CHALMERS
UNIVERSITY OF TECHNOLOGY

Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots

Citation for the original published paper (version of record):
Popolizio, F., Vinetti, M., Combrink, A. et al (2024). Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots. IEEE International Conference on Automation Science and Engineering: 3063-3068. http://dx.doi.org/10.1109/CASE59546.2024.10711693

(article starts on next page)

2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)
August 28 - September 1, 2024. Bari, Italy

2024 IEEE 20th International Conference on Automation Science and Engineering (CASE) | 979-8-3503-5851-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/CASE59546.2024.10711693

Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots

Francesco Popolizio[1], Martina Vinetti[1], Alvin Combrink[1],
Sabino Francesco Roselli[1], Maria Pia Fanti[2] and Martin Fabian[1]

***Abstract*— This work presents a *Fleet Manager* for a fleet of Autonomous Mobile Robots (AMRs) that perform material handling tasks in a shared environment. The *Fleet Manager* assigns AMRs to newly released tasks, computes paths for them to travel to the task's locations, and schedules their travel along the computed paths so that conflicts with other AMRs are avoided. The objective is for each AMR to complete its task as quickly as possible, to then be assigned a new task.**

The *Fleet Manager* works *online*, assigning a released task to the AMR closest to the task's location, and then computing the path and schedule to fit in with the already assigned and executing AMRs. Conflicts occur when, in order to reach their targets, AMRs would have to simultaneously occupy the same space. Resolving this is done by appropriate scheduling, or by moving idle AMRs out of the way. For fleet management to be practicable, the computation time for assigning an AMR to a task and computing its path and schedule must be negligible compared to other system times.

Tests were conducted to evaluate the performance of the *Fleet Manager* on a number of benchmark problem instances, counting up to hundreds of AMRs. The results show that the presented *Fleet Manager* can handle these systems quickly enough to be practically useful in real industrial scenarios.

I. INTRODUCTION

Recent years have seen a growing interest in the use of Autonomous Mobile Robots (AMRs) in applications such as manufacturing plants, automated warehouses [1], airport operations [2], office buildings [3, 4], power grids [5], etc. Potentially, AMRs increase flexibility, robustness, and efficiency of the systems [6], especially so with fleets of AMRs simultaneously performing tasks in shared environments. However, as the number of AMRs in a system grows, so does the complexity of planning and controlling their movements to ensure conflict-free interaction between them [7].

Managing a fleet of AMRs relates to *Multi-Agent Path Finding* (MAPF), which is the problem of deciding collision-free paths for multiple agents from their start locations to goal locations in a shared, known, environment [8]. Goal locations are defined by *tasks* that are completed when the agent assigned to the task reaches the specified location. In the *offline* MAPF, all tasks are known beforehand, thus a static task assignment can be computed, and the computation time is typically not critical. In the *online* MAPF, by contrast, tasks are not known beforehand but are released to the system at a priori unknown times. Thus, agents that have completed their tasks have to be assigned new tasks, so the computation time must be short enough for the new assignment not to be obsolete once it has been computed.

Though the offline MAPF has received a lot of interest [9, 10, 11, 12, 13], the online MAPF has received less so. Still, several algorithms to solve the online MAPF have emerged in recent times. *Token Passing* (TP) [14] solves the online Multi-Agent Pickup and Delivery problem (MAPF with two goal locations). The algorithm is *decoupled* and paths of already assigned agents are not changed. It also assumes *stay at target* [8], where an idle agent stays in its current location until assigned a task or moved out of another agents' way. *Rolling-Horizon Collision Resolution* (RHCR) [15] re-plans paths at fixed intervals and uses the concept introduced in WHCA* [12] of resolving conflicts within a time-window only. RHCR is combined with the offline MAPF solvers CA* [12], CBS [10], *Enhanced CBS* [16], and *Priority-Based Search* [17].

For a more in-depth overview of MAPFs proposed methods and variants, see for example [8, 18, 19].

