Receding Horizon Re-Ordering of Multi-Agent Execution Schedules

Berndt, Alexander; Van Duijkeren, Niels; Palmieri, Luigi; Kleiner, Alexander; Keviczky, T.

DOI
10.1109/TRO.2023.3344051

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Robotics

Receding Horizon Re-Ordering of Multi-Agent Execution Schedules

Alexander Berndt, Niels van Duijkeren, Luigi Palmieri, *Member, IEEE*, Alexander Kleiner, and Tamás Keviczky, *Senior Member, IEEE*

***Abstract*—The trajectory planning for a fleet of automated guided vehicles (AGVs) on a roadmap is commonly referred to as the multi-agent path finding (MAPF) problem, the solution to which dictates each AGV's spatial and temporal location until it reaches its goal without collision. When executing MAPF plans in dynamic workspaces, AGVs can be frequently delayed, e.g., due to encounters with humans or third-party vehicles. If the remainder of the AGVs keeps following their individual plans, synchrony of the fleet is lost and some AGVs may pass through roadmap intersections in a different order than originally planned. Although this could reduce the cumulative route completion time of the AGVs, generally, a change in the original ordering can cause conflicts, such as deadlocks. In practice, synchrony is therefore often enforced by using a MAPF execution policy employing, e.g., an action dependency graph (ADG) to maintain ordering. To safely re-order without introducing deadlocks, we present the concept of the switchable action dependency graph (SADG). Using the SADG, we formulate a comparatively low-dimensional mixed-integer linear program that repeatedly re-orders AGVs in a recursively feasible manner, thus maintaining deadlock-free guarantees, while dynamically minimizing the cumulative route completion time of all AGVs. Various simulations validate the efficiency of our approach when compared to the original ADG method as well as robust MAPF solution approaches.**

***Index Terms*—Mixed integer programming, multi-agent path finding (MAPF), robust plan execution, scheduling and coordination.**

I. Introduction

MULTIPLE autonomous mobile robots (AMRs) have been shown to significantly increase the efficiency of performing intralogistics tasks, such as moving inventory in distribution centers [1]. Coordinating AMRs navigating a shared environment can be formulated as the multi-agent path finding (MAPF) problem [2]. The MAPF problem is to find trajectories for each AMR along a roadmap such that each AMR reaches its goal without colliding with the others, while minimizing a cost metric, such as the makespan or cumulative route completion time (also referred to as sum-of-costs). Throughout this manuscript, we refer to AMRs as automated guided vehicles (AGVs) to be consistent with the MAPF literature.

Minimizing temporal cost metrics when solving the MAPF problem has received a lot of attention in the literature [3]. However, even if an optimal MAPF solution is found, blindly executing the plans can still result in deadlocks when the AGVs experience delays. This introduces the need for plan execution policies, used to maintain the ordering between AGVs and thus avoiding deadlocks. Hönig et al. [4] proposed compiling an action dependency graph (ADG) from an MAPF solution to enforce the ordering during plan execution. However, this work, and most other works, such as the authors in [5], [6], and [7] considered AGVs that are only marginally delayed. This means that delays are seen as a lack of synchronization between AGVs, rather than significantly affecting the overall route completion times. With the advent of Industry 4.0 (such as the VDA5050 protocol [8] and the Robot Middleware Framework [9]), we turn our attention to AGV fleets navigating dynamic and complex environments occupied by humans and third-party vehicles. These dynamic environments are far less predictable than those typically considered in the MAPF literature, implying that AGVs can experience large delays when waiting for, e.g., a human to move out of its path.

These large, unpredictable delays can result in inefficient plan execution because the implicit ordering of the original MAPF solution requires AGVs to wait for largely delayed AGVs. Not adhering to this implicit ordering, however, can result in deadlocks. Approaches, such as in [10] and [11] propose re-ordering schemes, which maintain the deadlock-freeness properties of the original plan. The decision to switch the order between two robots or not, however, is based only on performance measures of the two involved AGVs. This means that although these switches are performed throughout the fleet, they do not necessarily lead to an overall performance increase in terms of a sum-of-costs or a makespan metric.

Contributions: To address the shortcomings of robust MAPF approaches and local path repair methods, we present the following.

Manuscript received 20 September 2023; accepted 14 November 2023. Date of publication 18 December 2023; date of current version 18 January 2024. This paper was recommended for publication by Associate Editor J. Alonso-Mora and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. This work was supported in part by the Robert Bosch GmbH and in part by the European Union's Horizon 2020 research and innovation program under Grant 101017274 (DARKO). *(Corresponding author: Niels van Duijkeren.)*

Alexander Berndt is with the Overstory B.V., 1018 VN Amsterdam, The Netherlands (e-mail: berndtae@gmail.com).

Niels van Duijkeren, Luigi Palmieri, and Alexander Kleiner are with the Robert Bosch GmbH, Corporate Research, 71272 Renningen, Germany (e-mail: Niels.vanDuijkeren@de.bosch.com; Luigi.Palmieri@de.bosch.com; Alexander.Kleiner@de.bosch.com).

Tamás Keviczky is with the Delft Center for Systems and Control (DCSC), TU Delft, 2628 CN Delft, The Netherlands (e-mail: T.Keviczky@tudelft.nl).

Digital Object Identifier 10.1109/TRO.2023.3344051

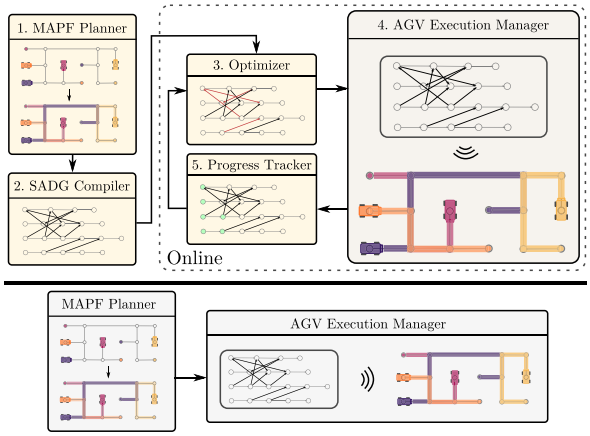

Fig. 1. Top: Our proposed optimization-based feedback control scheme. Bottom: Typical MAPF plan execution schemes. Our approach significantly reduces the cumulative route completion of AGVs subjected to large delays by optimizing the ordering of AGVs based on their progress in a receding horizon fashion, while maintaining collision- and deadlock-free plan execution guarantees.

1) The switchable action dependency graph (SADG), a novel data structure, which formalizes the definition of switchable dependencies between AGVs in multi-agent plans.
2) An online optimization-based shrinking horizon control (SHC) scheme, which re-orders AGVs based on the AGVs' current progress along their paths while maintaining collision- and deadlock-freeness guarantees.
3) Extension of the SHC to a receding horizon control (RHC) scheme, which significantly reduces computation times and thereby enables real-time applications for all of the presented maps and team sizes without sacrificing collision- and deadlock-freeness guarantees.

Our approach is illustrated in Fig. 1. We compare our approach to the baseline ADG method presented in [4] as well as the state-of-the-art robust MAPF solver K-CBSH-RM [5], yielding up to a 25% overall decrease in average route completion times as robots are confronted with large delays. Our method is available online as an open-source software package called *sadg-controller*.[1]

Outline: The rest of this article is organized as follows. Section II presents existing solutions and their capabilities and shortcomings in the context of our proposed solution. Preliminaries regarding the routing of multiple AGVs, as well as the problem formulation, is presented in Section III. We present the concept of the switchable action dependency graph (SADG) in Section IV and formulate the mixed-integer OCP in Section V. The method is extended to a receding horizon feedback control scheme in Section VI. We evaluate our approach in Section VII. Finally, Section VIII concludes this article.

II. Related Work

Recently, solving the MAPF problem has garnered widespread attention [2], [13]. This is mostly due to the abundance of application domains, such as intralogistics, airport taxi scheduling [14], and computer games [15]. Solutions to the MAPF problem include conflict-based search (CBS) [16], prioritized planning [17], declarative optimization approaches using answer set programming [18], heuristic-guided coordination [19], and graph-flow optimization approaches [20].

Algorithms, such as CBS, have been improved by exploiting properties, such as geometric symmetry [21], or using purpose-built heuristics [22]. Lam et al. [23] reformulated the multi-agent path finding (MAPF) as a mixed-integer linear program (MILP) and solve it using a branch-cut-and-price approach. MILP formulations have also been used in numerous binary-decision-based RHC problems, referred to as hybrid control systems [24]. Practical applications using these formulations include coordinating agents in urban road networks [25], coordinating autonomous cars at intersections [26], [27], UAV trajectory planning [28], multi-agent persistent coverage [29], and train scheduling [30]. Similarly, the development of bounded suboptimal solvers, such as enhanced conflict-based search (ECBS) [31] have further improved planning performance for higher dimensional state spaces. In turn, continuous conflict-based search extends CBS by enabling planning on roadmap graphs with weighted edges and considering continuous time intervals to describe collision avoidance constraints, albeit with increased solution times [32].

The abstraction of the MAPF to a graph search problem requires simplifying assumptions to manage complexity. These assumptions include the use of very crude vehicle motion models and neglecting most of the effects of unpredictable delays in stochastic and dynamic environments. In order to maintain validity of the MAPF plan during execution, it is required to synchronize the progress of all AGVs by closely monitoring the fleet. This synchronization can be achieved using a so-called execution policy to manage the AGVs according to their individual plans.

An ADG encodes the ordering between AGVs as well as their kinematic constraints in a post-processing step after solving the MAPF [6]. Combined with a plan execution policy, this allows AGVs to execute MAPF plans successfully despite kinematic constraints and unforeseen delays. Closely related to the ADG-based execution policy to account for disturbances is RMTRACK [10]. For every pair of robots, RMTRACK identifies collision regions (i.e., relative delays that lead to collisions) in their coordination space. It is imposed that the trajectory in the coordination space remains homotopic to the undisturbed trajectory. This leads to an equivalent coordination approach as [6] and guarantees deadlock-freeness. Follow-up work [33] proposes to relax the homotopy equivalence condition and allow flipping the order, in which robots pass through a certain region considering two different optimization strategies. The latter approach was later extended to guarantee deadlock-free plan execution [11] by asserting that the so-called "segment graph" that results from flipping the order has no cycles. Note that the "segment graph" is closely related to the ADG [6]. Moreover, the concept that flipping the order of two robots results in a different "segment graph" in turn relates to the SADG presented in [12] and this article. Contrary to these works, Coskun et al. [11] did not consider changing the order of robots at every possible conflict

[1][Online]. Available: https://github.com/alexberndt/sadg-controller

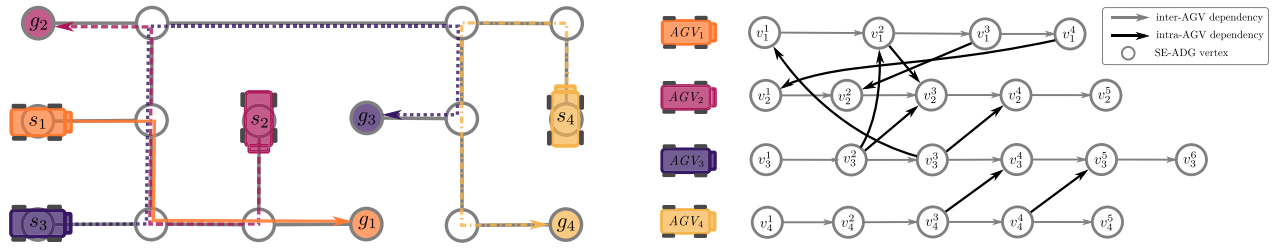

Fig. 2. Running example: Roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ occupied by $N = 4$ AGVs overlayed with the MAPF plan $\mathcal{P}$ indicated by the colored routes. AGV ordering is indicated by the z-height relative to other routes, i.e., AGV_3 before AGV_1 before AGV_2. This ordering is more explicitly indicated by the SE-ADG, $\mathcal{G}_{\text{SE-ADG}} = (\mathcal{V}_{\text{SE-ADG}}, \mathcal{E}_{\text{SE-ADG}})$, constructed from $\mathcal{P}$ using Algorithm 1. (a) Roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{P}$. (b) SE-ADG $\mathcal{G}_{\text{SE-ADG}} = (\mathcal{V}_{\text{SE-ADG}}, \mathcal{E}_{\text{SE-ADG}})$.

simultaneously. They thereby avoid the need to solve a costly MILP, but sacrifice optimality of the overall plan execution. Since the MAPF considers a fleet of AGVs each with a unique start and goal position, an additional framework is required to allow for the persistent planning of AGVs. Such a framework is proposed in [4], where the aforementioned ADG can be used to anticipate where AGVs will be in a future time-step (called a *commit*), allowing the MAPF to be solved from there, while the AGVs execute the plans up until this *commit*.

Several MAPF methods have been introduced to particularly handle delays. kR-MAPF solvers such as K-CBSH-RM address this by permitting delays up to a duration of k time-steps [5], [34]. Stochastic AGV delay distributions are considered in [7], where the MAPF is solved by minimizing the expected overall delay. These robust MAPF formulations and solutions inevitably result in more conservative plans compared with their nominal counterparts. A robust approach to handle communication delays and packet losses for AGVs with second-order dynamics is considered in [35]. However, all these solutions do not specifically address the effects of significantly large delays. These approaches typically view delays as a bounded lack of synchronization between AGVs, rather than as significantly impacting the route completion time.

The contribution of this article is a method that extends the concept of an ADG by modeling the allowed re-orderings of AGVs at intersections, obtaining an SADG. The routes for each AGV are considered given and remain unaltered. Typically they would be computed using an existing MAPF solver. The result is a comparatively low-dimensional decision-making problem, to continuously and reactively modify the MAPF plan online to improve the cumulative route completion time. We formulate the problem as an MILP that can be solved using off-the-shelf—commercial as well as open-source—solvers. By our re-ordering approach we allow AGVs to continue with their tasks without needing to unnecessarily wait for delayed AGVs, while guaranteeing deadlock- and collision-free execution.

