container terminal. *Journal of Marine Science and Technology*, 24:5, 2016.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1995.

Guomin Li, Xinyu Li, Liang Gao, and Bing Zeng. Tasks assigning and sequencing of multiple AGVs based on an improved harmony search algorithm. *Journal of Ambient Intelligence and Humanized Computing*, 10:4533–4546, 2019.

Kexin Li, Qianwang Deng, Like Zhang, Qing Fan, Guiliang Gong, and Sun Ding. An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. *Computers & Industrial Engineering*, 155:107211, 2021.

Wiem Mouelhi-Chibani and Henri Pierreval. Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58(2): 249–256, 2010.

A Noorul Haq, T Radha Ramanan, Kulkarni Sarang Shashikant, and R Sridharan. A hybrid neural network–genetic algorithm approach for permutation flow shop scheduling. *International Journal of Production Research*, 48(14): 4217–4231, 2010.

Gabriela Ochoa, Sébastien Verel, and Marco Tomassini. First-improvement vs. best-improvement local optima networks of NK landscapes. In *International Conference on Parallel Problem Solving from Nature*, pages 104–113. Springer, 2010.

Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12:417–431, 2009.

Tao Qiuyun, Sang Hongyan, Guo Hengwei, and Wang Ping. Improved particle swarm optimization algorithm for AGV path planning. *IEEE Access*, 9:33522–33531, 2021.

Ana Carolina Queiroz, Alex Vieira, and Heder Bernardino. Solving multi-agent pickup and delivery problems using multiobjective optimization. *Journal of Intelligent & Robotic Systems*, 109(2):26, 2023.

Liu Renke, Rajesh Piplani, and Carlos Toro. A review of dynamic scheduling: Context, techniques and prospects. *Implementing Industry 4.0: The Model Factory as the Key Enabler for the Future of Manufacturing*, pages 229–258, 2021.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

David Silver. Cooperative-A*. *https://www.davidsilver.uk/wp-content/uploads/2020/03/coop-path-AIIDE.pdf*, 2005.

Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research: A classification and literature review. *OR Spectrum*, 26:3–49, 2004.

Lucija Ulaga, Marko urasević, and Domagoj Jakobović. Local search based methods for scheduling in the unrelated parallel machines environment. *Expert Systems with Applications*, 199:116909, 2022.

Gary R Weckman, Chandrasekhar V Ganduri, and David A Koonce. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19:191–201, 2008.

Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. Multi-goal multi-agent pickup and delivery. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9964–9971. IEEE, 2022.

Xiyan Zheng, Chengji Liang, Yu Wang, Jian Shi, and Gino Lim. Multi-AGV dynamic scheduling in an automated container terminal: A deep reinforcement learning approach. *Mathematics*, 10(23):4575, 2022.