This work presents a *Fleet Manager* to solve a decoupled variant of the *warehouse model* [8] from the online MAPF problem, with the *stay at target* assumption, where an agent receives a new task when completing its assigned one. In this context, since incoming tasks need to be completed as soon as possible, the arrival and release times of tasks coincide. Tasks are released online without the knowledge of future tasks, and decisions are made in real-time so that short computation time is essential. The proposed method manages one task at a time, computing the path and the schedule for the assigned AMR, while keeping existing schedules for other AMRs unchanged.

In this context, a newly released task is assigned to the nearest available AMR and a path connecting the AMR's current location to the task location is computed. Such a path should be as short as possible, while at the same time avoiding, if possible, the task locations of the other AMRs and the current locations of idle AMRs. For this path, a *schedule* is computed, that is, each location along the path is designated a time when the AMR should be there. Only discrete time steps are considered.

For AMRs with non-overlapping paths, computing schedules is trivial since each AMR can operate independently, without any risk of conflict. However, scheduling becomes complicated when the paths of different AMRs overlap, due to having to avoid conflicts. Conflicts are avoided by

We gratefully acknowledge the Vinnova project CLOUDS (Intelligent algorithms to support Circular soLutions fOr sUstainable proDuction Systems) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Division of Systems and Control, Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden {frapop, vinetti, combrink, rsabino, fabian}@chalmers.se

[2]Department of Electrical and Information Engineering, Polytechnic of Bari, Bari, Italy mariapia.fanti@poliba.it

guaranteeing that AMRs never occupy the same location at the same time. Moreover, the *Fleet Manager* ensures that idle AMRs do not obstruct moving AMRs. When necessary, special tasks are assigned to idle AMRs to move them out of the way to resolve conflicts.

The scheduling of the AMRs is formulated as an optimization problem that, given as input the path of an AMR to schedule together with the paths and schedules of the other already scheduled AMRs, results in a conflict-free schedule. This paper presents a Tailor-Made Scheduler (TMS) specifically designed to solve this problem and, for comparison, a scheduler with the optimization solver Z3 [20] (Z3S), available under free academic license.

A set of small and medium-sized problem instances are constructed, varying the number of tasks released and the number of AMRs available, within the same plant. The *Fleet Manager* is benchmarked using both TMS and Z3S, and TMS outperforms Z3S in all experiments in terms of computation time.

Furthermore, a set of large problem instances are generated to test the *Fleet Manager* using TMS, to evaluate the performance in terms of throughput. The results are compared with others reported in the literature, demonstrating superior performance for the presented *Fleet Manager*.

The outline of this paper is as follows: Section II includes the problem definition with the inputs and assumptions, including the mathematical formulation of the scheduling problem, the management of the conflicts and the entire fleet management; Section III presents the results obtained under different simulation scenarios; final remarks and conclusions are given in Section IV.

II. Problem Definition

The plant in which the fleet operates is abstracted into a weighted, undirected, connected graph, $\langle \mathcal{N}, \mathcal{E}, \mathcal{W} \rangle$, where $\mathcal{N}$ is the set of nodes, representing possible locations for AMRs and tasks, $\mathcal{E}$ is the set of edges, representing the road segments between the nodes, and $\mathcal{W} : \mathcal{E} \to \mathbb{N}^+$ maps edges to the time it takes to traverse them. AMRs do not stop on an edge while traversing it, and all AMRs have the same, constant speed. An AMR $a \in \mathcal{A}$ (with $\mathcal{A}$ the set of all AMRs) occupies a node $n_t^a \in \mathcal{N}$ at time $t \in \mathbb{N}^+$. AMRs cannot simultaneously occupy the same nodes or traverse the same edges in opposite directions, as this leads to collisions. However, multiple AMRs can simultaneously traverse the same edge in the same direction.

A *task* is a triple $\tau = \langle n_\tau, r_\tau, z_\tau \rangle \in \mathcal{N} \times \mathbb{N}^+ \times \mathbb{N}^+$, with n_τ the *task node* where the AMR assigned to the task is to go, r_τ the *task release time*, the earliest time when the assigned AMR can start, and z_τ the *task service time*, the time that the assigned AMR has to stay at the task node before the task is completed. Let $\mathcal{T}$ be the set of all tasks.