III. Coordinating Multiple AGVs

In this section, we introduce the concepts of a valid MAPF plan as well as a formal introduction of the spatially exclusive action dependency graph (SE-ADG), a concept derived from the ADG originally proposed in [4]. The SE-ADG and properties introduced will form the foundation of the methods introduced in subsequent sections.

A. Valid MAPF Plans

Consider a workspace represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which is occupied by a fleet of N AGVs, e.g., as in Fig. 2(a). Each AGV has a unique start and goal position $s_i \in \mathcal{V}, s_i \neq s_j$ if $i \neq j \; \forall \, i, j \in \{1, \ldots, N\}$ and $g_i \in \mathcal{V}, g_i \neq g_j$ if $i \neq j \; \forall \, i, j \in \{1, \ldots, N\}$, respectively. The task is for AGV_i to navigate from s_i to g_i without collisions $\forall \; i \; \in \{1, \ldots, N\}$. A solution to this task is called a MAPF solution, which we represent as a set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$, where $\mathcal{P}_i = \{p_i^1, \ldots, p_i^{N_i}\}$ is a sequence of N_i *plan tuples* representing the actions AGV_i must take to navigate from s_i to g_i. A *plan tuple* $p_i^k = (\hat{v}(p_i^k), \hat{t}(p_i^k))$ where the operators $\hat{v}(p_i) : \mathcal{P}_i \to \mathcal{V}$, and $\hat{t}(p_i) : \mathcal{P}_i \to \mathbb{N}_0$ return the roadmap vertex and *planned* time when AGV_i must be at vertex $\hat{v}(p_i)$, respectively. Note that we consider the *planned* time in a discrete fashion, as used in almost all MAPF formulations and algorithms [2]. Definition 1 lists the conditions for a valid MAPF solution. Essentially, for a MAPF solution to be valid, all AGVs must reach their goals in finite time, and there can be no collisions between the AGVs along their planned routes.

Definition 1 (Valid MAPF solution): A valid MAPF solution is a set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ such that the vertices $\hat{v}(p_i^k) \neq \hat{v}(p_j^l)$ if $\hat{t}(p_i^k) = \hat{t}(p_j^l), \;\; k \in \{1, \ldots, N_i\}, \;\; l \in \{1, \ldots, N_j\} \; \forall \, i, j \in \{1, \ldots, N\}$. In addition, AGV_i and AGV_j if $i \neq j$ must never traverse an edge $e \in \mathcal{E}$ in opposite directions in the same time-step. Finally, $\hat{v}(p_i^1) = s_i$ and $\hat{v}(p_i^{N_i}) = g_i \; \forall \, i, j \in \{1, \ldots, N\}$.

B. Spatially Exclusive Action Dependency Graph

Although a valid MAPF solution guarantees that all AGVs reach their respective goals in finite time without collision, the underlying assumption is that all AGVs execute the plan without time-delays. To relax this assumption, we introduce the SE-ADG, a graph-based data-structure used to define the ordering of AGVs as they navigate $\mathcal{G}$. The idea is that a valid MAPF solution can be used to generate an SE-ADG, which can be used to execute the MAPF plans while maintaining collision-avoidance and route-completion guarantees.

Note that the SE-ADG we present here is directly borrowed from the original ADG presented in [4], but with the additional property that each vertex must involve a movement from two locations, which are spatially exclusive of one another.

Definition 2 (Spatially Exclusive Action Dependency Graph): An SE-ADG is a directed graph $\mathcal{G}_{\text{SE-ADG}} = (\mathcal{V}_{\text{SE-ADG}}, \mathcal{E}_{\text{SE-ADG}})$. $\mathcal{V}_{\text{SE-ADG}}$ is a set of vertices $v = (\{p_1, \ldots, p_q\}, \mathit{status})$, which define the movement of AGV_i from $\hat{v}(p_1)$, via intermediate locations, to $\hat{v}(p_q)$. The variable $\mathit{status} \in \{\mathit{staged}, \mathit{in\text{-}progress}, \mathit{completed}\}$ indicates the current status of each movement. $\mathcal{E}_{\text{SE-ADG}}$ is a set of directed edges $e = (v, v')$ with $v, v' \in \mathcal{V}_{\text{SE-ADG}}$.

The SE-ADG represents the implicit ordering of the vertex visitation by each AGV using a valid MAPF plan $\mathcal{P}$ and can be constructed using Algorithm 1. Aside from the usage of different data structures, Algorithm 1 is practically identical to the ADG algorithm in [4], except for lines 7–13. $loc(p_i) : \mathcal{P}_i \to \mathbb{R}^2$ returns location of plan tuple p, $S_{\text{AGV}} \subset \mathbb{R}^2$ represents the area occupied by an AGV, $\oplus$ is the Minkowski-sum. Finally, $s(v) : \mathcal{V}_{\text{SE-ADG}} \to \mathcal{V}$ returns the roadmap vertex associated with the *first* plan tuple in v; $g(v) : \mathcal{V}_{\text{SE-ADG}} \to \mathcal{V}$ the *last* plan tuple in v. $\hat{t}_g(v) : \mathcal{V}_{\text{SE-ADG}} \to \mathbb{N}_0$ returns $\hat{t}(p_q)$ where p_q is the last plan tuple in v. Once again, we use the hat to indicate that we are dealing with *planned* times. The subscript g in $\hat{t}_g(\cdot)$ is short for *goal*. Going forward, we use the following notation to refer to AGVs and SE-ADG vertices: i and j are both AGV indices such that $i, j \in \{1, \ldots, N\}$. Furthermore, k and l are the SE-ADG vertex index of AGV i and j, respectively, i.e., $k \in \{1, \ldots, N_i\}$ and $l \in \{1, \ldots, N_j\}$.

Algorithm 1 takes a valid MAPF solution as input, and uses a two-stage approach to convert this into an SE-ADG. In the first stage (cf., lines 1–13), each AGV's plan is considered individually. Spatial exclusivity of each vertex is guaranteed in line 7. The interaction between AGVs is considered in the second stage (cf., lines 14–20). Here, dependencies are generated between AGVs if their planned routes cover the same location at any point in time.

Initially, the *status* of v_i^k is *staged* $\forall\, i, k$. The directed edges $e \in \mathcal{E}_{\text{SE-ADG}}$, from here on referred to as dependencies, define event-based constraints between two vertices. Specifically, (v_i^k, v_j^l) implies that v_j^l cannot be *in-progress* or *completed* until $v_i^k = \mathit{completed}$. A dependency $(v_i^k, v_j^l) \in \mathcal{E}_{\text{SE-ADG}}$ is classified as *intra-AGV* if $i = j$ and *inter-AGV* if $i \neq j$. If an SE-ADG execution policy is used to execute a valid MAPF solution, an important property to ensure finite-time task completion times for all AGVs is that the SE-ADG must be acyclic. In this context, finite-time task completion for all AGVs is the same as deadlock-free plan execution.

The key difference between Algorithm 1 and the ADG algorithm in [4] is the fact that subsequent vertices are spatially exclusive cf., lines 7–14. The plans referenced in [4] contain actions, such as *in-place rotations*. Despite an AGV not changing location by performing such an action, the ADG will have two inter-AGV edges. Using spatial exclusivity, as in the SE-ADG, an *in-place rotation* is merged with a spatially transitional action in one vertex, resulting in one inter-AGV edge to express this inter-AGV dependency. Not only does this mean that the SE-ADG has fewer vertices and edges, it will also be an important component for the definition of a *switched dependency* presented in the next section.

Algorithm 1: Compiling a SE-ADG. Notable functional differences to the ADG Algorithm in [4] in lines 7–13.

Input: $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ // valid MAPF solution
Result: $\mathcal{G}_{\text{SE-ADG}}$
// Add sequence of events for each AGV
1: **for** $\mathcal{P}_i = \{p_i^1, \ldots, p_i^{N_i}\}$ **in** $\mathcal{P}$ **do**
2: $p \leftarrow p_i^1$
3: $v \leftarrow (\{p\}, \mathit{staged})$
4: $v_{\text{prev}} \leftarrow$ None
5: **for** $k = 2$ to N_i **do**
6: Append p_i^k to sequence of plan tuples of v
// Check for spatial exclusivity
7: **if** $loc(p) \oplus S_{\text{AGV}} \cap loc(p_i^k) \oplus S_{\text{AGV}} = \emptyset$ **then**
8: Add v to $\mathcal{V}_{\text{SE-ADG}}$
9: **if** v_{prev} not *None* **then**
10: Add edge $e = (v_{\text{prev}}, v)$ to $\mathcal{E}_{\text{SE-ADG}}$
11: **end if**
12: $v_{\text{prev}} \leftarrow v$
13: $p \leftarrow p_i^k$
14: $v \leftarrow (\{p\}, \mathit{staged})$
15: **end if**
16: **end for**
17: **end for** // Add inter-AGV ordering constraints
18: **for** $i = 1$ to N **do**
19: **for** $j = 1$ to $N, i \neq j$ **do**
20: **for** $k = 1$ to N_i **do**
21: **for** $l = 1$ to N_j **do**
22: **if** $s(v_i^k) = g(v_j^l)$ **and** $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$ **then**
23: Add edge $e = (v_i^k, v_j^l)$ to $\mathcal{E}_{\text{SE-ADG}}$
24: **end if**
25: **end for**
26: **end for**
27: **end for**
28: **end for**
29: **return** $\mathcal{G}_{\text{SE-ADG}} = (\mathcal{V}_{\text{SE-ADG}}, \mathcal{E}_{\text{SE-ADG}})$

C. SE-ADG as a Plan Execution Policy

In the following we detail how an SE-ADG can be used as an execution policy to coordinate the AGVs and to accommodate for possible delays. From Definition 2, we recall that the operator $\text{status}(v) : \mathcal{V}_{\text{SE-ADG}} \to \mathit{status}$ returns the status of a vertex v. An AGV is said to be *executing* an SE-ADG vertex v if it is performing the actions defined by the plan tuple sequence $\{p_1, \ldots, p_q\}$ of v.

Definition 3 (SE-ADG plan execution policy): Consider a valid MAPF plan $\mathcal{P}$ and the corresponding SE-ADG, $\mathcal{G}_{\text{SE-ADG}}$, constructed using Algorithm 1. The SE-ADG-execution policy is defined as follows.

1) Initially, status$(v) = staged\ \forall v \in \mathcal{V}_{\text{SE-ADG}}$.
2) Each AGV$_i$'s first vertex is $v_i^1\ \forall\ i\ \in\ \{1, \ldots, N\}$.
3) AGV$_i$ can only start *executing* v_i^k if $status(v) = completed\ \forall\ e = (v, v_i^k) \in \mathcal{E}_{\text{SE-ADG}}$.

The SE-ADG vertex statuses are updated by this policy as follows.

1) $status(v_i^k)$ changes from *staged* to *in-progress* if AGV$_i$ is busy executing v_i^k.
2) $status(v_i^k)$ changes from *in-progress* to *completed* if AGV$_i$ has finished executing v_i^k.

D. Properties of the SE-ADG

With the SE-ADG execution policy from Definition 3, we now show that if the SE-ADG is acyclic, we can guarantee that AGVs can execute their plans in finite time in a collision-free manner. In this context, we anticipate that guaranteeing finite-time plan completion is equivalent to ensuring the plan execution is persistently deadlock-free. First, we make the following assumption.

Assumption 1: A single AGV can navigate the workspace (represented by roadmap $\mathcal{G}$) occupied by static and dynamic obstacles in a collision-free manner using on-board navigation methods.

In the context of multiple AGVs and MAPF, Assumption 1 is relatively nonconstraining. It requires AGVs to be able to follow the roadmap $\mathcal{G}$ and navigate around or wait for static and dynamic third-party obstacles, which might partially or temporarily block its path, respectively. This single AGV navigation problem has already been addressed in numerous works [36], [37], [38], [39].

If Assumption 1 is satisfied, guaranteeing collision-free task execution in the multi-AGV case requires us to ensure that AGVs do not collide with each other. Results in resource allocation of concurrent system analysis, such as [40], show that if a dependency graph is *acyclic*, plan execution is guaranteed to be deadlock-free. In this context, if the SE-ADG is constructed from a valid MAPF plan, we know that following the execution policy in Definition 3 will ensure each AGV will complete its task in a collision- and deadlock-free manner.

Corollary 1 (SE-ADG guarantees collision-free plan execution): Consider a valid MAPF plan abstracted to an SE-ADG using Algorithm 1. AGVs are guaranteed to execute the SE-ADG plans in a collision-free manner if all AGVs adhere to the execution policy in Definition 3.

Proof: Consider the nominal execution of a MAPF plan: from Definition 1, $\hat{v}(p_i^k) \neq \hat{v}(p_j^l)$ if $\hat{t}(p_i^k) = \hat{t}(p_j^l)$, implying that no two AGVs will occupy the same location at the same time. In addition, by Definition 1, each i'th AGV reaches its goal $\hat{v}_g(p_i^{N_i})\ \forall\ i \in \{1, \ldots, N\}$. Next, consider Algorithm 1 lines 22 and 23, which ensure that $\forall i, j \in 1, \ldots, N$ and $k \in 1, \ldots, N_i$, $l \in 1, \ldots, N_j$ where $s(v_i^k) = g(v_j^l)$ and $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$, a dependency $e = (v_i^k, v_j^l)$ is added to $\mathcal{E}_{\text{SE-ADG}}$. By the execution policy Definition 3, item 3, each AGV will only move from a vertex $s(v)$ to $g(v)$ if all edges pointing to v have status completed. Because the AGVs can only collide by visiting the same location at the same time, and each instance of an AGV occupying the same location as another yields an edge in the SE-ADG, no AGV will occupy the same location at the same time, implying zero collitions during plan execution.

Corollary 2 (Acyclic SE-ADG is sufficient to guarantee deadlock- and collision-free plan execution): Consider an SE-ADG, $\mathcal{G}_{\text{SE-ADG}}$, constructed from a valid MAPF plan $\mathcal{P}$ using Algorithm 1. If $\mathcal{G}_{\text{SE-ADG}}$ is acyclic and Assumption 1 holds, each AGV$_i$ will reach g_{N_i} in finite time without collisions for all $i \in \{1, \ldots, N\}$.

