A Local Search-Based Heuristic for Optimizing AGV Routing in Automated Port Environment

Tess Nouy[1], Ghassen Cherif [2], Sandra Ulrich Ngueveu[3] and Mikhail Zakharov [4]

***Abstract*— The use of Automated Guided Vehicles for container loading and unloading has emerged as a key solution in response to the increasing demand for port efficiency, driven by the growing volume of port operations. We formulate a routing problem, extended from existing literature, that determines the trajectories of these vehicles in an automated port environment while ensuring collision-free paths. We show that when the physical paths of the AGVs is fixed the problem reduces to a job shop scheduling problem with blocking constraints and no swap allowed, very little studied in the literature. We also develop an Iterative Path Reinsertion algorithm to improve the trajectories of the AGV for a given solution. Results show the efficiency of our approach both in terms of solution quality and of computational time for instances of various sizes. The conclusion outlines future research directions.**

I. INTRODUCTION

Over the past decades, the reduction in sea transportation costs, driven by the increasing scale of freight operations, has led to a significant rise in transported volumes. This trend has resulted in larger vessel sizes and, consequently, increased port infrastructure capacities to accommodate ship docking. To address these challenges, ports must improve the efficiency and speed of loading and unloading operations.

A promising approach involves the implementation of automated, fast, and robust cargo handling solutions. In this context, Automated Guided Vehicles (AGVs) and containers are used to facilitate the transfer of goods. The objective is to minimize the overall operational time required to transfer containers between ships and customers in both directions.

To achieve this, AGV trajectories must be optimized while ensuring deadlock-free operations (e.g., preventing collisions between AGVs). In this regard, authorizing bidirectional paths into AGV routing presents a major challenge.

The remainder of this paper is structured as follows: Section 2 describes the problem and provides a literature review. Section 3 introduces the mathematical formulation and underlying assumptions. Section 4 presents the proposed heuristic algorithm. Finally, Section 5 analyzes the numerical results and compares the performance of different approaches.

II. PROBLEM DESCRIPTION

The routing of Automated Guided Vehicles (AGVs) in automated port environments is a critical topic in the scientific literature. However, most existing studies assume unidirectional transportation network, where each segment can only be traversed in a single direction [1], [2], [3]. To better reflect real-world terminal constraints and optimize routing decisions, bidirectional structures are more relevant but have been less studied in the literature. Such structure also introduces additional challenges, in particular regarding collision avoidance. Thus, [4] and [5] adopt a bidirectional approach to assign containers to AGVs and determine their trajectories while avoiding conflicts. [4] proposes a dynamic routing approach to minimize the overall container transportation time, while [5] developed a branch-and-bound approach to determine task assignments and sequences, while a greedy heuristic algorithm is used to generate conflict-free routes. These two approaches address both scheduling and trajectory routing, relying on the same port model, whereas our study specifically focuses on trajectory routing using a different port model. The port under consideration consists of three main zones [6]: storage areas, loading and unloading zones for vessels, and an intermediate circulation area dedicated to AGVs. Its transportation network is structured using blocks and crossroads. A block represents a segment of the roadway that can be occupied by only one AGV at a time to prevent collisions. AGVs move from one block to another via crossroads. Each AGV has a designated starting point (loading) and a destination (unloading), and these points can only be blocks.

To tackle this problem, various approaches have been proposed in the literature. One approach from [7] relies on the use of Petri nets. Others focus on Mixed-Integer Linear Programming (MILP) models developed to minimize the total operational time required for AGV routing [8], [9]. Another MILP-based model, incorporating structural modifications, was proposed by [6]. While these models are effective for small-scale instances, their computational performance deteriorates significantly as the problem size increases. Given these scalability limitations, local search methods offer a computationally efficient alternative, capable of finding high-quality solutions even for larger problem instances.

III. PROBLEM FORMULATION

In this section, a Mixed Integer Linear Programming (MILP) model is proposed to address the AGV routing problem. This model is inspired by the formulation presented in [6], with some modifications related to the objective function and the formulation of some constraints.

The objective of this problem is to minimize the makespan.

[1] Tess Nouy is with the LAAS-CNRS laboratory and the Université Paul Sabatier, Toulouse, France. tnouy@laas.fr

[2] Ghassen Cherif is with the LAAS-CNRS laboratory and the Université Paul Sabatier, Toulouse, France. gcherif@laas.fr

[3] Sandra Ulrich Ngueveu is with the LAAS-CNRS laboratory and Toulouse INP, Toulouse, France. ngueveu@laas.fr

[4] Mikhail Zakharov is with the LAAS-CNRS laboratory and INSA Toulouse, Toulouse, France. mzakharov@laas.fr

The following section provides a detailed description of the network topology and the problem modeling.

