A Feedback Scheme to Reorder a Multi-Agent Execution Schedule by Persistently Optimizing a Switchable Action Dependency Graph

Alexander Berndt[*]   **Niels van Duijkeren**[†]   **Luigi Palmieri**[†]   **Tamás Keviczky**[*]

Abstract

In this paper we consider multiple Automated Guided Vehicles (AGVs) navigating a common workspace to fulfill various intralogistics tasks, typically formulated as the Multi-Agent Path Finding (MAPF) problem. To keep plan execution deadlock-free, one approach is to construct an Action Dependency Graph (ADG) which encodes the ordering of AGVs as they proceed along their routes. Using this method, delayed AGVs occasionally require others to wait for them at intersections, thereby affecting the plan execution efficiency. If the workspace is shared by dynamic obstacles such as humans or third party robots, AGVs can experience large delays. A common mitigation approach is to re-solve the MAPF using the current, delayed AGV positions. However, solving the MAPF is time-consuming, making this approach inefficient, especially for large AGV teams. In this work, we present an online method to repeatedly modify a given acyclic ADG to minimize route completion times of each AGV. Our approach persistently maintains an acyclic ADG, necessary for deadlock-free plan execution. We evaluate the approach by considering simulations with random disturbances on the execution and show faster route completion times compared to the baseline ADG-based execution management approach.

Index terms— *Robust Plan Execution, Scheduling and Coordination, Mixed Integer Programming, Multi-Agent Path Finding, Factory Automation*

1   Introduction

Multiple Automated Guided Vehicles (AGVs) have shown to be capable of efficiently performing intra-logistics tasks such as moving inventory in distribution centers (Wurman, D'Andrea, and Mountz 2008). The coordination of AGVs in shared environments is typically formulated as the
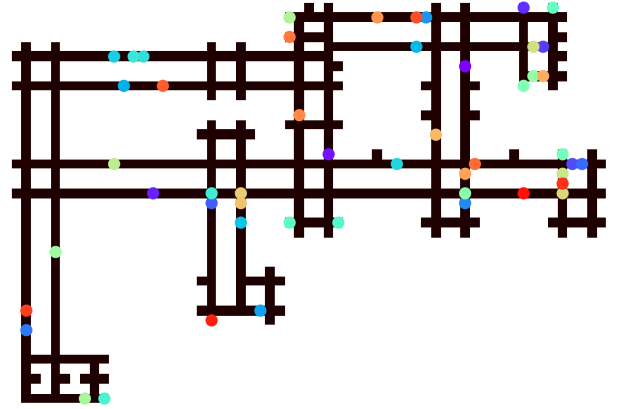

Figure 1: A roadmap occupied by 50 AGVs (represented by colored dots). AGVs must efficiently navigate from a start to a goal position while avoiding collisions with one another, despite being subjected to delays.

Multi-Agent Path Finding (MAPF) problem, which has been shown to be NP-Hard (Yu and LaValle 2012). The problem is to find trajectories for each AGV along a roadmap such that each AGV reaches its goal without colliding with the other AGVs, while minimizing the makespan. The MAPF problem typically considers an abstraction of the workspace to a graph where vertices represent spatial locations and edges pathways connecting two locations.

Recently, solving the MAPF problem has garnered widespread attention (Stern et al. 2019), (Felner et al. 2017). This is mostly due to the abundance of application domains, such as intralogistics, airport taxi scheduling (Morris et al. 2016) and computer games (Ontanón et al. 2013). Solutions to the MAPF problem include Conflict-Based Search (CBS) (Sharon et al. 2015), Prioritized Planning using Safe Interval Path Planning (SIPP) (Yakovlev and Andreychuk 2017), declarative optimization approaches using answer set programming (Bogatarkan, Patoglu, and Erdem 2019), heuristic-guided coordination (Pecora et al. 2018) and graph-flow optimization approaches (Yu and LaValle 2013).

---

[*]Alexander Berndt and Tamás Keviczky are with the Delft Center for Systems and Control (DCSC), TU Delft, 2628 CN Delft, The Netherlands `A.E.Berndt@student.tudelft.nl`, `T.Keviczky@tudelft.nl`.

[†]Niels van Duijkeren and Luigi Palmieri are with Bosch Corporate Research, Robert Bosch GmbH, Renningen, 71272, Germany `{ Niels.vanDuijkeren, Luigi.Palmieri } @de.bosch.com`.

Algorithms such as CBS have been improved by exploiting properties such as geometric symmetry (Li et al. 2019), using purpose-built heuristics (Felner et al. 2018), or adopting a Mixed-Integer Linear Program (MILP) formulation where a branch-cut-and-price solver is used to yield significantly faster solution times (Lam et al. 2019).

Similarly, the development of bounded sub-optimal solvers such as Enhanced Conflict-Based Search (ECBS) (Barer et al. 2014) have further improved planning performance for higher dimensional state spaces. Continuous Conflict-Based Search (CCBS) can be used to determine MAPF plans for more realistic roadmap layouts (Andreychuk et al. 2019). As opposed to CBS, CCBS considers a weighted graph and continuous time intervals to describe collision avoidance constraints, albeit with increased solution times.

The abstraction of the MAPF to a graph search problem means that executing the MAPF plans requires monitoring of the assumptions made during the planning stage to ensure and maintain their validity. This is because irregularities such as vehicle dynamics and unpredictable delays influence plan execution. kR-MAPF addresses this by permitting delays up to a duration of k time-steps (Atzmon et al. 2018). Stochastic AGV delay distributions are considered in (Ma, Kumar, and Koenig 2017), where the MAPF is solved by minimizing the expected overall delay. These robust MAPF formulations and solutions inevitably result in more conservative plans compared to their nominal counterparts.

An Action Dependency Graph (ADG) encodes the ordering between AGVs as well as their kinematic constraints in a post-processing step after solving the MAPF (Hönig et al. 2017). Combined with an execution management approach, this allows AGVs to execute MAPF plans successfully despite kinematic constraints and unforeseen delays. This work was extended to allow for persistent re-planning (Hönig et al. 2019).