Proof: Individually, the completion time of each vertex $v \in \mathcal{V}_{\text{SE-ADG}}$ is finite by Assumption 1. If $\mathcal{G}_{\text{SE-ADG}}$ is acyclic, it has a topological ordering, implying that at each point, at least one SE-ADG vertex can be executed, until all vertices are *completed*. This proves deadlock-free execution. Collision-free movement is proven in Corollary 1

Note that we are not able to extend Corollary 2 to be a *necessary* condition for deadlock-free plan execution, since methods guaranteeing deadlock-free plan execution exist which do not use an SE-ADG approach. Nevertheless, this *sufficient* condition is still very useful, as will be shown in subsequent sections.

Finally, we address the fact that not all valid MAPF solutions yield an acyclic SE-ADG. A cyclic SE-ADG comes from a plan, which essentially requires perfect synchronization among AGVs. However, with little limitation to practical cases (see Remark 1), we assume that MAPF solutions will yield acyclic SE-ADGs.

Assumption 2: MAPF problems are such that they initially yield an acyclic SE-ADG.

Remark 1: Acyclicity of the SE-ADG can be ensured when the roadmap vertices outnumber the AGV fleet size i.e.,$|\mathcal{V}| > N$ (as is typically the case in warehouse robotics) or a MAPF solver, such as kR-MAPF, is used with $k = 1$[5]. Alternatively, simple modification (e.g., an extra edge constraint in CBS) to existing MAPF solvers is sufficient to ensure acyclicity [4].

IV. Reordering AGV Plans: Introducing the SADG

In Section III, we introduced the SE-ADG and showed that an acyclic SE-ADG is a sufficient condition to guarantee deadlock- and collision-free plan execution for multiple AGVs executing a valid MAPF plan. In this section, we address the core challenge we are tackling throughout this manuscript: how can we adjust plans online to account for delays while maintaining deadlock- and collision-free plan execution guarantees? To this end, we introduce a new data-structure, the SADG, which facilitates the systematic re-ordering of AGVs based on time-delays. We go on to show that the SADG provides the ability to maintain deadlock- and collision-free guarantees of the original SE-ADG on which it is based.

A. Switched Dependencies

Before introducing the SADG, we need to introduce the fundamental building block on which it is based: the *switched dependency*. Consider an inter-AGV (i.e., $i \neq j$) dependency $e_{\text{original}} = (v_j^l, v_i^k) \in \mathcal{E}_{\text{SE-ADG}}$. From here on, we refer to an *inter-AGV dependency* simply as a *dependency*. As per Definition 3, e_{original} implies status$(v_j^l) = completed$ before status$(v_i^k) = in - progress$ when executing the SE-ADG-based plans. In

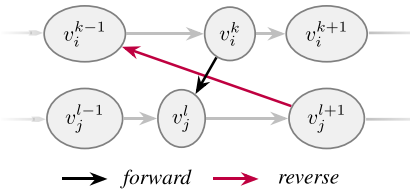

Fig. 3. Visual illustration of a switched dependency: Original dependency (black) and its reversed counterpart (red).

terms of the roadmap $\mathcal{G}$, this is equivalent to requiring AGV_j to leave $s(v_j^l)$ (reach $g(v_j^l)$) before AGV_i can advance to $g(v_i^k)$. Note the implicit ordering that AGV_j must go to $s(v_j^l) = g(v_i^k)$ before AGV_i. The idea of a switched dependency is to reverse this implicit ordering while ensuring AGVs do not occupy the same vertex in $\mathcal{G}$ at the same time.

Definition 4 (Switched inter-AGV dependency): Given a dependency $e_{\text{fwd}} = (v_i^k, v_j^l) \in \mathcal{E}_{\text{SE-ADG}}$, a switched dependency is an edge $e_{\text{rev}} = (v, v')$, which ensures AGV_i reaches $s(v_j^l) = g(v_i^k)$ before AGV_j without collision.

Given a dependency e_{fwd}, a switched dependency which fulfills Definition 4 can be determined using Lemma 1. A dependency and its reversed counterpart are illustrated in Fig. 3.

Lemma 1 (Switched inter-AGV dependency): Let $v_i^k, v_j^l, v_j^{l+1}, v_i^{k-1} \in \mathcal{V}_{\text{SE-ADG}}$, where $\mathcal{G}_{\text{SE-ADG}} = (\mathcal{V}_{\text{SE-ADG}}, \mathcal{E}_{\text{SE-ADG}})$ from Definition 2. Then, $d' = (v_j^{l+1}, v_i^{k-1})$ is the switched counterpart of $d = (v_i^k, v_j^l)$ which adheres to Definition 4.

Proof: The dependency $d = (v_i^k, v_j^l)$ encodes the constraint $t_s(v_j^l) \geq t_g(v_i^k)$. The switched counterpart of d is denoted as $d' = (v_j^{l+1}, v_i^{k-1})$. d' encodes the constraint $t_s(v_i^{k-1}) \geq t_g(v_j^{l+1})$. By definition, $t_s(v_i^k) \geq t_g(v_i^{k-1})$ and $t_s(v_j^{l+1}) \geq t_g(v_j^l)$. Since $t_g(v) \geq t_s(v)$, this implies that d' encodes the constraint $t_s(v_i^k) \geq t_g(v_j^l)$, satisfying Definition 4.

Note that, as discussed in Section III, the SE-ADG is spatially exclusive, allowing us to use only a single dependency as the switched counterpart of one dependency.

B. Switchable Action Dependency Graph

Lemma 1 provides us with a method to maintain collision avoidance while re-ordering AGVs. We are now in the position to extend the SE-ADG to enable the re-ordering of AGVs using *switchable dependencies*. To this end, we introduce the SADG, a mapping from a binary vector, $\mathbf{b}$, to an SE-ADG, $\mathcal{G}_{\text{SE-ADG}}$.

Definition 5 (Switchable Action Dependency Graph): An SADG is a mapping $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}) : \{0,1\}^{m_T} \rightarrow \mathbb{G}_{\text{SE-ADG}}$, which outputs the resultant SE-ADG based on the dependency selection represented by $\boldsymbol{b} = \{b_1, \ldots, b_{m_T}\}$, where $b_m = 0$ and $b_m = 1$ imply selecting the forward and reverse dependency of pair m, respectively, $m \in \{1, \ldots, m_T\}$.

In Definition 5, $\mathbb{G}_{\text{SE-ADG}}$ refers to the set of all possible $\mathcal{G}_{\text{SE-ADG}}$'s. Depending on the value of $\boldsymbol{b}$, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ will result in a different $\mathcal{G}_{\text{SE-ADG}}$. Similarly to the SE-ADG, an SADG can be constructed from a valid MAPF solution $\mathcal{P}$ using

Algorithm 2: Compiling an SADG.

Input: $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ // valid MAPF solution
Result: $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$
// Add sequence of events for each AGV
1: **for** $\mathcal{P}_i = \{p_i^1, \ldots, p_i^{N_i}\}$ **in** $\mathcal{P}$ **do**
2: $p \leftarrow p_i^1$
3: $v \leftarrow (\{p\}, staged)$
4: $v_{\text{prev}} \leftarrow None$
5: **for** $k = 2$ to N_i **do**
6: Append p_i^k to sequence of plan tuples of v
// Check for spatial exclusivity
7: **if** $loc(p) \oplus S_{\text{AGV}} \cap loc(p_i^k) \oplus S_{\text{AGV}} = \emptyset$ **then**
8: Add v to $\mathcal{V}_{\text{SADG}}(\boldsymbol{b})$
9: **if** v_{prev} not *None* **then**
10: Add (v_{prev}, v) to $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$
11: **end if**
12: $v_{\text{prev}} \leftarrow v$
13: $p \leftarrow p_i^k$
14: $v \leftarrow (\{p\}, staged)$
15: **end if**
16: **end for**
17: **end for** // Add switchable dependency pairs
18: **for** $i = 1$ to N **do**
19: **for** $j = 1$ to $N, j \neq i$ **do**
20: **for** $k = 1$ to N_i **do**
21: **for** $l = 1$ to N_j **do**
22: **if** $s(v_i^k) = g(v_j^l)$ **and** $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$ **then**
23: $e_{\text{fwd}} \leftarrow (v_i^k, v_j^l)$
24: **if** $(v_i^{k-1}, v_j^{l+1}) \in \mathcal{V}_{\text{SADG}}(\boldsymbol{b})$ **then**
25: $e_{\text{rev}} \leftarrow (v_j^{l+1}, v_i^{k-1})$
26: Add $depSwitch(b_m : \{e_{\text{fwd}}, e_{\text{rev}}\})$ to $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$
27: **else**
28: Add e_{fwd} to $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$
29: **end if**
30: **end if**
31: **end for**
32: **end for**
33: **end for**
34: **end for**
35: **return** $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}) = (\mathcal{V}_{\text{SADG}}(\boldsymbol{b}), \mathcal{E}_{\text{SADG}}(\boldsymbol{b}))$

Algorithm 2. Like Algorithm 1, Algorithm 2 also consists of two-stages. In fact, the only difference between Algorithm 2 and Algorithm 1 is in lines 19–25. Instead of just creating the dependency e_{fwd}, a check is done to validate if its reverse counterpart, e_{rev}, can be constructed (cf., line 24). If so, a binary switching function $depSwitch : \{0,1\} \rightarrow \mathcal{E}_{\text{SE-ADG}}$ is appended to the set of dependencies $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$ (cf., line 26), where, as per Definition 5, $depSwitch(b_m = 0 : \{e_{\text{fwd}}, e_{\text{rev}}\}) = e_{\text{fwd}}$, and $depSwitch(b_m = 1 : \{e_{\text{fwd}}, e_{\text{rev}}\}) = e_{\text{rev}}$. Conversely, if the reverse counterpart of e_{fwd} cannot be constructed, dependency switching is not possible for e_{fwd} and there is no need for a binary variable b_m. In this case, the dependency e_{fwd} is appended to $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$ (cf., line 28).

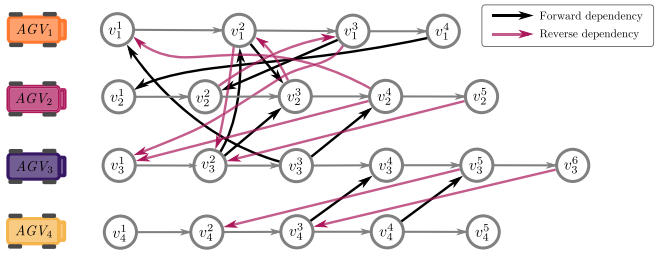

Fig. 4. SADG $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ constructed from the same valid MAPF solution used to construct the SE-ADG in Fig. 2(b).

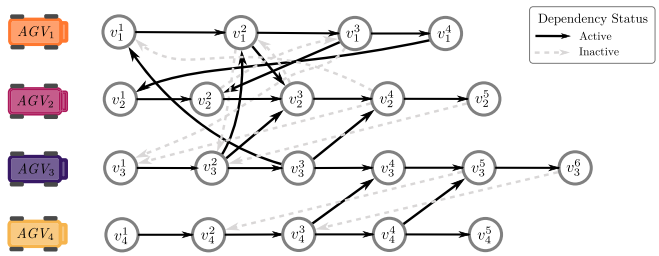

Fig. 5. Illustration of $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ for $\boldsymbol{b} = \mathbf{0}$. The resultant $\mathcal{G}_{\text{SE-ADG}}$ is depicted with solid black arrows. The dependencies not part of $\mathcal{G}_{\text{SADG}}(\mathbf{0})$ are indicated by dotted lines.

Fig. 4 shows an illustration of the SADG for the running example illustrated in Fig. 2. Note that all the forward dependencies (solid, black arrows), are identical to those in Fig. 2(b). This is because Algorithm 2 follows the same process to generate the forward dependencies as Algorithm 1. Note also how the forward dependency $e_{\text{fwd}} = (v_1^3, v_3^1)$ has no reverse counterpart. This is because its reverse counterpart, as specified by Lemma 1, would be $e_{\text{rev}} = (v_1^4, v_3^0)$, which does not exist (as checked in Algorithm 2, line 24).

C. SADG Properties and Execution Policy

With the aim of introducing an SADG-based control scheme in Section V, we need to ensure the SE-ADG execution policy in Definition 3 can be applied to SADGs. To this end, recall that $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ yields an SE-ADG for a particular $\boldsymbol{b}$. When all AGVs are at their starting positions and $\boldsymbol{b} = \mathbf{0}$, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}) = \mathcal{G}_{\text{SE-ADG}}$, the same SE-ADG as obtained with Algorithm 1, which was shown to ensure collision-free and deadlock-free plans in Corollary 2.

Although we have proven collision- and deadlock-avoidance if AGVs follow the execution policy in Definition 3, this only holds if the underlying SE-ADG remains constant. We will now show that if $\boldsymbol{b}$ is chosen such that the all edges in $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$ in $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ do not violate the assumptions in the execution policy of Definition 3, Corollary 1 can be extended to SADGs.

Corollary 3 (Persistent collision-free plan execution for SADGs): For varying $\boldsymbol{b}$, the resultant SE-ADG from $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ will guarantee collision-free plan execution as long as

1) $\boldsymbol{b}$ is chosen such that $\forall\, e = (v_i^k, v_j^l) \in \mathcal{E}_{\text{SADG}}(\boldsymbol{b})$ the head of e, v_j^l, has $status(v_j^l) = staged$ at the time $\boldsymbol{b}$ is changed and
2) AGVs follow the plan execution policy Definition 3 based on the changing $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$ at all times.

