Combining Deep Reinforcement Learning with Search Heuristics for Solving Multi-Agent Path Finding in Segment-based Layouts

1st Robbert Reijnen
Jheronimus Academy of Data Science
's-Hertogenbosch, The Netherlands
reijnen.robbert@gmail.com

2nd Yingqian Zhang
Dept. of Industrial Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands
yqzhang@tue.nl

3rd Wim Nuijten
Dept. of Math. and Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
w.p.m.nuijten@tue.nl

4th Caglar Senaras
Vanderlande Industries B.V.
Veghel, The Netherlands
caglar.senaras@vanderlande.com

5th Mariana Goldak - Altgassen
Vanderlande Industries B.V.
Veghel, The Netherlands
mariana.goldak@vanderlande.com

***Abstract*—A multi-agent path finding (MAPF) problem is concerned with finding paths for multiple agents such that certain objectives, such as minimizing makespan, are reached in a conflict-free manner. In this paper, we solve a practical MAPF problem with automated guided vehicles (AGVs) for the conveying of luggage in segment-based layouts (MAPF-SL).**

Most existing algorithms for MAPF are mainly focused on grid environments. However, the conflict prevention problem is more challenging with segment-based layouts in which software is constrained to oversee that vehicles remain on predefined travel paths. Hence, the existing multi-agent path finding algorithms cannot be applied directly to solve MAPF-SL. In this paper, we propose an algorithm, called WHCA*S-RL, that combines Deep Reinforcement Learning (DRL) with a heuristic approach for solving MAPF-SL. DRL is used for determining travel directions while the heuristic approach oversees the planning in a segment-based layout. Our experiment results show that the proposed WHCA*S-RL approach can be successfully used for making path plans in which traffic congestion is both avoided and relieved. In this way, individual vehicles are found to reach goal destinations faster than the approach with search only.

***Index Terms*—multi-agent path finding, path planning, deep reinforcement learning, automated guided vehicles**

I. Introduction

The growth of worldwide passenger volumes and the requirement to facilitate more transfer flights with shorter connection times stimulate investments of airports in automated baggage handling solutions. For the next generation of automation solutions for the sector, the use of Automated Guided Vehicles (AGVs) has been identified, due to its flexibility and reliability [18]. AGVs are battery-powered, cell-driven vehicles that transport materials and items from one location to another without any accompanying operator [3]. In AGV-based baggage handling systems, AGVs are used to carry luggage through handling areas of airports; each item of luggage is dropped on a single AGV and is brought to its goal destination. The automated vehicles are typically constrained by the layout of travel paths that they can follow, which is overseen by a system that prevents conflicts from occurring. This problem of finding conflict-free paths for multiple AVGs is a multi-agent path finding (MAPF) problem [16].

Various algorithms have been introduced in the literature for solving the MAPF problem. Fast algorithms, such as Prioritized Planning [5] and Windowed Hierarchical Cooperative A* (WHCA*) [13] are found to be efficient and robust for finding conflict-free paths in grid-based layout environments. These algorithms are widely used by practitioners in real-world environments despite their sub-optimal solutions. Recent works have been devoted to solve the grid-based MAPF problem with Deep Reinforcement Learning (DRL) (e.g., [7], [9]). These works have shown that learning-based algorithms are more robust to uncertainties in real-world environment.

In this paper, we solve a practical MAPF problem with automated guided vehicles (AGVs) for the conveying of luggage. We call this problem MAPF-SL. Most existing approaches to solve MAPF are focused on grid-based layouts. However, in our problem, the typical layouts in the AVG-based luggage handling system are segmented-based (see Figure 1). In these types of layouts, AGVs are constrained to remain on predefined paths and occupy surrounding segments instead of single grid-tiles. These characteristics of segment-based layouts cause different types of conflicts compared to the grid-based MAPF problem variants addressed in the literature.

In this work, we adapt the popular search algorithm WHCA* [13] to handle segment-based layouts. WHCA* is cooperative planning algorithm, which breaks down the MAPF problem into series of single-agent problems to find paths of limited length. In this way, computation is spread throughout

978-1-7281-2547-3/20/$31.00 ©2020 European Union

execution and inconsistencies are addressed in successive plans. Our proposed algorithm WHCA*S adapts WHCA* to handle potential conflicts in the segment-based layout.

Despite the advantages of computational efficiency and simplicity in implementation, one disadvantage of WHCA* based approaches is on their sub-optimal paths in terms of makespan and flowtime. This is due to their greedy nature, i.e., future traffic situations are not considered in the search for paths for single agents when search is performed with a limited window length. This causes AGVs to pursue their own shortest paths and prevents the usage of potential faster detours. To overcome this, we propose WHCA*S-RL that improves the proposed WHCA*S algorithm for determining what direction should be pursued by vehicles using a Deep Reinforcement Learning (DRL) model. In this way, different paths can be pursued such that congestion can be avoided for reaching goal destinations faster.