The aforementioned plan execution solutions such as (Hönig et al. 2019), (Atzmon et al. 2018), (Ma, Kumar, and Koenig 2017) do not specifically address the effects of significantly large delays. These approaches typically view delays as a lack of synchronization between AGVs, rather than as significantly impacting the route completion time. The result is that plan execution is unnecessarily inefficient when a single AGV is largely delayed and others are on schedule, since AGVs still occasionally need to wait for the delayed AGV before continuing their plans. We observe that to efficiently mitigate the effects of such large delays the plans should be adjusted continuously in an online fashion. The main challenge being to optimize the plan efficiently while maintaining deadlock- and collision-free execution.

In this paper, we present such an online approach capable of reordering AGVs based on any given MAPF solution, thus allowing for efficient MAPF plan execution even in the presence of large delays. This approach is fundamentally different from the aforementioned approaches (Hönig et al. 2017), (Ma, Kumar, and Koenig 2017), (Atzmon et al. 2018), (Hönig et al. 2019) in that delays can be accounted for as they occur, instead of anticipating them *a priori*. The feedback nature of our approach additionally means solving the initial MAPF can be done assuming nominal plan execution, as opposed to solving a robust formulation which necessarily results in longer initial plans.

Our contributions include the introduction of reverse agent dependencies leading to the Switchable Action Dependency Graph (SADG), a mixed-integer linear program formulation to optimally select agent dependencies, and showing that execution management based on this approach guarantees collision- and deadlock-free execution.

Working towards our proposed solution, we formally define the Multi-Agent Path Finding problem and the concept of an Action Dependency Graph (ADG) in Section 2. Based on a modified version of this ADG, we introduce the concept of a reverse agent dependency in Section 3. This will allow an alternative ordering of Automated Guided Vehicles, while maintaining collision avoidance constraints. In Section 4, we formulate the choice of selecting between forward or reverse ADG dependencies as a closed-loop optimization problem. This optimization formulation guarantees that the resulting ADG allows plan execution to be both collision- and deadlock-free, while minimizing the predicted plan completion time. Finally, we compare this approach to the baseline ADG method in Section 5.

2 Preliminaries

Let us now introduce the fundamental concepts on which our approach is based, facilitated by the example shown in Fig. 2. Consider the representation of a workspace as a roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, e.g., as in Fig. 2a. A MAPF solution is defined as in Definition 1.

***Definition* 1** (MAPF Solution). *Consider a roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ occupied by a set of N AGVs where the ith AGV has start $s_i \in \mathcal{V}$ and goal $g_i \in \mathcal{V}$, such that $s_i \neq s_j$ and $g_i \neq g_j \ \forall \ i, j \in \{1, \ldots, N\}, \ i \neq j$, an MAPF solution $\mathcal{P} = \{P_1, \ldots, P_N\}$ is a set of N plans, each defined by a sequence of tuples $p = (l, t)$ consisting of a location $l \in \mathcal{V}$, and a time $t \in [0, \infty)$. $P_i = \{p_i^1, \ldots, p_i^{N_i}\}$ refers to the plan for AGV_i. If every AGV perfectly follows its plan given by the MAPF solution, then all AGVs will reach their respective goals in finite time without collision.*

In Fig. 2a, AGV_1 and AGV_2 have start and goal $s_1 = A$, $g_1 = H$ and $s_2 = E$, $g_2 = D$, respectively. For this example, using CCBS (Andreychuk et al. 2019) yields $\mathcal{P} = \{P_1, P_2\}$ as

$$P_1 = \{(A, 0), (B, 1.0), (C, 2.2), (G, 3.1), (H, 3.9)\},$$

$$P_2 = \{(E, 0), (F, 1.1), (G, 3.9), (C, 4.8), (D, 5.9)\}.$$

Note the implicit ordering in $\mathcal{P}$, stating AGV_1 traverses $C - G$ before AGV_2.

Given a plan tuple $p = (l, t)$, define the operators $l = loc(p)$ and $t = \hat{t}(p)$ which return the location $l \in \mathcal{V}$ and *planned* time of plan tuple p respectively. Let $S(p) \mapsto S \subset \mathbb{R}^2$ be an operator which maps $l = loc(p)$ to a spatial region in the physical workspace in $\mathbb{R}^2$. Finally, let $S_{\text{AGV}} \subset \mathbb{R}^2$ refer to the physical area occupied by an AGV.

2.1 Modified Action Dependency Graph

Based on a MAPF solution $\mathcal{P}$, we can construct a modified version of the original Action Dependency Graph (ADG), formally defined in Definition 2. This modified ADG encodes the sequencing of AGV movements to ensure the plans are executed as originally planned despite delays.

Definition 2 (Action Dependency Graph). *An ADG is a directed graph $\mathcal{G}_{ADG} = (\mathcal{V}_{ADG}, \mathcal{E}_{ADG})$ where the vertices represent events of an AGV traversing the roadmap $\mathcal{G}$. A vertex $v_i^k = (\{p_1, \ldots, p_q\}, status) \in \mathcal{V}_{ADG}$ denotes the event of AGV_i moving from $loc(p_1)$, via intermediate locations, to $loc(p_q)$, where $q \geq 2$ denotes the number of consecutive plan tuples encoded within v_i^k. Finally, status $\in \{staged, in\text{-}progress, completed\}$.*

Initially, the *status* of v_i^k are *staged* $\forall\ i, k$. The directed edges in an ADG, from here on referred to as dependencies, define event-based constraints between two vertices. Formally, (v_i^k, v_j^l) implies that v_j^l cannot be *in-progress* or *completed* until $v_i^k = completed$. A dependency $(v_i^k, v_j^l) \in \mathcal{E}_{ADG}$ is classified as *Type 1* if $i = j$ and *Type 2* if $i \neq j$.