Proof: Proof by induction. Initially, at time $t = T_0$, $\boldsymbol{b} = \mathbf{0}$ and $\mathcal{G}_{\text{SE-ADG}}$ is acyclic which guarantees collision-free plan execution by Corollary 1. Consider at time $t = T_1 > T_0$, $\boldsymbol{b}$. Let $\boldsymbol{b}_{\text{switched}} \subset \boldsymbol{b}$ refer to the subset of $\boldsymbol{b}$ that is different between $t \leq T_1$ and $t > T_1$. Since $\forall\, b \in \boldsymbol{b}_{\text{switched}}$, $status(v_j^l) = staged$, $status(v_j^{l'})\ \forall l' \geq l$, by the construction of intra-AGV dependencies in Algorithm 1, cf., lines 1–13. Hence, at time $t > T_1$, the constraints imposed by all the newly active edges introduced by $\boldsymbol{b}_{\text{switched}}$ have not been violated. Since all constraints are adhered, collision-avoidance is guaranteed. The same logic applies for a subsequent switching at time $t = T_2 > T_1$, thus proving persistent collision-avoidance.

Note that although Corollary 3 provides us constraints on the changing of $\boldsymbol{b}$ at any time to ensure persistent collision-avoidance guarantees, no guarantees are made regarding deadlocks. Hence, it is possible that, following Corollary 3, $\boldsymbol{b}$ is chosen which causes the AGVs to enter into a deadlock. Viewed from the perspective of the SE-ADG: it is possible that a value of $\boldsymbol{b}$ causes a cycle in the resultant SE-ADG. Finding a value of $\boldsymbol{b}$ which also guarantees deadlock-free plans, i.e., persistent acyclicity of $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$, will be considered in Section V.

For our running example, in the case that $\boldsymbol{b} = \mathbf{0}$, the active SE-ADG is shown in Fig. 5. Note that Fig. 5 is practically identical to Fig. 2(b) because $\boldsymbol{b} = \mathbf{0}$ corresponds to the original SE-ADG constructed using Algorithm 1.

V. Shrinking Horizon Control (SHC) Scheme

Consider AGVs executing their respective plans, as described by an SE-ADG $\mathcal{G}^0_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\mathbf{0})$, adhering to execution policy in Definition 3. In the case that any subset of the AGVs is delayed, the cumulative route completion times of the AGVs could be reduced if the SE-ADG is modified at a time $T_1 > T_0$, where T_0 refers to the time the AGVs started executing the plans. In Section IV, we showed that if, at time T_1, $\boldsymbol{b}$ is chosen in accordance with Corollary 3, persistent collision-avoidance of the AGVs is guaranteed when following the execution policy in Definition 3. However, Corollary 3 does not guarantee plan completion (i.e., deadlock-free movement). This is because changing $\boldsymbol{b}$ at T_1 could result in a cyclic SE-ADG, causing a deadlock. Therefore, the objective is to find $\boldsymbol{b}^*$ at T_1 to ensure $\mathcal{G}^*_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*)$ is acyclic, while minimizing the route-completion time of the AGV fleet, based on the individual AGV delays at time T_1.

In this section, we show that finding $\boldsymbol{b}^*$ is equivalent to solving an optimal control problem (OCP). We go on to solve this OCP using a MILP formulation, which we integrate into a shrinking horizon feedback control scheme.

A. Optimal Control Problem

At any time during the execution of their respective plans, given an initial SE-ADG, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}_0)$, the OCP can be formulated

as follows:

$$f_{\text{OCP}}(\boldsymbol{b}) = \min_{\boldsymbol{b},\boldsymbol{t}_g,\boldsymbol{t}_s} \sum_{i=1}^{N} t_g(v_i^{N_i}) \tag{1a}$$

s.t.

$$t_g(v_i^k) > t_s(v_i^k) + \Delta T(v_i^k) \quad \forall\, v_i^k \in \Pi\left(\mathcal{V}_{\text{SADG}}(\boldsymbol{b})\right) \tag{1b}$$

$$t_s(v_i^{k+1}) > t_g(v_i^k) \quad \forall\, v_i^{k+1} \in \Gamma\left(\mathcal{V}_{\text{SADG}}(\boldsymbol{b})\right) \tag{1c}$$

$$\mathit{status}(v') = \mathit{staged} \quad \forall\, (v, v') \in \Psi(\boldsymbol{b}, \boldsymbol{b}_0) \tag{1d}$$

$$t_g(v_i^k) < t_s(v_j^l) \quad \forall\, (v_i^k, v_j^l) \in \mathcal{E}_{\text{SADG}}(\boldsymbol{b}) \text{ if } i \neq j \tag{1e}$$

where $\Psi(\boldsymbol{b}, \boldsymbol{b}_0)$ returns only the edges in $\mathcal{E}_{\text{SADG}}(\boldsymbol{b})$, which do not exist in $\mathcal{E}_{\text{SADG}}(\boldsymbol{b}_0)$

$$\Psi\left(\boldsymbol{b}, \boldsymbol{b}_0\right) = \{e \mid e \in \mathcal{E}_{\text{SADG}}(\boldsymbol{b}) \wedge e \notin \mathcal{E}_{\text{SADG}}(\boldsymbol{b}_0)\}.$$

Furthermore, $\Pi(\cdot)$ and $\Gamma(\cdot)$ are filters such that

$$\Pi\left(\mathcal{V}\right) = \{v \in \mathcal{V} \mid \mathit{status}(v) \in \{\mathit{staged}, \mathit{in\text{-}progress}\}\}$$

$$\Gamma\left(\mathcal{V}\right) = \{v \in \mathcal{V} \mid \mathit{status}(v) = \mathit{staged}\}\,.$$

Finally, $\Delta T(v_i^k)$ is the estimated time AGV_i will take to complete v_i^k, defined as

$$\Delta T(v_i^k) = \begin{cases} T_{\text{est}}(v_i^k) & \text{if } \mathit{status}(v_i^k) = \mathit{staged} \\ \mu T_{\text{est}}(v_i^k) & \text{if } \mathit{status}(v_i^k) = \mathit{in\text{-}progress} \\ 0 & \text{if } \mathit{status}(v_i^k) = \mathit{completed} \end{cases}$$

where $T_{\text{est}}(v_i^k)$ is the total estimated time it will take AGV_i to complete v_i^k, and $\mu \in [0, 1]$ is the fraction of v_i^k that still needs to be completed. Since $t_g(v_i^{N_i})$ refers to the time where AGV_i will reach its goal position, the cost function in (1a) is the cumulative route completion time of all N AGVs. Note that (1b) and (1c) enforce the route sequence of each individual AGV, whereas (1d) and (1e) enforce ordering constraints between AGVs. Moreover, (1d) ensures the heads of all *switched* dependencies point to *staged* vertices.

Corollary 4 (Cyclic SE-ADG yields constraint violation): A cycle in the SE-ADG violates constraints of OCP (1) and therefore any feasible solution of OCP (1) is acyclic.

Proof: Without loss of generality, consider the cyclic dependency chain formed by a dependency from v_i^k to v_j^l, and from v_j^l back to v_i^k. The dependencies forming this cycle translate to the following constraints, as specified in (1e):

$$v_i^k \to v_j^l \;:\; t_g(v_i^k) < t_s(v_j^l) \tag{2a}$$

$$v_j^l \to v_i^k \;:\; t_g(v_j^l) < t_s(v_i^k). \tag{2b}$$

Furthermore, by (1b), we require that

$$t_g(v_i^k) > t_s(v_i^k) + \Delta T(v_i^k) \tag{3a}$$

$$t_g(v_j^l) > t_s(v_j^l) + \Delta T(v_j^l). \tag{3b}$$

Since $\Delta T(v_*^*) \geq 0$, observe that (2) and (3) lead to the contradiction that both $t_g(v_i^k) < t_s(v_l^j)$ and $t_g(v_i^k) > t_s(v_l^j)$ must hold. Such a contradiction appears for every (possibly longer) cycle within the SE-ADG. This result directly implies that any feasible solution to OCP (1) is acyclic.

Next, we show that if the initial SE-ADG is acyclic, the OCP is feasible and in turn yields an acyclic SE-ADG.

Corollary 5 (A solution to (1) exists if $\mathcal{G}_{SADG}(\boldsymbol{b}_0)$ is acyclic): If $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}_0)$ is acyclic, the minimizer to (1), $\boldsymbol{b}^*$, exists, $f_{\text{OCP}}(\boldsymbol{b}^*)$ is finite., implying $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*)$ is acyclic.

Proof: A direct result from Corollary 2 is that if $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}_0)$ is acyclic, the route completion time of all AGVs is finite. Because the cost function of (1) equals the cumulative route completion time of all AGVs, $f_{\text{OCP}}(\boldsymbol{b}_0)$ is necessarily finite and $\boldsymbol{b}_0$ is a solution to (1). Consequently, the minimizer $\boldsymbol{b}^*$ exists and is a solution of (1). From Corollary 4, this means that $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*)$ is acyclic.

B. Formulation as MILP

Working toward the definition of a feedback control scheme, we now formulate the OCP in (1) as an MILP as follows:

$$\min_{\boldsymbol{b},\boldsymbol{t}_g,\boldsymbol{t}_s} \sum_{i=1}^{N} t_g(v_i^{N_i}) \tag{4a}$$

s.t.

$$t_g(v_i^k) > t_s(v_i^k) + \Delta T(v_i^k) \quad \forall\, v_i^k \in \Pi\left(\mathcal{V}_{\text{SADG}}(\boldsymbol{b})\right) \tag{4b}$$

$$t_s(v_i^{k+1}) > t_g(v_i^k) \quad \forall\, v_i^{k+1} \in \Gamma\left(\mathcal{V}_{\text{SADG}}(\boldsymbol{b})\right) \tag{4c}$$

$$\mathit{status}(v') = \mathit{staged} \quad \forall\, (v, v') \in \Psi(\boldsymbol{b}, \boldsymbol{b}_0) \tag{4d}$$

$$t_g(v_i^k) < t_s(v_j^l) \quad \forall\, (v_i^k, v_j^l) \in \mathcal{E}_{\text{SADG}}(\boldsymbol{b}) \text{ if } i \neq j \tag{4e}$$

$$t_s(v_j^l) > t_g(v_i^k) - bM \quad \forall\; b \in \boldsymbol{b} \tag{4f}$$

$$t_s(v_i^{k-1}) > t_g(v_j^{l+1}) - (1-b)M \quad \forall\; b \in \boldsymbol{b} \tag{4g}$$

where $M \gg 0$ is a large constant, and i and j are both indices referring to a specific AGV such that $i, j \in \{1, \ldots, N\}$. Furthermore, k and l are the SE-ADG vertex index of AGV i and j, respectively, i.e., $k \in \{1, \ldots, N_i\}$ and $l \in \{1, \ldots, N_j\}$. The constraints (4f) and (4g) encode the switching decision for each of the switchable dependencies pairs using the big-M binary decision formulation [26]. Consider a $b \in \boldsymbol{b}$, if $b = 1$, (4f) is relaxed because of the $-bM$ factor. Conversely, if $b = 0$, (4g) is relaxed, because of the $-(1-b)M$ factor.

C. Shrinking Horizon Feedback Control Scheme

Having defined the MILP in (4), we present an optimization-based shrinking horizon feedback control scheme to minimize the cumulative route completion times of the AGVs based on delays as they occur. The scheme consists of an initial planning phase followed by an online phase. During the planning phase, the roadmap and AGV start- and goal-positions are used to define a MAPF problem. The MAPF problem is solved using an algorithm, such as CBS, ECBS, and the solution $\mathcal{P}$ is used to construct an SADG using Algorithm 2. Once constructed, the execution policy in Definition 3 is used to navigate the the SADG's trivial solution, $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{0})$. As AGVs traverse the roadmap, potentially incurring delays, the MILP formulation in (4) is parameterized based on the current AGV route progress, and solved. This solution is then used to update

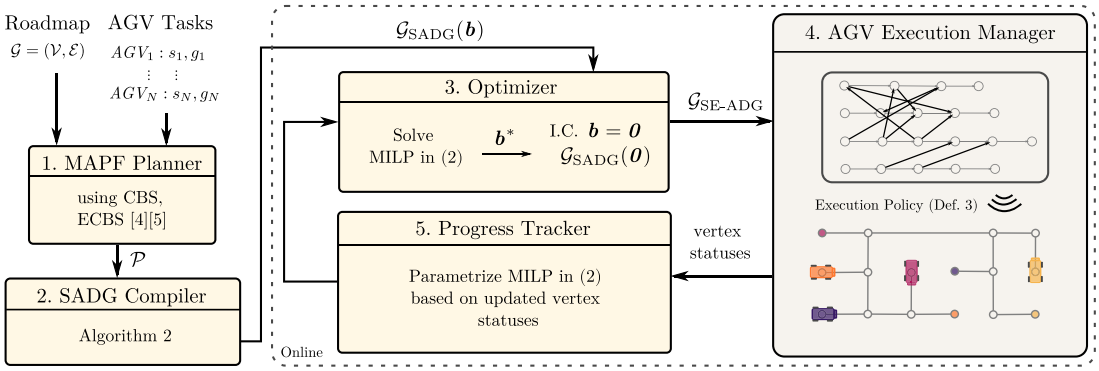

Fig. 6. Shrinking horizon feedback control diagram: The planning phase consists of *1. MAPF Planner* and *SADG Compiler*, which yields an SADG given a roadmap and AGV tasks. Initially, in *3. Optimizer*, the SADG is initialized with $\boldsymbol{b} = \boldsymbol{0}$ yielding an initial SE-ADG which can be used by AGVs to execute their plans. As AGVs progress, and are inevitably delayed, their status is tracked in *5. Progress Tracker*, and an optimization iteration is performed to re-order the AGVs in *3. Optimizer*. The solution to the MILP is used to update the active SE-ADG used for plan execution in a feedback loop. The feedback loop runs at a sampling frequency of h.

the SE-ADG used by the execution policy, until the next optimization iteration, where the MILP is reparameterized, and the SE-ADG is updated. This iterative loop repeats until all AGVs reach their respective goals. This scheme is illustrated in Fig. 6.

Having defined the feedback control scheme, we now prove that the feedback scheme is recursively feasible. This implies that if the initial planning phase is completed, the MILP will remain feasible until all AGVs have reached their respective goal positions. We use the notation $\boldsymbol{b}^*_{T_m}$ to refer to the minimizer of (4), parameterized by the AGV positions and solved at some time $t = T_m$, $m \in \mathbb{N}$.