A. Topology

The studied port environment is represented as a network composed of blocks and crossroads (FIG. 1), within which missions are defined.

Blocks: A block is one of the two main components of the network. The set of blocks is denoted as $\mathcal{B} = \{b_1, b_2, \ldots, b_{|\mathcal{B}|}\} = \{b_i\}_{i\in[1,\ldots,|\mathcal{B}|]}$. Each block $b_i \in \mathcal{B}$ has an associated traversal time $d_i^b \in \mathbb{N}^+$. To avoid conflicts between vehicles, an AGV is allowed to wait within a block. The set of blocks directly accessible from block b_i is denoted as $\mathcal{B}_i^A$ and the set of blocks from which an AGV can reach block b_i is denoted as $\mathcal{B}_i^{\overleftarrow{A}}$.

Crossroads: Crossroads constitute the second component of the network. They ensure connectivity between blocks, meaning an AGV must traverse a crossroad to move from one block to another. The set of crossroads is denoted as $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\} = \{c_p\}_{p\in[1,\ldots,|\mathcal{C}|]}$. Each crossroad $c_p \in \mathcal{C}$ has an associated crossing duration $d_p^c \in \mathbb{N}^+$. Furthermore, for each block $b_i \in \mathcal{B}$ and each block $b_j \in \mathcal{B}_i^A$, there exists a unique crossroad $c_{i,j}$ that an AGV must pass through to reach b_j from b_i. The set of crossroads accessible from the block b_i is denoted by $\mathcal{C}^i$. The set of blocks that can be reached from block b_i via crossroad c_p is denoted by $\mathcal{B}_{i,p}^A$, while the set of blocks from which block b_i is accessible via crossroad c_p is denoted by $\mathcal{B}_{i,p}^{\overleftarrow{A}}$. Unlike blocks, waiting is not allowed on crossroads to prevent congestion.

AGVs missions: An AGV mission $m \in \mathcal{M}$ consists of transporting a container from an origin point s_m to a destination point t_m, where s_m is a block located in the storage area (resp. at the quay) and t_m is a block located at the quay (resp. in the storage area). Each mission m is assigned to a unique AGV a_m. The set of AGVs is defined as $\mathcal{A} = \{a_1, a_2, \ldots, a_{|\mathcal{M}|}\} = \{a_m\}_{m\in[1,\ldots,|\mathcal{M}|]}$.
A key constraint of the problem is that AGVs cannot backtrack: they are not allowed to traverse the same block or crossroad more than once.

B. Modelling

1) Decision variables: The decision variables of the proposed model are defined as follows:

- $X_m^{i,j}$: $\begin{cases} 1 & \text{if AGV } a_m \text{ moves from block } b_i \text{ to block } b_j \\ 0 & \text{otherwise.} \end{cases}$
- Y_m^i : $\begin{cases} 1 & \text{if AGV } a_m \text{ passes through block } b_i \\ 0 & \text{otherwise.} \end{cases}$
- Z_m^p : $\begin{cases} 1 & \text{if AGV } a_m \text{ passes through crossroad } c_p \\ 0 & \text{otherwise.} \end{cases}$
- α_m^i, β_m^i: Arrival (resp. Departure) time of AGV a_m at (resp. from) block b_i ($\in \mathbb{N}$)
- ϕ_m^p, ψ_m^p: Arrival (resp. Departure) time of AGV a_m at (resp. from) crossroad b_p ($\in \mathbb{N}$)
- ϵ_m^i : Waiting time of AGV a_m in block b_i ($\in \mathbb{N}$)
- $B_{m,n}^i$: $\begin{cases} 1 & \text{if AGV } a_m \text{ passes through block } b_i \\ & \text{before AGV } a_n \\ 0 & \text{otherwise.} \end{cases}$

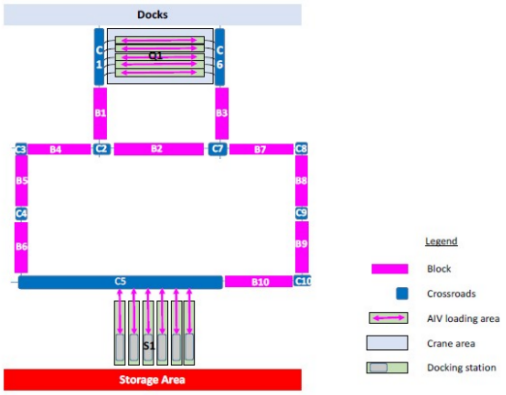

Fig. 1: Example of the topology of the transportation network in a seaport terminal