An ADG can be constructed from a plan $\mathcal{P}$ using Algorithm 1. Fig. 2b shows an example of an ADG plan being executed, based on the AGV positions in Fig. 2a. Observe how AGV_2 cannot start v_2^2 before v_1^4 has been completed.

As discussed in (Hönig et al. 2019), there exist valid MAPF solutions for which the resulting ADG is cyclic. It is possible to modify MAPF planning algorithms to avoid obtaining such plans, which require precise synchronous execution. We therefore introduce the following assumption.

Assumption 1 (Acyclic ADG). *The MAPF solution as defined in Definition 1 is such that the ADG constructed by Algorithm 1 is acyclic.*

Let us introduce notation facilitating the differentiation between *planned* and *actual* ADG vertex completion times. Let $\hat{t}_s(v_i^k)$ and $\hat{t}_g(v_i^k)$ denote the *planned* time that event $v_i^k \in \mathcal{V}_{ADG}$ is expected start and be completed respectively. This *planned* time refers to times specified in a MAPF solution. Due to delays, the *planned* and *actual* ADG vertex times may differ. We therefore introduce $t_s(v_i^k)$ and $t_g(v_i^k)$ which denote the *actual* start and completion times of event $v_i^k \in \mathcal{V}_{ADG}$ respectively. With regards to the status of ADG vertex v_i^k, $\hat{t}_s(v_i^k)$ and $t_s(v_i^k)$ refer to when the status of v_i^k changes from *staged* to *in-progress*, and $\hat{t}_g(v_i^k)$, $t_g(v_i^k)$ refer to changes from *in-progress* to *completed*. Note that if the MAPF solution is executed nominally, i.e. AGVs experience no delays, $t_s(v_i^k) = \hat{t}_s(v_i^k)$ and $t_g(v_i^k) = \hat{t}_g(v_i^k)$ for all $v \in \mathcal{V}_{ADG}$.

Unlike the originally proposed ADG algorithm, Algorithm 1 ensures subsequent plan tuples which are not spatially exclusive are contained within a single ADG vertex, cf. line 8 of the algorithm. This property will prove to be useful with the introduction of reverse dependencies in Section 3.1. Referring to Algorithm 1, we introduce $plan(v_i^k)$ which returns the sequence of plan tuples $\{p_1, \ldots, p_q\}$ for $v_i^k \in \mathcal{V}_{ADG}$. Let the operators $s(v_i^k)$ and $g(v_i^k)$ return the start and goal vertices $loc(p_1)$ and $loc(p_q)$ of vertex v_i^k respectively. Finally, $\oplus$ denotes the Minkowski sum.
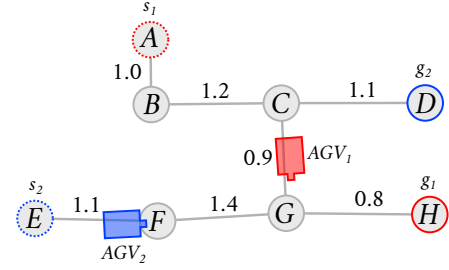
(a) A roadmap graph occupied by two AGVs with start s_i and goal g_i for $i = \{1, 2\}$. The start and goal vertices are highlighted with dotted and solid colored circle outlines respectively. The edge weights indicate the expected traversal times.
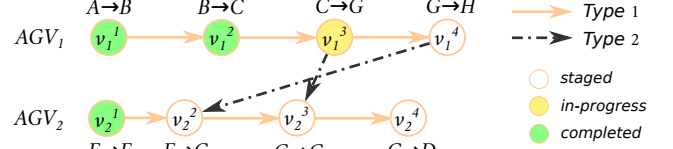
(b) Illustration of the ADG where each vertex status is color coded. It reflects the momentary progress of the AGVs in Fig. 2a.

Figure 2: Illustrative MAPF problem example alongside the constructed Action Dependency Graph

---

Algorithm 1 Modified ADG construction based on (Hönig et al. 2019)

Input: MAPF solution $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$
Result: $\mathcal{G}_{ADG}$

    // Add ADG vertices and *Type 1* dependencies
1: **for** $i = 1$ to N **do**
2:     $p \leftarrow p_i^1$
3:     $v \leftarrow (\{p\}, staged)$
4:     $v_{prev} \leftarrow None$
5:     $k = 2$
6:     **for** $k = 2$ to N_i **do**
7:         Append p_i^k to $plan(v)$
8:         **if** $S(p) \oplus S_{AGV} \cap S(p_i^k) \oplus S_{AGV} = \emptyset$ **then**
9:         Add v to $\mathcal{V}_{ADG}$
10:         **if** v_{prev} not *None* **then**
11:         Add edge (v_{prev}, v) to $\mathcal{E}_{ADG}$
12:         $v_{prev} \leftarrow v$
13:         $p \leftarrow p_i^k$
14:         $v \leftarrow (\{p\}, staged)$

    // Add *Type 2* dependencies
15: **for** $i = 1$ to N **do**
16:     **for** $k = 1$ to N_i **do**
17:         **for** $j = 1$ to N **do**
18:         **if** $i \neq j$ **then**
19:           **for** $l = 1$ to N_j **do**
20:           **if** $s(v_i^k) = g(v_j^l)$ **and** $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$ **then**
21:           Add edge (v_i^k, v_j^l) to $\mathcal{E}_{ADG}$

22: **return** $\mathcal{G}_{ADG}$

Despite these modifications, Algorithm 1 maintains the original algorithm's time complexity of $\mathcal{O}(N^2\bar{n}^2)$ where $\bar{n} = \max_i N_i$.

Furthermore, let us introduce an important property of an ADG-managed plan-execution scheme, Proposition 1, concerning guarantees of successful plan execution.