Proposition 1 (Recursive Feasibility of SHC scheme): If the initial SE-ADG, $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{0})$, obtained from the planning phase, is acyclic, the shrinking horizon feedback control scheme will guarantee that $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*)$ is acyclic at each subsequent optimization step.

Proof: Proof by induction. Initially the SE-ADG, $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_0})$ is acyclic. If the MILP in (4) is solved at $t = T_1 > T_0$, Corollary 5 guarantees that the SE-ADG at T_1, $\mathcal{G}_{\text{SE-ADG}}{}^{T_1}$, obtained from $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_1})$, is acyclic. Subsequent optimization steps will always result in acyclic SE-ADGs, proving recursive feasibility of the feedback control scheme.

To illustrate this feedback control scheme, an example of the online optimization step is shown in Fig. 7. The example starts at time t, where $T_1 \leq t \leq T_2$, and AGV_4 has been delayed. However, because of the dependencies in $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_1})$, AGV_3 must wait for AGV_4 before it can proceed, as illustrated in Fig. 7(a). At $t = T_2$, the optimization step is performed and a new SE-ADG is obtained, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_2})$, as illustrated in Fig. 7(b). Note that $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_2})$ has switched the dependencies between AGV_3 and AGV_4, and remains acyclic. For $t > T_2$, the AGVs continue executing their plans, but using the newly obtained SE-ADG, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_2})$.

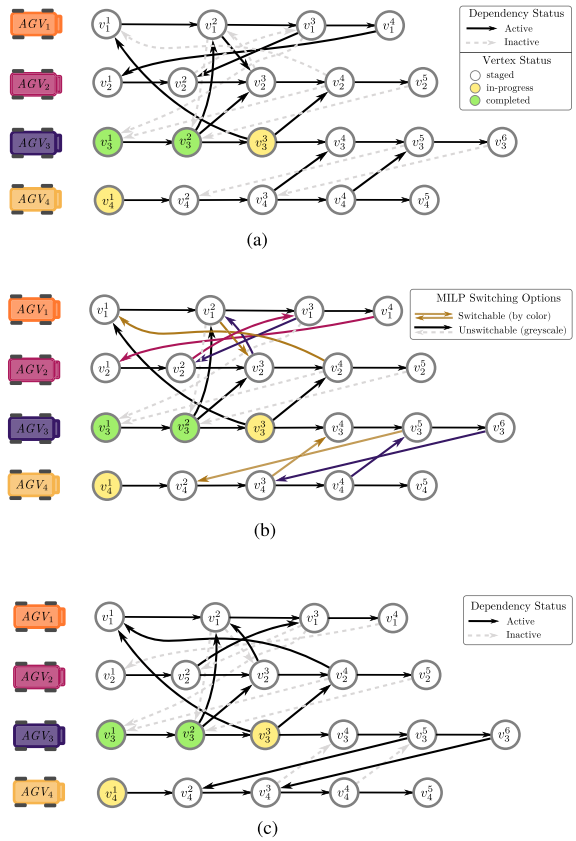

Fig. 7. Illustrative example of switching performed by the shrinking horizon feedback control scheme. (a) AGV progress illustrated by vertex statuses. AGV_4 is delayed. (b) Based on AGV progress, ordering can be adjusted by considering the valid switching pairs within the MILP formulation. (c) Switching enables AGV_3 to continue without waiting for AGV_4, and AGV_2 without waiting for AGV_1.

D. Grouping Switchable Dependency Pairs

Two patterns of switchable dependencies often occur with multi-agent planning problems. The first pattern is shown in Fig. 8(a) and occurs when a MAPF plan requires two AGVs to travel across multiple roadmap vertices in the same direction. This pattern can be found by sequentially visiting each inter-AGV dependency (v_i^k, v_j^l) in the SADG and searching for any

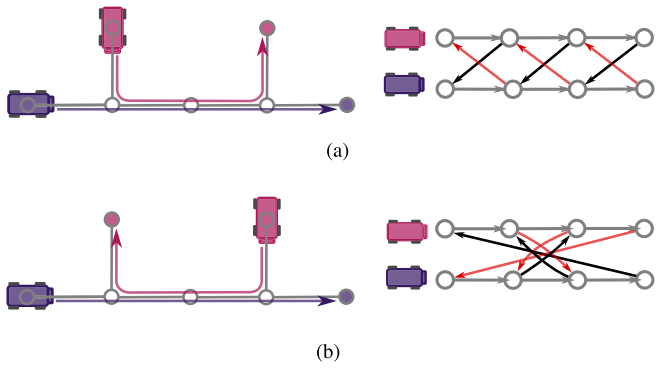

Fig. 8. Dependency grouping patterns for AGVs planned to cross roadmap vertices in the (a) same and (b) opposite directions. .

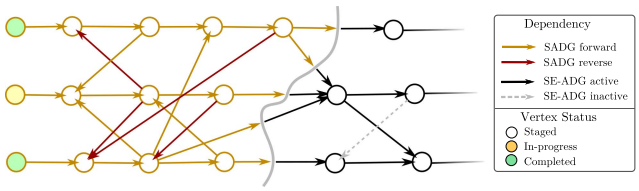

Fig. 9. Graphical illustration of how the original SADG can be split into a finite horizon SADG subset and an SE-ADG, while ensuring if the finite horizon SADG subset yields an acyclic SE-ADG subset, the resultant SE-ADG will be acyclic.

sequence of dependencies of the pattern

$$(v_i^{k+n}, v_j^{l+n}),\ n \in 1, 2, \ldots.$$

A dependency group DG then consists of the sequence

$$\text{DG} = ((v_i^k, v_j^l), (v_i^{k+1}, v_j^{l+1}), (v_i^{k+2}, v_j^{l+2}), \ldots).$$

Similarly, the second pattern is shown in Fig. 8(b), corresponding to two AGVs travelling in the opposite direction. This pattern can be found by sequentially visiting each inter-AGV dependency (v_i^k, v_j^l) in the SADG and searching for any sequence of dependencies of the pattern

$$(v_i^{k+n}, v_j^{l-n}),\ n \in 1, 2, \ldots.$$

A dependency group DG then consists of the sequence

$$\text{DG} = ((v_i^k, v_j^l), (v_i^{k+1}, v_j^{l-1}), (v_i^{k+2}, v_j^{l-2}), \ldots).$$

In both cases, a single binary variable is sufficient to express the switching for all the dependency pairs in the group, since the only way for the switching to yield an acyclic SE-ADG is if either all the forward dependencies and none of the reverse dependencies are active (or vice-versa). Depending on the roadmap topography, the size of $\boldsymbol{b}$ can be significantly reduced, greatly reducing the complexity of the MILP problem at each iteration. Identifying dependency groups is an $\mathcal{O}(n)$ operation, and can be done during SADG construction, i.e., before plan execution.

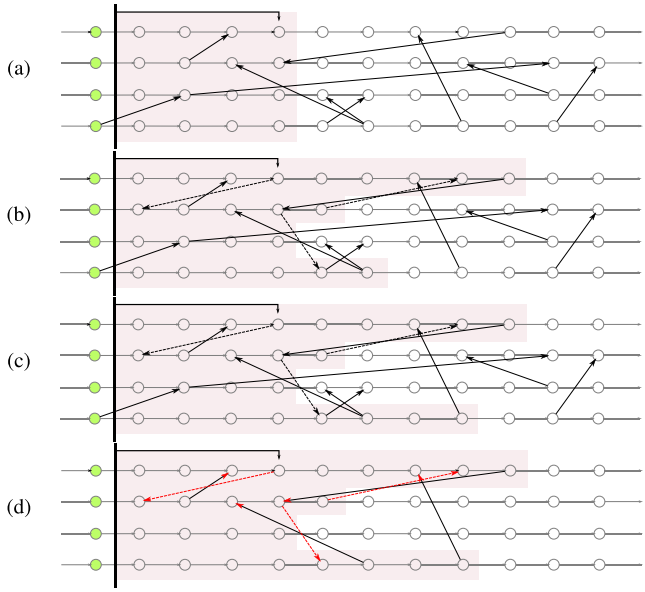

Fig. 10. Illustrative example of how Algorithm 3 extracts the finite horizon SADG subset from the original SADG for a user-specified horizon H. The magenta region refers to the stack $\mathcal{V}_{\text{FH}}$ in Algorithm 3. (a) Refers to line 1, with the vertices within H added to $\mathcal{V}_{\text{FH}}$. (b) Refers to line 2, where all forward or reverse dependencies pointing to within H are added to $\mathcal{V}_{\text{FH}}$. (c) Refer to lines 3 and 4, where any forward dependencies pointing to $\mathcal{V}_{\text{FH}}$ in (b) are included. (d) Switchable dependencies, in red, as identified in lines 5–8.

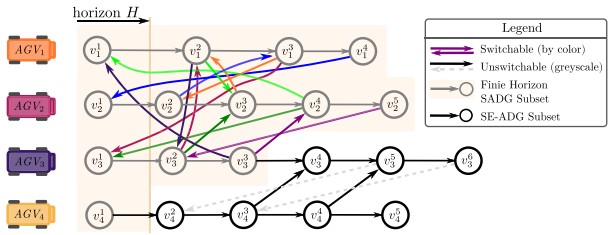

Fig. 11. Highlighted finite horizon SADG subset connected to the acyclic SE-ADG subset with unidirectional dependencies. The horizon H is indicated by the horizontal black arrow.

E. Summary

The result of this section is that we have an SADG from which an OCP is derived which can be solved at any time-step to re-order the AGVs based on their progress of the plans. A feasible solution is guaranteed to yield a deadlock-free, collision free plan for the AGVs.

VI. RHC Scheme

We now address the fact that the OCP in (1) could yield an optimization problem too large for feasible real-time implementation as it considers the entire, finite-length plans. Specifically, we show how the OCP in (1) can be approximated by a receding horizon variant of the MILP in (4). The motivation for this is predicated on the fact that MILP problems are exponentially complex in the number of discrete variables. Furthermore, a receding horizon formulation allows for ad-hoc re-planning of AGVs without needing to wait for all AGVs to reach their goals.

To this end, we first introduce a method to split an SADG into a smaller SADG and an acyclic SE-ADG. We then show how reformulating the MILP to consider the smaller SADG approximates a solution to the original OCP and maintains recursive feasibility guarantees.

Note that we extend the approach originally presented in [12] by introducing a method to split the SADG. This enables the persistent resolving of the MAPF mid-route as shown for the original ADG method in [4].

A. Introducing the Finite Horizon SADG Subset

RHC approaches are predicated on the fact that a sufficiently accurate solution to an OCP can be obtained by only considering the system trajectories within a finite horizon. For discrete systems, such as the one described by a SADG, defining the finite horizon which guarantees the RHC presents a unique challenge: reordering AGVs within a finite horizon could still yield a cyclic SE-ADG, despite the portion of the SE-ADG within the horizon being acyclic.

To this end, we present Algorithm 3 to split an SADG into 1) a *finite horizon SADG* and 2) an *acyclic SE-ADG* such that if the *finite horizon SADG* (a subset of the original SADG) yields an acyclic, finite horizon SE-ADG for a particular $\boldsymbol{b}_{\text{rhc}} \subset \boldsymbol{b}$, the entire SE-ADG will be acyclic. This finite horizon SADG can then be used, instead of the original SADG, to parameterize MILP formulation, greatly reducing the computational load and maintaining collision- and deadlock-free plan execution guarantees.

To this end, we introduce Lemma 2, a result which enables the splitting of an SADG into a finite horizon SADG subset and an SE-ADG subset, the resulting SE-ADG will be acyclic if the switching within the finite horizon SADG subset yields an acyclic SE-ADG. Lemma 2 builds on the more general fact that a graph $\mathcal{G}$ is acyclic if 1) it is constructed from two acyclic graphs, $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, and 2) all edges connecting $\mathcal{G}_1$ and $\mathcal{G}_2$ point from $\mathcal{V}_1$ to $\mathcal{V}_2$. This result is proven in Appendix A, Lemma 3. The result in Lemma 2 is illustrated by the example in Fig. 9. The gray partition line splits the SADG into a SADG subset and an SE-ADG. Note how all the dependencies connecting the SADG subset and the SE-ADG go from the SADG subset to the SE-ADG.

The application of Algorithm 3 to our running example is illustrated in Fig. 11. Here, the finite horizon SADG subset is highlighted in orange.

Extracting a finite horizon SADG subset from the SADG can be done with Algorithm 3. To illustrate the intuition behind Algorithm 3, consider Fig. 10. In line 1, Algorithm 3 navigates the graph $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$ by first pushing all the vertices estimated to be completed within a user-specified time horizon H to a stack $\mathcal{V}_{\text{FH}}$, shown in Fig. 10(a). Next, each forward–reverse dependency pair pointing to a vertex in $\mathcal{V}_{\text{FH}}$ is identified, and its associated binary variable is appended to $\boldsymbol{b}_{\text{rhc}}$. This is illustrated by the magenta region in Fig. 10(b). The remaining forward dependencies pointing to vertices in $\mathcal{V}_{\text{FH}}$ are appended to $\mathcal{E}_{\text{FH}}$, until no dependencies point from a vertex in $\mathcal{V}_{\text{FH}}$ to a vertex outside $\mathcal{V}_{\text{FH}}$. This is shown in Fig. 10(c) The resulting switchable dependencies, which are considered within $\boldsymbol{b}_{\text{rhc}}$ are shown in Fig. 10(d).

Algorithm 3: Extracting a Finite Horizon SADG Subset.