- $C_{m,n}^p$: $\begin{cases} 1 & \text{if AGV } a_m \text{ passes through crossroad } c_p \\ & \text{before AGV } a_n \\ 0 & \text{otherwise.} \end{cases}$

2) Constraints: The constraints of this model are organized into different categories: routing constraints, time constraints, and safety constraints that prevent collisions between AGVs.
For the sake of linearization, big-M constraints have been used. In this model, M represents a sufficiently large constant, while ω is a strictly positive and sufficiently small constant.

a) Routing Constraints:

$$X_m^{i,j} = 0, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \notin \mathcal{B}_i^A \tag{1}$$

$$X_m^{i,i} = 0, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B} \tag{2}$$

$$\sum_{b_j \in \mathcal{B}_{s_m}^A} (X_m^{s_m,j} - X_m^{j,s_m}) = 1, \qquad \forall a_m \in \mathcal{A} \tag{3}$$

$$\sum_{b_j \in \mathcal{B}_{t_m}^A} (X_m^{t_m,j} - X_m^{j,t_m}) = -1, \qquad \forall a_m \in \mathcal{A} \tag{4}$$

$$Y_m^{s_m} = 1, \qquad \forall a_m \in \mathcal{A} \tag{5}$$

$$Y_m^{t_m} = 1, \qquad \forall a_m \in \mathcal{A} \tag{6}$$

$$\sum_{b_j \in \mathcal{B}\setminus\{b_i\}} X_m^{i,j} \leq 1, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B} \tag{7}$$

$$\sum_{b_j \in \mathcal{B}_i^A} X_m^{i,j} = Y_m^i, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B} \setminus \{t_m\} \tag{8}$$

$$\sum_{b_i \in \mathcal{B}} X_m^{i,j} = Y_m^j, \qquad \forall a_m \in \mathcal{A},\ \forall b_j \in \mathcal{B} \setminus \{s_m\} \tag{9}$$

$$X_m^{i,j} \leq Z_m^{c_{i,j}}, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A \tag{10}$$

$$X_m^{i,j} + X_m^{j,i} \leq 1, \qquad \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A \tag{11}$$

$$\sum_{b_j \in \mathcal{B}\setminus\{b_i\}} (X_m^{i,j} - X_m^{j,i}) = 0,\ \forall a_m \in \mathcal{A}, \forall b_i \in \mathcal{B}\setminus\{s_m, t_m\} \tag{12}$$

Constraint (1) ensures that an AGV cannot move from block b_i to block b_j if b_j is not accessible from b_i. Constraint (2) prevents a block from being accessible to itself. Constraint (3) ensures that each AGV departs from its starting point and does not return to it. Constraint (4) ensures that each AGV reaches its destination and does not leave it once arrived. Constraints (5) and (6) enforce that each AGV necessarily passes through its starting and destination points. Constraint (7) ensures that AGV a_m can access at most one block from b_i. The inequality accounts for cases where the AGV does not pass through b_i, setting all variables $X_m^{i,j}$ to zero. Constraints (8) and (9) link the decision variables X_m and Y_m: if AGV a_m moves from block b_i to block b_j, it necessarily traverses both blocks. Constraint (10) establishes the connection between the variables X_m and Z_m: if AGV a_m moves from block b_i to block b_j, it must necessarily pass through the crossroad $c_{i,j}$. Constraint (11) indicates that an AGV cannot move in reverse; each AGV can only travel through a block in one direction throughout its entire mission. Constraint (12) ensures that if AGV a_m enters block b_i (with $b_i \neq t_m$), it must also exit. Conversely, for it to exit block b_i (with $b_i \neq s_m$), it must have previously entered it.