A. Fleet Manager Overview

The *Fleet Manager*, Fig. 1, is the central algorithm for handling the fleet of AMRs. To begin with, at time step t, all released tasks $\tau \in \{i \in \mathcal{T} \mid r_i = t\}$ are added to the

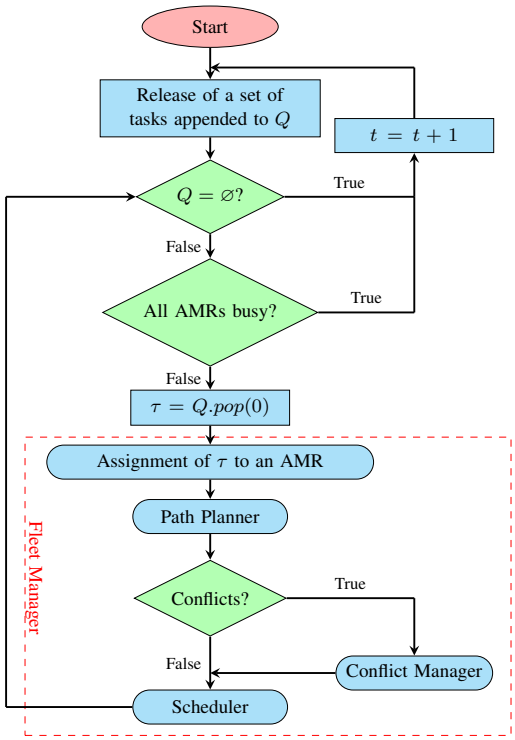

Fig. 1: Flowchart of the *Fleet Manager*.

waiting list $\mathcal{Q}$, an ordered set containing the tasks to be performed. The tasks in $\mathcal{Q}$ are then handled one at a time until $\mathcal{Q}$ is empty, or there are no more available AMRs. The *Fleet Manager* handles a task as follows: the first task in $\mathcal{Q}$ is assigned to the available AMR closest to the task node, and a path from the AMR's current location to the task node is computed by the *Path Planner*. *Conflicts*, referring to situations where idle AMRs along the computed path must be moved, are checked for and solved by the *Conflict Manager*.

Finally, the *Scheduler* decides the time steps at which the assigned AMR travels along the path, dependent on existing schedules of other AMRs. At this point, the task τ has been handled.

B. The Path Planner

A path $p = \langle n_0, n_1, \ldots, n_k \rangle \in \mathcal{N}^*$ is an ordered set of nodes with $n_i \in p$ for $i = 0 \ldots k$. Given an AMR a, its position n_t^a at time t, and an assigned task τ, the *Path Planner* computes a path where $n_0 = n_t^a$ and $n_k = n_\tau$, and each pair $\langle n_i, n_{i+1} \rangle \in \mathcal{E}$. Besides finding a short path, the *Path Planner* tries to avoid conflicts with other AMRs by avoiding nodes where idle AMRs could currently or eventually be. For this objective, a *Path Planning Heuristic* determines whether to circumvent a node that would cause a conflict or to include it in the path and resolve the conflict, aiming to minimize task completion duration. This determination involves considering estimated durations to move the conflict AMR out of the way. The strategy is to temporarily increase the weights of edges connected to nodes where either idle AMRs $a_I \in \mathcal{A}_I$ are located (i.e. $n_t^{a_I}$) or are task nodes of assigned but uncompleted tasks

$\tau_U \in \mathcal{T}_{A\&U}$, i.e. where idle AMRs will be in the future. The weight penalty w_i^+ for node n_i is calculated as:

$$w_i^+ = \frac{|\{a_I \in \mathcal{A}_I \mid n_t^{a_I} = n_i\}|}{2} + \sum_{\tau_U \in \mathcal{T}_{A\&U} | n_{\tau_U} = n_i} \frac{1 + z_{\tau_U}}{2}$$

where $\mathcal{A}_I$ is the set of idle AMRs and $\mathcal{T}_{A\&U}$ is the set of task nodes of the last assigned and uncompleted task for each AMR. Note that w_i^+ can only be either 0, 1/2, or $(1+z_{\tau_U})/2$, since only one AMR can occupy a node n_i at a time. w_i^+ is added to the weight of all edges connected to node n_i. Dijkstra's algorithm [21] is then used to find the shortest path from n_t^a to n_τ in the graph with modified edge-weights.