Input: $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$, H
Result: $\mathcal{G}^{\text{rhc}}_{\text{SADG}}(\boldsymbol{b}_{\text{rhc}})$
// Add switching dependency pairs within horizon H
1: Add vertices to $\mathcal{V}_{\text{FH}}$ within H
2: Add all edges to/from $\mathcal{V}_{\text{FH}}$ to $\mathcal{E}_{\text{FH}}$
3: Add $b \in \boldsymbol{b}$ to $\boldsymbol{b}_{\text{rhc}}$ if b is related to edges in $\mathcal{E}_{\text{FH}}$
4: Add all edges pointing to $v \in \mathcal{V}_{\text{FH}}$ to $\mathcal{E}_{\text{inwards}}$
// Expand $\mathcal{V}_{\text{FH}}$ until no dependencies point into $\mathcal{V}_{\text{FH}}$
5: **while** $\mathcal{E}_{\text{inwards}} \neq \emptyset$ **do**
6: $e \leftarrow \text{pop}(\mathcal{E}_{\text{inwards}})$
7: Add all v's pointing to e
8: Add all edges pointing from v out of $\mathcal{V}_{\text{FH}}$ to $\mathcal{E}_{\text{inwards}}$
9: **end while**
10: **return** $\mathcal{G}^{\text{rhc}}_{\text{SADG}}(\boldsymbol{b}_{\text{rhc}}) = (\mathcal{V}_{\text{FH}}, \mathcal{E}_{\text{FH}}(\boldsymbol{b}_{\text{rhc}}))$

Lemma 2 (Finite horizon SADG subset solution guarantees acyclicity of resultant SE-ADG): Consider a SADG, $\mathcal{G}_{\text{SADG}}(\boldsymbol{b})$, split into a finite horizon SADG subset and an SE-ADG subset using Algorithm 3. If $\boldsymbol{b}^*_{\text{rhc}}$ is such that $\mathcal{G}^{\text{rhc}}_{\text{SADG}}(\boldsymbol{b}^*_{\text{rhc}})$ is acyclic, the resultant SE-ADG is acyclic.

Proof: By lines 5–9, Algorithm 3 ensures all dependencies connecting the finite horizon SADG subset, $\mathcal{G}^{\text{rhc}}_{\text{SADG}}()$, and the SE-ADG subset, $\mathcal{G}^{\text{rhc}}_{\text{SE-ADG}}$, are directed from $\mathcal{G}^{\text{rhc}}_{\text{SADG}}()$ to $\mathcal{G}^{\text{rhc}}_{\text{SE-ADG}}$ only. Using the result in Lemma 3, if a particular $\boldsymbol{b}_{\text{rhc}}$ is chosen such that $\mathcal{G}^{\text{rhc}}_{\text{SADG}}(\boldsymbol{b}_{\text{rhc}})$ is acyclic, the resultant SE-ADG will be acyclic, since $\mathcal{G}^{\text{rhc}}_{\text{SE-ADG}}$ is acyclic as well.

B. Reformulation of MILP

We now reformulate the MILP to consider the finite horizon SADG subset instead of the entire SADG as was done in (4)

$$\min_{\boldsymbol{b}, \boldsymbol{t}_g, \boldsymbol{t}_s} \sum_{i=1}^{N} t_g(v_i^{n_i}) \tag{5a}$$

s.t.

$$t_g(v_i^k) > t_s(v_i^k) + \Delta T(v_i^k) \quad \forall\, v_i^k \in \Pi\left(\mathcal{V}_{\text{FH}}(\boldsymbol{b})\right) \tag{5b}$$

$$t_s(v_i^{k+1}) > t_g(v_i^k) \quad \forall\, v_i^{k+1} \in \Gamma\left(\mathcal{V}_{\text{FH}}(\boldsymbol{b})\right) \tag{5c}$$

$$status(v') = staged \quad \forall\, (v, v') \in \Psi(\boldsymbol{b}, \boldsymbol{b}_0, \mathcal{V}_{\text{FH}}(\boldsymbol{b})) \tag{5d}$$

$$t_g(v_i^k) < t_s(v_j^l) \quad \forall\, (v_i^k, v_j^k) \in \mathcal{E}_{\text{FH}}(\boldsymbol{b}) \text{ if } i \neq j \tag{5e}$$

$$t_s(v_j^l) > t_g(v_i^k) - bM \quad \forall\ b \in \boldsymbol{b} \tag{5f}$$

$$t_s(v_i^{k-1}) > t_g(v_j^{l+1}) - (1-b)M \quad \forall\ b \in \boldsymbol{b} \tag{5g}$$

where $M \gg 0$ is a large constant and n_i is the horizon length as determined for AGV_i as determined by Algorithm 3.

C. Receding Horizon Feedback Control Scheme

Having reformulated the MILP in (5) such that a subset of the SADG is considered at each optimization step, we present the

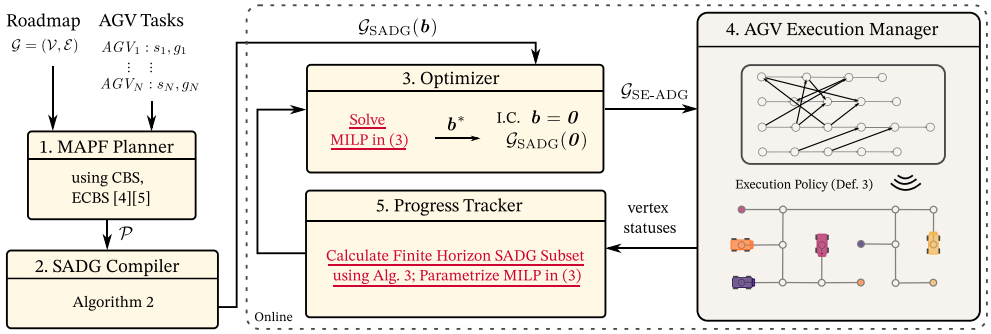

Fig. 12. Receding horizon feedback control diagram. *Differences from Fig. 6 are highlighted in red.* As AGVs progress, and are inevitably delayed, their status is tracked, and an optimization iteration is performed to re-order the AGVs based on a Finite Horizon SADG subset calculated by Algorithm 3. The solution of the MILP is used to update the SE-ADG used for plan execution in a feedback loop.

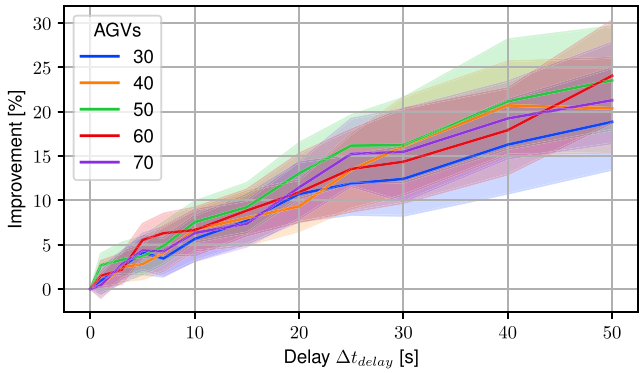

Fig. 13. Improvement for various simulated AGV delays navigating roadmaps in Fig. 14. Each Δt_{delay} seconds, a randomly selected subset of 20% of the AGVs are stopped for Δt_{delay} seconds. Horizon $H = 5$. Bounds correspond to $\pm 1\sigma$.

receding horizon feedback control scheme as shown in Fig. 12. The main difference between the RHC scheme and the SHC scheme presented in Section V is the inclusion of Algorithm 3 in the step *5. Track AGV Progress*. Finally, we prove recursive feasibility of the RHC scheme.

Proposition 2 (Recursive feasibility of RHC scheme): If the initial SE-ADG, $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\mathbf{0})$, obtained from the planning phase, is acyclic, the RHC scheme will guarantee that $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*)$ is acyclic at each subsequent optimization step.

Proof: Proof by induction. Initially the SE-ADG, $\mathcal{G}_{\text{SE-ADG}} = \mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_0})$ is acyclic. Algorithm 3 is used to extract the finite horizon SADG subset at $t = T_1 > T_0$. If the MILP in (5) is solved at $t = T_1 > T_0$, Corollary 5 guarantees that the SE-ADG at T_1, $\mathcal{G}_{\text{SE-ADG}}{}^{T_1}$, obtained from $\mathcal{G}_{\text{SADG}}(\boldsymbol{b}^*_{T_1})$, is acyclic. Subsequent optimization steps will always result in acyclic SE-ADGs, proving recursive feasibility of the feedback control scheme.

VII. Evaluation

We perform extensive evaluations of our proposed feedback control scheme. Our simulations consist of multiple AGVs occupying various roadmaps with randomized start and goal locations. Each simulation starts with a planning step where the MAPF is solved given the start and goal locations.

To gain insight into our proposed control schemes, we perform evaluations in three different settings. In the first, we aim to gain statistical insight into the performance gains of our approach compared with the original ADG baseline by considering various roadmap topologies and AGV fleet sizes. In the second setting, we compare our method to the state-of-the-art robust MAPF planner K-CBSH-RM [34]. In the third setting, we showcase our method in a high-fidelity ROS and Gazebo simulation environment. In all three settings, the RHC SADG is used in combination with dependency grouping as described in Section V-D. All simulations were conducted on a Lenovo Thinkstation with an Intel Xeon E5-1620 3.5 GHz processor and 64 GB RAM.

A. Setting 1: Statistical Analysis

We compare our proposed receding horizon feedback control scheme, referred to from here on as the RHC SADG approach, to the original ADG approach in [4]. We consider the roadmaps in Fig. 14, inspired by intralogistic warehouse layouts [1], [8], [9]. For each simulation run, AGVs are given randomized start and goal locations on the map. The execution policy in Definition 3 is used for both the ADG and RHC SADG approaches. The AGVs are simulated as simple differential-drive robots with constant rotational and translational velocities of $3\,\text{rad}\cdot\text{s}^{-1}$ and $1\,\text{m}\cdot\text{s}^{-1}$, respectively. The coordination of AGVs is performed using ROS, where a central coordination ROS node solves the MAPF, constructs the RHC SADG (or ADG), and communicates the vertex statuses to each AGV. The feedback loop sampling time h is set to 2 s. AGVs are artificially delayed as follows: at the start of each interval of length Δt_{delay}, 20% of the AGV fleet is *randomly selected* and stopped for the next Δt_{delay} seconds. For the next interval, a different subset of the AGVs is randomly selected and delayed, and so on. Improvement is quantified by comparing the cumulative route completion time of all the AGVs for the same MAPF plan when using the baseline ADG approach to our receding horizon feedback control scheme, defined as

$$\text{improvement} = \frac{\sum t_{\text{ADG}} - \sum t_{\text{RHCSADG}}}{\sum t_{\text{ADG}}} \cdot 100\% \quad (6)$$

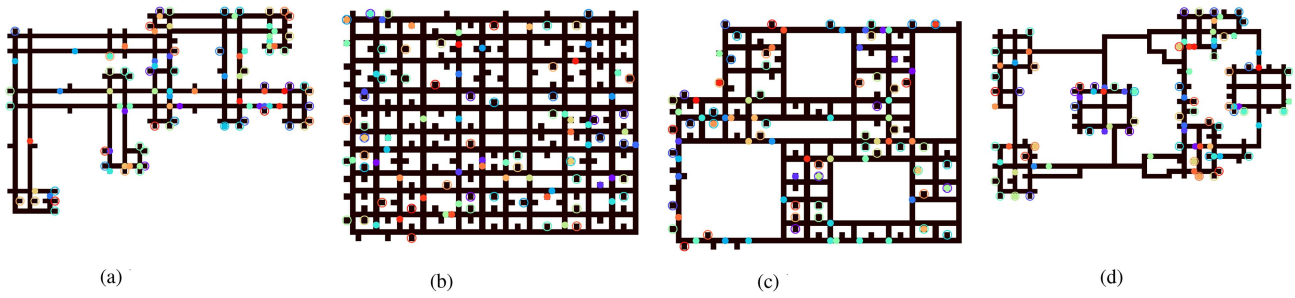

Fig. 14. Roadmaps used for the statistical analysis in Section VII-A, inspired by [1], [8], and [9]. AGVs are indicated by colored circles, and their corresponding goal location by the same colored ring. Roadmap vertices are represented by black squares. (a) Warehouse. (b) Full Maze. (c) Half Maze. (d) Islands.

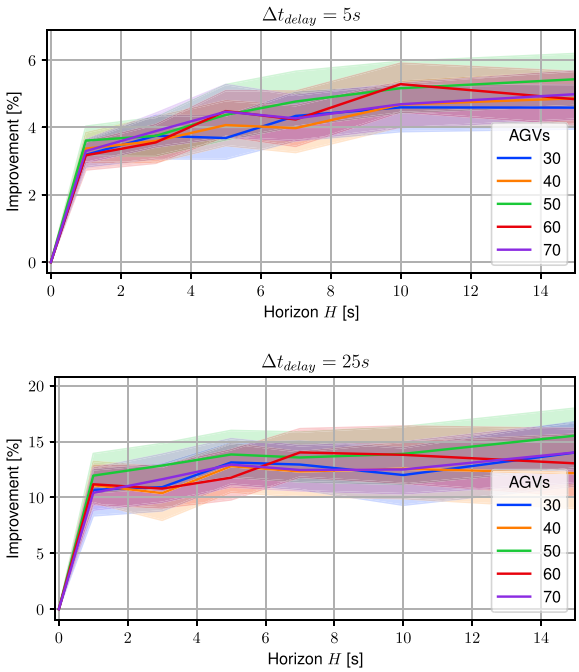

Fig. 15. Improvement for delays Δt_{delay} of 5 and 25 s given various horizons H. Bounds correspond to $\pm 1\sigma$.

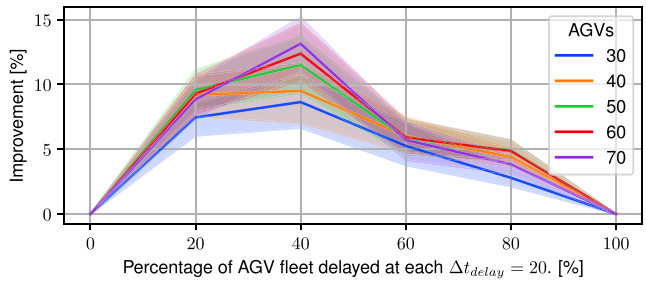

Fig. 16. Improvement for various percentages of AGVs delayed at each interval of Δt_{delay}. $\Delta t_{\text{delay}} = 20$ s and horizon $H = 5$ s. Bounds correspond to $\pm 1\sigma$.

where $\sum t_*$ refers to the cumulative plan completion time for all AGVs using approach $*$.