b) Time Constraints:

$$\begin{aligned}\beta_j - \alpha_j - \epsilon_j - d_j^b \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A \backslash \{s_m\}\end{aligned} \tag{13}$$

$$\begin{aligned}\alpha_j + \epsilon_j + d_j^b - \beta_j \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A \backslash \{s_m\}\end{aligned} \tag{14}$$

$$\beta_m^{s_m} = \alpha_m^{s_m} + \epsilon_m^{s_m} + d_{s_m}^b, \qquad \forall a_m \in \mathcal{A} \tag{15}$$

$$\alpha_m^{s_m} = 0, \qquad \forall a_m \in \mathcal{A} \tag{16}$$

$$\begin{cases}\alpha_m^i \leq M Y_m^i \\ \epsilon_m^i \leq M Y_m^i\end{cases}, \qquad \forall a_m \in \mathcal{A}, \forall b_i \in \mathcal{B} \tag{17}$$

$$\begin{aligned}\alpha_m^j - \beta_m^i - d_{i_{c^j}}^c \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A\end{aligned} \tag{18}$$

$$\begin{aligned}\beta_m^i + d_{i_{c^j}}^c - \alpha_m^j \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A\end{aligned} \tag{19}$$

$$\phi_m^{c_{i,j}} - \beta_m^i \leq M(1 - X_m^{i,j}),\ \forall a_m \in \mathcal{A}, \forall b_i \in \mathcal{B}, \forall b_j \in \mathcal{B}_i^A \tag{20}$$

$$\beta_m^i - \phi_m^{c_{i,j}} \leq M(1 - X_m^{i,j}), \quad \forall a_m \in \mathcal{A}, \forall b_i \in \mathcal{B}, \forall b_j \in \mathcal{B}_i^A \tag{21}$$

$$\begin{aligned}\psi_m^{c_{i,j}} - \phi_m^{c_{i,j}} - d_{c_{i,j}}^c \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A\end{aligned} \tag{22}$$

$$\begin{aligned}\phi_m^{c_{i,j}} + d_{c_{i,j}}^c - \psi_m^{c_{i,j}} \leq M(1 - X_m^{i,j}),\\ \forall a_m \in \mathcal{A},\ \forall b_i \in \mathcal{B},\ \forall b_j \in \mathcal{B}_i^A\end{aligned} \tag{23}$$

$$\phi_m^p \leq M Z_m^p, \qquad \forall a_m \in \mathcal{A},\ \forall c_p \in \mathcal{C} \tag{24}$$

Constraints (13) and (14) define the time intervals during which blocks are occupied when traversed by AGVs. For origin blocks, which do not have preceding blocks, constraint (15) is introduced. Constraint (16) initializes the departure times for all AGVs to 0.

To improve the efficiency of the solution process, constraint (17) is introduced. It ensures that the time intervals for unoccupied blocks are set to zero.

Finally, constraints (18) and (19) manage the sequence of time intervals during which an AGV occupies a block, specifically when AGV a_m moves from block b_i to block b_j. The arrival time in block b_j must be equal to the departure time from block b_i plus the duration required to traverse the crossroad $c_{i,j}$. The constraints introduced bellow are similar to those presented just above, which deal with the occupation time intervals of blocks. Constraints (20) and (21) define the arrival times of AGVs at crossroads, which are based on the departure times from the preceding block.

Once the arrival times are defined, the departure times from crossroads must also be determined. This is achieved through constraints (22) and (23), which specify the intervals for both arrival and departure.

To further optimize the computation time, constraint (24) is added to the model. It ensures that the arrival times at crossroads are set to zero when no AGV passes through them, effectively eliminating unnecessary time intervals.

c) Safety constraints:

$$B_{m,n}^i + B_{n,m}^i = 1, \qquad \forall a_m \neq a_n \in \mathcal{A},\ \forall b_i \in \mathcal{B} \tag{25}$$

$$C_{m,n}^p + C_{n,m}^p = 1, \qquad \forall a_m \neq a_n \in \mathcal{A},\ \forall c_p \in \mathcal{C} \tag{26}$$

$$\begin{aligned}\beta_m^i + \omega \sum_{j \in \overleftarrow{\mathcal{B}_{i,p}^A}} X_n^{j,i} \leq\\ \alpha_n^i + M\left(4 - Y_m^i - Y_n^i - B_{m,n}^i - \sum_{j \in \mathcal{B}_{i,p}^A} X_m^{i,j}\right),\\ \forall a_m \neq a_n \in \mathcal{A},\ \forall b_i \in \mathcal{B} \backslash \{t_m\},\ \forall c_p \in \mathcal{C}^i\end{aligned} \tag{27}$$

$$\beta_n^{t_m} \leq \alpha_m^{t_m} + M(1 - Y_n^{t_m}), \qquad \forall a_m \neq a_n \in \mathcal{A} \tag{28}$$

$$\begin{aligned}\psi_n^p + \omega \leq \phi_m^p + M\left(3 - Z_m^p - Z_n^p - C_{m,n}^p\right),\\ \forall a_m \neq a_n \in \mathcal{A},\ \forall c_p \in \mathcal{C}\end{aligned} \tag{29}$$

Constraints (25) and (26) define the precedence relationships in block and crossroad occupancy for each pair of AGVs. Thus, one AGV must necessarily pass before the other. Finally, constraints (27), (28), and (29) prevent collisions between AGVs passing through the same block or crossroad.