***Proposition* 1** (Collision- and deadlock-free ADG plan execution)**.** *Consider an ADG, $\mathcal{G}_{ADG}$, constructed from a MAPF solution as defined in Definition 1 using Algorithm 1, satisfying Assumption 1. If the AGV plan execution adheres to the dependencies in $\mathcal{G}_{ADG}$, then, assuming the AGVs are subjected to a finite number of delays of finite duration, the plan execution will be collision-free and completed in finite time.*

Proof 1*: Proof by induction. Consider that AGV_i and AGV_j traverse a common vertex $\bar{p} \in \mathcal{G}$ along their plans $\mathcal{P}_i$ and $\mathcal{P}_j$, for any $i, j \in \{1, \ldots, N\}, i \neq j$. By lines 1-14 of Algorithm 1, this implies $g(v_i^k) = s(v_j^l) = \bar{p}$ for some $v_i^k, v_j^l \in \mathcal{V}_{ADG}$. By lines 15-21 of Algorithm 1, common vertices of $\mathcal{P}_i$ and $\mathcal{P}_j$ in $\mathcal{G}$ will result in a Type 2 dependency (v_j^l, v_i^k) if $p = s(v_j^l) = g(v_i^k)$ and $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$. For the base step: initially, all ADG dependencies have been adhered to since v_i^1 is staged $\forall~i \in \{1, \ldots, N\}$. For the inductive step: assuming vertices up until v_i^{k-1} and v_j^{l-1} have been completed in accordance with all ADG dependencies, it is sufficient to ensure AGV_i and AGV_j will not collide at $\bar{p}$ while completing v_i^k and v_j^l respectively, by ensuring $t_s(v_i^k) > t_g(v_j^l)$. By line 20 of Algorithm 1 the Type 2 dependency (v_i^k, v_j^l) guarantees $t_s(v_i^k) > t_g(v_j^l)$. Since, by Assumption 1, the ADG is acyclic, at least one vertex of the ADG can be in-progress at all times. By the finite nominal execution time of the MAPF solution in Definition 1, despite a finite number of delays of finite duration, finite-time plan completion is established. This completes the proof.* $\square$

3 Switching Dependencies in the Action Dependency Graph

We now introduce the concept of a reversed ADG dependency. In the ADG, *Type 2* dependencies essentially encode an ordering constraint for AGVs visiting a vertex in $\mathcal{G}$. The idea is to switch this ordering to minimize the effect an unforeseen delay has on the task completion time of each AGV.

3.1 Reverse *Type 2* Dependencies

We now introduce the notion of a reverse *Type 2* dependency. Definition 3 states that a dependency and its reverse contain the same collision avoidance constraints, but with a reversed AGV ordering. Lemma 1 can be used to obtain a dependency which conforms to Definition 3. Lemma 1 is illustrated graphically in Fig. 3.

***Definition* 3** (Reverse *Type 2* dependency)**.** *Consider a Type 2 dependency $d = (v_i^k, v_j^l)$. d requires $t_s(v_j^l) \geq t_g(v_i^k)$. A reverse dependency of d is a dependency d' that ensures $t_s(v_i^k) \geq t_g(v_j^l)$.*
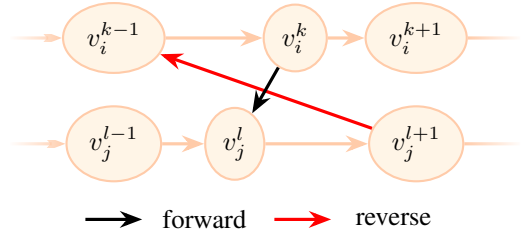

Figure 3: A subset of an ADG with a dependency (black) and its reverse (red)

***Lemma* 1** (Reversed Type 2 dependency)**.** *Let $v_i^k, v_j^l, v_j^{l+1}, v_i^{k-1} \in \mathcal{V}_{ADG}$. Then $d' = (v_j^{l+1}, v_i^{k-1})$ is the reverse dependency of $d = (v_i^k, v_j^l)$.*

Proof 2*: The dependency $d = (v_i^k, v_j^l)$ encodes the constraint $t_s(v_j^l) \geq t_g(v_i^k)$. The reverse 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 3.* $\square$

The modified ADG ensures that reverse dependencies maintain sufficient collision avoidance constraints since adjacent vertices in $\mathcal{V}_{ADG}$ refer to spatially different locations, cf. line 8 in Algorithm 1.

3.2 Switchable Action Dependency Graph

Having introduced reverse *Type 2* dependencies, it is necessary to formalize the manner in which we can select dependencies to obtain a resultant ADG. A cyclic ADG implies that two events are mutually dependent on each other, implying a deadlock. To ensure deadlock-free plan execution, it is sufficient to ensure the selected dependencies result in an acyclic ADG. Additionally, to maintain the collision-avoidance guarantees implied by the original ADG, it is sufficient to select at least one of the forward or reverse dependencies of each forward-reverse dependency pair in the resultant ADG. Since selecting both a forward and reverse dependency always results in a cycle within the ADG, we therefore must either select between the forward or the reverse dependency. To this end, we formally define a Switchable Action Dependency Graph (SADG) in Definition 4 which can be used to obtain the resultant ADG given a selection of forward or reverse dependencies.

***Definition* 4** (Switchable Action Dependency Graph)**.** *Let an ADG as in Definition 2 contain m_T forward-reverse dependency pairs determined using Definition 3. A Switchable Action Dependency Graph (SADG) is a mapping $SADG(\boldsymbol{b}) : \{0, 1\}^{m_T} \mapsto \mathcal{G}_{ADG}$ which outputs the resultant ADG based on the selected 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, $m \in \{1, \ldots, m_T\}$.*

***Corollary* 1** (SADG plan execution)**.** *Consider an SADG, $SADG(\boldsymbol{b})$, as in Definition 4. If $\boldsymbol{b}$ is chosen such that $\mathcal{G}_{ADG}$*

$= SADG(\bm{b})$ *is acyclic, and no dependencies in* $\mathcal{G}_{ADG}$ *point from vertices that are staged or in-progress to vertices that are completed,* $\mathcal{G}_{ADG}$ *will guarantee collision- and deadlock-free plan execution.*

Proof 3: By definition, any $\bm{b}$ *will guarantee collision-free plans, since at least one dependency of each forward-reverse dependency pair is selected, by Proposition 1. If* $\bm{b}$ *ensures* $ADG = SADG(\bm{b})$ *is acyclic, and the resultant ADG has no dependencies pointing from vertices that are staged or in-progress to vertices that are completed, the dependencies within the ADG are not mutually constraining, guaranteeing deadlock-free plan execution.*

The challenge is finding $\bm{b}$ which ensures $ADG_{\bm{b}}$ is acyclic, while simultaneously minimizing the cumulative AGV route completion times. This is formulated as an optimization problem in Section 4.