Improvement and Delays: We consider delays of $\Delta t_{\text{delay}} \in \{1, 3, 5, 10, 15, 20, 25, 30, 40, 50\}$ seconds for AGV fleet sizes of $\{30, 40, 50, 60, 70\}$ navigating the four roadmaps in Fig. 14. We run 100 simulations for each fleet size and delay time permutation. For each simulation, the MAPF is solved using ECBS with suboptimality bound w chosen such that the initial planning time is below ten minutes. Fig. 13 shows the improvement for various AGV fleet sizes and delays. The horizon H is set to 5 s. We observe that the improvement is almost linear with respect to the delays of the AGVs, for all the considered fleet sizes. The improvement standard deviation, σ, indicated by the lightly colored bands, is relatively small, indicating that improvement is consistent for a given fleet size and delay configuration.

Improvement and Horizon: For the RHC SADG approach, we consider various horizons $H \in \{1, \ldots, 15\}$ s. We consider delays $\Delta t_{\text{delay}} = 5$ s and $\Delta t_{\text{delay}} = 25$ s, with the improvement shown in Fig. 15. We run 100 simulations for each fleet size, horizon, and delay time permutation. Notice how for both shorter and longer delays, the improvement already significantly increases with low horizons H, indicating that the RHC MILP in (5) performs a consistently good approximation of the OCP in (1) for small H.

Varying Percentage of Delayed AGV Fleet: We evaluate different proportions of the AGVs delayed at each $\Delta t_{\text{delay}} = 20$ s, with the improvement for an horizon $H = 5$ s, shown in Fig. 16. When 0% of the AGVs are delayed, the improvement is 0% since none of the AGVs are re-ordered with the RHC SADG approach. Similarly, when 100% of the AGVs are delayed, the cumulative route completion times for both the ADG and SADG methods is ∞, yielding an improvement of 0%. Improvement is highest when 40% randomly selected AGVs are delayed each Δt_{delay}.

Different Roadmaps: We evaluate the improvement separately for each roadmap in Fig. 14. Results are shown in Fig. 17 for horizon $H = 5$ s and $\Delta t_{\text{delay}} = 10$ s. We note that the least improvement is seen for the sparser Islands map, and the best improvement is seen for the denser Full Maze and Warehouse maps. We conclude that our method is best suited to maps which present the *opportunity* for switching, as this increases the binary decision space of the OCP.

RHC MILP Computation Times: For our simulations, the MILP was solved using the coin-or branch-and-cut solver [41]. The computation times for different AGV fleet sizes and horizon lengths are shown in Fig. 18. We note that the MILP can be solved

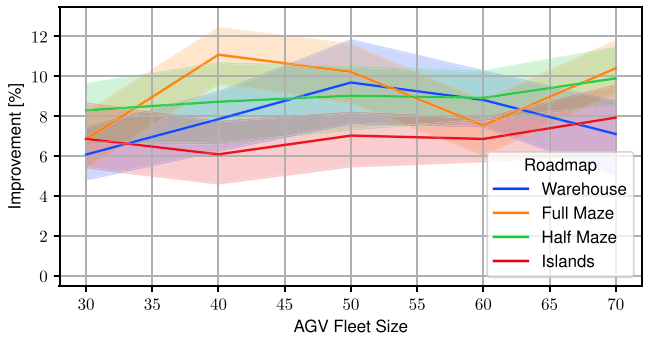

Fig. 17. Improvement for the roadmaps shown in Fig. 14 for various AGV fleet sizes. 20% of the AGVs are delayed by $\Delta t_{\text{delay}} = 20$ s and the $H = 5$ s. Bounds correspond to $\pm 1\sigma$.

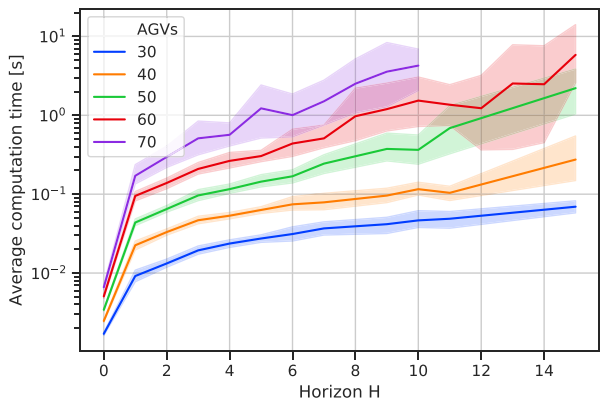

Fig. 18. Computation times of 3. Optimizer in Fig. 12 for varying horizon lengths and AGV fleet sizes. Bounds correspond to $\pm 3\sigma$.

below 1 s for fleet sizes of up to 70 AGVs for horizons below $H = 5$ s. Recall that even smaller horizons can yield significant improvement as in Fig. 15.

Summary: We observe significant reductions in average route completion times for AGVs. Specifically, the larger the delays observed by the AGVs, the larger the improvement is observed when using the RHC SADG approach compared with the ADG baseline using the same initial MAPF solution. Significant improvements are observed for small RHC SADG horizons, across multiple roadmap topologies.

B. Setting 2: Comparison With Robust MAPF Solver

In this section we compare our proposed receding horizon feedback control scheme to the state-of-the-art robust MAPF solver K-CBSH-RM [34]. We consider the roadmap in Fig. 19(a) with AGV fleets of size 20 and 25 with the horizon of the RHC SADG approach set to $H = 10$ s. The smaller roadmap and fleet sizes were chosen because K-CBSH-RM and CBS failed to yield valid MAPF solutions for larger maps or fleet sizes. Recall that we used the bounded suboptimal equivalent of CBS, ECBS, in Setting 1, allowing us to find feasible solutions to the MAPF for larger solution spaces. As in Setting 1, improvement is measured using the original ADG approach with CBS as the initial MAPF planner.

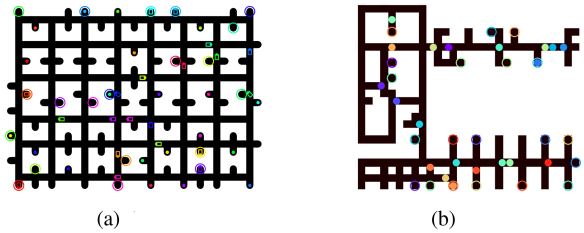

Fig. 19. Roadmaps with AGVs indicated by colored circles, and their correspondingly colored rings denoting their goal locations. (a) Roadmap for the comparison with K-CBSH-RM in Section VII-B. (b) Roadmap used for the Gazebo evaluation in Section VII-C.

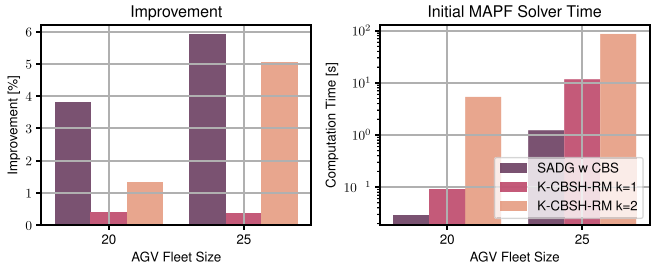

Fig. 20. Average improvement and initial MAPF solver computation times for 100 simulations with AGVs velocity profiles from Fig. 21 comparing the RHC SADG method with the K-CBSH-RM planner for $k \in \{1, 2\}$s for $H = 10$ s.

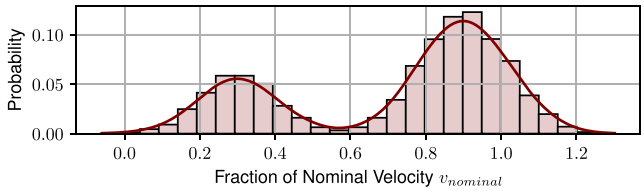

Fig. 21. Normalized velocity distribution for Setting 2. Each AGV is given a randomly sampled velocity to complete its next SADG event from this distribution.

Simulating Interactions with Dynamic Obstacles: To simulate random interactions with dynamic obstacles, each AGV is prescribed a different, randomly sampled velocity when completing an SE-ADG event. These velocities are sampled from the distribution shown in Fig. 21. This distribution was created to simulate the fact that, most often, AGVs move at a velocity close to a nominal velocity v_{nominal}, but occasionally travel at significantly slower velocities when navigating around dynamic obstacles, modeled here by the velocity distribution centered around $0.3v_{\text{nominal}}$, with $v_{\text{nominal}} = 1\text{m}\cdot\text{s}^{-1}$. Fig. 20 shows the comparison of the RHC SADG approach with K-CBSH-RM for $k \in \{1, 2\}$ seconds for 100 simulations. We observe that the RHC SADG approach yields a higher improvement during plan execution for both AGV fleet sizes. As expected, we observed that improvement increased for increasing values of k in K-CBSH-RM. However, for $k > 2$, K-CBSH-RM only found valid MAPF solutions for 1% of the simulations within the 240 s cutoff time, making a comparison with our approach for $k > 2$ impossible. The success rate of K-CBSH-RM for different values of k is shown in Fig. 22. These results highlight the fact

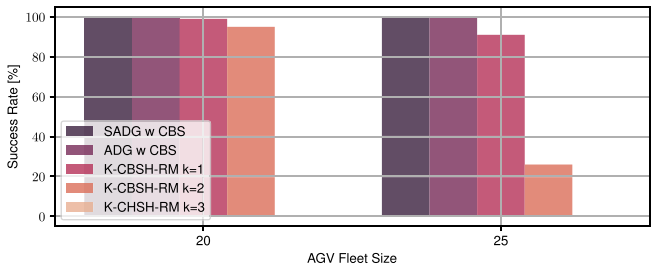

Fig. 22. Initial MAPF planner success for 100 simulations using CBS and K-CBSH-RM with $k \in \{1, 2, 3\}s$.

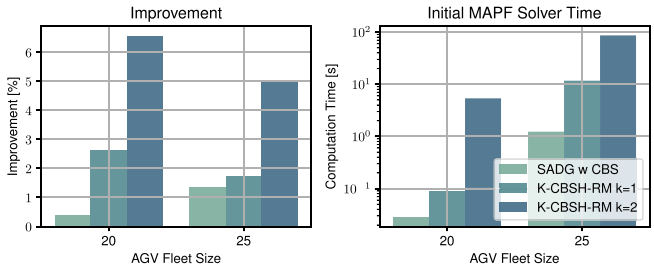

Fig. 23. Average improvement and initial MAPF solver computation times for 100 simulations with AGVs velocity profiles from Fig. 24 comparing the RHC SADG method with the K-CBSH-RM planner for $k \in \{1, 2\}$s for $H = 10$ s.

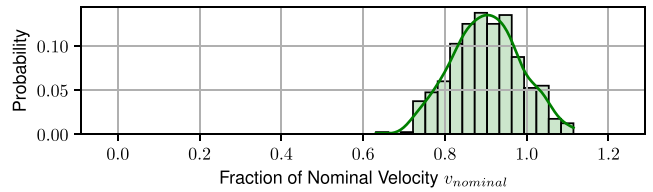

Fig. 24. Normalized velocity distribution for Setting 2. Each AGV is given a randomly sampled velocity to complete its next SADG event from this distribution.

that our approach can be used with a nonrobust planner, such as CBS or ECBS, to ensure a valid solution to the MAPF is found, while adding robustness to delays in an online fashion through the re-ordering of AGV plans.

Simulating Workspaces with Bounded Delays: The family of robust MAPF solvers assume delays of all AGVs are upper-bounded by k seconds [5], [34]. We compare our proposed approach to K-CBSH-RM with a velocity distribution in Fig. 24 resulting in smaller delays. Fig. 23 shows that K-CBSH-RM performs better than our approach in this case, which only yields marginal improvements compared with the ADG baseline.

Remark 2: Our approach is agnostic to the original MAPF planner. Hence, it would be possible to use K-CBSH-RM to solve the initial MAPF, and the RHC SADG to re-order AGVs based on delays observed during plan execution.

Summary: We observe that although K-CBSH-RM yields theoretically lower cumulative route completion times when delays are bounded by k seconds, our proposed RHC SADG approach yields better improvement when delays are larger. In addition, the robust MAPF planners require significantly more time to solve the robust MAPF problem, as opposed to the standard MAPF solver, CBS, or ECBS, used by our approach.

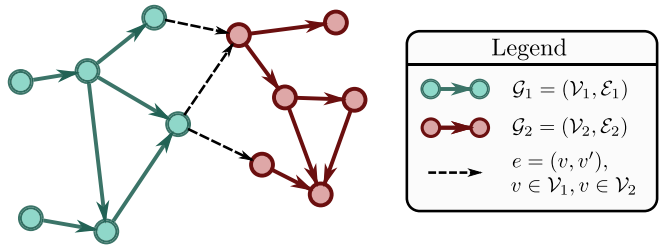

Fig. 25. Graphical illustration of Lemma 3: two acyclic graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, connected by unidirectional edges, indicated by the dotted lines, yield a larger, acyclic graph.

TABLE I
AVERAGE IMPROVEMENT FOR 20 SIMULATIONS USING RHC SADG WITH $H = 10$S IN GAZEBO

	Initial MAPF Solver Time [s]	**Improvement [%]**
High-Fidelity Gazebo Simulation	3.45 ± 0.31	8.4 ± 2.2
Setting 2: Simulating Interactions with Dynamic Obstacles	3.65 ± 0.25	6.7 ± 0.8

C. Setting 3: High-Fidelity Gazebo Simulations

We evaluate our proposed approach in a high-fidelity Gazebo simulation environment. We consider 20 AGVs with randomly selected goal and start locations navigating the roadmap in Fig. 19(b). AGVs use the open-source ROS *move_base* motion planner to execute the SE-ADG events. Delays occur naturally as AGVs navigate the workspace and negotiate interactions with other dynamic obstacles. Improvement is quantified as in (6) and the results are given in Table I. We observe positive improvements for all 20 simulations with slightly higher improvement in the Gazebo simulation compared with the equivalent start-goal positions when simulating dynamic obstacles in Setting 2. The Gazebo simulations yielded a higher improvement because AGVs were found to experience larger delays than modeled in Fig. 21.