IV. SOLUTION APPROACH

Solving the MILP model using CPLEX presents scalability issues; therefore, a local search heuristic has been developed, offering a computationally efficient alternative that quickly converges to high-quality solutions, especially for large-scale instances where MILP faces significant performance challenges. To perform this local search, an initial solution is required. Thus, the first step consists of computing an initial solution, which is then used as the input for the Iterative Path Reinsertion (IPR) local search algorithm.

A. Computation of the Initial Solution

The determination of the initial solution follows two main steps: computing the shortest length paths without considering other AGVs and managing conflicts.

1) Computation of the Shortest Length Paths Without Considering Other AGVs: The first step in computing the initial solution involves determining the shortest length path for each mission (and thus for each AGV) using Dijkstra's algorithm. For all AGVs in the instance, the sequence of blocks and crossroads they will traverse is determined.

2) Conflict Management: Once the shortest length path for each AGV has been computed (i.e., the sequence of blocks and crossroads), conflict management must be incorporated. This requires handling the occupancy intervals of blocks and crossroads, as well as computing waiting times for each AGV to prevent collisions.

To achieve this, the problem is formulated as a job shop scheduling problem with blocking constraints and no swap allowed [10]. A swap denotes a situation where two AGVs exchange their respective positions, such that the block initially assigned to one becomes that of the other, and vice versa. The jobs correspond to the shortest length paths computed in the previous step, while the machines represent the set of blocks and crossroads. Each machine can process only one job at a time. The blocking constraint ensures that there is no intermediate storage capacity between two consecutive machines: an operation must wait and cannot release the current machine until the next one becomes available. The swap constraint prevents two conflicting operations from being executed simultaneously.

This problem is known to be NP-complete and has received very little study in the literature, making it highly challenging to solve. However, by leveraging the solver Tempo [11], a generic solver designed to efficiently handle various classes of scheduling problems including job shop problems, conflict management is effectively addressed, and an initial feasible solution is obtained.

B. Iterative Path Reinsertion (IPR) Algorithm

The IPR algorithm is a local search heuristic that starts from an initial solution and aims to improve the trajectories of AGVs. The algorithm begins by sorting the list of AGVs based on a prioritization policy. Then, following this predefined order, the path of a selected AGV is removed and subsequently reinserted in an optimized manner. These steps are iterated until one of the stopping criteria is met. The IPR algorithm is summarized in Algorithm 1.

1) Prioritization Policies: Prioritization policies define an ordering strategy for AGV paths based on a specific criterion. Four prioritization policies have been implemented:

- Lexicographic (LEXI): sorts the AGVs in lexicographical order.
- Latest Arrival Time First (LATF): prioritizes the AGV that finishes its mission last.

Algorithm 1 Iterative Path Reinsertion Algorithm

Require: $|\mathcal{M}| \geq 0$ = number of AGVs; S = initial solution; policy = selected prioritization policy, time_limit =
$Order$ = sort_AGV($policy$) ▷ Sorts the list of AGVs according to the chosen policy
i = 1
while $i \leq |\mathcal{M}|$ and time_limit not reached **do**
  v = find_AGV($Order$, i)
  $S_{s,v}$ = remove(S,v) ▷ Removes the path of AGV v from the solution
  S_{temp} = bestinsert($S_{s,v}$, v) ▷ Finds the best path to insert AGV v in $S_{s,v}$
  if path(v,S) $\neq$ path(v,S_{temp}) **then**
    i=1
    S = S_{temp}
    $Order$ = sort_AGV($policy$) ▷ Sorts the list of AGVs according to the chosen policy
  else
    i += 1
  end if
end while

- Longest Wait Time First (LWTF): prioritizes the AGV with the longest wait time.
- Most Untimed Paths Conflict First (MUPCF): prioritizes the AGV with the highest number of conflicts in the absence of temporal constraints.

2) Path Reinsertion: To optimally reinsert the selected AGV's path, a space-time graph is constructed. This graph consists of temporally defined nodes. A node is therefore characterized by its departure time, cost, type (block or crossroads), name, and a list of neighboring nodes.

Occupied nodes are removed from the space-time graph. For example, if a node is occupied by another AGV during a certain time period, then the corresponding time-stamped nodes are removed from the space-time graph. In addition to nodes being occupied for a certain period, events that may lead to swaps, where two AGVs attempt to exchange positions simultaneously, must be blocked. This corresponds to the set of forbidden movements.

