Planning and Control of Multiple Mobile Robots for Intralogistics

an optimization-based reordering strategy

Alexander Berndt

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Planning and Control of Multiple Mobile Robots for Intralogistics

an optimization-based reordering strategy

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Alexander Berndt

June 22, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Robert Bosch GmbH. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Planning and Control of Multiple Mobile Robots for Intralogistics

by

Alexander Berndt

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: June 22, 2020

Supervisor(s):

Prof.dr.ir. Tamás Keviczky

dr.ir. Niels van Duijkeren

Reader(s):

dr.ir. Riccardo Ferrari

drs.ir. Gabriel de Albuquerque Gleizer

Abstract

In this thesis we consider multiple Automated Guided Vehicles (AGVs) navigating a common workspace to fulfill 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, due to its inherent complexity, solving the MAPF is time-consuming, making this approach inefficient, especially for large AGV teams.

To address this challenge, we present a novel concept called a Switchable Action Dependency Graph (SADG) which is used as the basis for a shrinking and receding horizon control scheme to repeatedly modify an acyclic ADG to minimize route completion times of each AGV using an optimization based approach. Our control strategies persistently maintain an acyclic ADG, necessary for deadlock-free plan execution.

The proposed control strategies are evaluated in a simulation environment and show a reduction in route completion times when a fleet of AGVs is subjected to random delays. Finally, the methods are also implemented using ROS and validated in the Gazebo simulation environment to illustrate practical feasibility when applied to real systems.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my supervisor dr. Tamás Keviczky for his guidance and support throughout this masters thesis, as well as offering me the opportunity to do my thesis with Robert Bosch GmbH. Thank you for your guidance and always providing input when I needed it, but simultaneously allowing me the freedom to also make decisions myself, permitting me to make mistakes and learn in the process.

I am also grateful to the CR/AAS group at Robert Bosch GmbH Zentrum für Forschung und Vorausentwicklung in Renningen, Germany, where this thesis work was carried out. In particular, thank you to dr. Niels van Duijkeren for our constructive weekly meetings, enthusiastic guidance, frequent discussions and meticulous critique which was a key factor in producing this final work. You taught me a great deal by the way you tackled problems and always posed questions to improve your understanding of things. Also, thank you for your persistence in making this thesis at CR/AAS a possibility, despite all the hurdles we faced along the way.

Additionally, I would like to thank Dr. Alexander Kleiner for welcoming me into the CR/AAS team. I am also grateful to dr. Luigi Palmieri, dr. Ralph Lange for their general guidance and fruitful discussions on mobile robotics and planning.

Thank you to the colleagues at CR/AAS for providing a fun and relaxed yet hard-working environment filled with numerous interesting conversations on a wide variety of topics. Thanks to the doctoral students, master's students and interns who made the coffee-breaks all the more fun. Also, thank you to all my friends back in Delft, especially Daniel Freyr Hjartarson, Ilario Azzolini, Wilhelm van der Kamp and Matthijs Rebel.

Last but not least, thank you to my Mom and Dad for their unending support and belief in me over the past years, even if I didn't always believe in myself.

Delft, University of Technology
June 22, 2020

Alexander Berndt

"Beim Menschen ist es wie beim Velo. Nur wenn er faehrt, kann er bequem die Balance halten."

— *Albert Einstein*

Chapter 1

Introduction

Continuous advancements in the fields of robotics, algorithm development and decreasing hardware costs are opening up possibilities in automation for a large number of application areas. Specifically, consider the logistical challenges of organizing and displacing inventory in large warehouses, commonly referred to as *intralogistics*. Traditionally, intralogistics tasks have predominantly been performed by humans and/or highly supervised human-operated machinery. However, the logistics and warehousing industries have seen a large increase in automation, as shown by a recent study by Elms *et al.* [5]. Two examples of automated distribution centers are shown in Figure 1-1. The study goes on to predict an exponential increase in the potential use of Automated Guided Vehicles (AGVs) to perform such intralogistics tasks and subsequently emphasizes the increasing demand for solutions to efficiently coordinate AGVs in warehouses and distribution centers.

An AGV is a general description given to a mobile robotic platform capable of navigating a workspace autonomously. In the case of the intralogistics use-case, an AGV is typically configured to carry a payload such that it can transport goods throughout the warehouse. The *Active Shuttle*, shown in Figure 1-2a, is an example of an AGV which was developed by

(a) Amazon Robotics [6] **(b)** Robert Bosch GmbH [1]

Figure 1-1: Examples of distribution centers which pose significant intralogistics challenges.

(a) Active Shuttle AGV navigating a dynamic environment. The orange and red regions represent its collision-avoidance regions.

(b) Multiple Active Shuttle AGVs navigating an intersection while carrying payloads (blue crates).

Figure 1-2: An example of AGVs used to perform intralogistics tasks [1].

Bosch Rexroth, a subsidiary of Robert Bosch GmbH. It uses a variety of sensors including two laser scanners and a camera to localize itself in a dynamic environment. Battery-powered electric motors coupled to a differential drive system are used to allow the Active Shuttle to navigate throughout the workspace. A payload can be carried by driving under a shelf and lifting it up using an actuated loading bay at the rear of the vehicle.

The pioneering work of Wurman *et al.* [6] has illustrated that multiple AGVs are capable of efficiently performing intralogistics tasks, assuming a defined factory layout with no dynamic obstacles other than the AGVs occupying the workspace. In this thesis, the aim is to contribute to an extension of this concept by contributing to the coordination of multiple AGVs navigating a workspace shared by third-party static and dynamic obstacles such as humans. The robotics community has already developed effective controllers which allow individual AGVs to efficiently navigate dynamic environments, as illustrated in Figure 1-2a. However, coordinating multiple AGVs efficiently within dynamic environments remains a challenging task. The coordination of multiple AGVs in a shared environment can be formulated as the Multi-Agent Path Finding (MAPF) problem. The MAPF problem typically considers an abstraction of the workspace to a graph, called a *roadmap*, where vertices represent spatial locations and edges pathways connecting two locations. The problem is to find trajectories for each AGV along this roadmap such that they reach their goal positions without colliding with the other AGVs. MAPF solvers determine such solutions while minimizing some metric such as cumulative route completion time or, alternatively, the makespan. The makespan of a plan is the maximum time it takes all the AGVs to complete their tasks.

Despite the abstraction of the MAPF to a significantly reduced decision space as a multi-agent graph search problem, solving the MAPF is a computationally demanding task. This is substantiated by the fact that the problem has been shown to be NP-Hard [7]. As is typically the case with combinatorial challenges such as vehicle routing or job shop scheduling, researchers formulate the problem by introducing a variety of constraints on the solution space in order to reduce the computational effort required to obtain solutions. In the case of the MAPF problem, AGVs traverse a graph structure representing the workspace, while assuming discrete graph traversal movements and ignoring external disturbances such as delays imposed on the AGVs [8]. Even though some more recent solutions do accommodate slight delays and

asynchronous AGV movement such as in [9, 10], these methods still result in inefficient plan execution when delay duration and frequency increases, since the ordering of AGVs along their paths remains fixed. Currently, most approaches require the MAPF problem to be re-solved when AGVs experience significant delays. However, due to the MAPF problem's exponential complexity, this approach is often very inefficient, especially when we consider a large number of AGVs.

In this thesis, the aim is to address this challenge by developing approaches that permit the re-ordering of AGV plans without needing to re-solve the entire MAPF plan, while simultaneously minimizing the route completion time of the AGV fleet. In contrast to existing approaches which consider delays a-priori, this thesis will focus on developing an online optimization scheme, which allows the plan ordering to be adjusted based on the delays that AGVs experience as they carry out their plans.

Finally, the MAPF problem can be used to address a number of practical challenges, not only the coordination of AGVs in a warehouse. Other practical applications are numerous and have been addressed by the literature. These include the routing of airport crew vehicles [11], ships in a harbor [12], mobile mining robots [13] and even artificial armies in computer games [14], as illustrated in Figure 1-3. This thesis will focus on general solutions which will essentially allow the developed methods to be applied to any of the aforementioned use cases.

(a) Routing airport crew vehicles [11].

(b) Ships in a harbor [12].

(c) Mobile mining robots [13].

(d) Computer game armies [14].

Figure 1-3: Various applications of the MAPF problem

1-1 Research Objectives, Focus and Scope

It is clear that a lot of work has been done on improving solutions to the MAPF. This thesis is focused on developing solutions which don't compete, but rather complement these recent advances to ultimately yield a more efficient and robust overall system. This will be done by looking into an online adjustment method to augment already determined plans such that AGVs can efficiently adjust their ordering based on delays while carrying out these plans. This brings us to the first research objective.

Research Objective 1. *Adjust the ordering of AGVs based on their current delays, while maintaining the collision-free and deadlock-free guarantees of the original plan.*

Next, the MAPF has recently been generalized into the so-called Multi-Agent Pickup and Delivery (MAPD) in [15]. The MAPD is essentially a persistent planning variant of the MAPF. In the MAPD, instead of a start-goal location for each AGV, AGVs are provided a stream of start-goal locations which need to be fulfilled as they appear. With the anticipation that the MAPD will become increasingly relevant, the methods developed in this thesis should be implementable for persistent plans as well. This requirement is encompassed in the second research objective.

Research Objective 2. *Extend the ordering of AGVs to be applicable to persistent planning architectures using a receding horizon control approach.*

Finally, to ensure that the developed methods are not only theoretically interesting, but practically implementable and beneficial, it is desirable to validate the developed theory by considering a realistic use-case simulating an environment where these methods will most likely be used. This validation is specified in the third and final research objective.

Research Objective 3. *Validate the developed methods in an extensive and realistic simulation environment.*

1-2 Contributions

Despite the increased attention multi-agent planning algorithms have garnered in recent years, adapting plans based on disturbances such as AGV delays in an efficient manner remains an open research question. This thesis aims to address this by presenting a new concept, called a Switchable Action Dependency Graph (SADG), to allow for the persistent adjustment of AGV schedules in an online fashion using an optimization-based approach.

Specifically, our contributions include the introduction of reverse agent dependencies leading to the SADG, an Optimal Control Problem (OCP), formulated as a Mixed-Integer Linear Program (MILP), to optimally select agent dependencies, and showing that execution management based on this approach guarantees collision- and deadlock-free plan execution.

Additionally, this thesis presents a method to reformulate the aforementioned OCP into a receding horizon scheme which is guaranteed to maintain the original collision- and deadlock-free plan execution guarantees. Finally, the presented methods are evaluated in an extensive

statistical simulation framework as well as a realistic simulation environment using the Robot Operating System (ROS) and Gazebo.

The theoretical contributions are summarized as follows:

1. **Switchable Action Dependency Graph**
   A novel data structure which can be used to re-order AGVs based on trajectory completion, while maintaining collision-free guarantees of the original plan.

2. **Shrinking Horizon Optimal Control Problem**
   The formulation of an Optimal Control Problem based on the SADG which can be solved in a feedback scheme to maintain the original deadlock-free guarantees of the plan despite re-ordering of AGVs.

3. **Receding Horizon Optimal Control Problem**
   Translation of the shrinking horizon OCP into a receding horizon formulation, while maintaining deadlock-free guarantees despite only considering a subset of the original plan within the OCP.

4. **Recursive Feasibility Guarantees**
   Both the switching and receding horizon feedback schemes are proven to be recursively feasible, meaning that the OCP can be guaranteed to have a feasible solutions at each iteration, which implies that plans are guaranteed to be completed in a collision- and deadlock-free manner.

1-3 Outline

This thesis consists of eight chapters, organized as depicted in Figure 1-4. Following the introduction in this chapter, Chapter 2 provides an overview of relevant literature regarding the coordination of multiple AGVs in shared environments. The goal of this chapter is to provide the reader with a concise overview of the state-of-the-art in order to put the contributions of this thesis into context.

Chapter 3 forms the foundation of this thesis. This chapter introduces the Switchable Action Dependency Graph (SADG), a novel data structure which forms the basis of the developed feedback schemes in the rest of this thesis. The SADG is presented following the introduction of switchable dependencies facilitating the re-ordering of AGVs despite significant delays in plan execution. The SADG is used as the basis of an Optimal Control Problem (OCP) formulation which can be solved in a feedback scheme such that the cumulative plan completion times for the AGVs is decreased.

Introducing the need for a persistent planning variant of the approach in Chapter 3, Chapter 4 extends the aforementioned concepts to be used in a receding horizon fashion, essentially allowing our approach to be used on schedules of theoretically infinite length.

Chapter 5 considers the OCPs presented in both Chapters 3 and 4 and presents a Mixed-Integer Linear Program (MILP) formulation which can be used to solve the OCPs in an efficient manner. This chapter also presents various cost functions and introduces several

heuristics for obtaining faster solutions exploiting the structure of the SADG. Finally, recursive feasibility is proven for both the shrinking and receding horizon control schemes of the previous chapters.

Chapter 6 provides a statistical evaluation of the presented methods in this thesis. The aim of this chapter is to gain insight into the performance benefits of the proposed methods from a statistical point of view.

In Chapter 7, the approaches presented in Chapters 3 through 5 are validated in a realistic simulation using the Gazebo simulation environment. A complete analysis of this simulation is performed to provide insight into the practical applicability of the methods developed in this thesis.

Finally, Chapter 8 presents a discussion of the results presented in Chapters 6 and 7. These results are analyzed in detail and placed in the context of the current state-of-the-art in the literature. This chapter also summarizes the contributions of this thesis, and provides insight into potential future work.

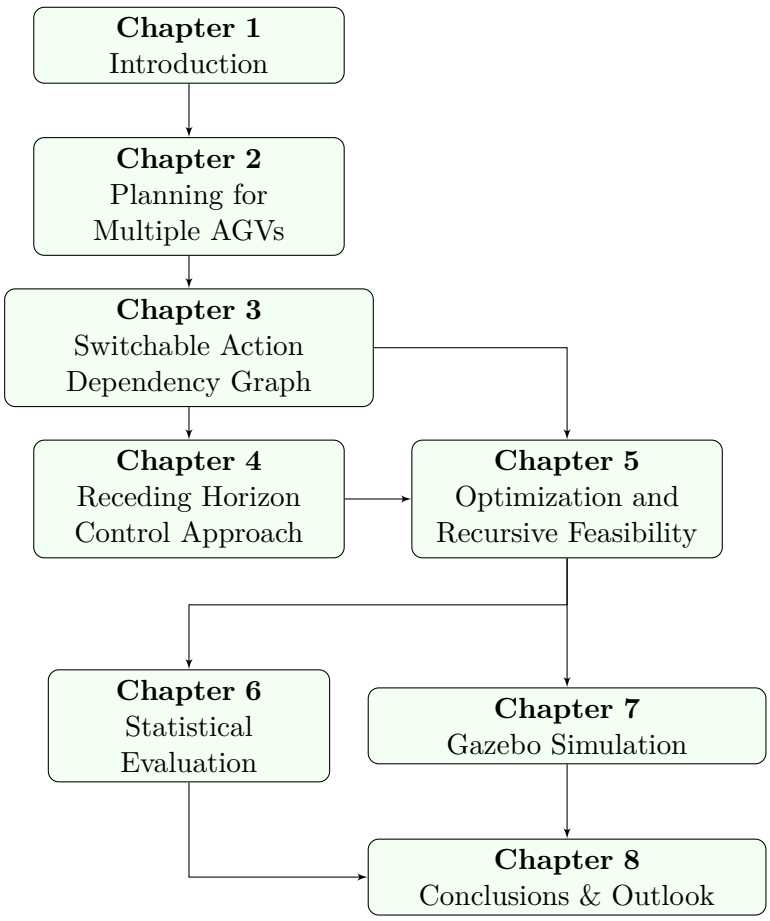


Figure 1-4: Outline of this thesis

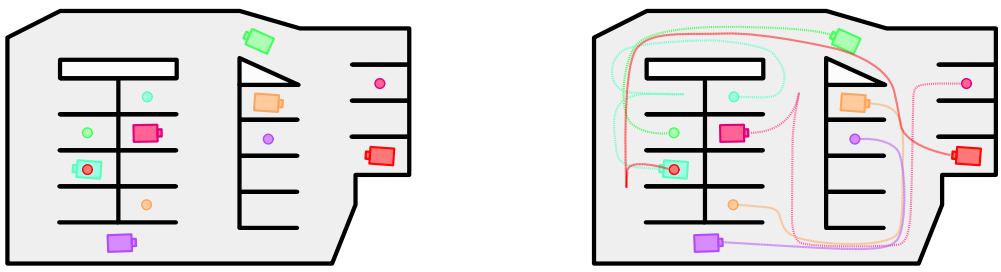
Chapter 2

Planning for Multiple AGVs

In this chapter, we introduce and describe the most notable developments related to the planning and coordination of multiple Automated Guided Vehicles (AGVs) relevant to this thesis work. The aim of this chapter is two-fold: firstly, to provide the reader with a brief, yet detailed overview of the current literature dedicated to the coordination of multiple AGVs in a shared environment; secondly, to place the contributions of this thesis into the wider context of existing solutions. These two points are important because the work in this thesis aims at complementing existing planning methods. To this end, a comprehensive understanding of the capabilities and limitations of existing methods is necessary.