4 Optimization-Based Approach

Having introduced the SADG, we now formulate an optimization problem which can be used to determine $\bm{b}$ such that the resultant ADG is acyclic, while minimizing cumulative AGV route completion times. The result is a Mixed-Integer Linear Program (MILP) which we solve in a closed-loop feedback scheme, since the optimization problem updates the AGV ordering at each iteration based on the delays measured at that time-step.

4.1 Translating a Switchable Action Dependency Graph to Temporal Constraints

Regular ADG Constraints   Let us introduce the optimization variable $t_{i,s}^k$ which, once a solution to the optimization problem is determined, will be equal to $t_s(v_i^k)$. The same relation applies to the optimization variable $t_{i,g}^k$ and $t_g(v_i^k)$. The event-based constraints within the SADG can be used in conjunction with a predicted duration of each event to determine when each AGV is expected to complete its plan. Let $\tau(v_i^k)$ be the expected time it will take AGV_i to complete event $v_i^k \in \mathcal{V}_{ADG}$ based solely on dynamical constraints, route distance and assuming the AGV is not blocked. For example, we could let τ equal the edge distance divided by the expected nominal AGV velocity. We can now specify the temporal constraints corresponding to the *Type 1* dependencies of the plan of AGV_i as

$$
\begin{aligned}
t_{i,g}^1 &\geq t_{i,s}^1 + \tau(v_i^1), \\
t_{i,s}^2 &\geq t_{i,g}^1, \\
t_{i,g}^2 &\geq t_{i,s}^2 + \tau(v_i^2), \\
t_{i,s}^3 &\geq t_{i,g}^2, \\
&\vdots \qquad\qquad \vdots \\
t_{i,s}^{N_i} &\geq t_{i,g}^{N_i-1}, \\
t_{i,g}^{N_i} &\geq t_{i,s}^{N_i} + \tau(v_i^{N_i}).
\end{aligned}
\tag{1}
$$

Consider a *Type 2* dependency (v_i^k, v_j^l) within the ADG. This can be represented by the temporal constraint

$$
t_{j,s}^l > t_{i,g}^k,
\tag{2}
$$

where the strict inequality is required to guarantee that AGV_i and AGV_j never occupy the same spatial region.

Adding Switchable Dependency Constraints   We now introduce the temporal constraints which represent the selection of forward or reverse dependencies in the SADG. Initially, consider the set $\mathcal{E}_{ADG}^{\text{Type 2}} = \{e \in \mathcal{E}_{ADG} | e \text{ is Type 2}\}$ which represents the sets of all *Type 2* dependencies. The aim here is to determine a set $\mathcal{E}_{ADG}^{\text{switchable}} \subset \mathcal{E}_{ADG}^{\text{Type 2}}$ containing the dependencies which could potentially be switched and form part of the MILP decision space, as required by Corollary 1. $\mathcal{E}_{ADG}^{\text{switchable}}$ is determined by considering the dependencies in $\mathcal{E}_{ADG}^{\text{Type 2}}$ which match the following criteria:

1. Where the tail points to a *staged* vertex,

2. Where the head of the original and reverse dependency point to a *staged* vertex.

An illustrative example of which dependencies to consider in the MILP is shown in Fig. 4. Having determined $\mathcal{E}_{ADG}^{\text{switchable}}$, the next step is to include the switched dependencies as temporal constraints. Directly referring to Section 3.2, we assume m_T forward-reverse dependency pairs in $\mathcal{E}_{ADG}^{\text{switchable}}$, where the Boolean b_m is used to select the forward or reverse dependency of the mth forward-reverse dependency pair, $m \in \{1, \ldots, m_T\}$. These temporal constraints can be written as

$$
\begin{aligned}
t_{j,s}^l &> t_{i,g}^k - b_m M, \\
t_{i,s}^{k-1} &> t_{j,g}^{l+1} - (1 - b_m) M,
\end{aligned}
\tag{3}
$$

where M is a large, positive constant such that $M > \max_i t_i^{N_i}$. Note that $\max_i t_i^{N_i}$ can be approximated by estimating the maximum anticipated delays experienced by the AGVs. In practice, however, finding such an upper bound on delays is not evident, meaning we choose M to be a conservatively high value.

4.2 Optimization Problem Formulation

We have shown that an SADG is represented by the temporal constraints in Eq. (1) through Eq. (3) for $i \in \{1, \ldots, N\}$, $m \in \{1, \ldots, m_T\}$. Minimizing the cumulative route completion time of all AGVs is formulated as the following optimization problem

$$
\begin{aligned}
\min_{\bm{b}, \, \bm{t}_s, \, \bm{t}_g} \quad & \sum_{i=1}^{N} t_i^{N_i} \\
\text{s.t. } & Eq.~(1) \; \forall \, i = \{1, \ldots, N\}, \\
& Eq.~(2) \; \forall \, e \in \mathcal{E}_{ADG}^{\text{Type 2}} \setminus \mathcal{E}_{ADG}^{\text{switchable}}, \\
& Eq.~(3) \; \forall \, e \in \mathcal{E}_{ADG}^{\text{switchable}},
\end{aligned}
\tag{4}
$$

where $\bm{b} : \{0,1\}^{m_T}$ is a vector containing all the binary variables b_m and the vectors $\bm{t}_s$ and $\bm{t}_g$ contain all the variables $t_{i,s}^k$ and $t_{i,g}^k$ respectively $\forall \; k \in \{1, \ldots, N_i\}, i \in \{1, \ldots, N\}$.
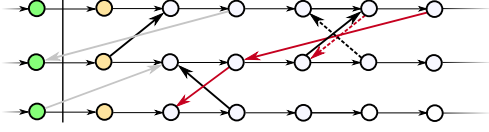
Figure 4: Illustrative example of dependencies considered in the optimization problem. Considered dependencies are black (forward) and red (reverse). Grey dependencies are ignored.