Once the space-time graph is built and both occupied nodes and forbidden movements are identified, a variant of Dijkstra's algorithm is applied between two points in the graph. Indeed, for each AGV a_m, the departure point is known and corresponds to s_{m0}. The arrival point belongs to the set $\{t_{m0}, ..., t_{m,t_{max}}\}$, where the second index represents the time variable ranging from 0 to t_{max} . Given this information, the shortest path is computed between the departure point and the set of possible arrival points. Among all valid paths, the one arriving the earliest is selected.

The resulting path is reconstructed by backtracking from the node corresponding to the final destination, following the recorded predecessor nodes, and forming a sequence of segments that indicate both movements and waiting times.

3) Termination Policies: Several termination criteria have been defined to stop the algorithm. The algorithm terminates if no shorter path is found for any AGV. Another stopping criterion is based on a predefined time limit.

V. NUMERICAL EXPERIMENT

To evaluate the performance of the proposed model and algorithm, experiments were conducted on instances of various sizes, ranging from small to large configurations. The objective is to minimize the makespan.
To assess the quality of the obtained solutions, we compute the gap between the solution values and a lower bound, defined as the sum of the shortest length path for all AGVs. This lower bound disregards AGV conflicts and, consequently, does not account for waiting times at the blocks. While it may not represent an achievable value, it serves as a reference for evaluating the quality of the solutions.
All experiments were conducted with a time limit of one hour per instance.

A. Port Environment

The considered port environment consists of 205 blocks, 72 crossroads, 8 docks, each made up of 5 blocks, and 12 storage areas, each consisting of 6 blocks.

B. Data sets

Two classes of instances are generated. The first class corresponds to the unidirectional case, where the AGVs perform their tasks in the same direction (X_u, with X representing the number of AGVs). The second class corresponds to the bidirectional case, where the AGVs transport their containers in both directions (X_d, with X representing the number of AGVs). The number of AGVs per instance ranges from 4 to 80. For each number of AGVs and each class of instances, 50 instances are generated randomly, ensuring that no more than one AGV departs from the same starting point or arrives at the same destination.

C. Method used

The algorithm was coded in Python (Python 3.10.12), with the use of PyPy 3.10. To calculate the initial solution, the Tempo solver was used.
To compare the results, the MILP model was also coded in Python 3.10.12, using the CPLEX solver (CPLEX 22.1.1).
All instances were executed on one of the cores of the computing platform at the Laboratory of Systems Analysis and Architecture (LAAS-CNRS), which uses an AMD EPYC 9654 processor running at 3.7 GHz and has 1024 GB of RAM.

D. Results and analysis

The results are analyzed according to two main factors: first, by comparing the solutions obtained with our method to those obtained by solving the MILP model, and second, by evaluating the improvement achieved through the IPR algorithm relative to the initial solutions. To illustrate these comparisons, Table I presents a comprehensive summary of the numerical results across various instance types and sizes. The table is organized into three main sections: results obtained using the MILP model, those computed when generating the initial solution, and those obtained after applying the IPR algorithm under different prioritization policies (LEXI, LATF, LWTF, MUPCF). Each row corresponds to a specific instance type (e.g., 4d, 8u), while the columns report: #_inst, the number of unsolved instances out of 50; sol_i and gap_i, the average cost and percentage gap of the initial solution; sol_f and gap_f, the average cost and percentage gap of the final solution obtained after applying IPR.
The percentage gap is computed using the standard formula:

$$gap = \frac{solution\ cost - lower\ bound}{lower\ bound} * 100$$

1) Comparison with the MILP Model: The first objective is to verify that the developed algorithm addresses the scalability issues associated with the MILP model. Therefore, it is necessary to compare the results obtained with this algorithm to those obtained by solving the MILP model.
Two criteria can be studied: the time required to obtain the best feasible solution and the obtained solutions.
For the first criterion, Fig. 2 shows the number of feasible solutions found as a function of time for 750 instances, ranging from 4 to 80 AGVs. It appears that the MILP model reaches its time limit for all instances. Furthermore, feasible solutions are obtained only for 363 instances. The heuristic produces a larger number of feasible solutions in just a few seconds. And by the end of the time limit, the heuristic is able to identify almost twice as many feasible solutions as those determined by the MILP model. It is important to note that the LEXI prioritization policy is much less effective than the other prioritization policies: it takes significantly more time to improve the initial solution. This can be explained by the fact that the LEXI policy does not prioritize AGVs whose trajectories present the most issues.
Examining the results in TABLE I more in detail, it is noteworthy, from the columns "nbe_inst" representing the number of unresolved instances, that for instances with 36 AGVs or more, the MILP model fails to find any solution. Furthermore, starting from the 4u instances, the solutions obtained with the heuristic are better than those obtained with the MILP model. This indicates that the heuristic is more efficient than the MILP model, both for small and large instances.
For larger instances, the number of unresolved instances is non-zero. This is due to the method used to calculate the initial solution: some cases are impossible to resolve. Four such cases have been identified: (i) *Initial swap case* when two AGVs exchange positions at the beginning of their trajectories; (ii) *Opposing path case* when two AGVs follow the same length path in opposite directions; (iii) *Inclusion path case* when the path of one AGV is entirely contained within the path of another; (iv) *Loop case* when four or more AGVs are involved in a circular dependency, where each AGV's start point corresponds to the endpoint of another, and all share nearly the same path. In these four cases, the solver is unable to find an initial solution because collisions can not be prevented by adding waiting times at the blocks or crossroads used by the shortest length paths. Future work will focus on alternative methods for generating feasible initial solutions.

	MILP			SOL INI			IPR - LEXI		IPR - LATF		IPR - LWTF		IPR - MUPCF	
Inst	#_inst	sol_f	gap	#_inst	sol_i	gap	sol_f	gap	sol_f	gap	sol_f	gap	sol_f	gap
4d	16	142.4	1.3	0	144.4	2.5	144.4	2.5	144.4	2.5	144.4	2.5	144.4	2.5
4u	20	136.0	2.4	0	133.3	2.4	133.3	2.4	133.3	2.4	133.3	2.4	133.3	2.4
8u	1	144.1	6.0	0	142.8	2.7	142.7	5.8	142.8	5.8	142.8	5.9	142.8	5.9
8d	2	174.9	3.4	0	172.4	3.0	172.0	2.7	172.0	2.7	172.0	2.7	172.4	3.0
12u	0	153.1	9.7	0	150.4	9.3	149.2	8.4	149.3	8.4	149.3	8.4	149.4	8.5
12d	0	207.2	14.2	0	188.9	8.9	179.7	3.2	179.9	3.3	180.0	3.4	180.0	3.4
16u	0	166.9	15.7	0	163.8	17.0	156.3	11.6	156.6	11.9	156.8	12.0	156.5	11.8
16d	0	229.1	22.8	0	204.8	19.3	182.8	6.1	181.8	5.5	183.6	6.6	182.0	5.7
24d	48	636.0	56.6	5	423.6	54.7	323.6	18.0	282.9	3.3	306.3	11.5	284.7	4.0
36d	50	–	–	4	482.4	70.2	420.5	48.4	304.0	7.3	352.1	24.3	304.3	7.4
48d	50	–	–	12	568.9	100.3	479.2	68.9	342.2	20.5	395.9	39.4	352.7	24.2
56d	50	–	–	13	615.1	112.4	518.5	79.0	387.8	33.9	427.0	47.5	385.2	33.1
64d	50	–	–	22	655.3	124.8	599.9	105.7	424.1	45.7	474.2	62.9	428.9	47.4
72d	50	–	–	25	699.1	139.5	591.6	102.7	477.5	63.8	503.5	72.6	472.0	61.8
80d	50	–	–	32	745.4	126.9	667.2	126.9	528.3	79.8	576.8	96.2	529.9	80.2

TABLE I: Comparison of results obtained with MILP and IPR

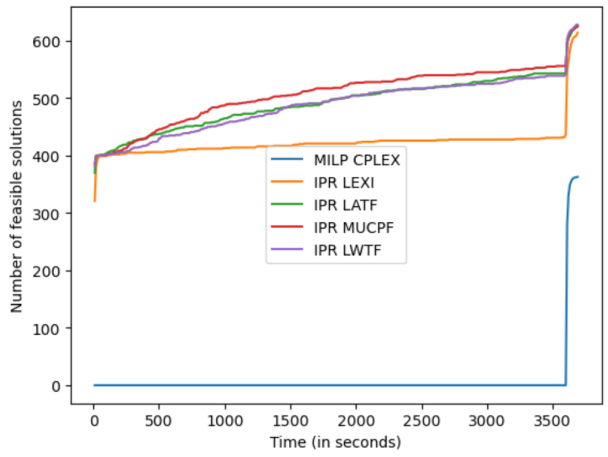

Fig. 2: Comparison between the IPR algorithm and the MILP model solved with CPLEX on 750 instances ranging from 4 to 80 AGVs

2) Comparison with the initial solution: To validate the effectiveness of the IPR algorithm, it is essential to ensure that the solutions obtained after its application are superior to the initial solution.