2-1 The Multi-Agent Path Finding Problem

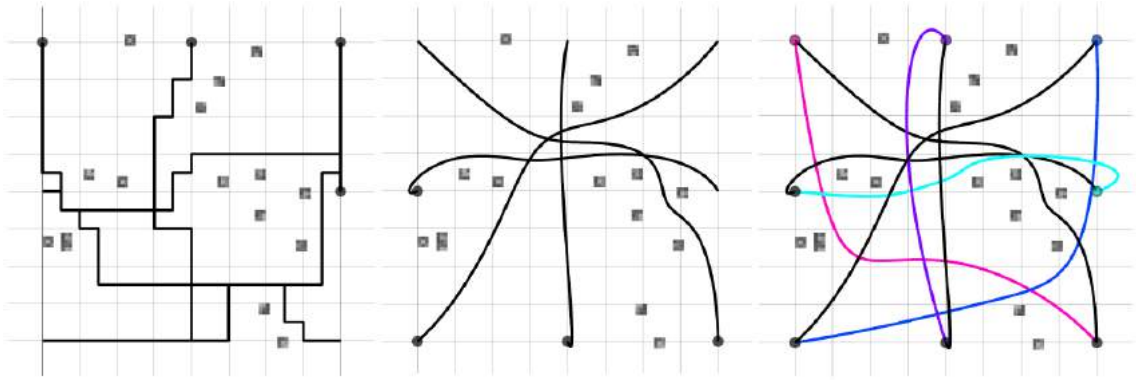
Consider a shared workspace, occupied by a set of AGVs, each with a unique start and goal pose as pictured in Figure 2-1a. In an intralogistics context, these start and goal poses could refer to inventory drop-off and pickup locations respectively.



(a) AGVs with start and goal positions.

(b) Trajectories for each AGV.

Figure 2-1: AGVs occupying a workspace, with starting positions (rectangular shapes) and their corresponding goal positions (circles), as well as possible collision-free trajectories.



(a) Graph-based roadmap. **(b)** Continuous trajectories. **(c)** Comparison of trajectories.

Figure 2-2: A continuous workspace with obstacles and multiple UAVs with graph-based solutions along the roadmap, continuous trajectories based on the graph-based solution and a comparison of these trajectories with those obtained from planning in the continuous workspace directly [2].

The task is to determine trajectories for these AGVs such that they can each reach their goal pose without colliding with one another. Additionally, it is desirable for these trajectories to be minimized by some metric such as expected route completion time or route distance. An example of trajectories resulting in minimum cumulative route completion time is shown in Figure 2-1b.

Planning in a continuous space is a very challenging task. To alleviate this, a common approach is to abstract this continuous space into a graph structure. A recent example of work which uses this approach is [2], where the authors address the challenge of coordinating multiple Unmanned Aerial Vehicles (UAVs) in a shared environment. Such a graph structure is commonly referred to as a *roadmap*, and it is assumed that AGVs can navigate the continuous space by traversing the edges of this roadmap. Referring to the work in [2], Figure 2-2a shows an example of a roadmap extracted from the continuous workspace littered with obstacles. In this manner, the problem of finding collision-free trajectories for AGVs in a common workspace is abstracted to a multi-agent graph search problem. Based on the graph-based solution, continuous trajectories can be extracted as shown in Figure 2-1b. It is worth noting that the abstraction of a workspace to a graph can yield longer trajectories compared to if these trajectories were planned in the original continuous space. An example is shown in Figure 2-2c, which shows trajectories based on a graph-based solution (in black), compared to the solution determined directly in continuous space (in color). Another example of a roadmap occupied by AGVs is shown in Figure 2-3. In this case, there are 70 AGVs occupying the workspace, and despite reducing the problem to a graph-search problem, it remains a challenging task which requires a significant amount of computational effort. This will be discussed in more detail in Section 2-2-1.

Note that there are some approaches which plan directly in continuous space. An example is the work of Pecora *et al.* [16]. However, this work assumes some general trajectory has already been determined for the AGV, and the task is to coordinate AGV movement to avoid collisions while following these pre-determined trajectories. This is different to the more general case we are considering here, where AGVs have more than one possible general trajectory they can follow to their respective goals.

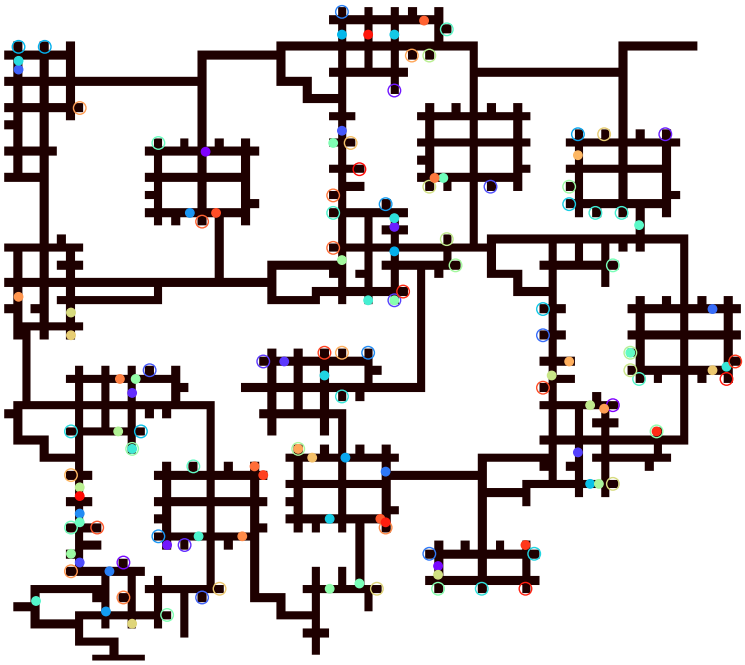


Figure 2-3: A roadmap occupied by 70 AGVs (represented by colored dots). AGVs must efficiently navigate from a start to a goal position (shown by colored circles) while avoiding collisions with one another, despite being subjected to delays.

Having briefly introduced the MAPF problem, let us now formalize this concept. The formal MAPF problem is defined in Problem 2.1.

Problem 2.1 (Multi-Agent Path Finding) *The MAPF problem for N AGVs is a tuple $(\mathcal{G}, AGV)$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph and $AGV = \{AGV_1, \ldots, AGV_N\}$ is a list of AGVs occupying $\mathcal{G}$. AGV_i is a tuple (s_i, g_i), where $s_i \in \mathcal{V}$ and $g_i \in \mathcal{V}$ denote the start and goal positions respectively, such that $s_i \neq s_j$ and $g_i \neq g_j \; \forall \; i, j \in \{1, \ldots, N\}$, $i \neq j$. Time is discretized such that each AGV is at one of the graph vertices and can only perform one action at each time step. An action is a function $a : V \mapsto V$ such that $a(v) = v'$ such that if AGV_i is at v and performs a, it will be at v' in the next time-step. At each time-step, an AGV can either wait at the current vertex v or move to an adjacent vertex v' if $(v, v') \in \mathcal{E}$.*

It is worth noting that the problem formulation in Problem 2.1 assumes that AGVs perform actions at fixed time-steps. Next, we define a valid solution to the MAPF problem in Definition 2.1.

Definition 2.1 (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.*

Essentially, a valid solution to the MAPF is a sequence of adjacent nodes for each AGV, from start to goal, such that none of the AGVs occupy the same roadmap node or traverse the same edge of the roadmap at the same time. Definition 2.1 only specifies the requirements for a valid MAPF solution, without placing any requirements on plan length, as long as they are finite. The idea is that a solver solving the MAPF problem will do so by minimizing a metric such as cumulative trajectory length or the makespan. The makespan is the maximum plan length or route completion time of each of the AGVs.

2-2 Challenges Related to Multi-Agent Path Finding

There are numerous challenges related to the MAPF problem. In this section, we present the challenges which need to be considered by the solution in this thesis work. Given that the MAPF is NP-hard [7], the first obvious challenge is that of developing efficient solvers to solve the MAPF in an efficient manner. The second challenge which needs to be considered is motivated by the fact that a valid MAPF solution as in Definition 2.1 requires an execution policy. This is because most MAPF solutions assume that the AGVs executing the plan do so in a perfectly synchronous manner. This requires AGVs to have perfect, global communication, and assumes AGVs will never be delayed. Naturally, these assumptions are difficult to adhere to in practical applications, meaning that MAPF solution execution policies need to be developed to ensure correct execution of theoretically collision-free MAPF plans. Another challenge is the fact that the MAPF problem assumes AGVs have a start and goal pose. This is a limiting assumption because, typically, AGVs get assigned new tasks once their current tasks are completed, implying the need for a persistent MAPF formulation.

The aim of this section is to provide more details about these aforementioned challenges, starting with efficient solutions to the MAPF in Section 2-2-1, extensions to persistent MAPF formulations in Section 2-2-2, existing MAPF solution execution policies in Section 2-2-3 and methods which consider delays in the planning algorithm in Section 2-2-4.

2-2-1 Efficient Solutions to the MAPF

Due to its increasing relevance and important application domains, solving the MAPF problem has garnered widespread attention over the years. Comprehensive overviews of existing MAPF solutions have been presented by Felner *et al.* [17] and, more recently, Stern *et al.* [8], of which the most relevant to this thesis will be presented here. Since the MAPF complexity grows exponentially with the number of decision variables, existing solutions either attempt to solve the problem optimally in an efficient manner using heuristics, or solve a bounded sub-optimal version of the problem. We are also limiting our focus to complete algorithms, meaning that these algorithms are guaranteed to find a solutions, if one exists.

Conflict-Based Search

A popular approach is called Conflict-Based Search (CBS), introduced by Sharon *et al.* [18]. Consider a MAPF problem with a set of N AGVs denoted by $AGVs = (AGV_1, \ldots, AGV_N)$ navigating a roadmap $\mathcal{G}_{\text{roadmap}} = (\mathcal{V}_{\text{roadmap}}, \mathcal{E}_{\text{roadmap}})$. The CBS algorithm uses a two-level

solution scheme. The high level component performs a search along a so-called *conflict tree* (CT). Each node within the CT consists of the following elements:

1. **Constraints**
   Each constraint is associated with a single AGV. Each node in the CT inherits the constraints of its parent nodes. The root node constraints is an empty set. Constraints can either be related to vertices or edges within the roadmap $\mathcal{G}_{\text{roadmap}}$. A vertex constraint is given by the tuple

   $$(AGV_i, v, t) \text{ where } AGV_i \in AGVs, v \in \mathcal{V}_{\text{roadmap}}, t \in \mathbb{R}_{\geq 0}. \tag{2-1}$$

   Similarly, an edge constraint is given by the tuple

   $$(AGV_i, v_k, v_l, t) \text{ where } AGV_i \in AGVs, v_k, v_l \in \mathcal{V}_{\text{roadmap}}, t \in \mathbb{R}_{\geq 0}. \tag{2-2}$$

   These constraints refer to a portion of the roadmap $\mathcal{G}_{\text{roadmap}}$ which should not be visited by AGV_i at time t.

2. **Solution**
   A set of N paths where the path of AGV_i adheres to all the constraints applicable to that AGV.

3. **Total cost**
   The cost of the node solution. This is the sum of the path length of each AGV.

The solution for a node within the CT is determined at the low level. This is done by performing a shortest-path search for the AGVs associated with the constraints of that node. This shortest path search is performed while adhering to the constraints defined in the current CT node.

Once a solution has been determined for a node in the CT, it is validated. This means that it is checked to see if it contains plans which occupy the same vertex or edge at the same time-step, which is referred to as a conflict. Conflicts can either be vertex- or edge conflicts. A vertex conflict is a tuple of the form

$$(AGV_i, AGV_j, v, t) \text{ where } AGV_i, AGV_j \in AGVs, v \in \mathcal{V}_{\text{roadmap}}, t \in \mathbb{R}_{\geq 0}. \tag{2-3}$$

Similarly, an edge conflict is a tuple of the form

$$(AGV_i, AGV_j, v_k, v_l, t) \text{ where } AGV_i, AGV_j \in AGVs, v_k, v_l \in \mathcal{V}_{\text{roadmap}}, t \in \mathbb{R}_{\geq 0}. \tag{2-4}$$

If a solution contains a conflict, this conflict is resolved by creating two constraints, one for each AGV in the conflict. A conflict $(AGV_i, AGV_j, \cdot)$ yields the constraints $(AGV_i, \cdot)$ and $(AGV_j, \cdot)$, where the dot is used to represent the same vertex and time of both vertex and edge conflicts and constraints. Note the difference between the conflicts in (2-3) and (2-4) which refer to two AGVs, compared to constraints in (2-1) and (2-2) which are used in the low-level shortest-path search algorithm. The aforementioned discussion is contained within the pseudo code for the CBS algorithm in Algorithm 1.

To illustrate the multi-agent graph-search problem addressed in the MAPF, consider the illustration shown in Figure 2-4. This illustrations shows the trajectories some of the AGVs

Algorithm 1 Conflict-Based Search High-Level [18]

Input: MAPF problem.
Result: $\mathcal{P}$

1: $Root$.constraints $\leftarrow \emptyset$
2: $Root$.solution $\leftarrow$ low level shortest-path search for each AGV
3: $Root$.cost = getCost($Root$.solution)
4: CT.insert($Root$)
5: **while** CT not empty **do**
6: $P \leftarrow$ lowest cost node in CT
7: Validate paths in P until conflict occurs
8: **if** P has no conflicts **then**
9: return P.solution
10: $C \leftarrow$ first conflict (AGV_i, AGV_j, v, t) in P
11: **for** AGV_i in C **do**
12: $A \leftarrow$ new CT node
13: A.constraints $\leftarrow P$.constraints + (AGV_i, v, t)
14: A.solution $\leftarrow P$.solution
15: Update A.solution using low-level search for AGV_i
16: A.cost = getCost(A.solution)
17: **if** A.cost $< \infty$ **then**
18: Insert A to CT

must follow in order to guarantee collision-free movement to their respective goals. Note, for example, the first portion of yellow trajectory, where the AGV must temporarily move to the right to make way for another AGV before moving to the left again, in the direction of its goal position. This kind of behavior may seem sub-optimal for a single AGV, but is optimal when considering all the AGVs collectively.