The contributions of this work are summarized as follows:

- We propose a WHCA* based algorithm (WHCA*S) that can handle additional conflicts arising from the segment-based layout.
- we propose WHCA*S-RL, which is the first approach to combine the fast search algorithm with DRL to solve the MAPF problem with a segment-based layout.
- We shown the benefit of WHCA*S-RL by a number of experiments with various settings.

The remaining of the paper is structured as follows: Section II summarizes the related work on MAPF. We detail the problem addressed in this work in Section III and describe the proposed methods in Section IV. In Section V we present the experiment results, and Section VI concludes.

II. Related work

The multi-agent path finding (MAPF) problem is referred to as the problem of finding a joint path plan for multiple agents for reaching goal destinations [16]. In the classical MAPF problem, time is discretized into time steps in which agents perform a single action. These actions are movements to adjacent position locations or decisions to wait at current occupied positions and take exactly one time step. A valid solution to a MAPF problem instance is a joint plan in which all agents move to their goal destinations without conflicting with each other. The definition of what constitutes a conflict depends on the application and its corresponding environment. This is dependent of the dimensions of the agents and the configuration of the layout environment.

A. Conventional solution approaches

Existing solution approaches for tackling a MAPF problem can be categorized into three classes: *Fast Heuristics*, *Optimal Algorithms*, and *Approximation Algorithms* [15]. In most fast heuristics the problem is decoupled to sub-problems in which paths should be found for individual agents. In this way, sub-optimal solutions can be obtained at relatively low computation costs. Popular fast heuristics are *Prioritized Planning* [5] and various *Cooperative Planning* approaches such as WHCA* [13]. Optimal algorithms are solvers that are guaranteed to be optimal with respect to a given objective. These solvers are typically centralized planning solutions, which take goal objectives of all agents into account at once. This enables the solvers to find all possible solutions including the optimal one, with a high computational cost. The work of [15] classifies optimal MAPF algorithms to four high-level approaches: *Extensions of A** (e.g. [14], [19]), *Increased Cost Tree Search* (e.g. [12]), *Conflict-Based Search* (e.g. [1], [11]) and *Constraint Programming* (e.g. [8]). Approximation algorithms are solutions that provide a trade-off between the fast heuristics and the optimal algorithms, as they can solve the problem in a reasonable amount of time while maintaining high solution quality. These solution methods typically relax the strict requirements of optimal algorithms to reduce the required computation time (e.g. [19]). Ideally, the algorithms can be controlled such that a trade-off can be found between computation time and solution quality. Unfortunately, no guidelines have been identified for predicting which algorithms work best for the various MAPF problem variants.

B. Reinforcement Learning based approaches

The potential use of Reinforcement Learning (RL) in environments that require cooperation between agents was initially investigated in [17]. This work describes multiple approaches for enabling cooperation, including the sharing of value functions. This was found to be an effective and efficient approach for learning in non-hierarchical settings with homogeneous agents. In [6], the authors apply this successfully to a path finding problem for the routing of taxi vehicles. The work of [4] investigates the implementation of neural networks for path finding related jobs with multiple agents. In the work, a Deep Q-learning network (DQN) is used for learning cooperative and competitive behavior in a pursuit-evasion game. Learning is performed by training only one agent at a time while the remaining agents utilize an earlier trained policy. After several iterations, the updated policy is distributed to the other agents. The work was correspondingly extended by [7], implementing the procedure in a fleet control problem. Policies of the agents are learned with the same network simultaneously, enabling faster convergence. With this, and with the progress booked in Deep Reinforcement Learning, the authors of [9] tackle the classical MAPF problem with Deep Reinforcement Learning. In the work, an asynchronous advantage actor-critic (A3C) model is provided with an observation of the surrounding environment of agents to learn the routing of individual vehicles. With this, the work has shown that models can be trained for the routing of up to 1024 agents in grid-based environments, outperforming classical MAPF planners in terms of processing speed and solution quality.

III. Problem formulation

The multi-agent path finding problem addressed in this work is concerned with a *segment-based layout* $G = (V, E)$ with position location vertices V and directed segment edges E.

Each agent i is assigned to an initial position location $s_i \in V$ and a goal destination $g_i \in V$. A path planning solution needs to find a valid joint plan Π that consists of single-agent plans π_i that guide individual vehicles from s_i to g_i in a *conflict-free* manner. The objective of our MAPF problem is to minimize the average duration of single-agent plans π_i, which is defined as the *average flowtime*. With this, the number of items transported per time unit can be maximized. Another frequently used performance metric in the literature is *makespan*, which is the total elapsed time required for all agents to reach goal destinations.