C. Scheduler

Once the path is defined, the arrival time at each node along the path is decided, i.e. the AMR is scheduled.

A mathematical model of the scheduling problem is presented below, where at a time t the AMR a is scheduled along the path p to execute task τ. Note that at time t there may be some other AMRs moving in the plant along previously computed paths, according to previously computed schedules.

Let the integer decision variable x_n, for $n \in \mathcal{N}$, model the discrete time step at which a arrives at node n. The *objective function* to minimize is the arrival time at the task node n_τ. The list of parameters used to formulate the problem is given as:

- p, path of the assigned AMR, with p_0 and p_*, respectively, first and last node of p
- p', currently scheduled path traversed by $a' \in \mathcal{A}$, with p'_0 and p'_*, respectively, first and last node of p'
- $\mathcal{P}$, set of currently scheduled paths, with $p' \in \mathcal{P}$
- n_{prev} and n_{next}, respectively, the previous and the next node of n, with $n \in p$
- $p_{*,prev}$ is the node preceding $p_* \in p$
- t, integer variable indicating the current discrete time step
- $z_{p'}$, service time of the task located at $p'_*, \forall p' \in \mathcal{P}$
- $w_{n,n_{next}} = \mathcal{W}(\langle n, n_{next} \rangle)$, integer weight of the edge $\langle n, n_{next} \rangle \in \mathcal{E}$
- $s_{n'}$, integer variable that indicates the time at which node n' of path $p' \in \mathcal{P}$ is visited by a'
- n'_e, first node in p' upstream of the edges shared between p' and p

Model: The optimality criterion is to minimize the time a reaches its task node, p_*:

$$\min x_{p_*} \tag{1}$$

subject to:

$$x_n = -1 \qquad \forall n \in (\mathcal{N} \setminus p) \tag{2}$$

$$x_{p_0} = t \tag{3}$$

$$x_{n_{next}} - x_n \geq w_{n,n_{next}} \qquad \forall n \in p \setminus \{p_*\} \tag{4}$$

$$\big[(x_n < s_{n'}) \wedge (x_{n_{next}} - w_{n,n_{next}} < s_{n'})\big] \vee \big[x_n > s_{n'_{next}} - w_{n',n'_{next}}\big], \quad \forall n \in p \setminus \{p_*\}, n' \in p' \setminus \{p'_*\}, n = n' \tag{5}$$

$$x_{p_*} > s_{n'_{next}} - w_{n',n'_{next}} \qquad \forall n' \in p', p_* = n' \tag{6}$$

$$\big[(x_n < s_{p'_*}) \wedge (x_{n_{next}} - w_{n,n_{next}} < s_{p'_*})\big] \vee \big[x_n > s_{p'_*} + z_{p'}\big], \quad \forall n \in p \setminus \{p_*\}, n = p'_* \tag{7}$$

$$\big[x_n < s_{n'_e} - \sum_{r=n'}^{n'_{e,next}} w_{r,r_{prev}}\big] \vee \big[x_n > s_{n'}\big], \quad \forall n \in p \setminus \{p_*\}, n' \in p', n = n', n_{next} = n'_{prev} \tag{8}$$

$$x_{p_*} > s_{n'_e} \qquad \forall n'_e \in p', p_* = n'_e, p_{*,prev} = n'_{e,next} \tag{9}$$

Constraint (2) sets an invalid arrival time for all the nodes $n \in \mathcal{N}$ that are not involved in p; (3) sets the arrival time of a in p_0 equal to the current time step; (4) imposes the minimum arrival times for each $n \in p$, by considering the edge's travel time. Constraints (5), (6), (7) guarantee that in each node there is only one AMR at a time. When p and p' share a node ($n = n'$), there are two general possibilities defined by (5): the first case is when a can pass through n before a', but it has to leave n before a'; the second is that a has to wait for a' to traverse n. A different case arises when the shared node between p and p' is p_*, (6). In that case, a will visit p_* after a' to avoid blocking it. The last case occurs when the shared node between p and p' is p'_*, (7): a can pass through $n = p'_*$ before p' if it can leave that node before a'; otherwise, a could visit n at least after a' completes its task (also considering its service time $z_{p'}$). In this case, after a' completes its task, the *Conflict Manager* will move it from that node. Constraints (8), (9) deal with avoiding multiple AMRs on the same edges in opposite directions. In fact, based only on the constraints (5), (6), (7), a situation as in Figure 2 would not be resolved and would cause a collision between a_0 and a_1 in the time between time steps 3 and 4.