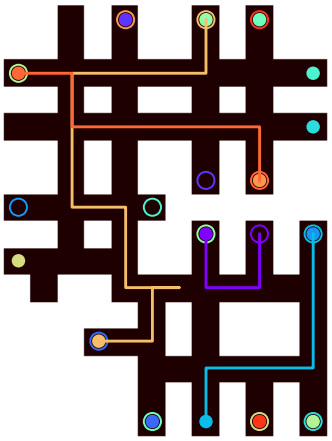


Figure 2-4: Visualization of conflict-free paths determined by CBS. Only a subset of the trajectories are shown for clarity. These paths are collision free in space-time.

Improvements and Variants of Conflict-Based Search

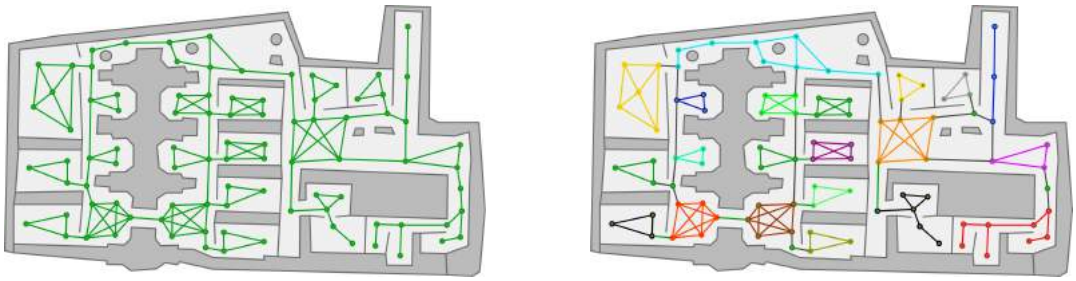
The original CBS algorithm has been improved by exploiting properties such as geometric symmetry by Li *et al.* in [19] or using purpose-built heuristics as by Felner *et al.* in [20]. Another significant increase in performance was presented more recently by Lam *et al.* in [21]. Lam *et al.* identified the similarity of CBS to the branch-and-bound architecture used in Mixed-Integer Linear Program (MILP) solvers, where the linear program relaxation corresponds to the low level shortest-path graph search in CBS. This work goes on to formulate the MAPF as an MILP which is subsequently solved using a purpose-designed branch-cut-and-price solver.

Another approach to ensure fast solution times is by finding sub-optimal solutions that are bounded with respect to the optimal solution. Depending on the chosen sub-optimality ratio w, this can lead to significantly faster solution times. Barer *et al.* introduced a variant of CBS, called Bounded Sub-Optimal Conflict-Based Search (ECBS) [22], which uses a focal search variant of A* at the low level planner of CBS to yield solutions with a makespan upper-bounded by a factor w compared to the optimal solution.

Other notable solutions to the MAPF include Prioritized Planning using Safe Interval Path Planning (SIPP) [23], declarative optimization approaches using answer set programming [24, 25], heuristic-guided coordination [16] and graph-flow optimization approaches [26]. The work by Ryan *et al.* [3] manually decomposes the overall map into a set of simpler sub-graphs. The plans are then determined in a hierarchical fashion: first, robots plan movements from one sub-graph to another; second, purpose-specific planners are used to determine paths within each sub-graph. This method is sound and complete, however, the increase in search performance is highly dependent on the (manually performed) graph decomposition. Figure 2-5b shows the overall graph of the original factory in Figure 2-5a with sub-graphs indicated by the different colors. Wilt *et al.* [27] follow a similar approach where congested portions of the graph are highlighted where a dedicated agent is used to solve the routing problem locally. The aforementioned works and references are summarized in Table 2-1.

Name	Optimal	Reference
Conflict-Based Search	yes	[18]
Prioritized Planning using SIPP	no	[23]
Declarative Programming	yes	[25]
Graph flow optimization	yes	[26]
Exploiting symmetry in CBS	yes	[19]
Purpose-built heuristics CBS	yes	[20]
Enhanced-CBS	bounded sub-optimal	[22]
Subgraph optimization	no	[3]

Table 2-1: Summary of MAPF solutions.



(a) A *roadmap* graph showing the abstraction of the workspace into a roadmap.

(b) Reduction of the roadmap into sub-graphs indicated by different colors.

Figure 2-5: Top-view of a real warehouse workspace layout as used by Ryan *et al.* in [3].

2-2-2 Extensions to MAPF

Despite the numerous works already dedicated to solving the MAPF, the original formulation as presented in Problem 2.1 has a number of shortcomings regarding its generality. More specifically, the MAPF assumes all AGVs move in discrete, constant time steps across an unweighted graph. Additionally, all AGVs are assumed to be given a single task, and stay in their goal position once it is reached. When considering certain implementation examples, these assumptions can sometimes be too strict, which has inspired researchers to consider relaxations of these problems. We discuss two notable extensions in this section which are most relevant to this work.

Roadmaps as Weighted Graphs

The authors of [28] address the fact that the MAPF is originally defined for unweighted roadmaps where AGVs only move in discrete time-steps by introducing Continuous Conflict-Based Search (CCBS). CCBS allows the roadmap to be defined as a weighted graph. The CCBS algorithm also makes use of a two-level approach like CBS. However, the lower level path planning is solved using SIPP [23] instead of the much faster A* search or Dijkstra's algorithm as in CBS. This results in increased computation times for the same sized problem, as well as solutions which allow multiple AGVs on the same edge, as long as they do not collide.

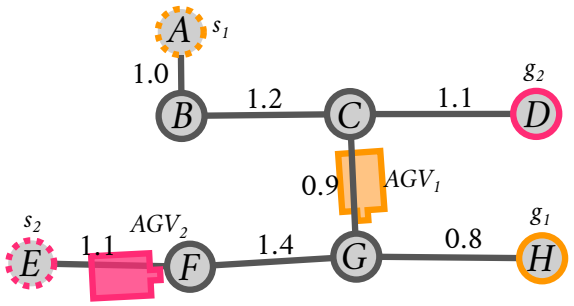


Figure 2-6: A roadmap graph occupied by two AGVs with start s_i and goal g_i for $i = \{1, 2\}$. The edge weights indicate the expected traversal times.

Multi-Agent Pickup and Delivery

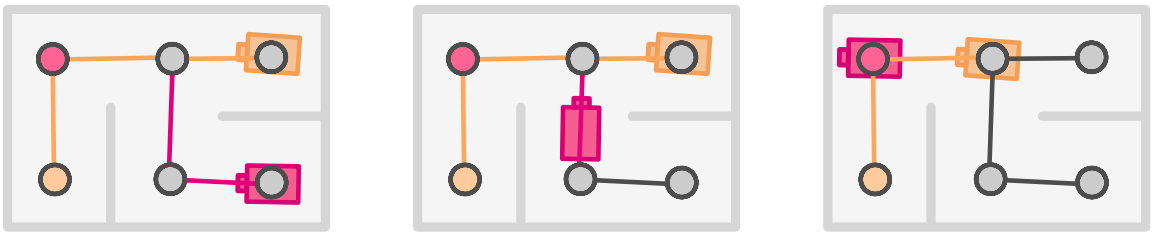
Another important assumption in the original MAPF problem is that AGVs have a single task defined by a start-goal pair and, once this task is complete, the AGV simply remains at the goal position until all the AGVs have reached their goals. However, a more realistic assumption, especially when considering intralogistics, is that tasks appear in a continuous stream, and AGVs must fulfill each task as they appear. This is the motivation behind the recently formalized Multi-Agent Pickup and Delivery (MAPD) problem. The MAPD problem is essentially a persistent extension of the original MAPF which was formally introduced by Ma *et al.* in [15]. The authors use a method where the basic idea is that AGVs assign themselves to a stream of incoming tasks based on their relative location and availability. The tasks and assignment information is stored in globally-shared memory and accessed by each AGV when necessary. Despite results showing that this method is far less effective than the MAPF solution in the case that all the tasks appear at once, this method is a good starting point when considering persistent task assignment. Initially omitted from the first MAPD solution, Ma *et al.* go on to address kinematic constraints imposed on each AGV in [29].

2-2-3 Executing MAPF Solutions using Plan Execution Policies

A core assumption in many MAPF algorithms is that AGVs acting out the plans are perfectly synchronized and are thus able to be in constant, perfect communication with one another. The result is that MAPF planners sometimes yield solutions which cannot easily be executed in reality, unless the aforementioned can somehow be met, which is rarely the case in practice [30]. To illustrate this point, consider the scenario shown in Figure 2-7. In this case, the roadmap consists of a 4-node rectangular graph, with one AGV at each node. The goal position for each AGV is the next node, looking in an anti-clockwise orientation, implying that the MAPF solution only requires each AGV to move forward by one node. In theory, executing such a plan is trivial. However, to avoid collisions, the AGVs must be perfectly synchronized and move at exactly the same velocity to ensure that none of them collide. Since AGVs are typically controlled by motion planners using controllers based on feedback laws, each AGV is waiting on the next AGV blocking its path to move. This will result in a deadlock because each AGV is waiting on another, forming a cycle of inter-AGV dependencies.

(a) AGVs in starting positions. **(b)** AGVs moving synchronously towards goals.

Figure 2-7: Example illustrating the need for synchronous coordination among AGVs to execute a trivial MAPF solution. Colored edges and nodes indicate trajectory and goal positions respectively.



(a) A valid MAPF solution for each AGV indicated with colored edges/nodes.

(b) The orange AGV is delayed but the red AGV follows its path.

(c) The red AGV is now blocking the orange AGV because of the delay.

Figure 2-8: Example illustrating the need for plan execution policies to account for AGV delays. Colored edges and nodes indicate trajectory and goal positions for the corresponding AGV.

Another example that illustrates the need for plan execution policies is shown in Figure 2-8. In this case, two AGVs have independent goals, and a MAPF plan is solved to determine the routing for each AGV such that collisions will not occur assuming nominal behavior. The original plan requires that the red AGV waits for the orange AGV. However, as shown in Figure 2-8b, the orange AGV could be delayed due to an external disturbance, meaning that the red AGV waits, and then starts following its dedicated plan. The result is that the red AGV ends up blocking the orange AGV since its goal position is along the path of the orange AGV.

The conclusion of these two examples is that an execution policy is required to ensure the original plan is followed correctly. Referring to the second example, informing the red AGV of the delayed orange AGV could allow it to first wait until the orange AGV makes up for the delay before continuing along its path. The original MAPF problem does not consider these challenges, and the solutions presented in Section 2-2-1 do not account for these aforementioned complications. However, some recent works have addressed this challenge of which the most notable and applicable to this thesis work are discussed next.

Robust Multi-Robot Trajectory Tracking Strategy

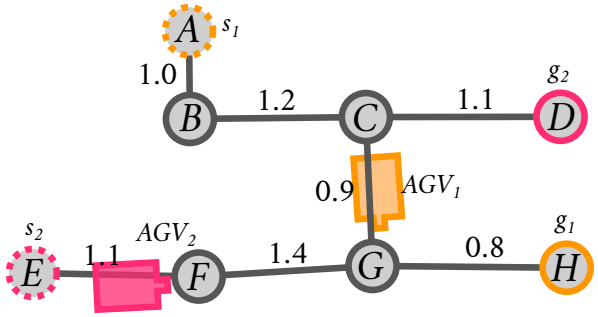
In [30], the authors present a control scheme which considers the progress of each AGV along its path, as dictated by a MAPF plan, and determines if each AGV can proceed, or whether it should stop and wait on other AGVs. The control law considers the AGVs in coordination space, a space formed by taking the product of each AGV's trajectory in space-time according to the plan. This method, dubbed by the authors as RMTRACK, ensures that AGVs can continue with their individual plans up until the point where they block another, possibly delayed, AGV from safely following its plan. RMTRACK is compared to the a so-called ALLSTOP approach, which is a naive control law that requires all AGVs to stop and wait until a delayed AGV has made up for this delay, showing lower overall cumulative route completion times in simulation.

As with all literature detailing plan execution policies for multiple AGVs, the authors of RMTRACK analyze the collision-free and liveness properties of their method. The core assumption for the liveness guarantees of RMTRACK is the fact that the ordering of AGVs, as

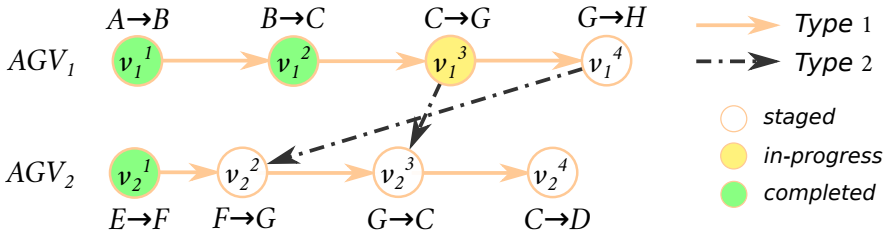
dictated by the original MAPF plan, remains unchanged. Hence, RMTRACK only improves the efficiency of AGVs as they complete their plans by potentially allowing them to move despite other AGVs being delayed. However, if e.g. AGV_1 is delayed by a significant amount of time, and the implicit ordering dictated by the MAPF plan requires AGV_1 to reach a point $S \in \mathbb{R}^2$ along its path before AGV_2 can reach that same point, RMTRACK will force AGV_2 to wait until AGV_1 has made up for its delay. At the limit, this implies that a single AGV could still significantly delay all other AGVs.

Action Dependency Graph Approach

In a somewhat parallel approach to RMTRACK, the authors in [31] make use of a so-called Action Dependency Graph (ADG) to maintain the implicit ordering of AGVs along their MAPF. 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 in [9]. The ADG is will be used as the foundation for work in this thesis, and we therefore introduce this concept in more detail, starting with the example shown in Figure 2-9. Consider the representation of a workspace as a roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, shown in Figure 2-9a.



(a) A roadmap 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 Figure 2-9a.

Figure 2-9: Illustrative MAPF problem example alongside the constructed Action Dependency Graph

Solving this MAPF problem with the previously introduced CCBS method 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. It is important to note that we modified the CCBS algorithm such that AGVs are not permitted to occupy the same edge at the same time. Originally, CCBS allows two AGVs to occupy the same edge, as long as the circles describing their position-envelopes do not intersect. The ADG method assumes that AGVs cannot occupy the same edge at the same time, hence our modification.

To facilitate discussion, let us first introduce the relevant notation. Given a plan tuple $p = (l, t)$, we 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.

The ADG is defined in Definition 2.2. This definition is a modification of the originally proposed ADG in [9] by allowing multiple plan tuples to be contained within a single ADG vertex. The motivation behind this modification will become clear in Section 3-2-1, since it allows us to introduce a spatial exclusivity property to vertices making up the ADG.

Definition 2.2 (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 $(v_i^k, v_j^l) \in \mathcal{E}_{\text{ADG}}$ 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 2. Figure 2-9b shows an example of an ADG plan being executed, based on the AGV positions in Figure 2-9a. Observe how AGV_2 cannot start v_2^2 before v_1^4 has been completed because of the *Type 2* dependency pointing from v_1^4 to v_2^2.