VIII. CONCLUSION

In this manuscript, we propose an optimization-based receding horizon feedback control scheme to re-order AGVs subject to delays when executing MAPF plans. When compared with the state-of-the-art MAPF planners, our approach yields a significant reduction in cumulative route completion times for AGVs subjected to large delays, often experienced in uncertain environments with dynamic obstacles. Our optimization-based re-ordering scheme is derived to obtain approximate solutions to a newly formulated optimal control problem (OCP). This OCP is described using a SADG, a novel data-structure introduced in this manuscript. The SADG extends the ADG introduced in [4] by enabling the re-ordering of AGVs while provably maintaining collision-avoidance guarantees of the original MAPF plan. Moreover, our approach guarantees deadlock-free plan execution while simultaneously minimizing the cumulative route completion time of the AGVs.

We evaluate our approach in three settings. In Setting 1, we illustrate the efficiency of our approach, reducing the cumulative route completion time for AGVs by up to 25% compared with the baseline ADG approach. Here, we also illustrate the real-time implementability of our feedback scheme, showing that the RHC MILP problem can consistently be solved under 1 s, even for AGV fleet sizes of up to 70 AGVs, all the while significantly reducing route completion times. In Setting 2, we compare our approach to the state-of-the-art in robust MAPF planner K-CBSH-RM, showing a significant reduction in cumulative route completion times for AGVs subjected to larger delays, and comparable cumulative route completion times for smaller delays. In Setting 3, we showcased our approach in a high-fidelity Gazebo simulation environment with 20 AGVs navigating around dynamic obstacles, reducing the cumulative route completion time by 8%, thereby validating the results obtained in Settings 1 and 2.

In all simulation settings, the AGVs exhibited collision- and deadlock-free plan execution, a result we prove for both the SHC and RHC feedback schemes. Our approach is also agnostic with respect to the planner used to solve the initial MAPF problem. This means that most MAPF planners, e.g., CBS, ECBS, K-CBSH-RM, can be used, as long as the initial MAPF solution yields an acyclic SE-ADG, a constraint which is easy to adhere to as long as the number of roadmap vertices is larger than the number of AGVs. Although we only consider AGVs executing intralogistics tasks, our approach can be extended to other use-cases covered in the MAPF literature. For future work, we recommend a detailed comparison of our approach to real-time re-planning of the MAPF using bounded, suboptimal MAPF solvers as in [2].

APPENDIX A

Lemma 3 (Two acyclic graphs connected by unidirectional edges yield an acyclic graph): Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ subdivided into two subgraphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ such that $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ and $\mathcal{E}_1 \cap \mathcal{E}_2 = \emptyset$, $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$ and the edges connecting vertices in $\mathcal{G}_1$ and $\mathcal{G}_2$ are contained within the set $\mathcal{E}_{12}$, such that $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_{12} = \mathcal{E}$. If both $\mathcal{G}_1$ and $\mathcal{G}_2$ are acyclic and $e = (v_1, v_2)$ is such that $v_1 \in \mathcal{V}_1$ and $v_2 \in \mathcal{V}_2$ for all $e \in \mathcal{E}_{12}$, then the graph $\mathcal{G}$ is also acyclic.

Proof: Consider two acyclic graphs, $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, illustrated in Fig. 25. For $\mathcal{G}_1$, consider an inbound edge $e = (v, v')$, which implies that $v \notin \mathcal{V}_1$ and $v' \in \mathcal{V}_1$. Any number of inbound edges e will not cause $\mathcal{G}_1$ to be cyclic. Similarly, consider an outbound edge $e = (v, v')$, which implies that $v \in \mathcal{V}_1$ and $v' \notin \mathcal{V}_1$. Any number of outbound edges e will not cause $\mathcal{G}_1$ to be cyclic. The same arguments apply to $\mathcal{G}_2$. Since neither $\mathcal{G}_1$ nor $\mathcal{G}_2$ have an internal cycle, the only possibility for a cycle within $\mathcal{G}$ is a cycle through both subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Since all edges connecting $\mathcal{G}_1$ and $\mathcal{G}_2$ can be defined by edge $e = (v, v')$ such that $v \in \mathcal{V}_1$ and $v' \in \mathcal{V}_2$, such a cycle cannot exist. This guarantees that the entire graph is acyclic, completing the proof.

ACKNOWLEDGMENT

The authors would like to thank Musa Morena Marcusso Manhães for her help with setting up the ROS and Gazebo frameworks as well as Zhe Chen for the help setting up the code to compare our approach with K-CHSH-RM solver.

REFERENCES

[1] P. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, pp. 9–20, 2008.
[2] R. Stern et al., "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proc. Int. Symp. Combinatorial Search*, vol. 12, pp. 151–158, 2019.
[3] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," *Algorithmic Foundations Robot. X*, pp. 157–173, 2013.
[4] W. c, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1125–1131, Apr. 2019.
[5] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding and executing," *J. Artif. Intell. Res.*, vol. 67, pp. 549–579, 2020.
[6] W. Hönig et al., "Summary: Multi-agent path finding with kinematic constraints," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 4869–4873.
[7] H. Ma, S. Kumar, and S. Koenig, "Multi-agent path finding with delay probabilities," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 3605–3612.
[8] *Assoc. Automot. Ind.*, "Interface for the communication between automated guided vehicles (AGV) and a master control," Assoc. Automot. Ind. (VDA) Tech. Rep. v2.0, 2022. Accessed: Oct. 4, 2023. [Online]. Available: https://github.com/VDA5050/VDA5050
[9] Open Source Robotics Foundation, Inc., "Open-RMF," 2022. Accessed: Oct. 15, 2022. [Online]. Available: https://www.open-rmf.com
[10] M. Čáp, J. Gregoire, and E. Frazzoli, "Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 5113–5118.
[11] A. Coskun, J. O'Kane, and M. Valtorta, "Deadlock-free online plan repair in multi-robot coordination with disturbances," in *Proc. Int. FLAIRS Conf. Proc.*, vol. 34, pp. 1–6, 2021.
[12] A. Berndt, N. van Duijkeren, L. Palmieri, and T. Keviczky, "A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph," in *Proc. Distrib. Multi-Agent Plan. Workshop ICAPS*, vol. 6, pp. 1–9, 2020.
[13] A. Felner et al., "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Proc. 10th Int. Symp. Combinatorial Search*,, 2017, pp. 29–37.
[14] R. Morris et al., "Planning, scheduling and monitoring for airport surface operations," in *Proc. AAAI Workshop: Plan. Hybrid Syst.*, 2016, pp. 608–614.
[15] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.
[16] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, 2015.
[17] M. Cáp, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 835–849, Jul. 2015.
[18] A. Bogatarkan, V. Patoglu, and E. Erdem, "A declarative method for dynamic multi-agent path finding," in *Proc. 5th Glob. Conf. Artif. Intell.*, 2019, pp. 54–67.
[19] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," in *Proc. 28th Int. Conf. Automated Plan. Scheduling*, 2018, pp. 485–493.
[20] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 3612–3617.
[21] J. Li, D. Harabor, P. Stuckey, H. Ma, and S. Koenig, "Symmetry-breaking constraints for grid-based multi-agent path finding," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 6087–6095.
[22] A. Felner et al., "Adding heuristics to conflict-based search for multi-agent path finding," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2018, pp. 83–87.

[23] E. Lam, P. L. Bodic, D. Harabor, and P. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 28, pp. 1289–1296, 2019.
[24] A. Bemporad, W. Heemels, and B. D. Schutter, "On hybrid systems and closed-loop MPC systems," *IEEE Trans. Autom. Control*, vol. 47, no. 5, pp. 863–869, May 2002.
[25] S. Lin, B. De Schutter, Y. Xi, and H. Hellendoorn, "Fast model predictive control for urban road networks via MILP," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 3, pp. 846–856, Sep. 2011.
[26] R. Hult, G. R. Campos, P. Falcone, and H. Wymeersch, "An approximate solution to the optimal coordination problem for autonomous vehicles at intersections," in *Proc. Amer. Control Conf.*, 2015, pp. 763–768.
[27] S. Ravikumar, R. Quirynen, A. Bhagat, E. Zeino, and S. Di Cairano, "Mixed-integer programming for centralized coordination of connected and automated vehicles in dynamic environment," in *Proc. IEEE Conf. Control Technol. Appl.*, 2021, pp. 814–819.
[28] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *AIAA J. Guid., Control, Dyn.*, vol. 25, pp. 755–764, 2001.
[29] M. Charitidou and T. Keviczky, "An MILP approach for persistent coverage tasks with multiple robots and performance guarantees," *Eur. J. Control*, vol. 64, 2022, Art. no. 100610.
[30] T. van den Boom and B. D. Schutter, "Dynamic railway network management using switching max-plus-linear models," *IFAC Symp. Control Transp. Syst.*, vol. 11, pp. 343–348, 2006.
[31] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proc. Eur. Conf. Artif. Intell.*, 2014, pp. 961–962.
[32] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 39–45.
[33] A. Coskun and J. M. O'Kane, "Online plan repair in multi-robot coordination with disturbances," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 3333–3339.
[34] Z. Chen, D. Harabor, J. Li, and P. J. Stuckey, "Symmetry breaking for k-robust multi-agent path finding," in *Proc. 35th AAAI Conf. Artif. Intell.*, 2021, pp. 12267–12274.
[35] A. Mannucci, L. Pallottino, and F. Pecora, "Provably safe multi-robot coordination with unreliable communication," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3232–3239, Oct. 2019.
[36] T. Schoels, L. Palmieri, K. O. Arras, and M. Diehl, "An NMPC approach using convex inner approximations for online motion planning with guaranteed collision avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3574–3580.
[37] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 5681–5686.
[38] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *J. Guid., Control, Dyn.*, vol. 40, no. 2, pp. 344–357, 2017.
[39] R. Triebel et al., "Spencer: A socially aware service robot for passenger guidance and help in busy airports," in *Field and Service Robotics*, Berlin, Germany: Springer, 2016, pp. 607–622.
[40] E. Schrock, "Dynamic lock dependency analysis of concurrent systems," Ph.D. thesis, Dept. Comput. Sci., Brown Univ., Providence, RI, USA, 2003.
[41] J. Forrest et al., "coin-or/Cbc: Version 2.9.9," 2018, doi: 10.5281/zenodo.1317566. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.2720283

Alexander Berndt received the M.Sc. degree in systems and control from the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, in 2020.

He is currently a ML/Software Engineer with Overstory B.V. working on geospatial data intelligence applied to vegetation management. He has authored articles published in ECC, ICAPS, and ISTVS spanning the domains of data-driven control, set-based estimation, and multi-agent robotics and coordination. His research interests include data-driven control and estimation schemes with guarantees, the control of multi-agent systems, and learning-based control.

Niels van Duijkeren received the M.Sc. degree in systems & control in systems and control from the Delft University of Technology, The Netherlands, in 2014, and the Ph.D. degree in mechanical engineering from the Motion Estimation Control and Optimization group, Mechanical Engineering, KU Leuven, Belgium, in 2019.

He is currently a Research Scientist with Robert Bosch GmbH - Corporate Research. During his Ph.D., he focused on methods for time-optimal geometric motion control of robot manipulators and user-friendly efficient numerical solvers. He has co-authored papers in e.g., TAC, CDC, IROS, and RSS on topics spanning model predictive control, optimization methods, and machine learning. His current research interests include motion planning and control of mobile robots, software and methods for optimization-based control and estimation, robust motion planning in dynamic environments, system identification and model learning for adaptive robot control.

Luigi Palmieri (Member, IEEE) received the Ph.D. degree in robot motion planning from the University of Freiburg, Freiburg im Breisgau, Germany, in 2018.

He is a Senior Expert with Robert Bosch GmbH—Corporate Research. During his Ph.D., he was responsible for the motion planning task of the EU FP7 project Spencer. Since then, he has the same responsibility in the EU H2020 project ILIAD. He has co-authored multiple papers at RA-L, ICRA, IROS, FSR on the combinations of motion planning with control, search, machine learning, and human motion prediction. His research interests include kinodynamic motion planning in dynamic and crowded environments, control of non linear dynamic systems, hybrid systems of learning–planning–control, MPC and numerical optimization techniques, planning considering human motion predictions, and social constraints.

Alexander Kleiner received the M.Sc. degree in computer science from the Stafford University, London, U.K., in 2000, the Ph.D. degree in computer science from the University of Freiburg, Freiburg im Breisgau, Germany, and a docent degree (habilitation) from the Linköping University, Linköping, Sweden, in 2008.

He is Chief Expert for navigation and coordination of autonomous systems with Bosch Cooperate Research, Renningen, Germany. From 2017 to 2018 he worked as President for AI and Machine Learning at the startup FaceMap LLC, Malibu, CA, USA, and from 2014 until 2018 as Senior Principal Robotics Scientist with technical lead at iRobot in Pasadena, CA. From 2011 to 2014 he served as Associate Professor with the Linköping University, where he headed the research group on collaborative robotics. He worked as a Postdoctoral Fellow with Carnegie Mellon University, Pittsburgh, PA, USA, and at La Sapienza University, Rome, Italy. His research interests include collaborative robotics, multirobot navigation planning, and machine learning.

Tamás Keviczky (Senior Member, IEEE) received the M.Sc. degree in electrical engineering from the Budapest University of Technology and Economics, Budapest, Hungary, in 2001, and the Ph.D. degree in control science and dynamical systems from the Control Science and Dynamical Systems Center, University of Minnesota, Minneapolis, MN, USA, in 2005.

He was a Postdoctoral Scholar of control and dynamical systems with the California Institute of Technology, Pasadena, CA, USA. He is currently a Professor with Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands. His research interests include distributed optimization and optimal control, model predictive control, embedded optimization-based control and estimation of large-scale systems with applications in aerospace, automotive, mobile robotics, industrial processes, and infrastructure systems, such as water, heat, and power networks.

Dr. Keviczky was the co-recipient of the AACC O. Hugo Schuck Best Paper Award for Practice in 2005. He was an Associate Editor for Automatica from 2011 to 2017 and for IEEE TRANSACTIONS ON AUTOMATIC CONTROL since 2021.