Regarding of the percentage gap between initial and final solutions, no significant improvement is observed for small instances (comprising 4 to 8 AGVs). This can be attributed to the fact that the initial solutions obtained are already very close to the optimal solution. However, for larger instances, a significant improvement is observed, with the gap between the initial and final solution reaching up to 35.6%.

Among the different prioritization policies implemented, the LEXI policy performs best for small instances (ranging from 8 to 16 AGVs). However, the gap between the solutions obtained with the LEXI policy and those obtained with other policies remains below 1%, making this improvement negligible. In contrast, for larger instances, the LATF and MUPCF policies yield better results than the LEXI and LWTF policies. In particular, the LEXI policy becomes the least effective, with a solution gap reaching up to 41%. This can be explained by the fact that the LATF and MUPCF policies prioritize adjustements the trajectories of AGVs that necessitate the most modifications.

These results highlight a significant improvement in solution quality after applying the IPR algorithm proposed.

VI. CONCLUSION

This study proposed a mathematical formulation, a constructive heuristic and a dedicated local search approach to address the bidirectional conflict-free routing problem for Automated Guided Vehicles within a specific port layout composed of blocks and crossroads. The numerical experiments illustrate the quality of solutions obtained for both small and large instances. A few unsolved cases have been identified, that emerge when multiple AGVs have conflicting origin and destination blocks, or when the initial trajectory of an AGV is a subpath of another AGV. Future work will address these cases to ensure the identification of a feasible and efficient solution for all instances.

REFERENCES

[1] Yang, Y., Zhong, M., Dessouky, Y., & Postolache, O. (2018). *An integrated scheduling method for AGV routing in automated container terminals*. *Computers & Industrial Engineering*, 126, 482-493.

[2] Zhong, M., Yang, Y., Dessouky, Y., & Postolache, O. (2020). *Multi-AGV scheduling for conflict-free path planning in automated container terminals*. *Computers & Industrial Engineering*, 142, 206371.

[3] Ji, S., Luan, D., Chen, Z., & Guo, D. (2020). Integrated scheduling in automated container terminals considering AGV conflict-free routing. *Transportation Letters*, DOI: 10.1080/19427867.2020.1733199.

[4] Cao, Y., Yang, A., Liu, Y., Zeng, Q., & Chen, Q. (2023). *AGV dispatching and bidirectional conflict-free routing problem in automated container terminal*. *Computers & Industrial Engineering*, 184, 109611.

[5] Wang, Z., & Zeng, Q. (2022). *A branch-and-bound approach for AGV dispatching and routing problems in automated container terminals*. *Computers & Industrial Engineering*, 166, 107968.

[6] Terfasse, K., Cherif, G., & Huguet, M-J. (2024). *Integer Linear Programming for AGV Path Planning in Container Terminals*. In *International Conference on Control, Decision and Information Technologies (CoDIT 2024)*, Jul 2024, Valletta, Malta. ffhal-04704943f.

[7] Cherif, G., Trouillet, B., & Toguyeni, A. K. (2022). *Modeling and routing problems of automated ports using T-TPN and Beam search*. In *8th International Conference on Control, Decision and Information Technologies (CoDIT)*, Vol. 1, IEEE, 2022, pp. 1201-1206.

[8] Danloup, N., Trouillet, B., Bourdeaud'huy, T., & Toguyéni, A. (2018). *Optimisation de tournée de véhicule dans un environnement portuaire*. In *GOL 2018 - International Conference on Logistics Operations Management*, Apr 2018, Le Havre, France (pp. 1-7). ffhal-01728605v2f.

[9] Danloup, N., Trouillet, B., Toguyéni, A., & Bourdeaud'Huy, T. (2019). *Gestion d'une flotte de véhicules automatisés dans un environnement portuaire*. In *MSR 2019 - 12ème Colloque sur la Modélisation des Systèmes Réactifs*, Nov 2019, Angers, France. ffhal-02432674f.

[10] Pranzo, M., & Pacciarelli, D. (2016). *An iterated greedy metaheuristic for the blocking job shop scheduling problem*. *Journal of Heuristics*, 22, 587-611.

[11] `https://gitlab.laas.fr/roc/emmanuel-hebrard/tempo`.