A sufficient condition to guarantee liveness of the plans encoded within an ADG is for it to be acyclic. Recall that a vertex within the ADG corresponds to an action executed by an AGV. Since a dependency enforces that the vertex to which it is pointing from must be completed before the vertex to which it is pointing to can be in-progress, an acyclic ADG implies that no vertices are mutually dependent. This means that at each time-step, at least one of the vertices can be completed. However, as discussed in [9], there exist valid MAPF solutions for which the resulting ADG is cyclic, a simple example being the previously discussed MAPF problem illustrated in Figure 2-7. It is possible to modify MAPF planning algorithms to avoid obtaining such plans, which require precise synchronous execution. In CBS, this corresponds to an additional edge constraint which needs to be considered every time there is a node-conflict. Such an assumption is relatively easy to maintain, in theory as well as in practice, as detailed in [31], since we only consider additional constraints in the CT of CBS.

Algorithm 2 Action Dependency Graph Construction based on [9]

Input: Plan $\mathcal{P}_i$ for each robot.
Result: $\mathcal{G}_{\text{ADG}}$

// create vertices and *Type 1* edges
for $i = 1$ to N **do**
  $v_i^1 \leftarrow (\{p_i^1\}, staged)$
  Add v_i^1 to $\mathcal{V}_{\text{ADG}}$
  $v_{\text{prev}} \leftarrow v_i^1$
  for $k = 2$ to N_i **do**
    $v_i^k = (\{p_i^k\}, staged)$
    Add v_i^k to $\mathcal{V}_{\text{ADG}}$
    Add edge (v_{prev}, v_i^k) to $\mathcal{E}_{\text{ADG}}$
    $v_{\text{prev}} \leftarrow v_i^k$
// create *Type 2* edges
for $i = 1$ to N **do**
  for $k = 1$ to N_i **do**
    for $j = 1$ to N **do**
      if $i \neq j$ **then**
        for $l = 1$ to N_j **do**
          if $s(p_i^k) = g(p_j^l)$ **and** $\hat{t}_g(p_i^k) \leq \hat{t}_g(p_j^l)$ **then**
            Add edge (v_i^k, v_j^l) to $\mathcal{E}_{\text{ADG}}$
            break

As a result, we introduce Assumption 2.1, which allows us to guarantee that a constructed ADG will be acyclic.

Assumption 2.1 (Acyclic ADG) *The MAPF solution as defined in Definition 2.1 is such that the ADG constructed by Algorithm 2 is acyclic.*

2-2-4 Explicitly Considering Delays

In real applications, AGVs are inevitably going to experience delays. Such delays can be caused by the need for an AGV to avoid an obstacle, taking a longer time to cross a section of the road, or dynamic obstacles temporarily blocking the path of the AGV, or even in the form of a mismatch between the roadmap and the physical world, where the actual path the AGV must travel is longer than expected. Having previously discussed methods to account for delays while executing MAPF plans, some works in the literature have considered delays within the MAPF formulation itself. The duration of these delays are typically modeled as an arbitrary deviation from a given value bounded within a range, or as a value sampled from a probability distribution. In this section, we present the most notable solutions which consider delays viewed from either of these two perspectives.

Robust Solutions to the MAPF

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 [10]. The authors introduce kR-CBS, an extension to CBS, where the constraints as presented in (2-3) and (2-4) are extended such that they are defined over a range, subsequently called *range constraints*. A range constraint can be represented as follows

$$(AGV_i, v, [t_1, t_2]) \text{ where } AGV_i \in AGV, v \in \mathcal{V}_{\text{roadmap}},\ t_1, t_2 \in \mathbb{R}_{\geq 0}, t_1 < t_2.$$

The result of this work is a more conservative solution to the MAPF problem, since each AGV is allocated a larger time window to visit the vertices along its path as it drives towards its goal. By varying the value of k, the robustness of the solution is increased, but at the cost of deterministic optimality. Note that, in the case that an AGV is delayed by more than k time-steps, other AGVs which share a path with this AGV need to wait in order to avoid collisions.

Modeling Delays Stochastically

An alternative representation of delays is by modeling them in a stochastic manner. The most notable work on this is that of Ma *et al.* in [32], where the authors present an extension of the MAPF, called the multi-agent path finding problem with delay probabilities (MAPF-DP). This problem formulation essentially considers the probability on whether an AGV will advance to the next position along its path or not. The MAPF-DP is then solved using a 2-level algorithm, much like the CBS approach, consisting of a high-level and low-level search. The main difference is that the low-level search uses a minimization in expectation approach to determine individual paths with the lowest expected routing time. The authors go on to introduce two robust plan-execution policies to execute a valid MAPF-DP solution. The two policies are a communication-heavy fully synchronized policy (FSP) and a minimal communication policy (MCP). These policies are similar to the RM Track approach discussed in Section 2-2-3, albeit with global or only local information sharing respectively.

Shortcomings

A clear shortcoming of both the robust and stochastic delay methods discussed in this section are the fact that information of the delays are assumed *a priori*, and once the re-formulated MAPF problem (kR-MAPF or MAPF-DP) has been solved, no adjustment to the plans can be made to account for unexpected disturbances. This means that these methods are highly reliant on a good representation of the expected delays which AGVs will experience while completing their tasks. Both these formulations are therefore not well suited to highly dynamic environments with unpredictable delays.

2-3 Problem Outline

Based on the literature presented in the previous sections, we now present an outline of the shortcomings of these approaches in order to define the challenges to address in this thesis work. Four important realizations from the literature presented in this chapter are:

1. **Solving the MAPF is time consuming**
   Despite extensive research on solving the MAPF, finding an efficient solution to this problem remains time-consuming in the case that a large number of AGVs are considered. The main realization is that including re-planning within a feedback loop is not a realistic approach to account for disturbances. Hence, an alternative form of re-planning is required which considers a significantly smaller subset of the original MAPF decision space.

2. **MAPF execution policies are a necessity**
   The MAPF is a highly abstracted interpretation of the physical problem of routing AGVs in a shared environment. Despite such an abstraction begin necessary to make finding MAPF solutions a tractable problem, many of the assumptions thereof cannot always be guaranteed. Hence, a reliable, efficient and robust plan execution policy is necessary to execute a MAPF solution.

3. **Large delays are best addressed in an online fashion**
   Existing work which take delays into account consider them in an *a-priori* fashion. 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 challenge being to optimize the plan in an efficient manner while maintaining deadlock- and collision-free execution guarantees.

4. **Liveness guarantees assume constant AGV ordering**
   The presented literature guarantees liveness by assuming the initial MAPF solution is collision-free and executable in finite time, and then maintaining the implicit ordering of the AGVs to maintain this liveness property. However, in order to make plan execution more efficient when considering large delays, a re-ordering strategy is highly desirable. The challenge is to maintain the original liveness guarantees while allowing for the re-ordering of AGVs.

Based on these four realizations, we are now ready to present the solutions developed in this thesis, starting with a Switchable Action Dependency Graph in Chapter 3. This solution addresses these four points by (1) only solving a subset of the MAPF decision space, (2) providing an execution policy for AGVs allowing real AGVs to execute the MAPF plans, (3) accounting for large delays in a feedback loop and (4) maintaining liveness despite re-ordering the AGVs.

2-4 Summary

In this chapter, a review of literature related to the planning and routing for multiple AGVs was presented. The MAPF problem was formalized and presented, as well as an overview of existing solutions and extensions developed to address the challenges posed by this problem. It was observed that a solution which allows for the natural re-ordering of AGVs based on unpredictable delays has not been addressed in the literature yet, as far as the author is aware. Finally, the problem outline which will be addressed in this thesis was presented.

Chapter 3

Switchable Action Dependency Graph

In this chapter, the initial contributions of this thesis are presented. Having identified in Chapter 2 that persistent plan re-ordering is a crucial component in ensuring efficient execution of MAPF plans given large delays, we present a novel concept called a Switchable Action Dependency Graph (SADG) which will allow the re-ordering of AGVs while maintaining the collision- and deadlock-free guarantees of the original plan. We prove that the SADG maintains these guarantees given realistic plan assumptions. This SADG can be used to enable the re-ordering of AGVs as they complete their planned trajectories to ultimately decrease plan completion times when these AGVs are subject to delays.

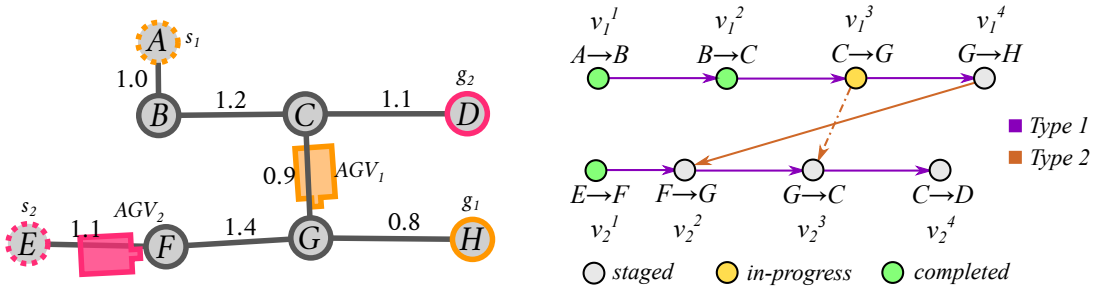
Working towards this result, we introduce the concept of a reverse dependency, which brings forth the necessity for spatial exclusivity among Action Dependency Graph (ADG) vertices. We then present an algorithm which uses these concepts to construct an SADG which can be used as an efficient tool to execute a MAPF plan despite significant AGV delays. We prove that the obtained SADG can maintain the original plan's collision- and deadlock-free execution guarantees.

3-1 Preliminaries

Recall the introduction of the Action Dependency Graph (ADG) in Section 2-2-3. This concept facilitates the successful execution of a MAPF plan despite small delays experienced by the AGVs. This is done by imposing inter-agent dependencies on the events implicitly contained within the MAPF solution. The ADG is a directed graph where each node represents an action associated with an AGV, and the edges indicate the dependence on different events within this ADG. An event within the ADG can only change from *staged* to *in-progress* if all the dependencies pointing to that event are *completed*.

Given the simple example in Figure 3-1a, the MAPF solution can be obtained using a dedicated solver and the ADG constructed using Algorithm 2. Now, let us consider the case where one of the AGVs is delayed for a significant amount of time. As both AGVs complete their events, AGV_1 (in red) is delayed (possibly due to a dynamic obstacle temporarily blocking

the way). In Figure 3-2b, AGV_2 cannot proceed due to the dependency (v_1^4, v_2^2), and since v_1^4 is not *completed*. We therefore switch the two dependencies, as shown in Figure 3-2c, allowing AGV_2 to continue with its path without having to wait for AGV_1. Finally, in Figure 3-2f, the AGVs reach their respective goals. Note that these ideas apply seamlessly to any number of AGVs, as long as the switched dependencies do not form a cycle (causing a deadlock). Only two AGVs are used in this example to illustrate the concept in an intuitive manner.



(a) Roadmap **(b)** Action Dependency Graph

Figure 3-1: Example of an illustrative MAPF problem in (a) as well as the constructed Action Dependency Graph in (b).

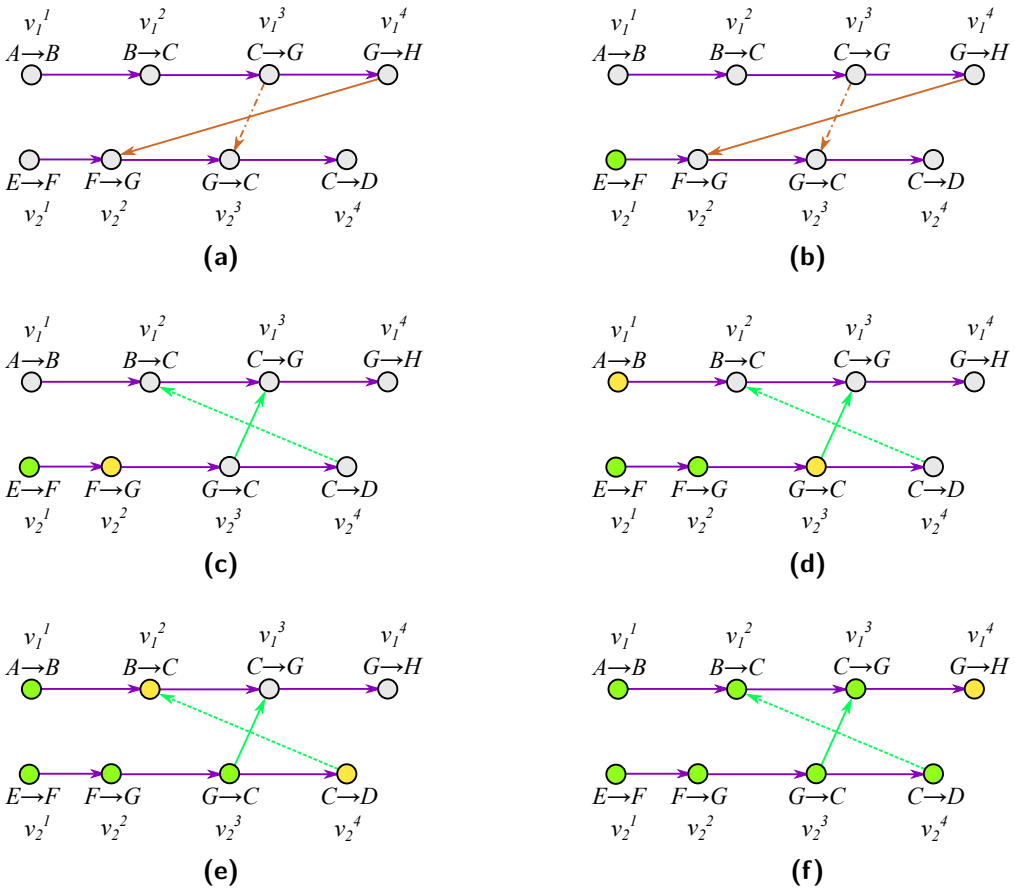


Figure 3-2: Illustrative example of switching based on event completion within the ADG. Dependencies are switched between (b) and (c), emphasized by the differently colored arrows.

It is important to note that this example made use of various properties of the ADG which we did not specifically mention. These properties are necessary to define a systematic manner to allow for the switching of dependencies, and are as follows:

1. **Switching dependency**
   The switching dependency should maintain the collision-free guarantees of the original dependency.

2. **Spatial exclusivity**
   Consider a single dependency, switching the dependency will only require the introduction of one new dependency if the ADG vertices are spatially exclusive, meaning that each vertex requires the transition of an AGV to a new area.

3. **Switching which did not cause another deadlock** The switching must not cause a cycle in the ADG, as this implies a deadlock since two or more vertices are mutually dependent on each other to change from *staged* to *in-progress*.

Based on these properties, we will modify the original ADG to ensure spatial exclusivity, which will lead to an introduction of a systematic manner in which to define reverse dependencies for switching. The selection of the dependency must be done in such a way that the resultant ADG is acyclic. These concepts are presented in Section 3-2.

3-2 Systematically Re-ordering AGVs

Having introduced the basic idea of switching, we now work towards a formal introduction and description of a Switchable Action Dependency Graph (SADG). An SADG is a data structure which we will utilize to optimize AGV re-ordering based on the AGV delays at a given time-step. The core component of the SADG is the reverse dependency. However, since the reverse dependency makes use of a spatial exclusivity assumption in an ADG, we formally present this concept. To facilitate discussion, let us first introduce the relevant notation. 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 $\mathbb{R}^2$. Let $S_{\text{AGV}} \subset \mathbb{R}^2$ refer to the physical area occupied by an AGV.