A. Segment-based layout

The implementation of AVG-based systems requires the design of travel paths that can be followed by vehicles. The system addressed in this work is operating with an uni-directed segment-based layout in which individual position location points are interconnected by segments. The location points are single locations in the layout, and the segments connect two adjacent points with a defined heading. Each segment is defined with a radius which is aligned such that subsequent segments follow the same curvation. This allows vehicles to drive continuously without stopping at the location points. Figure 1 illustrates an example layout, in which the pick-up points are represented in blue, the drop-off points in green, and all remaining position locations in red.

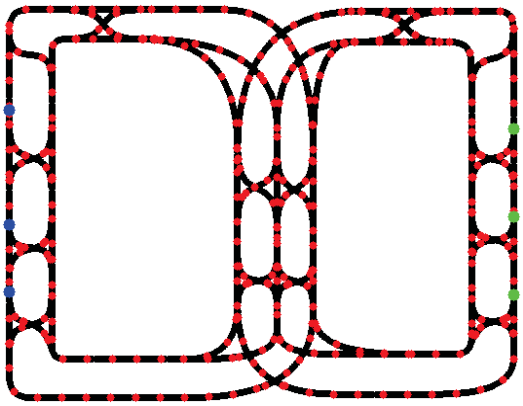


Fig. 1: A segment-based layout with pick-up points (blue), drop-off points (green) and other position locations (red)

B. Collision- and deadlock prevention

The concept of *blockings* is introduced for preventing the occurrence of collisions. This is a property of a segment that states which other segments are prevented from being used by other AGVs when that segment is occupied. Blockings are used because AGVs in the system occupy space around the segment on which they are driving. An example of the blockings corresponding to an AGV positioned on a certain position can be found in Figure 2. The *control point* in the figure is the current position of a vehicle and the *bounding box* indicates the area of influence around the vehicle. In a situation in which a vehicle is located on a certain segment, it virtually occupies the area within its bounding box. Other AGVs and their corresponding bounding boxes are not allowed in there. The segments that lay within the bounding box of a vehicle on a certain position are therefore considered as the blockings of a segment. When blockings are respected in a joint path plan for the AGVs, a system is guaranteed to be free of collisions.

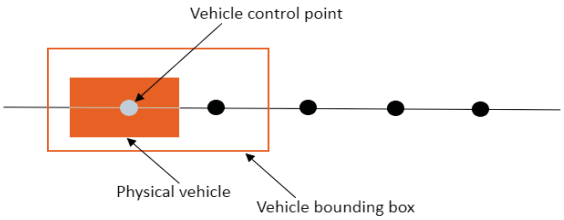


Fig. 2: The vehicle bounding box for determining blockings

Other situations that need to be prevented from occurring are deadlock situations. These are situations in which all AGVs in a set of vehicles are waiting on at least one other AGV from the set for continuing the path to the goal destination. The result of this is that the AGVs are waiting indefinitely and that none of the AGVs involved can continue its path. A deadlock situation in which two AGVs are trying to cross an intersection is displayed in Figure 3. In this situation, both AGVs are occupying a part of the blockings required to be available to cross the intersection with their bounding box. The result of this is that both vehicles are waiting for each other for making the cross.

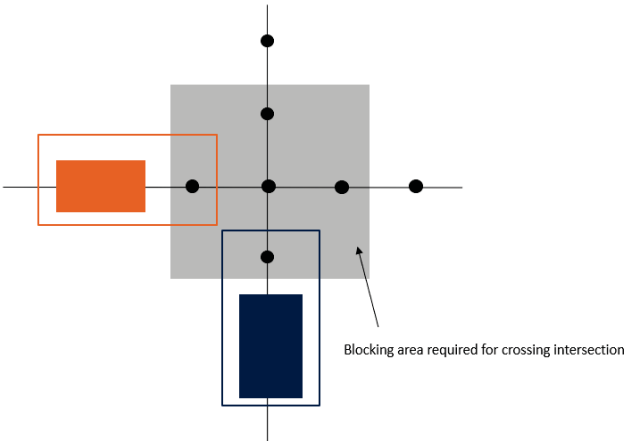


Fig. 3: A deadlock situation in which two AGVs are trying to cross an intersection

IV. The WHCA*S-RL algorithm

Various approaches have been identified in Section II-A for solving the problem formulated above. These approaches typically find a joint path plan over a long time horizon. However, in real-world environments in which future jobs are often unknown, path plans over long time ranges are found