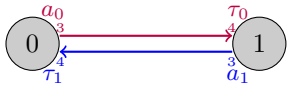

Fig. 2: Collision between two AMRs on an edge.

Constraint (8) defines the two general cases to avoid two AMRs on an edge at the same time: the first case is when a can cross the edge or sequence of shared edges before a' enters it. If this solution is not feasible, then the second case is that a enters the sequence of shared edges after a' passes through it. The latter is the only possible solution when the sequence of shared edges extends up to p_*, see (9).

Once a has been scheduled for the path p, i.e. the decision variables x_n of the problem have been determined, the schedule is stored in S, the set of all the computed schedules s. The set S contains the history of all the schedules computed up to the current step; this is used to know the position of the scheduled AMRs at each instant of time, so as to schedule the following AMRs so that there are no collisions between the moving AMRs.

The problem (1)–(9) can be solved by any general-purpose optimization solver, such as Z3. However, since only one AMR is scheduled at a time, on a predefined path, and the schedules of other AMRs are fixed ($s \in S$), a greedy algorithm can find a schedule without an optimization solver. Thus, a Tailor-Made Scheduler (TMS) that enforces constraints (2)–(9) was developed. In the TMS, those constraints are represented by conditions to compute all the x_n variables, and to be sure that all the constraints are met they are incorporated into a loop. Within this loop, all conditions are evaluated sequentially, over all x_n. The loop persists until all conditions are satisfied. During each iteration, the loop checks whether the values of the variables have changed from the previous iteration. If changes are detected, the loop recommences the evaluation of all conditions; otherwise, the loop terminates.

D. The Conflict Manager

A conflict arises when the path found by the Path Planner does not avoid nodes with idle AMRs or task nodes of currently executing tasks. Conflicts cannot be resolved by the *Scheduler*, as it is only capable of deciding the timing of the AMR a's movements to avoid occupying the same nodes or edges simultaneously with other moving AMRs. When conflicts are detected, the *Fleet Manager* invokes the *Conflict Manager* to move the AMRs causing the conflicts out of the way. Two different types of conflicts can be identified:

- *Type I*, conflict with an idle AMR, $a_I \in \mathcal{A}_I$;
- *Type II*, conflict with a currently scheduled AMR, a_S, in its task node.

For Type I conflicts, a_I must certainly be moved. However, Type II could be resolved without moving the AMR causing the conflict. In the best case, a has enough time to arrive at and leave the task node of a_S before a_S. Otherwise, a_S arrives at its task node before a, becomes idle, and must be moved.

The *Conflict Manager* can be described by the flow chart in Figure 3, and receives as input:

- J, set of AMRs belonging to the Type I conflict;
- Y, set of AMRs belonging to the Type II conflict.

The first step of the *Conflict Manager* is to move all AMRs in J out of the way, one at a time. To move an AMR, it is first necessary to select the nearest node that is not the task node of a task assigned to another AMR, that the AMR can be moved to. Subsequently, a path to the selected node is calculated and saved in $\mathcal{P}$, which ideally does not cause further conflicts, else this becomes a *cascade conflict* (see below). Finally, the AMR is scheduled to travel along the calculated path. This schedule will also be recorded in the set S and the corresponding *conflict task* in $\mathcal{T}$, which is a specific task with zero service time.