3-2-1 Spatially Exclusive Action Dependency Graph

An ADG contains the sequence of events defined by a MAPF plan. However, the MAPF problem could be defined in a space including orientation or general robotic actions, meaning that successive plan tuples in the original plan do not always refer to a transition in the spatial domain. To allow for a natural definition of a reverse dependency, we introduce the concept of spatial exclusivity. A spatially exclusive ADG is an ADG where each vertex $v \in \mathcal{V}_{\text{ADG}}$ describes the movement of two spatially exclusive locations in the physical workspace. Two locations $S_1, S_2 \in \mathbb{R}^2$ are deemed spatially exclusive for AGVs of size S_{AGV} if

$$S_1 \oplus S_{\text{AGV}} \cap S_2 \oplus S_{\text{AGV}} = \emptyset, \tag{3-1}$$

where $\oplus$ denotes the minkowski sum. (3-1) essentially implies that if an AGV of footprint S_{ADG} is occupying a space S_1, another AGV of footprint S_{ADG} can occupy S_2 without any collision between the two AGVs. The footprint of an AGV refers to its size in all possible orientations, but for a fixed position at the origin.

Definition 3.1 (Spatially Exclusive Action Dependency Graph) *A spatially exclusive 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 $v_i^k = (\{p_1, \dots, p_q\}, status) \in \mathcal{V}_{ADG}$ denotes the event of AGV_i moving from $S(p_1)$, via intermediate locations $S(p_2), S(p_3), \dots, S(p_{q-1})$, to $S(p_q)$, where $q \geq 2$ denotes the number of consecutive plan tuples encoded within v_i^k and $S(p_1) \oplus S_{AGV} \cap S(p_q) \oplus S_{AGV} = \emptyset$. Finally, status $\in \{staged, in\text{-}progress, completed\}$.*

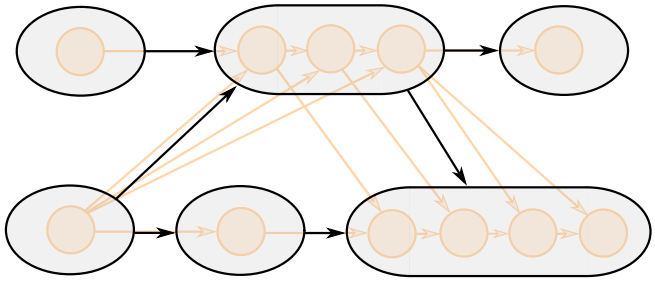


Figure 3-3: A schematic showing the difference between the originally proposed ADG in orange, overlaid by a spatially exclusive ADG used to construct the SADG in black. Note how a single dependency in the spatially exclusive ADG is sufficient to represent multiple dependencies in the original ADG.

We note that the definition of spatial exclusivity is rather general, specifically the definition of the footprint of an AGV S_{AGV}. A conservative approach to estimating S_{AGV} would be to simply determine the area covered by an AGV at a given point in all possible orientations. An alternative approach could consider the kinematic constraints of the AGV as well when moving towards a point. Such an approach would be necessary for highly constrained, non-holonomic AGVs. However, since this is mostly dependent on the kinematic layout of the AGV, these challenges are considered out of the scope of this chapter.

3-2-2 Reverse *Type 2* Dependencies

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

Let us first 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 $v_i^k \in \mathcal{V}_{\text{ADG}}$ is expected to 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 $v_i^k \in \mathcal{V}_{\text{ADG}}$ respectively. With regards to the status of ADG vertex

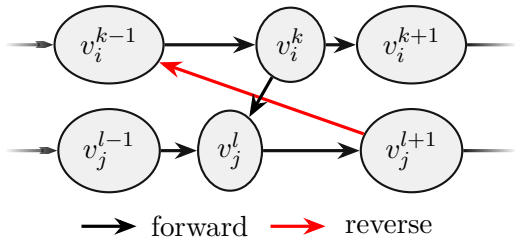


Figure 3-4: A subset of an ADG with a dependency (black) and its reverse (red)

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 $v \in \mathcal{V}_{\text{ADG}}$.

Definition 3.2 states that a dependency and its reverse contain the same collision avoidance constraints, but with a reversed AGV ordering. Lemma 3.1 can be used to obtain a dependency which conforms to Definition 3.2. Lemma 3.1 is illustrated graphically in Figure 3-4.

Definition 3.2 (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 $t_s(v_i^k) \geq t_g(v_j^l)$. If d has a reverse d', the pair (d, d') is referred to as a dependency pair.*

Lemma 3.1 (Reverse *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. 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 3.2. □

Based on the spatially exclusive ADG as defined in Definition 3.1, we are guaranteed that reverse dependencies maintain sufficient collision avoidance constraints since adjacent vertices in $\mathcal{V}_{\text{ADG}}$ refer to spatially different locations.

3-2-3 Introducing the Switchable Action Dependency Graph

Having introduced reverse *Type 2* dependencies in Section 3-2-2, 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 as shown in Definition 3.2. 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.

With this in mind, we are finally equipped to formally introduce the core contribution of this chapter, which we call the Switchable Action Dependency Graph (SADG), defined in Definition 3.3. Given a MAPF solution $\mathcal{P}$, we can construct an SADG using Algorithm 3. Unlike the originally proposed ADG algorithm, Algorithm 3 ensures that subsequent plan tuples which are not spatially exclusive are contained within a single ADG vertex, cf. line 8 of the algorithm. Referring to Algorithm 3, we introduce $plan(v_i^k)$ which returns the sequence of plan tuples $\{p_1, \dots, p_q\}$ for $v_i^k \in \mathcal{V}_{\text{ADG}}$. 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. Despite these modifications, Algorithm 3 maintains the original algorithm's time complexity of $\mathcal{O}(N^2\bar{n}^2)$ where $\bar{n} = \max_i N_i$.

Definition 3.3 (Switchable Action Dependency Graph) *Let a spatially exclusive ADG as in Definition 3.1 contain m_T forward-reverse dependency pairs determined using Definition 3.2. A Switchable Action Dependency Graph 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 $\boldsymbol{b} = \{b_1, \dots, b_{m_T}\}$, where $b_m = 0$ and $b_m = 1$ imply selecting the forward and reverse dependency of pair m respectively, $m \in \{1, \dots, m_T\}$.*

3-2-4 Plan Execution Guarantees of the SADG

Having formally introduced the SADG, the next step is to establish the formal guarantees of this concept regarding plan execution and completion. The objective is to prove that the SADG enables collision- and deadlock-free plan execution despite the AGVs being subjected to finite delays, while simultaneously allowing switching of dependencies to lower the cumulative plan completion time.

The approach to proving this property is done in two steps. First, we consider a spatially exclusive ADG without switching, as defined in Definition 3.1, and show that plan completion will be collision-free and the plan will be completed in finite time as long as the AGVs are only subjected to a finite number of delays, each of finite length. This is shown in Proposition 3.1. Note that even though Proposition 3.1 makes use of Algorithm 3, which is an SADG construction algorithm, it is still possible to discuss properties of a spatially exclusive ADG since Algorithm 3 yields such an ADG in the case that no switching is performed.

In the second step, we extend this idea to the SADG in Corollary 3.1. We provide the conditions under which switching should occur to maintain collision- and deadlock-free plan execution.

Proposition 3.1 (Collision- and deadlock-free ADG plan execution) *Consider a spatially exclusive ADG, $\mathcal{G}_{ADG}$, constructed from a MAPF solution as defined in Definition 2.1, using Algorithm 3, satisfying Assumption 2.1. 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. 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 $i, j \in \{1, \dots, N\}, i \neq j$. By lines 1-14 of Algorithm 3, 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 16-23 of Algorithm 3, 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 $p = s(v_j^l) = g(v_i^k)$

Algorithm 3 SADG construction

Input: MAPF solution $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$
Result: SADG(b)

// Add event vertices and *Type 1* dependencies
for $i = 1$ to N **do**
  $p \leftarrow p_i^1$
  $v \leftarrow (\{p\}, staged)$
  $v_{\text{prev}} \leftarrow None$
  $k = 2$
  for $k = 2$ to N_i **do**
    Append p_i^k to $plan(v)$
    // Check for spatial exclusivity
    if $S(p) \oplus S_{\text{AGV}} \cap S(p_i^k) \oplus S_{\text{AGV}} = \emptyset$ **then**
      Add v to $\mathcal{V}_{\text{ADG}}$
      if v_{prev} not *None* **then**
        Add edge (v_{prev}, v) to $\mathcal{E}_{\text{ADG}}$
      $v_{\text{prev}} \leftarrow v$
      $p \leftarrow p_i^k$
      $v \leftarrow (\{p\}, staged)$

// Add forward-reverse dependency pairs
$\mathcal{D} = \emptyset$ // init dependency pair list
for $i = 1$ to N **do**
  for $k = 1$ to N_i **do**
    for $j = 1$ to N **do**
      if $i \neq j$ **then**
        for $l = 1$ to N_j **do**
          if $s(v_i^k) = g(v_j^l)$ **and** $\hat{t}_g(v_i^k) \leq \hat{t}_g(v_j^l)$ **then**
            Add edge $e_{fwd} = (v_i^k, v_j^l)$ to $\mathcal{E}_{\text{ADG}}$
            Set edge to active
            // Add reverse *Type 2* dependency
            if $v_i^{k-1} \in \mathcal{V}_{\text{ADG}}$ **and** $v_j^{l+1} \in \mathcal{V}_{\text{ADG}}$ **then**
              Add edge $e_{rev} = (v_j^{l+1}, v_i^{k-1})$ to $\mathcal{E}_{\text{ADG}}$
              Set edge to inactive
            else
              $e_{rev} \leftarrow$ None
            Append (e_{fwd}, e_{rev}) to $\mathcal{D}$

return SADG

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 21 of Algorithm 3 the Type 2 dependency* (v_i^k, v_j^l) *guarantees* $t_s(v_i^k) > t_g(v_j^l)$. *Since, by Assumption 2.1, 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 2.1, despite a finite number of delays of finite duration, finite-time plan completion is established. This completes the proof.* □

Corollary 3.1 (SADG plan execution) *Consider an SADG,* $SADG(\boldsymbol{b})$, *as in Definition 3.3. If* $\boldsymbol{b}$ *is chosen such that* $\mathcal{G}_{ADG} = SADG(\boldsymbol{b})$ *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. By definition, any $\boldsymbol{b}$ *will guarantee collision-free plans, since at least one dependency of each forward-reverse dependency pair is selected, by Proposition 3.1. If* $\boldsymbol{b}$ *ensures* $ADG = SADG(\boldsymbol{b})$ *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 scenario with two AGVs in Figure 3-1 only shows a trivial example where identifying which switchable dependencies to activate can be done by inspection. Naturally, in more complex scenarios with a higher number of AGVs, the number of switchable dependencies is greatly increased as well as the possibilities of cycles within the resultant ADG. An example of a more complex SADG is shown in Figure 3-5. Therefore, the challenge is finding $\boldsymbol{b}$ which ensures $\mathcal{G}_{\text{ADG}} = \text{SADG}(\boldsymbol{b})$ is acyclic, for any $\text{SADG}(\boldsymbol{b})$, while simultaneously minimizing the cumulative AGV route completion times. This challenge is formalized as an Optimal Control Problem (OCP) in the next section.

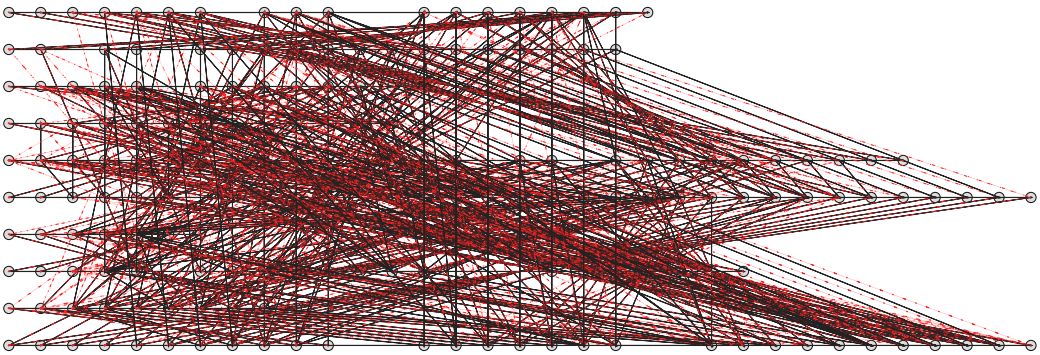


Figure 3-5: A larger SADG for 10 AGVs. This serves solely as an illustration to emphasize the complexity of SADGs even for a relatively small number of AGVs.

3-3 Shrinking Horizon Optimal Control Problem

In this section we show how an SADG can be translated into an Optimal Control Problem (OCP) based on the current progress of each AGV along each of their paths. The aim is to extract the temporal relations implicitly defined by the dependencies within an SADG and translate each into explicit temporal relations between AGV events in order to formulate the OCP.

Consider the ith AGV, *AGV*$_i$. Let the first vertex in its plan sequence of SADG vertices $\{v_i^1, \ldots, v_i^{N_i}\}$ which is either *staged* or *in-progress* be denoted by $v_i^{n_i}$. Let us then define the set $\mathcal{V}^i_{\text{staged}}$ which is a set of all the staged vertices of AGV$_i$. We then define $\mathcal{V}_{\text{staged}}$ as the union of these sets. Specifically,

$$\mathcal{V}_{\text{staged}} = \bigcup_{i=1}^{N} \mathcal{V}^i_{\text{staged}} \quad \text{where} \quad \mathcal{V}^i_{\text{staged}} = \{v_i^{n_i}, \ldots, v_i^{N_i}\}. \tag{3-2}$$

The OCP must consider *Type 1* dependencies and *Type 2* dependency pairs. These will be considered separately.

3-3-1 Type 1 Dependencies

This results in a sequence of staged vertices $\{v_i^{n_i}, \ldots, v_i^{N_i}\}$ for AGV$_i$. Let $\tau(v)$ be the estimated duration it will take an AGV to complete the transition specified within v. For example, $\tau(v)$ can be derived by considering the path length specified by v and the expected AGV velocity while traversing that path. The temporal constraints which are implied by *Type 1* dependencies can be expressed by the following linear inequalities

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

where $t_{i,s}$ refers to the expected time at which *AGV*$_i$ will start event $v_i^{n_i}$, which can be determined by local information based on the completion of a previous event.

3-3-2 Type 2 Dependency Pairs

According to Definition 3.3, the SADG has a set of m_T switchable dependency pairs, which encapsulate all inter-AGV dependencies. Note that if the reverse of a dependency does not

exist, the dependency pair is simply $(e_{fwd}, None)$, cf. line 28 of Algorithm 3. However, the OCP should only contain the dependencies which can still affect the ordering of AGVs.