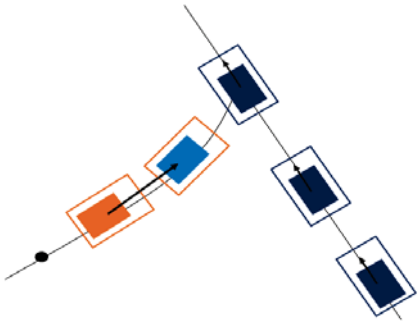
difficult to execute. These environments, therefore, rely on replanning for continued operability, such that found plans are continuously updated and overruled. To address this dependency on replanning, a window was introduced to the search in the Windowed Hierarchical Cooperative A* (WHCA*) of [13] for finding paths of limited length during execution. In this section we propose WHCA*S, a WHCA*-based algorithm that handles segment layout specific problems. In addition, we introduce WHCA*S-RL, an approach that combines WHCA*S with DRL to improve decision making using learning.

A. WHCA* for segment-based layouts (WHCA*S)

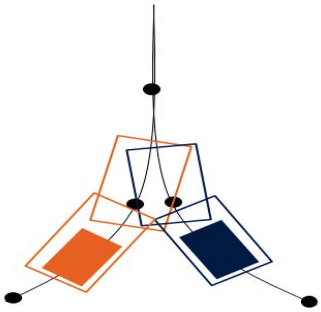
In the WHCA* heuristic, the search for paths is limited to a fixed length W. These partial paths towards goals are sequentially planned for agents according to priority rules and are executed in parallel up to time step K ($K \leq W$). By only executing a part of the paths, cooperativeness is ensured for the executed part, which is achieved by deviating agents from shortest paths when conflicts arise. Later planned steps are less cooperative as fewer future conflicts are taken into consideration. The determination of W and K requires careful consideration; a too-small planning window requires many planning iterations where a too long planning window demands more computation time before execution.

The heuristic is designed for a grid-based environment in which agents can travel in 4 directions: up, down, left, or right, or can remain idle. Conflicts in planning that arise in this classical MAPF problem variant are therefore different from the conflicts in the segment-based variant. Also, conflict handling by taking steps aside from the shortest path is not allowed in the segment-based layouts as agents are required to remain on the defined travel paths. Therefore, more idle actions need to be scheduled for preventing the occurrence of conflicts in the MAPF problem addressed in this work. To determine when to schedule these idle actions, the different types of conflicts have been identified and are displayed in Figure 4. The figure displays two planning conflicts: *Planning conflict A* & *Planning conflict B*.

Planning conflict A arises when a prior planned AGV (orange) leaves no space in the segment-based layout for another AGV (light blue) (the orange bounding box around the light blue AGV indicates the intended move of the orange AGV). In this situation, the prior planned AGV (orange) needs to be removed from the planning and needs to be replanned to take the planning of the conflicting AGV (light blue) into account. *Planning conflict B* also arises in situations in which no position locations remain available for an AGV due to prior planned AGVs. However, in this situation an AGV (orange) is not allowed to travel to its next intended position location and is not allowed to stay at its current position location because both locations or blockings aligned to these locations have been reserved by the same AGV (blue). In this situation, an idle action should be scheduled for the AGV in a prior step such that the conflict will be prevented. However, in situations in which there are no prior steps known, or only idle steps, the AGV is scheduled to take a step backwards. The described



(a) Planning conflict A



(b) Planning conflict B

Fig. 4: Conflicts in planning

conflict handling approaches have been implemented within the WHCA* heuristic approach of [13]. The pseudo-code of this updated WHCA* heuristic, defined as WHCA*S, can be found in Algorithm 1.

B. Improving WHCA*S using DRL (WHCA*S-RL)

A global search to goal destinations is conducted with Dijkstra's algorithm [2] to ensure that AGVs are planned to travel in the right direction. Other AGVs are ignored in this search, causing AGVs to pursue shortest paths. However, especially in congested situations, deviating from these shortest paths can be a more optimal strategy as congestion can be avoided and relieved. Given the recent developments within Deep Reinforcement Learning (DRL) and its successes in related work (see Section II-B), it is perceived as a promising technique for performing the path planning within the WHCA*S heuristic. With this, the proposed WHCA*S heuristic is used for overseeing plans, while the DRL model selects adequate direction decisions (actions) for the vehicles. We define this planning approach as WHCA*S-RL and Figure 5 displays how it combines a DRL agent with the WHCA*S heuristic for finding path plans.