Regarding Type II conflicts, among all AMRs in Y, only those that actually cause a conflict are moved. For this, a temporary schedule is calculated for the main task, i.e. the task that was being addressed before entering the *Conflict Manager*. From the temporary schedule it is determined

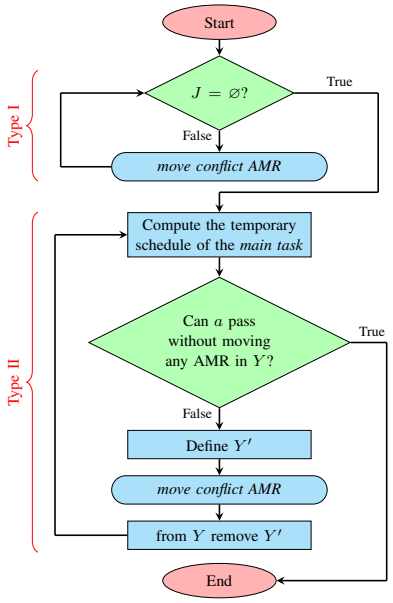

Fig. 3: Flowchart of the *Conflict Manager*.

which AMRs in Y will reach the corresponding task node before a does; so only these AMRs, grouped in the set $Y' \subseteq Y$, are moved. After moving the AMRs in Y' the *Conflict Manager* checks again if there are other conflicts with the AMRs in Y, caused by the schedules of the AMRs in Y', calculating again a temporary schedule of the main path. Provided that there is always at least one node to move the AMR causing the conflict to, the *Conflict Manager* ends when the temporary schedule no longer creates conflicts with the AMRs in Y.

When an AMR causing a conflict is moved, it cannot be guaranteed that it does not generate further conflicts. An example of this cascade conflict is shown in Figure 4.

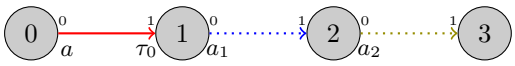

Fig. 4: Example of cascade conflict.

In this case, the AMR a wants to execute the task τ_0, released at time 0; in the task node of τ_0 there is an idle AMR, a_1. Therefore, to reach τ_0, it is necessary to move a_1. The only solution is to move a_1 to the right, to node 2; but node 2 is occupied by another idle AMR, a_2. To solve this situation, the *Conflict Manager* will move a_2 to node 3, a_1 to node 2, before it can execute the main task, i.e. τ_0. When managing cascade conflicts, in order to minimize delays, each AMR will move at its earliest possible time given the scheduling constraints. In the example of Figure 4, all AMRs move simultaneously at time 0, since this allows a to reach τ_0's task node at its earliest possible time.

The *Conflict Manager* recursively handles an arbitrary number of cascade conflicts, given that it is possible to move the last conflicting AMR in the cascade (and the computation does not exceed the hardware's memory).

Note that certain rare edge-cases cannot be handled by

the *Conflict Manager*, potentially leading to deadlocks. In the experiments in Section III this has occurred once, in a scenario involving 300 AMRs with 140 000 tasks to be executed over 5000 time steps. Currently, this issue is simply disregarded; however, in the future, strategies to handle such situations will be developed.

III. Experimental Evaluation

For experimental evaluation, the 33×46 *Fulfilment warehouse* map of [15, 22] is used, see Figure 5, containing 1278 nodes that can be occupied by AMRs. Initially, the AMRs are randomly positioned at any orange, gray, or blue cells. Given a time horizon T and release rate r_r, Tr_r tasks (rounded to the nearest integer) are generated with task nodes and release times uniformly sampled from blue positions and $[0, \ldots, T-1]$, respectively. Service times are set to 0.

All tests are performed on an Intel i7-1185G7 4.80 GHz CPU with 32 GB of 3200 MHz RAM.

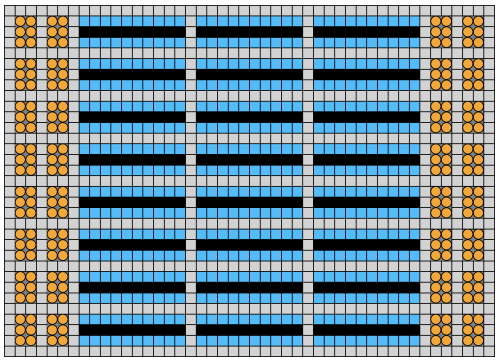

Fig. 5: Fulfilment warehouse map. Blue squares are possible task nodes, black squares are obstacles. Orange circles are used as initial positions in [15, 22], but here AMRs may start at any orange, blue or gray position.