There are different cases to consider. Iterating through the list of dependency pairs in $\mathcal{D}$, consider a dependency pair (e_{fwd}, e_{rev}), and let $e_{fwd} = (v^1_{fwd}, v^2_{fwd})$ and $e_{rev} = (v^1_{rev}, v^2_{rev})$. Finally, let $\mathcal{D}_{switchable}$ and $\mathcal{D}_{static}$ represent two lists which contain the switchable dependency-pairs and non-switchable dependency pairs respectively. These two lists are defined as follows:

Case 1: Forward and Reverse Dependency

If $v^2_{fwd} \in \mathcal{V}_{\text{staged}}$ and $v^2_{rev} \in \mathcal{V}_{\text{staged}}$, this dependency pair can be switched and can be added to $\mathcal{D}_{switchable}$. The dth switchable dependency pair can be represented by a binary variable b_d as

$$\begin{aligned} t_s(v^2_{fwd}) &\geq t_g(v^1_{fwd}) \quad \text{if } b_d = 0, \\ t_s(v^2_{rev}) &\geq t_g(v^1_{rev}) \quad \text{if } b_d = 1. \end{aligned} \tag{3-4}$$

Case 2: Only Forward Dependency

If $v^2_{fwd} \in \mathcal{V}_{\text{staged}}$ and $v^2_{rev} \notin \mathcal{V}_{\text{staged}}$, the forward dependency $e_{fwd} = (v^1_{fwd}, v^2_{fwd})$ can be added to $\mathcal{D}_{static}$. This can be represented by the temporal constraint

$$t_s(v^2_{fwd}) \geq t_g(v^1_{fwd}) \tag{3-5}$$

Case 3: No Dependency Constraints

No constraint is needed.

3-3-3 Formulation of Optimal Control Problem

Having translated the SADG into temporal constraints in (3-3) through (3-5) for $i \in \{1, \ldots, N\}$, $d \in \{1, \ldots, d_T\}$, we can formulate the OCP. The minimization of the cumulative route completion times of all AGVs can be written as

$$\begin{aligned} \min_{\boldsymbol{b},\, \boldsymbol{t}_s,\, \boldsymbol{t}_g} \quad & J(\boldsymbol{b},\ \boldsymbol{t}_s,\ \boldsymbol{t}_g) \\ \text{s.t.} \quad & (3-3)\ \forall\ i = \{1, \ldots, N\}, \\ & (3-4)\ \forall\ (e_{fwd}, e_{rev}) \in \mathcal{D}_{switchable}, \\ & (3-5)\ \forall\ e \in \mathcal{D}_{static}, \end{aligned} \tag{3-6}$$

where $\boldsymbol{b} : \{0,1\}^{d_T}$ is a vector containing all the binary variables b_d and the vectors $\boldsymbol{t}_s$ and $\boldsymbol{t}_g$ contain all the variables $t_s(v^k_i)$ and $t_g(v^k_i)$ respectively $\forall\ k \in \{1, \ldots, N_i\}, i \in \{1, \ldots, N\}$. The function $J(\cdot)$ is any positively affine function of the variables in $\boldsymbol{b}$, $\boldsymbol{t}_s$ and $\boldsymbol{t}_g$. Explicit functions for $J(\cdot)$ will be introduced in Chapter 5.

3-4 Feedback Control Scheme

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 4.

Algorithm 4 Switching ADG Feedback Scheme

1: Get goals and locations
2: Solve MAPF to obtain $\mathcal{P}$
3: Construct SADG($\boldsymbol{b}$) using Algorithm 3
4: ADG $\leftarrow$ SADG($\boldsymbol{0}$)
5: **while** Plans not done **do**
6: get current position along plans for each robot
7: $\boldsymbol{b} \leftarrow$ solve OCP in (3-6)
8: ADG $\leftarrow$ SADG($\boldsymbol{b}$)
9: Execute plans according to updated ADG

Lines 1-4 of Algorithm 4 refer to the initial MAPF formulation and SADG construction based on a set of AGVs, desired goal locations, and a roadmap. Once the initial plans have been determined, the AGVs start executing their plans, as dictated by the initial ADG cf. line 4. At each iteration, the AGVs' progress along their plans is registered by the controller and the OCP is solved based on the current progress. From the solution to the OCP, the ADG is updated, and the AGVs continue with their plans as dictated by this updated ADG.

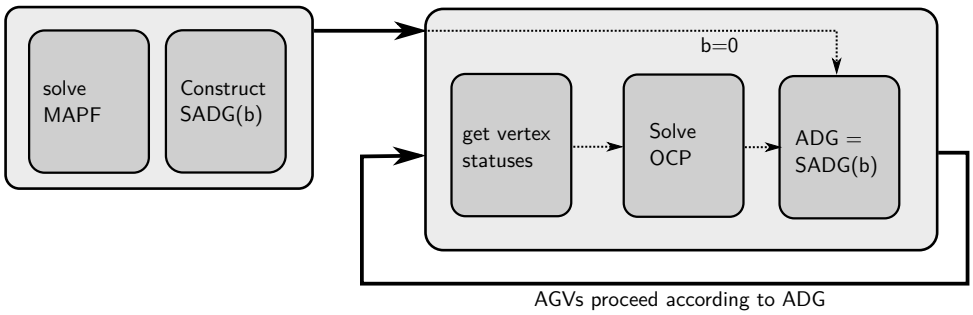


Figure 3-6: Shrinking horizon control loop diagram.

3-5 Decreasing Computational Effort

The time required to solve the OCP will directly affect the real-time applicability of this approach. In general, the complexity of the OCP increases exponentially in the number of binary variables. To render the OCP less computationally demanding, it is therefore most effective to, if possible, decrease the number of binary variables required to represent the same problem. We present two complementary methods to achieve this goal.

3-5-1 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 list $\mathcal{D}_{\text{switchable}}$ only contains dependency pairs pointing to within a receding horizon of future vertices. To carry this out, the conditions for dependency pairs in **Case 1** in Section 3-3-2 need to be adjusted as follows:

> **Case 1 RHC: Forward and Reverse Dependency**
>
> If $v^2_{fwd} \in \mathcal{V}$ and $v^2_{rev} \in \mathcal{V}$ and $t_g(v^2_{fwd}) \leq H_{\text{control}}$, this dependency pair can be switched and can be added to $\mathcal{D}_{switchable}$. The dth switchable dependency pair can be represented by a binary variable b_d as in (3-4). Conversely, if $t_g(v^2_{fwd}) > H_{\text{control}}$, only the forward dependency should be added as shown in (3-5)

An illustration of this selection of dependencies is shown in Figure 3-7. In this case, the region bounded by the switching horizon H_{control} is shaded in magenta. Note how only dependency pairs which have a forward dependency (denoted in black) pointing within this horizon are included in the optimization formulation. A dependency pair where the active dependency does not point to within this horizon (shown in green) is not added to $\mathcal{D}_{switchable}$, but rather to $\mathcal{D}_{static}$.

Note that although the selection of which dependencies to switch can be done in a receding horizon, the OCP still needs to consider all the *Type 1* dependency constraints within the SADG. This means that even though the amount of binary variables in the optimization problem can be reduced, this is still essentially a shrinking horizon control approach. This issue is discussed in more detail in Chapter 4, where a receding horizon framework for this problem is presented.

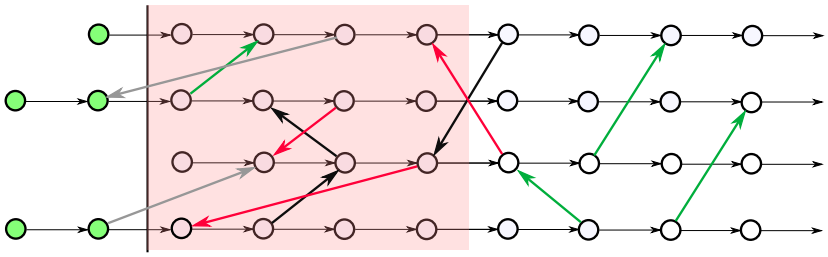


Figure 3-7: Dependency selection for a horizon of 4 vertices. Switchable dependency pairs are shown in black (forward) and red (reverse). Regular dependencies considered in the OCP are green. Dependencies not considered are grey.

3-5-2 Dependency Grouping

Consider the example of an ADG constructed from a MAPF plan for five AGVs shown in Figure 3-8. Note that multiple dependencies often form patterns, two of which are shown in Figure 3-9. These patterns are referred to as *same-direction* and *opposite-direction* dependency groups, shown in Figure 3-9a and Figure 3-9b respectively. Each of these dependency

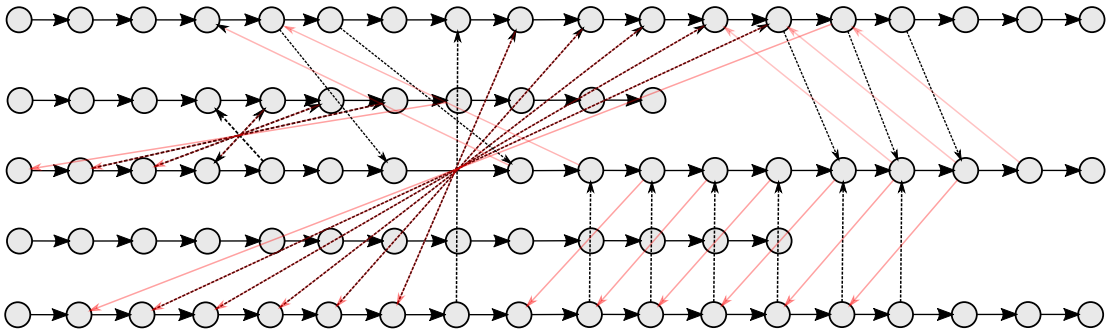
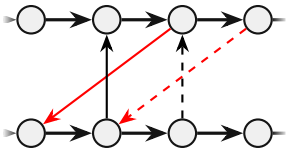
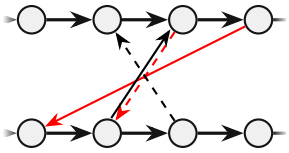


Figure 3-8: An SADG for five AGVs, clearly showing both *same* and *opposite* dependency group patterns as described in Figure 3-9.



(a) *same* dependency group.



(b) *opposite* dependency group.

Figure 3-9: Two commonly occurring dependency groups. Each dependency is either original (black) or reversed (red). Reverse and forward dependency pairings are differentiated by line styles.

groups share a common property: the resultant ADG can only be acyclic if either *all* the forward dependencies, 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 number of variables within the OCP in (3-6). Once such a dependency group has been identified, the temporal constraints can then be defined as

$$\begin{aligned} t^l_{j,s} > t^k_{i,g} \quad \forall\, (v^k_i, v^l_j) \in \mathcal{DG}_{\text{fwd}} \text{ if } b_{DG} = 0, \\ t^l_{j,s} > t^k_{i,g} \quad \forall\, (v^k_i, v^l_j) \in \mathcal{DG}_{\text{rev}} \text{ if } b_{DG} = 1, \end{aligned} \tag{3-7}$$

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.

3-6 Summary

In this chapter, we presented a novel concept called a Switchable Action Dependency Graph (SADG) which facilitates the re-ordering of AGVs based on their progress along their paths as dictated by the original MAPF solution. The SADG maintains the collision avoidance constraints of the original ADG by using the concept of switchable dependencies. The implicit constraints of the SADG can be used to formulate an OCP which, when a feasible solution is found, will guarantee the re-ordering to result in deadlock-free plan execution, thus maintaining liveness.

The next step for this work is to translate this method into a receding horizon optimization approach. In this chapter, ADG dependencies can be switched in a receding horizon fashion as discussed in Section 3-5-1, 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 such that the approach in this chapter can be applied. We address this shortcoming and present a solution to this problem in Chapter 4.

Chapter 4

Receding Horizon Control Approach

Chapter 3 introduced a feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a Switchable Action Dependency Graph (SADG). This method, however, assumed AGV plans to be of finite length. Furthermore, the number of variables considered within the resulting Optimal Control Problem (OCP) was directly related to the total number of events in the plans of the AGVs, which could potentially be significantly large. In this chapter, we consider the case where AGV plans can be of significant to infinite length, while ensuring the resultant OCP remains tractable. The result is a method which can be used to optimize persistent plans, while only considering a subset of the SADG, making the approach more feasible in a real-time implementation.

4-1 Motivating the Need for a Receding Horizon Scheme

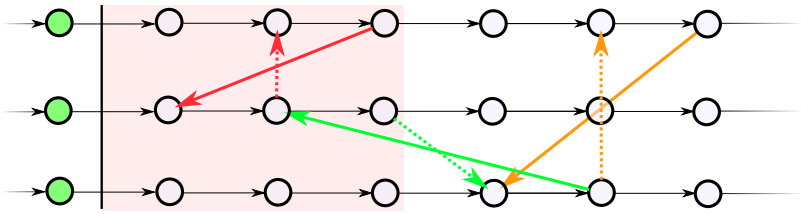
Recall from Chapter 3, where the plans $\mathcal{P}$ were used to construct an SADG. The dependencies within the SADG were translated into an OCP which was solved in a feedback scheme based on each AGV's progress along its individual plan. In this case, the OCP was based on the entire SADG since all future dependencies and events had to be considered to guarantee the resultant optimization problem would return a feasible solution that also maintains finite plan completion times for all AGVs, thus mitigating deadlocks. For longer plans, however, the resultant OCP would consist of an increasing number of variables, resulting in longer computational times at each iteration of the feedback scheme. At the limit, for infinite length plans, the OCP will become intractable, meaning the scheme in Chapter 3 can no longer be used.

These limitations motivate the need for a Receding Horizon Control (RHC) approach, which only considers events and dependencies within a pre-determined horizon to formulate the OCP. RHC schemes such as Model Predictive Control (MPC) are very popular in the control communities and a vast amount of literature is available specifically dedicated to the performance and stability analyses of such approaches [33]. The fundamentals of these MPC schemes will serve as inspiration for the development of the RHC framework in this chapter.

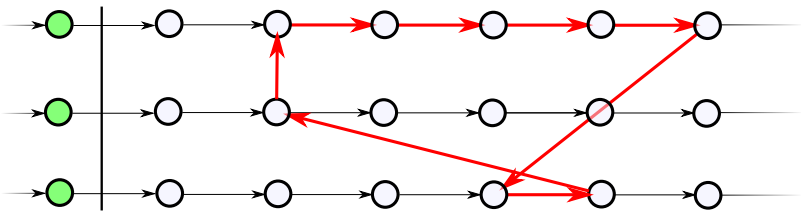
4-2 The Challenge of Receding Horizon Control

To use a RHC scheme to persistently optimize an SADG, it is necessary to only consider a subset of the SADG within a finite horizon. However, it must be ensured that optimizing over a subset of an SADG yields a solution that remains feasible for the entire SADG. This implies that we need to carefully select the edges, vertices and forward-reverse dependencies considered within the OCP. This is in contrast to typical RHC schemes for continuous-time systems, where the control horizon is defined *a-priori* and typically remains constant throughout the execution of this feedback scheme. To help illustrate this point, consider Example 4.1 where an illustrative SADG is depicted in Figure 4-1a. The optimization only considers the SADG within the shaded region which only includes the red dependency pair. In this example, the inactive red dependency is selected. Despite this selection being optimal for the SADG considered within the shaded region, it causes a cycle to appear in the resultant ADG due to the other dependency pairs not considered within the OCP, as seen in Figure 4-1b.

Example 4.1 Feasible solution to subset of SADG with infeasible ADG.
Consider the SADG depicted in Figure 4-1a. If the optimization only considers vertices within the control horizon, indicated by the shaded region, the optimal (feasible) solution is $b_{red} = 1$. *Since the green and orange dependencies are not considered, their associated variables remain* $b_{green} = 0, b_{orange} = 0$. *The result, however, is a solution which is infeasible when considering the entire resultant ADG. This is indicated graphically in Figure 4-1b. The induced cycle in the ADG is emphasized in red.*



(a) Example of an SADG with active (solid) and inactive (dashed) dependencies. Only the vertices within the magenta-shaded region are considered in the optimization.



(b) If the red dependency is switched, and the green and orange dependencies are not, the result is a cyclic $\mathcal{G}_{\text{ADG}}$ highlighted in red.