A Semi-Markov Decision Process (SMDP) is formulated for learning a DRL decision-making strategy (policy) for making path planning decisions (actions) within the WHCA*-RL approach. This SMDP can be considered as the environment and the DRL agent interacts with it to learn by trial and error. It is defined as a tuple $< S, A, P, F, R >$, where S represent

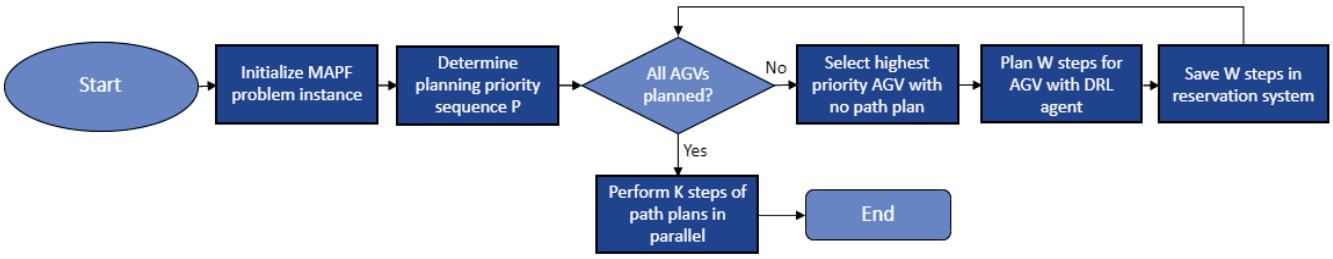


Fig. 5: Flowchart of the WHCA*S-RL heuristic framework with a DRL based path planning agent

Algorithm 1 WHCA*S for segment-based MAPF

Input: MAPF problem instance
reservation table $\tau \leftarrow \emptyset$
while *not all agents have reached goal* g_i **do**
  initialise planning priority P
  for *agent* $i \leftarrow 1$ *to* k *in* P **do**
    path $\pi_i \leftarrow$ *findPath*(i, τ, W)
    reserve π_i in τ
  for *each agent* $i \leftarrow 1$ *to* k *in parallel* **do**
    move agent i K steps on its path π_i
  reset reservation table τ
Function *findPath (i, τ, W)*
  path $\pi_i \leftarrow \emptyset$
  while *length of path* π_i != W **do**
    if *next step on shortest path to goal* g_i *computed with Dijkstra [2] is available* **then**
      plan new step for agent i to goal g_i in π_i
    else if *current position location is available* **then**
      schedule an idle step for agent i in π_i
    else
      agent $a \leftarrow$ agent blocking current position in τ
      agent $b \leftarrow$ agent blocking step on shortest path to g_i in τ
      if *agent a = agent b* **then**
        if *a non-idle action in plan* π_i **then**
          replace last non-idle step of agent i in π_i with an idle step and remove successive steps from π_i
        else if *previous location available in* τ **then**
          plan a step backwards for agent i in π_i
        else
          find Z agents in τ preventing agent i to take step backwards in π_i
          for *each agent* $x \leftarrow 1$ *to* Z **do**
            plan a step backwards for agent x in π_x
            remove successive steps from plan π_x
            add agent x to planning sequence
      else
        remove π_a from τ
        add agent a to planning sequence P
        schedule an idle action for agent i in π_i
  return π_i

the set of states, A the set of actions, P the state transition probability function, F the probability distribution of the time interval between two decisions and R the reward function. The state transition occurs after each action, such that the state S_t of the environment at time step t transforms to S_{t+1} at time step $t+1$, and the agents receive rewards according to the reward function R. Based on this, the multi-agent path finding task is defined from a mathematical perspective:

State space S - The state space is expressed as a one-dimensional vector. This vector contains information on all possible directions $j \in \{1, 2, \ldots, J\}$ that can be pursued by an individual AGV from the current occupied position location, where $j = 1$ represents the current position, i.e., remaining idle, and $j = 2, \ldots, J$ are predefined possible directions in a given segment-based layout. For each possible direction j, it contains two types of information: (1) the change in distance D_j towards the goal destination if direction j is taken, and (2) the number of available greedy steps C_j toward the goal destination if direction j is taken based on prior identified path plans of other AGVs. With this, the state space has a size of $2{\cdot}J$, where J is the maximum number of possible directions that can be pursued at any certain position in a segment-based layout. In the state representation, pairs of D_j and C_j for all possible directions $j \in \{1, \ldots, J\}$ are structured as follows:

$$< (D_1, C_1), \ldots, (D_j, C_j), \ldots, (D_J, C_J) >$$

With this, qualitative information is provided for all the directions that can be pursued. Change in distance D_j resulted by pursuing direction j is included as a normalized value that corresponds to the change in meters towards the goal destination. The normalized values are defined as follows:

$$D_j\text{:} \begin{cases} 0 & \text{if change in distance } = 0 \\ 1 & \text{if change in distance } > 0 \\ -1 & \text{if change in distance } \geq -2.5 \text{ and } < 0 \\ -2 & \text{if change in distance } \geq -10 \text{ and } < -2.5 \\ -3 & \text{if change in distance } \geq -25 \text{ and } < -10 \\ -4 & \text{if change in distance } < -25 \end{cases}$$

Information on plans of prior planned AGVs is included in C_j. The value indicates the number of available greedy actions towards the goal. With this, the state representation can for instance be structured as follows:

$$< (0, 8), (1, 8), (-1, 10), (-3, 8) >$$

From the example can be observed that the second direction that can be pursued (1,8) results in a positive change in the distance towards the goal destination, and when choosing the corresponding action, the AGV can take 8 consecutive greedy steps towards its goal. The third direction (-1,10) is also promising, as the AGV can continue its greedy path to the goal for 10 consecutive steps by taking a small detour. It is up to the DRL agents to learn a policy in which this consideration is effectively learned.