Table I shows the average *Fleet Manager* computation time per task, in seconds, using Z3S and TMS, respectively, on the same problem instances. The computation times at steady-state are evaluated, just considering the average computational time per task completed in the last 500 time steps, with $T = 1000$, $r_r = 1.5$, and the number of AMRs ranging from 25 to 750.

TABLE I: Average computation time (seconds) per task.

	Number of AMRs						
	25	50	200	400	500	600	750
Z3S	2.740	0.580	0.347	0.292	0.277	0.271	0.261
TMS	0.135	0.004	0.004	0.004	0.005	0.005	0.006

The results show superior performance of TMS over Z3S. The worst-case scenario is 25 AMRs, due to the limited number of AMRs resulting in lengthy paths. Tasks are typically assigned to the nearest available AMR; however, with a small number of AMRs and a high number of tasks, a long queue of tasks forms, leading to newly released tasks being assigned to the first available AMRs, even if they are distant from the task node. This issue decreases with more AMRs, allowing shorter *Fleet Manager* computation time.

While Z3S shows expected computational improvements with more AMRs, TMS does not confirm this expectation. Tests indicate that with a larger fleet, it takes longer for the Path Planner to find a path compared to the case of a smaller fleet. The reason could lie in the Path Planning Heuristics, which will have to iterate over a greater number of AMRs when modifying the edge weights. However, the computation time required for the Path Planner is in the order of milliseconds, so it is negligible when using Z3S; conversely, when using TMS, the computation time of the Path Planner becomes comparable to that required by the scheduler.

Given the significant computational time difference in Table I, further tests will use TMS.

Figure 6 shows results for five experiments for each release rate $r_r \in \{1.0, 1.2, 1.4\}$, with $T = 3000$ and 25 AMRs. Similar to the results in Table I, it is the steady-state metrics that are evaluated. Therefore, the computation time is defined as the average *Fleet Manager* computation time for tasks completed in the past 500 time steps. The throughput is defined as the average number of completed tasks over the past 500 time steps. It is shown that the throughput is able to match lower release rates ($r_r = 1.0$) but for higher release rates it becomes more common for the throughput to drop ($r_r = 1.4$ and one experiment with $r_r = 1.2$). Probably the system becomes saturated with tasks such that few AMRs are available at any given time. Since tasks are assigned to the nearest available AMR, having few to choose from likely results in longer average path lengths. A longer path means that more time is needed to complete a task, as well as more conflicts arising. This is supported by the correlation of growing queue sizes and computation times for the experiments where performance deteriorates.

Table II shows the average throughput for TMS and the reported throughput for RHCR [15] for different numbers of AMRs. The TMS throughput is the average over all experiments where neither performance deteriorates (defined here as where the queue size grows larger than $2\,|\mathcal{A}|$) nor deadlocks occur, and is calculated using the final steady-state throughput value, i.e. using the last 500 time steps. $T = 5000$ and the release rate is selected such that > 95 % of the 100 experiments can be used to calculate the throughput.

TABLE II: Throughput comparison.

Number of AMRs	25	60	100	140	200
TMS	1.10	4.01	8.51	12.97	19.50
RHCR [15]	-	2.33	3.56	4.55	-

IV. Conclusions

This paper presents an online *Fleet Manager* for AMR path planning and scheduling. Due to real-time requirements, short computation time is paramount, and experiments show