Figure 4-1: Example of a solution to an SADG where the resultant ADG from the SADG within the control horizon is acyclic (therefore feasible), but the entire ADG is cyclic.

In summary, the challenge is to determine a method to determine which portion of the SADG needs to be considered within the OCP to ensure the optimal solution of the OCP generated from this SADG portion remains feasible for the entire SADG.

4-3 Feasible SADG Subsets

In this section, we provide a systematic manner in which to address the challenge illustrated by Example 4.1. Recall from Chapter 3 that an SADG is a mapping from a binary vector to an ADG and can be visualized as a directed graph where a subset of the edges (the switchable *Type 2* dependencies) are either *active* or *inactive*. As is typical for RHC schemes, the idea is to only consider a finite, initial portion of the SADG within the OCP. To this end, we formally introduce the subset of an SADG in Definition 4.1.

Definition 4.1 (SADG Subset) *Let $S(\boldsymbol{b})$ be an SADG as defined in Definition 3.3. For a given $\boldsymbol{b}$, let the implicit ADG defined by $S(\boldsymbol{b})$ be denoted by $\mathcal{G}_{ADG} = (\mathcal{V}, \mathcal{E})$, as well as the set of m_T forward-reverse dependency pairs. A subset of $S(\boldsymbol{b})$ is an SADG, denoted by $\bar{S}(\bar{\boldsymbol{b}})$, which is a mapping $SADG(\boldsymbol{b}) : \{0,1\}^{\overline{m}_T} \mapsto \overline{\mathcal{G}}_{ADG} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$ where $0 \leq \overline{m}_T \leq m_T$, $\overline{\mathcal{V}} \subseteq \mathcal{V}$ and $\overline{\mathcal{E}} \subseteq \mathcal{E}$.*

Note that the subset of an SADG is an SADG itself. As a next step, we discern between admissible and inadmissible SADG subsets. An admissible SADG subset is formally defined in Definition 4.2. Recall that the goal is to ensure that the OCP derived from this SADG subset will have an optimal solution which is feasible for the entire SADG. If this is the case, such an SADG subset will be referred to as *admissible*, and can be determined using Algorithm 5 for a given SADG and horizon time H_{control}.

Definition 4.2 (Admissible SADG subset) *Given an SADG $S(\boldsymbol{b})$, an admissible SADG subset $\bar{S}(\bar{\boldsymbol{b}})$ is an SADG subset of $S(\boldsymbol{b})$ as per Definition 4.1 such that if $\bar{\boldsymbol{b}}$ is a feasible solution for $\bar{S}(\bar{\boldsymbol{b}})$, $\boldsymbol{b} = \{\bar{\boldsymbol{b}}, \boldsymbol{0}\}$ is a feasible solution for $S(\boldsymbol{b})$, where $\boldsymbol{0} = \{0\}^{\bar{m}_T - m_T}$.*

In the next step, we present Lemma 4.1, supported by the schematic in Figure 4-2, which states that if two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are acyclic, and any group of edges connecting the two graphs are directed only from any vertex $v_1 \in \mathcal{V}_1$ to any vertex $v_2 \in \mathcal{V}_2$, then the entire graph consisting of $\mathcal{G}_1$ and $\mathcal{G}_2$ is acyclic. The idea is to apply this result to two graphs which make up portions of an SADG, allowing the construction of an *admissible* SADG subset.

Lemma 4.1 (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 $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$. 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. □

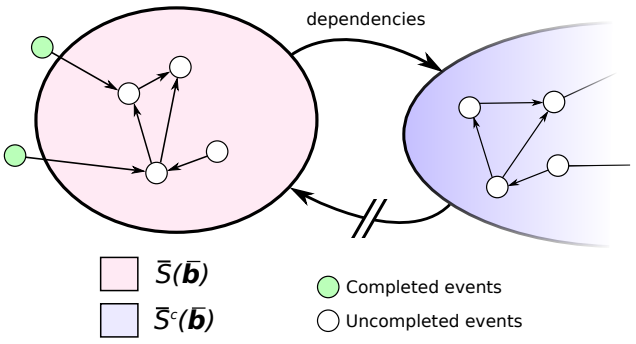


Figure 4-2: Graphical illustration of how Lemma 4.1 can be applied to two graphs which make up an ADG.

Algorithm 5 Determining an admissible SADG subset

Input: $S(\boldsymbol{b})$, H_{control}
Result: $\bar{S}(\bar{\boldsymbol{b}})$

// Loop through m_T switchable dependency pairs of entire SADG
$\mathcal{M} = \emptyset$
$\mathcal{V}_{\text{control}} \leftarrow \text{getVerticesInHorizon}(H_{\text{control}})$
for $m = 1$ to m_T **do**
  $(e_{\text{active}}, e_{\text{inactive}}) \leftarrow \text{getDependencyPair}(m)$
  $v_{active} \leftarrow \text{getHead}(e_{\text{active}})$
  if $\bar{t}_g(v_{active}) < H_{\text{control}}$ **then**
    $\mathcal{V}_{\text{dependency}} \leftarrow \text{getVertexSet}(m)$
    $\mathcal{V}_{\text{control}} \leftarrow \mathcal{V}_{\text{control}} \cup \mathcal{V}_{\text{dependency}}$
    $\mathcal{M} \leftarrow \mathcal{M} \cup (e_{\text{active}}, e_{\text{inactive}})$

// Include vertices until no dependencies point back into $\mathcal{V}_{\text{control}}$
$\mathcal{V}_{\text{check}} \leftarrow \mathcal{V}_{\text{control}}$
while $\mathcal{V}_{\text{check}}$ not empty **do**
  $v \leftarrow \text{pop}(\mathcal{V}_{\text{check}})$
  $\mathcal{V}_{\text{inbound}} \leftarrow \text{getTailofActiveInboundEdges}(v)$
  for $v_{\text{inbound}} \in \mathcal{V}_{\text{inbound}}$ **do**
    if $v_{\text{inbound}} \notin \mathcal{V}_{\text{control}}$ **then**
      Add v_{inbound} to $\mathcal{V}_{\text{check}}$ and $\mathcal{V}_{\text{control}}$

// Create subset SADG from $\mathcal{V}_{\text{control}}$ and $\bar{m}_T$ switchable dependency pairs
$\bar{S}(\bar{\boldsymbol{b}}) \leftarrow \text{constructSADG}(\mathcal{M}, \mathcal{V}_{\text{control}})$
return $\bar{S}(\bar{\boldsymbol{b}})$

Based on the previously introduced concepts, we are ready to introduce Algorithm 5, which provides a systematic manner to determine an admissible SADG subset given a pre-determined control horizon H_{control} based on an SADG of significant length. Concerning Algorithm 5, we introduce some additional notation and functions. Let the current time be denoted by t_c and let us introduce the simplifying notation $\bar{t}_s() = t_s() - t_c$ and $\bar{t}_g() = t_g() - t_c$ permitting the use of relative time so the current time is 0. The function *getVerticesInHorizon*(H_{control}) determines all the vertices in the SADG where $t_g(v) \leq H_{\text{control}}$ for $v \in \mathcal{V}$. The function *getDependencyPair*(m) returns the m^{th} dependency pair of the $SADG$ represented by an active and inactive edge. The function *getHead*(e) returns the vertex to which the directed edge e is pointing. The function *getVertexSet*(m) returns a set of vertices containing all the vertices specified within both the active and inactive edges for the m^{th} dependency pair. The function *getTailofActiveInboundEdges*(v) returns the set of vertices up until the tail vertex of all edges pointing towards v. This means that for every edge pointing towards v, all vertices up until the tail of this edge is added to the set.

Algorithm 5 Illustration and Walk-Through

As an intuitive illustration of the approach defined within Algorithm 5, consider the five sequential figures shown in Figure 4-3 showing how an admissible SADG subset is constructed from an existing SADG given a control horizon H_{control}. As per line 1, the vertices within the control horizon are selected, corresponding to Figure 4-3a. Following this, all dependency groups with a head pointing to a vertex within the region considered by H_{control} is selected as in Figure 4-3b, which corresponds to lines 3 to 9. At this point, the result of Lemma 4.1 is utilized. As in lines 10 to 16, all dependencies pointing to a vertex contained within the expanded control horizon and the associated vertices are added to the control horizon in a recursive fashion until the region of interest is constructed in such a manner that no dependencies are pointing to within it. This is shown in Figure 4-3c and 4-3d, where the highlighted gold dependency is an example of such a dependency. Finally, as in Figure 4-3e, a subset of the SADG is identified, from which the admissible SADG can be constructed. This corresponds to lines 17 to 18.

Computational Complexity of Algorithm 5

Since Algorithm 5 will be run at each time-step, it is vital that it is computationally tractable. Line 1 to 9 have a linear time complexity of $\mathcal{O}(N + m_T)$, since the function *getVerticesInHorizon*($\cdot$) is a simple graph traversal procedure which can be performed in linear time, and the switchable dependency pairs are evaluated in a single for-loop. The only potentially time-consuming portion of the algorithm lies in lines 11 through 16, since this consists of the recursive portion of the function. However, since all forward dependencies initially point away from the control horizon at initialization, the recursive step will not take considerable time. Once a forward dependency points inwards, towards the control horizon region, it will most likely be switched since it is constraining an AGV from advancing with its goals. An experimental validation of this statement is shown in the statistical simulation in Chapter 6.

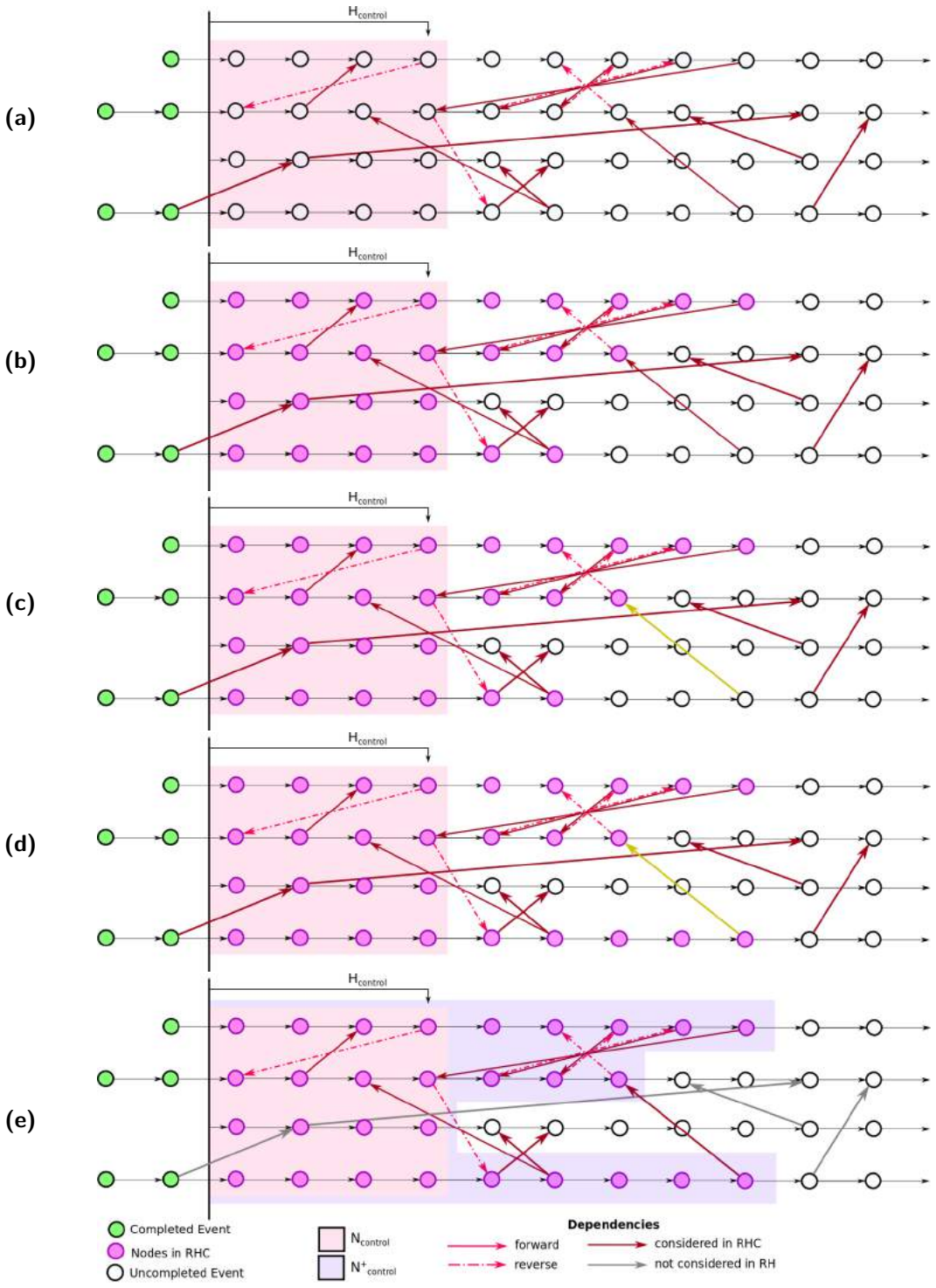


Figure 4-3: Graphical illustrations of the steps taken by Algorithm 5. (a) The vertices within the horizon are defined; (b) all dependency groups pointing into the horizon are selected; (c) dependencies pointing to the additionally added vertices are identified (highlighted in gold) and (d) associated vertices selected; (e) the vertices, static and switchable dependencies within this horizon are determined such that an admissible SADG subset can be constructed.

Feasible SADG Subset Guarantees

As a final step, we will apply the aforementioned concepts in order to determine the subset SADG with which to define the receding horizon OCP. To facilitate the explanation and proof of Proposition 4.1, we introduce the complement of an SADG subset in Definition 4.3. This is essentially the remaining portion of an SADG once a subset thereof is being considered.

Definition 4.3 (Complementary SADG Subset) *Given an SADG $S(\boldsymbol{b})$ and an SADG subset $\bar{S}(\bar{\boldsymbol{b}})$ as defined in Definition 4.1. The complementary SADG subset to $\bar{S}(\bar{\boldsymbol{b}})$ is denoted $\bar{S}^c$ and is itself a SADG subset defined by the mapping $SADG(\boldsymbol{b}) : \{0,1\}^{\overline{m}_T^c} \mapsto \overline{\mathcal{G}}^c_{ADG} = (\overline{\mathcal{V}}^c, \overline{\mathcal{E}}^c)$ where $\overline{m}_T^c = m_T - \overline{m}_T$, $\overline{\mathcal{V}}^c = \mathcal{V}/\overline{\mathcal{V}}$ and $\overline{\mathcal{E}}^c = \mathcal{E}/\overline{\mathcal{E}}$.*

Using Definition 4.3, we are now equipped to present the final result in Proposition 4.1. Proposition 4.1 states that a feasible solution to a subset SADG will ensure the entire resultant ADG from the whole SADG is acyclic, and therefore feasible, as long as the subset SADG is admissible, as determined using Algorithm 5.