Given the requirement to work with a consistently sized state space, dummy values ('9999') need to be included to fill up the state space in situations in which less than J directions can be pursued. With this, a state representation in which only two actions are feasible (1 for remaining idle and 2 for continuing a path) can for example be represented like this:

$$< (0,8), (1,8), (9999, 9999), (9999, 9999) >$$

Action space A - The action space maps the available directions an agent can pursue from the various position locations in a segment-based layout to corresponding moving actions. The action space is similar for every state, which is equal to the maximum number of possible directions that can be pursued in a state of a segment-based layout:

$$A = \{a_1, \ldots, a_j, \ldots, a_J\}$$

With this, the action space is equal in every state despite that some of the actions are invalid. These invalid actions are actions that direct the AGV in impossible directions and need to be adequately learned by the DRL algorithm. The action space in layouts in which maximum 5 directions can be pursued in a particular state is represented as follows:

$$A = \{1, 2, 3, 4, 5\}$$

State Transition Function P - A selected action will traverse an AGV to its next position location. This action takes one time step t, where t is defined as the action duration. In this work, we assume that the execution of all actions lasts equally long. Subsequently, the state of the AGV will be updated to the new state which is situated in the new position location.

Reward function R - To learn to schedule actions for individual AGVs towards goal destinations as fast as possible, a rewarding structure has been identified. This function returns rewards to AGVs which take actions towards goal destinations and penalizes AGVs for remaining idle or heading in the wrong direction. As not all actions are feasible in every state (can be invalid or blocked by a prior planned AGV) the following reward structure has been identified such that quality and applicability of the actions are learned:

$$R: \begin{cases} \text{Invalid action chosen:} & -10 \\ \text{Blocked action chosen:} & -5 \\ \text{Idle action chosen:} & -1.5 \\ \text{Valid action chosen:} & \begin{cases} 0 & \text{if change } = 0 \\ +3.5 & \text{if change } > 0 \\ -2.5 & \text{if change } \geq -2.5 \text{ and } < 0 \\ -5 & \text{if change } \geq -10 \text{ and } < -2.5 \\ -10 & \text{if change } \geq -25 \text{ and } < -10 \\ -25 & \text{if change } < -25 \end{cases} \end{cases}$$

A Proximal Policy Optimization algorithm (PPO) described in [10] was subsequently used for learning the applicability of the actions. The PPO algorithm is found to perform well and is widely used because of its ease of implementation. To successfully train the algorithm, a training algorithm is created which can be found in Algorithm 2. The training starts with the initialization of the model and a set of training scenarios. These training scenarios consist of conflict-free problem instances in which AGVs are assigned with an initial position location and goal destination, and path plans have been prior defined with WHCA*S for a random number of the AGVs. This allows for learning with realistic path plans of other AGVs. In each training episode, a training scenario is randomly chosen and the first AGV with no prior defined path plan is selected. This AGV is identified as the 'learning AGV'. Within the training episode, actions are predicted for this learning AGV based on its state representation S. Predicted actions are checked for feasibility, and if it is found to be infeasible, a penalty R is returned to the DRL agent after which a new action is predicted for the same state. If an action is feasible, a reward R is returned and the action is stored in the WHCA*S reservation system. The state is updated based on other path plans in the reservation system. The training episode continues until W feasible actions have been predicted. The updates of states and corresponding rewards are subsequently used for updating the policy of the PPO algorithm.

Algorithm 2 PPO for segment-based MAPF

```
PPO initialization
training scenarios initialization
for episode = 1 to M do
    random select training scenario
    initialize learning AGV
    initialize state S
    while Terminate == False do
        select action A based on S and policy π_θ
        if action == feasible then
            receive reward R
            update state S
        else
            receive penalty R for infeasible action
    compute advantage estimates Â_t, ..., Â_T [10]
    optimize L_t, via mini batch gradient descent [10]
```

V. Experiments

We test the performances of WHCA*S and WHCA*S-RL by solving problem instances with different numbers of AGVs in the segment-based layout displayed in Figure 1. We first describe the training details of the PPO DRL-model and the simulation model that is used for the conduction of the experiments. We then present the results of the experiments. Finally, some additional problem instances with varying congestion situations are used for elaborating on the performance of the deployed DRL model within WHCA*S-RL.