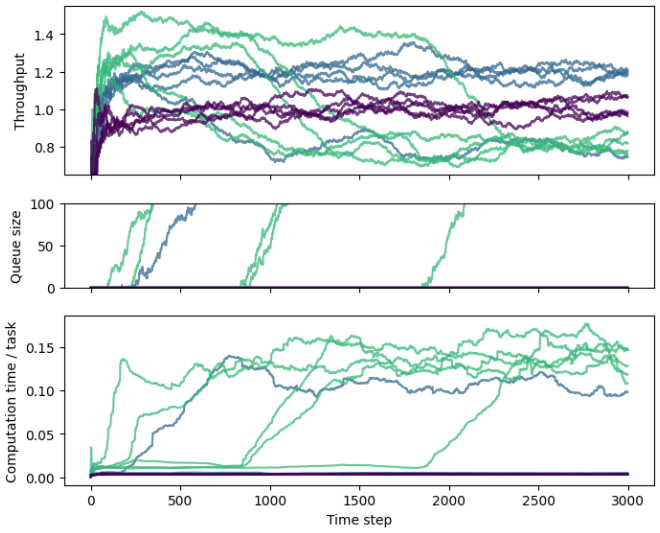

Fig. 6: Throughput (500 windowed), task queue size, and computation time (500 windowed, in seconds) of five experiments for each release rate $r_r = 1.0$ (purple), $r_r = 1.2$ (blue) and $r_r = 1.4$ (green).

that the *Fleet Manager* can handle large problem instances quickly, therefore being a suitable solution for real-world, large-scale applications.

In this work, the problem of scheduling AMRs along their paths is solved by means of a Tailor-Made Scheduler (TMS), and a freely available general-purpose optimization solver. These two approaches are compared over sets of benchmark instances, showing the TMS to be significantly faster than the general-purpose solver.

Further tests, using TMS, highlight the *Fleet Manager*'s performance in terms of throughput. Its performance surpasses those reported in existing literature. Moreover, the *Fleet Manager* scales well with respect to the number of AMRs and assigned tasks, for the same map size. However, limitations arise concerning the achievable throughput.

These findings collectively highlight the efficacy of the proposed *Fleet Manager* in real-world scenarios, offering enhanced computational efficiency and notable performance advantages over existing approaches.

REFERENCES

[1] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), pp. 9–19.

[2] Robert Morris et al. "Planning, Scheduling and Monitoring for Airport Surface Operations." In: *AAAI Workshop: Planning for Hybrid Systems*. 2016, pp. 608–614.

[3] Brian Coltin and Manuela Veloso. "Online pickup and delivery planning with transfers for mobile robots". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 5786–5791.

[4] Manuela Veloso et al. "Cobots: Robust symbiotic autonomous mobile service robots". In: *Twenty-fourth international joint conference on artificial intelligence*. AAAI Press, 2015, pp. 4423–4429.

[5] Rogério Sales Gonçalves and João Carlos Mendes Carvalho. "Review and latest trends in mobile robots used on power transmission lines". In: *International Journal of Advanced Robotic Systems* 10.12 (2013), p. 408.

[6] Giuseppe Fragapane et al. "Increasing flexibility and productivity in Industry 4.0 production networks with autonomous mobile robots and smart intralogistics". In: *Annals of operations research* 308.1-2 (2022), pp. 125–143.

[7] Kasim M Al-Aubidy, Mohammed M Ali, and Ahmad M Derbas. "Multi-robot task scheduling and routing using neuro-fuzzy control". In: *2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*. IEEE. 2015, pp. 1–6.

[8] Roni Stern et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 10. 1. 2019, pp. 151–158.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[10] Guni Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[11] Guni Sharon et al. "The increasing cost tree search for optimal multi-agent pathfinding". In: *Artificial intelligence* 195 (2013), pp. 470–495.

[12] David Silver. "Cooperative pathfinding". In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. Vol. 1. 1. 2005, pp. 117–122.

[13] Glenn Wagner and Howie Choset. "M*: A complete multirobot path planning algorithm with performance bounds". In: *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2011, pp. 3260–3267.

[14] Hang Ma et al. "Lifelong multi-agent path finding for online pickup and delivery tasks". In: (2017). arXiv: 1705.10868.

[15] Jiaoyang Li et al. "Lifelong multi-agent path finding in large-scale warehouses". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13. 2021, pp. 11272–11281.

[16] Max Barer et al. "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 5. 1. 2014, pp. 19–27.

[17] Hang Ma et al. "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7651–7658.

[18] Hang Ma. "Graph-based multi-robot path finding and planning". In: *Current Robotics Reports* 3.3 (2022), pp. 77–84.

[19] Oren Salzman and Roni Stern. "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems*. 2020, pp. 1711–1715.

[20] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[21] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[22] Minghua Liu et al. "Task and path planning for multi-agent pickup and delivery". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019, pp. 1152–1160.