4.3 Solving the MILP in a Feedback Loop

The aforementioned optimization formulation can be solved based on the current AGV positions in a feedback loop. The result is a continuously updated $\mathcal{G}_{\text{ADG}}$ which guarantees minimal path completion time based on current AGV delays. This feedback strategy is defined in Algorithm 2.

An important aspect to optimal feedback control strategies is that of recursive feasibility, which means that the optimization problem will remain feasible as long as the control law is applied. The control strategy outlined in Algorithm 2 is guaranteed to remain recursively feasible, as formally shown in Proposition 2.

Algorithm 2 Switching ADG Feedback Scheme

1: Get goals and locations
2: Solve MAPF to obtain $\mathcal{P}$
3: Construct ADG using Algorithm 1
4: Determine $SADG()$

5: **while** Plans not done **do**
6:     get current position along plans for each robot
7:     determine switchable dependencies
8:     $\boldsymbol{b} \leftarrow$ MILP in Eq. (4)
9:     $ADG \leftarrow SADG(\boldsymbol{b})$

Proposition 2 (Recursive Feasibility). *Consider an ADG, as defined in Definition 2, which is acyclic at time $t = 0$. Consecutively applying the MILP solution from Eq. (4) is guaranteed to ensure the resultant ADG remains acyclic for all $t > 0$.*

Proof 4: Proof by induction. Consider an acyclic ADG as defined in Definition 2, at a time t. The MILP in Eq. (4) always has the feasible solution $\boldsymbol{b} = 0$ if the initial ADG (from which the MILP's constraints in Eq. (1) through Eq. (3) are defined) is acyclic. Any improved solution of the MILP with $\boldsymbol{b} \neq 0$ is necessarily feasible, implying a resultant acyclic ADG. This implies that the MILP is guaranteed to return a feasible solution, the resultant ADG will always be acyclic if the ADG before the MILP was solved, was acyclic. Since the ADG at $t = 0$ is acyclic (a direct result of a MAPF solution), it will remain acyclic for $t > 0$. □

4.4 Decreasing Computational Effort

The time required to solve the MILP will directly affect the real-time applicability of this approach. In general, the complexity of the MILP increases expenentially in the number
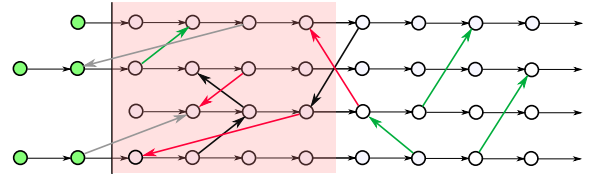

Figure 5: Dependency selection for a horizon of 4 vertices. Switchable dependency pairs are shown in black (forward) and red (reverse). Regular dependencies considered in the MILP are green. Dependencies not considered are grey.
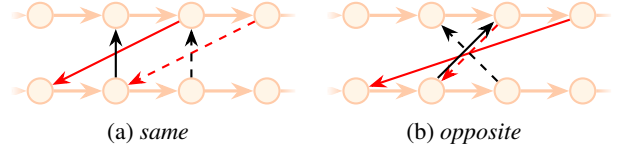

(a) *same*   (b) *opposite*

Figure 6: Dependency groups. Each dependency is either original (black) or reversed (red). Reverse and forward dependency pairings are differentiated by line styles.

of binary variables. To render the MILP less computationally demanding, it is therefore most effective to decrease the number of binary variables. We present two complementary methods to achieve this goal.

Switching Dependencies in a Receding Horizon The dependencies to switch will be selected in a receding horizon fashion, whereas the temporal dependencies are still considered for the entire plan length. Formally, this means that the set $\mathcal{E}_{\text{ADG}}^{\text{switchable}}$ only contains dependencies pointing to within H future vertices. An illustration of this selection is shown in Fig. 5.

Dependency Grouping We observed that multiple dependencies would often form patterns, two of which are shown in Fig. 6. These patterns are referred to as *same-direction* and *opposite-direction* dependency groups, shown in Fig. 6a and Fig. 6b respectively. These groups share the same property that the resultant ADG is acyclic if and only if either all the forward or all the reverse dependencies are active. This means that a single binary variable is sufficient to describe the switching of all the dependencies within the group, decreasing the variable space of the MILP in Eq. (4). Once such a dependency group has been identified, the temporal constraints can then be defined as

$$
\begin{aligned}
t_{j,s}^l > t_{i,g}^k - b_{DG}M & \quad \forall (v_i^k, v_j^l) \in \mathcal{DG}_{\text{fwd}}, \\
t_{j,s}^l > t_{i,g}^k - (1 - b_{DG})M & \quad \forall (v_i^k, v_j^l) \in \mathcal{DG}_{\text{rev}},
\end{aligned} \tag{5}
$$

where $\mathcal{DG}_{\text{fwd}}$ and $\mathcal{DG}_{\text{rev}}$ refer to the forward and reverse dependencies of a particular grouping respectively, and b_{DG} is a binary variable which switches all the forward or reverse dependencies in the entire group simultaneously.

5 Evaluation

We design a set of simulations to evaluate the approach in terms of re-ordering efficiency when AGVs are subjected to

delays while following their initially planned paths. We use the method presented by Hönig *et al.* (Hönig et al. 2019) as a comparison baseline. All simulations were conducted on a Lenovo Thinkstation with an Intel® Xeon E5-1620 3.5GHz processor and 64 GB of RAM.