A. Training details for PPO

The training of PPO was performed for 4.500.000 steps with 20.000 prior defined training scenarios in the layout presented in Figure 1 on a Processor Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz with 32GB of RAM. This took approximately 15 hours of training. In each episode, 20 valid steps need to be found for an individual AGV and training is performed in 10 parallel environments. The model parameters selected for training are similar to [10]. Since some actions are more often feasible than others, the training procedure was performed with an entropy loss coefficient value of 0.08. This encourages more exploration during training, ensuring the learning of the applicability of the less frequent feasible actions and prevents the over-fitting of more frequent feasible actions.

B. Simulation model

To investigate the performance of the different approaches a simulation model is developed. In this simulation model, a MAPF problem instance is initialized. This instance consists of a segment-based layout in which a set of AGVs are initiated on a position location and are assigned with a goal destination. The simulation model interacts with the path planning approaches for finding a valid joint path plan that directs AGVs in a conflict-free manner towards their goal destinations. The found joint plan is subsequently deployed in the simulation model, which simulates the individual path plans that AGVs pursue to reach their goal destinations. The simulation is terminated once all goal destinations have been reached. The interaction between the simulation model and path planning approaches is displayed in Figure 6.

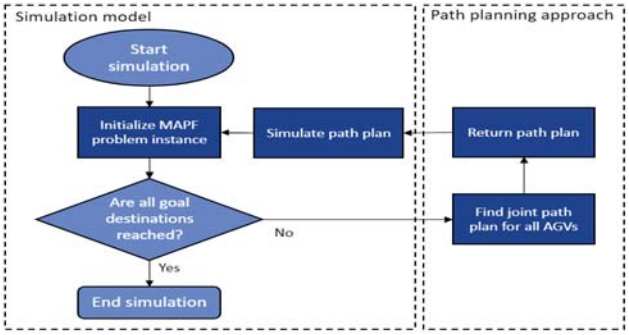


Fig. 6: Flowchart of interaction between simulation model and the path planning approach

C. Performance of WHCA*S and WHCA*S-RL

The proposed WHCA*S and WHCA*S-RL approaches are compared in terms of makespan (total elapsed time steps for all AGVs to reach the initialized goal destinations) and average flowtime (average time required for individual AGVs for reaching goal destinations). Different problem instances have been defined with up to 35 AGVs in the segment-based layout presented in Figure 1. This layout is selected to provide routing freedom to AGVs for reaching goal destinations. Problem instances are generated by randomly initializing AGVs in a conflict-free manner in the layout, and by assigning goal destinations to the vehicles at random. Each problem instance was subsequently solved 10 times with both methods (WHCA*S and WHCA*S-RL). The window length W for finding paths is set for 20 steps, of which 10 steps are executed (K). AGVs that reach goal destinations are required to stay at its goal destination location for one time step, representing the pick-up or drop-off time of an item.

A subset of the performance results can be found in Figure 7, displaying the performance results of both approaches in 12 selected problem instances (2 instances per fleet size configuration). The results are displayed in a box plot in order to summarize the distribution of performances. This variance in performance is caused by the random action selection for AGVs that have reached their initial goal. Also, the policy for predicting actions in the WHCA*S-RL approach can return different actions for similar state observations, causing variance in solution quality of a problem instance that is solved multiple times with the WHCA*S-RL approach.

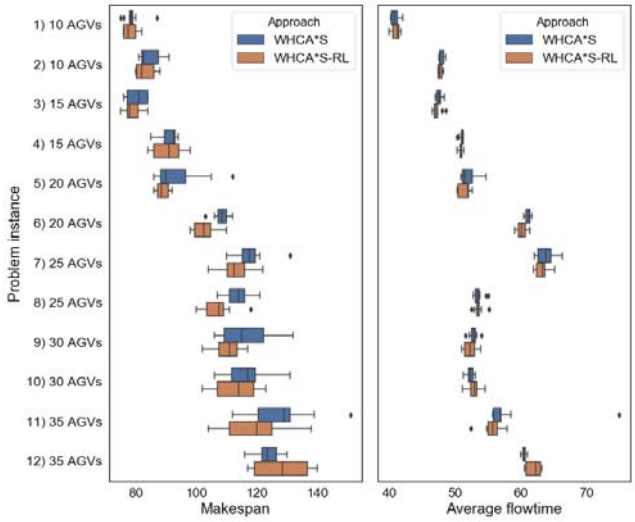


Fig. 7: Performance of WHCA*S and WHCA*S-RL for a subset of problem instances

What can be observed from Figure 7 is that the WHCA*S-RL approach is well capable of predicting actions to route AGVs faster towards goal destinations; path plans identified with WHCA*S-RL require on average a lower makespan and average flowtime for solving the identified problem instances. This improvement in performance is most significant in problem instance 11, in which the average joint path plan found by the WHCA*S-RL approach requires 7% less time for routing all AGVs towards goal destinations (makespan), while the average AGV reaches its goal destination 5% faster (average flowtime). This indicates that the policy learned is well capable to select actions to avoid and relieve congestion. This is found to be especially effective in problem instances in which more AGVs have been initialized (20+), as (more) congestion can be expected in these scenarios (e.g in problem instance 11).