5.1 Simulation Setup

The simulations consider a roadmap as shown in Fig. 1. A team of AGVs of size $\{30, 40, 50, 60, 70\}$ are each initialized with a random start and goal position. ECBS (Barer et al. 2014) is then used with sub-optimality factor $w = 1.6$ to solve the MAPF. We consider delays of duration $k = \{1, 3, 5, 10, 15, 20, 25\}$ time-steps. At each kth time-step, a random subset (20%) of the AGVs are stopped for a length of k. The MILP in Eq. (4) is solved at each time-step. We evaluate our approach using a Monte Carlo method: for each AGV team size and delay duration configuration, we consider 100 different randomly selected goal/start positions. The receding horizon dependency selection and dependency groups are used as described in Section 4.4.

5.2 Performance Metric and Comparison

Performance is measured by considering the cumulative plan completion time of all the AGVs. This is compared to the same metric using the original ADG approach with no switching as in (Hönig et al. 2019). The *improvement* is defined as

$$\text{improvement} = \frac{\sum t_{\text{baseline}} - \sum t_{\text{switching}}}{\sum t_{\text{baseline}}} \cdot 100\%,$$

where $\sum t_*$ refers to the cumulative plan completion time for all AGVs. The baseline is equivalent to forcing the solution of the MILP in Eq. (4) to $\boldsymbol{b} = \boldsymbol{0}$ at every time-step. Note that we consider cumulative plan completion time instead of the make-span because we want to ensure each AGV completes its plans as soon as possible, such that it can be assigned a new task.

Another important consideration is the time it takes to solve the MILP in Eq. (4) at each time-step. For our simulations, the MILP was solved using the academically orientated Coin-Or Branch-and-Cut (CBC) solver (Forrest et al. 2018). However, based on preliminary tests, we did note better performance using the commercial solver Gurobi (Gurobi Optimization, LLC 2020). This yielded computational time improvements by a factor 1.1 up to 20.

5.3 Results and Discussion

To showcase the efficacy of our approach, we first determine the average improvement of 100 random scenarios using the minimum switching horizon length of 1 for different AGV team sizes and delay lengths, shown in Fig. 7. The average improvement is highly correlated to the delay duration experienced by the AGVs.

Considering Fig. 7, it is worth noting how the graph layout and AGV-to-roadmap density affects the results: the AGV group size of 40 shows the best average improvement for a given delay duration. This leads the authors to believe there is an optimal AGV group size for a given roadmap, which ensures the workspace is both:
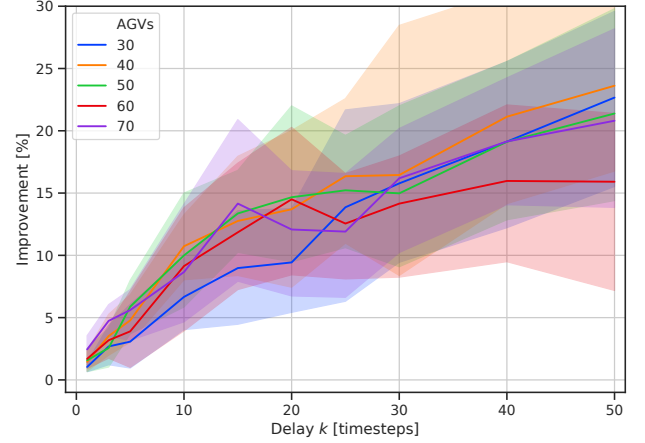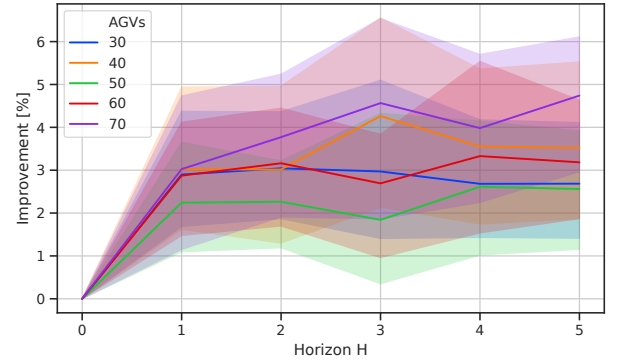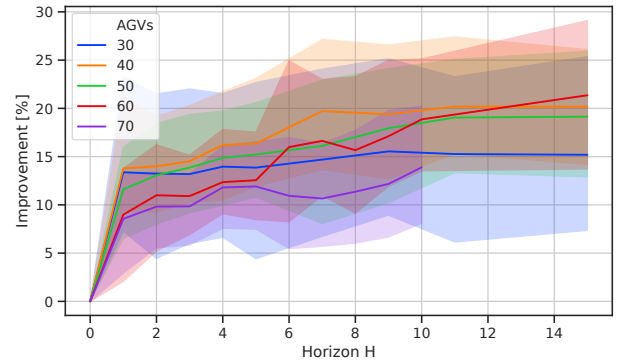

Figure 7: Average improvement of 100 scenarios for various delay lengths and AGV group sizes. Each scenario refers to different randomly generated starts/goals and a randomly selected subset of delayed AGVs. Solid lines depict the average, lighter regions encapsulate the min-max values.

(a) Delay $k = 3$.

(b) Delay $k = 25$.

Figure 8: Average improvement of 100 random start/goal positions and delayed AGV subset, for different switching horizon lengths, for different AGV group sizes. Solid lines depict the average, lighter regions encapsulate the min-max values.
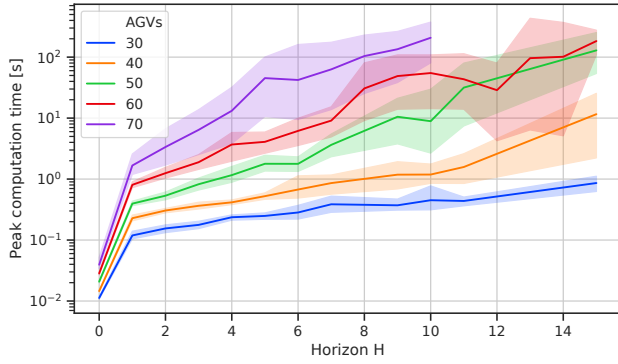
Figure 9: The peak computation to solve the optimization problem for different AGV team sizes and considered dependency horizon lengths. This plot considers the average improvement for delays $k = \{1, 3, 5, 10, 15, 20, 25\}$. Solid lines depict the average, lighter regions encapsulate the min-max values.

1. Not too congested to make switching of dependencies impossible due to the high density of AGVs occupying the map.

2. Not too sparse such that switching is never needed since AGVs are distant from each other, meaning that switching rarely improves task completion time.

Considering the switching dependency horizon, Fig. 8 shows the average improvement for 100 random start/goal positions and delayed AGV subset selection. We observe that a horizon length of 1 already significantly improves performance, and larger horizons seem to gradually increase performance for larger AGV teams.

Fig. 9 shows the peak computation time for various horizon lengths and AGV team sizes. As expected, the computation time is exponential with horizon size and AGV team size.

Two additional observations that were made are

1. *High variability in results.* Note the high variability in improvement indicated by the large lighter regions in Fig. 7. This means that for different random start/goal and delay configurations, the improvement varied significantly. This is due to the fact that each start/goal combination provides differing degrees-of-freedom from an ADG switching perspective.

2. *Occasional worse performance.* Occasionally, albeit rarely, our approach would yield a negative improvement for a particular random start/goal configuration. This was typically observed for small delay durations. The reason is that the optimization problem solves the switching assuming no future delays. However, it may so happen that the AGV which was allowed ahead of another, is delayed in the near future, additionally delaying the AGV it surpassed. We believe a robust optimization approach could potentially resolve this.

Finally, we emphasize that our proposed approach applies seamlessly to:

1. Directional road-maps (since ADG switching retains the direction of the original MAPF plan),

2. Any (weighted) graph layout, such as the MAPF formulation addressed by CCBS, as long as the MAPF can be formulated and solved for this configuration;

3. Persistent plans such as for Multi-Agent Pickup and Delivery (MAPD) (Ma et al. 2017), as long as the full plans are of finite length, since the optimization formulation in Eq. (4) must consider the final plan completion time.

6 Conclusions and Future Work

In this paper, we introduced a novel method which, given a MAPF solution, can be used to switch the ordering of AGVs in an online fashion based on currently measured AGV delays. This switching was formulated as an optimization problem as part of a feedback control scheme, while maintaining the deadlock- and collision-free guarantees of the original MAPF plan. Results show that our approach clearly improves the cumulative task completion time of the AGVs when a subset of AGVs are subjected to delays.

In future work, we plan to consider a receding horizon optimization approach. In this work, ADG dependencies can be switched in a receding horizon fashion, but the plans still need to be of finite length for the control problem to be formulated. For truly persistent plans (theoretically infinite length plans), it is necessary to come up with a receding horizon optimization formulation to apply the method proposed in this paper.

Another possible extension is complementing our approach with a local re-planning method. This is because our approach maintains the originally planned trajectories of the AGVs. However, we observed that under large delays, the originally planned routes can become largely inefficient due to the fact the AGVs are in entirely different locations along their planned path. This could potentially be addressed by introducing local re-planning of trajectories.

To avoid the occasional worse performance, we suggest a robust optimization approach to avoid switching dependencies which could have a negative impact on the plan execution given expected future delays.

Finally, to further validate this approach, it is desirable to move towards system-level tests in more realistic environments, such as by simulations in e.g., Gazebo, or to perform experiments on a real-world intralogistics setup.

References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-agent pathfinding with continuous time. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 39–45. International Joint Conferences on Artificial Intelligence Organization.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Bartak, R.; and Zhou, N. 2018. Robust multi-agent path finding. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2–9.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 961–962.

Bogatarkan, A.; Patoglu, V.; and Erdem, E. 2019. A declarative method for dynamic multi-agent path finding. In Calvanese, D., and Iocchi, L., eds., *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence*, volume 65 of *EPiC Series in Computing*, 54–67. EasyChair.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In Fukunaga, A., and Kishimoto, A., eds., *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA*, 29–37. AAAI Press.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 83–87.

Forrest, J.; Ralphs, T.; Vigerske, S.; LouHafer; Kristjansson, B.; jpfasano; EdwinStraver; Lubin, M.; Santos, H. G.; rlougee; and Saltzman, M. 2018. coin-or/cbc: Version 2.9.9.

Gurobi Optimization, LLC. 2020. Gurobi optimizer reference manual.

Hönig, W.; Kumar, S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2017. Summary: Multi-agent path finding with kinematic constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 4869–4873.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J.; and Ayanian, N. 2019. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters* 4(2):1125–1131.

Lam, E.; Bodic, P. L.; Harabor, D.; and Stuckey, P. 2019. Branch-and-cut-and-price for multi-agent pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, (in print).

Li, J.; Harabor, D.; Stuckey, P.; Ma, H.; and Koenig, S. 2019. Symmetry-breaking constraints for grid-based multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, (in print).

Ma, H.; Li, J.; Kumar, S.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 837–845.

Ma, H.; Kumar, S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3605–3612.

Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, scheduling and monitoring for airport surface operations. In *AAAI Workshop: Planning for Hybrid Systems*.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4):293–311.

Pecora, F.; Andreasson, H.; Mansouri, M.; and Petkov, V. 2018. A loosely-coupled approach for multi-robot coordination, motion planning and control. In *Twenty-eighth international conference on automated planning and scheduling*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40 – 66.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Barták, R. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. *CoRR* abs/1906.08291.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. volume 29 of *AI Magazine*, 9 – 19. Menlo Park, CA: AAAI.

Yakovlev, K. S., and Andreychuk, A. 2017. Any-angle pathfinding for multiple agents based on SIPP algorithm. *CoRR* abs/1703.04159.

Yu, J., and LaValle, S. M. 2012. Multi-agent path planning and network flow. *CoRR* abs/1204.5717.

Yu, J., and LaValle, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, 3612–3617.