In problem instances with fewer AGVs (see problem instance 1,2,3,4), the shortest paths to goal destinations obtained with WHCA*S remain optimal and no/less improvement is obtained by deviating from this path with WHCA*S-RL. However, in problem instances with very congested traffic situations the DRL tends to predict more idle actions, resulting in a drop in performance (see problem instance 12). This can be explained by the absence of these situations in the training scenarios.

D. Performance under varying congested situations

Additional problem instances were defined to investigate the performance of the DRL model for making decisions in different congested situations. In these instances, AGVs have been initiated in groups with initial position locations s_i and goal destinations g_i close to each other in a way that the groups of vehicles are expected to reach intersections at the same time. With this, congestion can be expected as only one AGV at a time is allowed to cross. Four different groups of AGVs have been initiated (one for each travel path towards the center, see Figure 1), and problem instances have been defined with up to 24 AGVs (6 per group). The results can be found in Table I, displaying the percentage change in makespan and average flowtime for WHCA*S-RL in comparison with WHCA*S.

TABLE I: Performance of WHCA*S-RL in increasing congested problem instances

Instance ID	Nr. of AGVs	Makespan	Average Flowtime
RZNSG001	4 AGVs	0%	0%
RZNSG002	8 AGVs	-3%	0%
RZNSG003	12 AGVs	-7%	-2%
RZNSG004	16 AGVs	-4%	-1%
RZNSG005	20 AGVs	+2%	+3%
RZNSG006	24 AGVs	+7%	+4%

Table I shows that the WHCA*S-RL approach is capable of finding path plans in which AGVs reach goal destinations faster in scenarios with modest congestion (see RZNSG003 & RZNSG004). The ability to improve plans decreases in more densely congested traffic situations; typically in situations in which multiple idle actions need to be predicted for the vehicles due to ongoing occupation of surrounding position locations. In these situations, additional idle actions are predicted by the DRL model for the vehicles. This causes the AGV to arrive later at goal destination and slows down oncoming traffic. This can be explained by the absence of these scenarios in the training scenarios used for learning the deployed decision making policy. Retraining with these severely congested training situations is therefore expected to improve the trained policy.

VI. Conclusion and future work

In this work we proposed a cooperative planning approach for solving a multi-agent path finding problem in segment-based layouts in which the search is windowed and conflicts are resolved by cooperation. In addition, we presented a new approach that relies on combining Deep Reinforcement Learning with the proposed cooperative planning approach. Through a set of experiments, we have shown that the Deep Reinforcement Learning approach for defining path plans successfully improves the cooperative planning approach by deviating from shortest paths such that traffic congestion is both avoided and relieved. On average, AGVs reach goal destination faster and it takes less time for the problem to be resolved, up to 5% and 7%.

For future work, we plan to retrain the DRL with more congested training situations. Subsequently, we plan to explore continuous routing of AGVs with updated goal destinations. This would allow us to see how the approaches perform in terms of throughput (transported items per time unit). Further, it would be interesting to see how the identified methods scale to more vehicles in larger layouts and to investigate how conflict handling can be improved such that conflicts are prevented from occurring rather that resolved in the planning.

References

[1] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[2] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[3] Barry Douglas and Mark Weninger. Automated guided vehicle system, January 9 2018. US Patent 9,864,371.

[4] Maxim Egorov. Multi-agent deep reinforcement learning. 2016.

[5] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[6] Neville Mehta, Prasad Tadepalli, and A Fern. Multi-agent shared hierarchy reinforcement learning. In *ICML Workshop on Rich Representations for Reinforcement Learning*, page 45, 2005.

[7] Cane Punma. Autonomous vehicle fleet coordination with deep reinforcement learning. 2018.

[8] Malcolm Ryan. Constraint-based multi-robot path planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 922–928. IEEE, 2010.

[9] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[11] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[12] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[13] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[14] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[15] Roni Stern. Multi-agent path finding–an overview. In *Artificial Intelligence*, pages 96–115. Springer, 2019.

[16] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[17] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[18] Vanderlande. Bagage handling vanderlande, 2019.

[19] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.