Proposition 4.1 (Feasible SADG from a feasible SADG subset solution) *Consider an SADG $S(\boldsymbol{b})$ as defined in Definition 3.3 and a control horizon $H_{control} \in \mathbb{R}_{\geq 0}$. If an SADG subset $\bar{S}(\bar{\boldsymbol{b}})$ is extracted from $S(\boldsymbol{b})$ using Algorithm 5, a feasible solution to $\bar{S}(\bar{\boldsymbol{b}})$ will result in an acyclic ADG for the entire SADG $S(\boldsymbol{b})$.*

Proof. Consider the SADG subset $\bar{S}(\bar{\boldsymbol{b}})$, assuming a feasible solution $\bar{\boldsymbol{b}}$, the resultant ADG from $\bar{S}(\bar{\boldsymbol{b}})$ necessarily requires the ADG to be acyclic. Since the SADG subset is determined using Algorithm 5, no dependencies point from the complementary subset SADG $\bar{S}^c$ to the subset SADG $\bar{S}(\bar{\boldsymbol{b}})$. Considering the complementary subset SADG, the trivial solution $\bar{S}^c(0)$ is acyclic by definition. The result is an $A\bar{D}G$ and $A\bar{D}G^{\mathcal{C}}$ which are both acyclic, and the only edges connecting the two are from $A\bar{D}G$ to $A\bar{D}G^{\mathcal{C}}$. By Lemma 4.1, the resultant ADG is acyclic, meaning that the overall SADG solution is feasible. □

4-4 Reformulation of the Optimal Control Problem

In this section, we formulate the OCP such that the solution thereof can be used within a receding horizon control scheme. Recall from Section 3-3 that for each AGV_i, we let the first *staged* or *in-progress* vertex in its plan sequence of SADG vertices $\{v_i^1, \ldots, v_i^{N_i}\}$ be denoted by $v_i^{n_i}$. For a given control horizon H_{control}, we can construct an SADG subset $\bar{S}(\bar{\boldsymbol{b}})$ using Algorithm 5. We additionally introduce $v_i^{n_f}$, which denotes the last vertex in the plan sequence of AGV_i considered within $\bar{S}(\bar{\boldsymbol{b}})$. Note the omission an index i in n_f for clarity. Let us then define the set $\mathcal{V}^{RHC,i}_{\text{staged}}$ which is a set of all the staged vertices of AGV_i. We then define $\mathcal{V}^{RHC}_{\text{staged}}$ as the union of these sets. Specifically,

$$\mathcal{V}^{RHC}_{\text{staged}} = \bigcup_{i=1}^{N} \mathcal{V}^{RHC,i}_{\text{staged}} \quad \text{where} \quad \mathcal{V}^{RHC,i}_{\text{staged}} = \{v_i^{n_i}, \ldots, v_i^{n_f}\}. \tag{4-1}$$

$\mathcal{V}^{RHC}_{\text{staged}}$ is a set containing all the vertices within the subset SADG. $\mathcal{V}^{RHC}_{\text{staged}}$ can also be used to determine which dependency pairs need to be considered within this OCP. This will be shown next by separately considering *Type 1* dependencies and *Type 2* dependency pairs.

4-4-1 Type 1 Dependencies

This results in a sequence of staged vertices $\{v_i^{n_i}, \ldots, v_i^{n_f}\}$ for AGV$_i$. Let $\tau(v)$ be the estimated duration it will take an AGV to complete the transition specified within $v \in \mathcal{V}_{\text{ADG}}$. For example, $\tau(v)$ can be derived by considering the path length specified by v and the expected AGV velocity while traversing that path. The *Type 1* constraints can be represented by the following linear inequalities

$$\begin{aligned} t_s(v_i^{n_i}) &= t_{i,s}, \\ t_g(v_i^{n_i}) &\geq t_s(v_i^{n_i}) + \tau(v_i^{n_i}), \\ t_s(v_i^{n_i+1}) &\geq t_g(v_i^{n_i}), \\ t_g(v_i^{n_i+1}) &\geq t_s(v_i^{n_i+1}) + \tau(v_i^{n_i+1}), \\ t_s(v_i^{n_i+2}) &\geq t_g(v_i^{n_i+1}), \\ \vdots \qquad & \qquad \vdots \\ t_s(v_i^{n_f}) &\geq t_g(v_i^{n_f-1}), \\ t_g(v_i^{n_f}) &\geq t_s(v_i^{n_f}) + \tau(v_i^{n_f}). \end{aligned} \tag{4-2}$$

4-4-2 Type 2 Dependency Pairs

Regarding the *Type 2* dependencies, there are three different cases to consider. Iterating through the list of dependency pairs in $\mathcal{D}$, consider a dependency pair $(e_{\text{fwd}}, e_{\text{rev}})$, and let $e_{\text{fwd}} = (v_{\text{fwd}}^1, v_{\text{fwd}}^2)$ and $e_{\text{rev}} = (v_{\text{rev}}^1, v_{\text{rev}}^2)$. Finally, let $\mathcal{D}_{\text{switchable}}^{RHC}$ and $\mathcal{D}_{\text{static}}^{RHC}$ represent two lists which contain the switchable dependency-pairs and non-switchable dependency pairs respectively. These two lists are defined as follows:

Case 1: Forward and Reverse Dependency

If $v_{\text{fwd}}^2 \in \mathcal{V}_{\text{staged}}^{RHC}$ and $v_{\text{rev}}^2 \in \mathcal{V}_{\text{staged}}^{RHC}$, this dependency pair can be switched and can be added to $\mathcal{D}_{\text{switchable}}^{RHC}$. The dth switchable dependency pair can be represented by a binary variable b_d as

$$\begin{aligned} t_s(v_{\text{fwd}}^2) &\geq t_g(v_{\text{fwd}}^1) \quad \text{if } b_d = 0, \\ t_s(v_{\text{rev}}^2) &\geq t_g(v_{\text{rev}}^1) \quad \text{if } b_d = 1. \end{aligned} \tag{4-3}$$

Case 2: Only Forward Dependency

If $v_{\text{fwd}}^2 \in \mathcal{V}_{\text{staged}}^{RHC}$ and $v_{\text{rev}}^2 \notin \mathcal{V}_{\text{staged}}^{RHC}$, the forward dependency $e_{\text{fwd}} = (v_{\text{fwd}}^1, v_{\text{fwd}}^2)$ can be added to $\mathcal{D}_{\text{static}}^{RHC}$. This can be represented by the temporal constraint

$$t_s(v_{\text{fwd}}^2) \geq t_g(v_{\text{fwd}}^1). \tag{4-4}$$

Case 3: No Dependency Constraints
No constraint is needed.

4-4-3 Formulation of Optimal Control Problem

Having translated the SADG into temporal constraints in (4-2) through (4-4) for $i \in \{1, \dots, N\}$, $d \in \{1, \dots, d_T\}$. Minimizing the cumulative route completion time of all AGVs is formulated as the following optimization problem

$$\begin{aligned} \min_{\boldsymbol{b},\, \boldsymbol{t}_s,\, \boldsymbol{t}_g} \quad & J(\boldsymbol{b},\ \boldsymbol{t}_s,\ \boldsymbol{t}_g), \\ \text{s.t.} \quad & (4-2)\ \forall\ i = \{1, \dots, N\}, \\ & (4-3)\ \forall\ (e_{\text{fwd}}, e_{\text{rev}}) \in \mathcal{D}^{RHC}_{\text{switchable}}, \\ & (4-4)\ \forall\ e \in \mathcal{D}^{RHC}_{\text{static}}, \end{aligned} \tag{4-5}$$

where $\boldsymbol{b} : \{0,1\}^{d_T}$ is a vector containing all the binary variables b_d and the vectors $\boldsymbol{t}_s$ and $\boldsymbol{t}_g$ contain all the variables represented by $t_s(v_i^k)$ and $t_g(v_i^k)$ respectively $\forall\ k \in \{1, \dots, N_i\}, i \in \{1, \dots, N\}$. Once again, the function $J(\cdot)$ is any positively affine function of the variables in $\boldsymbol{b}$, $\boldsymbol{t}_s$ and $\boldsymbol{t}_g$. Explicit functions for $J(\cdot)$ will be introduced in Chapter 5.

4-5 Receding Horizon Control Scheme

Based on the OCP presented in Section 4-4, we are now able to present the complete RHC feedback scheme. This framework is presented in Algorithm 6. The key difference between the shrinking horizon and receding horizon feedback schemes, described in Algorithm 4 and Algorithm 6 respectively, is the use of Algorithm 5 in Algorithm 6 cf. line 6.

Algorithm 6 Switching ADG RHC Feedback Scheme

1: Get goals and locations
2: Solve MAPF to obtain $\mathcal{P}$
3: Construct SADG($\boldsymbol{b}$) using Algorithm 3
4: **while** Plans not done **do**
5: get current position along plans for each AGV
6: Extract $\bar{S}(\bar{\boldsymbol{b}})$ using Algorithm 5
7: Construct OCP from $\bar{S}(b)$ as in Section 4-4
8: $\boldsymbol{b} \leftarrow$ solve OCP in (4-5)
9: ADG $\leftarrow$ SADG($\boldsymbol{b}$)
10: Execute plans according to updated ADG

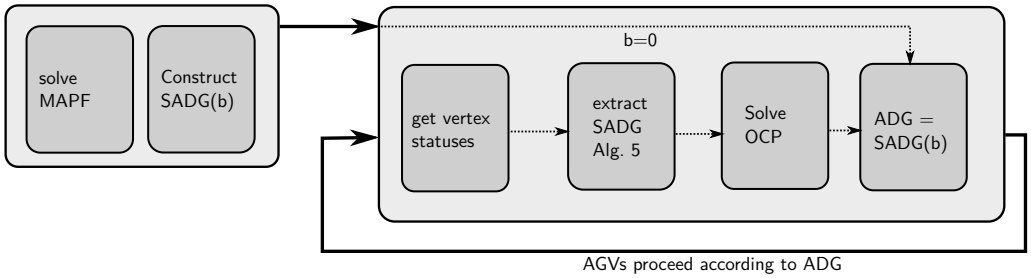


Figure 4-4: Receding horizon control loop diagram

4-6 Persistent Control Framework

The RHC scheme in Section 4-5 directly translates to a persistent planning setting in the case that the MAPF is re-solved based on updated AGV goal positions. The control architecture is largely similar to that shown in Figure 4-4, with the addition of a *re-plan* state which, when triggered, can by-pass the inner control loop and obtain new goals for the AGVs, re-solve the MAPF, update the SADG and then commence with the inner control strategy once more. This architecture is shown in Figure 4-5.

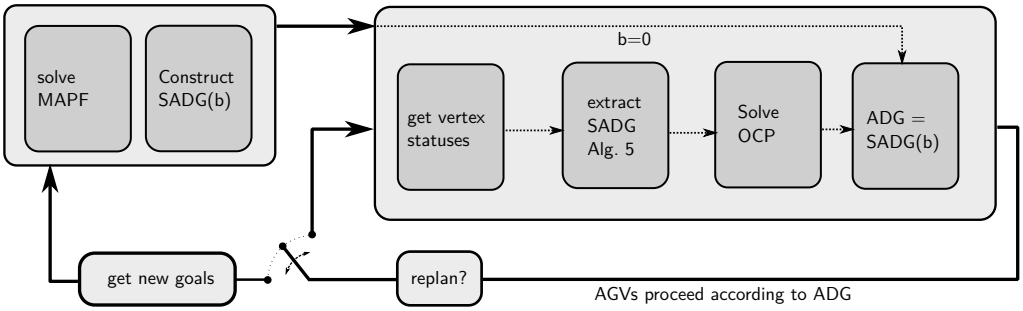


Figure 4-5: Persistent planning with receding horizon control loop diagram

4-7 Summary

In Chapter 3 we introduced a novel data-structure which was used as the foundation for a shrinking horizon feedback scheme. This scheme facilitated the re-ordering of AGVs while maintaining collision- and deadlock-free guarantees, but required the full length of the plans to be known *a-priori* such that the optimization scheme could guarantee a feasible solution to OCP in (3-6) is also feasible for the plan execution scheme. In the case of persistent planning, however, re-planning can result in new plans being appended to an existing plan, which is initially not considered within the OCP. Dynamically changing plans cannot be addressed using the previously developed shrinking horizon feedback scheme, introducing the necessity for a receding horizon scheme, which is presented in this chapter.

Initially, the challenges related to the development of a receding horizon scheme using a directional graph such as an SADG are presented. We then introduce an algorithm capable

of determining a subset of the SADG to consider within the OCP such that the optimal solution to the resultant subset OCP is still feasible for the entire SADG. This receding horizon approach not only reduces the computational effort required to solve the resulting OCP at each time step, but also allows the optimization of persistent plans with the guarantee of keeping the global ordering feasible as the AGVs complete their plans.

Chapter 5

Optimization and Recursive Feasibility

In this chapter, we consider a Mixed-Integer Linear Program (MILP) to solve the optimal control problems formulated in Chapters 3 and 4. Having introduced the SADG, we now formulate an optimization problem which can be used to determine $\boldsymbol{b}$ such that the resultant ADG is acyclic, while minimizing cumulative AGV route completion times. The result is a 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. Various cost functions are proposed which prioritize different properties of the obtained solution. Based on the inherent structure of the SADG, various heuristic methods are proposed which can be used to help find solutions faster. Finally, an analysis of the recursive feasibility of the shrinking and receding horizon schemes is presented.

5-1 Formulation as a Mixed-Integer Linear Program

For this section, we reserve the discussion for the method presented in Chapter 4, since the obtained formulation translates directly to the scheme in Chapter 3. Consider the OCP in (4-5). This optimization problem consists of a combination of linear constraints, as well as binary states representing the switching of dependencies. These components can be represented within an MILP where the so-called big-M approach can be used to generate the binary decision of selecting dependencies for a forward-reverse dependency switching pair. Consider the general form of an MILP as described below

$$\begin{aligned} &\min_{x} \ c^T x, \\ &\text{s.t. } Ax \leq b, \end{aligned} \tag{5-1}$$

where $x \in \mathbb{R}^{n_c} \times \{0,1\}^{n_d}$, $n = n_c + n_d$, $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^{m \times 1}$ and $c \in \mathbb{R}^{n \times 1}$. The objective of this section is to determine the variables which make up x, A, b and c. The *Type 1* dependency constraints are obtained directly from the temporal relations in Section 3-3 and Section 4-4

respectively. We show the equations for the RHC approach in Section 4-4, as this easily translates the approach in Section 3-3. Let us introduce the optimization variable $t^k_{i,s}$ which, once a solution to the optimization problem is determined, will be equal to $t_s(v^k_i)$. The same relation applies to the optimization variable $t^k_{i,g}$ and $t_g(v^k_i)$. The *Type 1* dependencies can be taken directly from (4-2) and written as

$$\begin{aligned} t^{n_i}_{i,s} &= t_{i,s}, \\ t^{n_i}_{i,g} &\geq t^{n_i}_{i,s} + \tau(v^{n_i}_i), \\ t^{n_i+1}_{i,s} &\geq t^{n_i}_{i,g}, \\ t^{n_i+1}_{i,g} &\geq t^{n_i+1}_{i,s} + \tau(v^{n_i+1}_i), \\ t^{n_i+2}_{i,s} &\geq t^{n_i+1}_{i,g}, \\ \vdots \qquad & \qquad \vdots \\ t^{n_f}_{i,s} &\geq t^{n_f-1}_{i,g}, \\ t^{n_f}_{i,g} &\geq t^{n_f}_{i,s} + \tau(v^{n_f}_i). \end{aligned} \tag{5-2}$$

Recall the definition of the set $\mathcal{D}^{RHC}_{\text{static}}$ which contains all the static dependencies within the SADG. Given a dependency $e \in \mathcal{D}^{RHC}_{\text{static}}$, where $e = (v^k_i, v^l_j)$, this *Type 2* dependency can be written as

$$t^l_{j,s} \geq t^k_{i,g} \;\; \forall \;\; e \;\in\; \mathcal{D}^{RHC}_{\text{static}}. \tag{5-3}$$

Finally, we consider the switchable dependency pairs in $\mathcal{D}^{RHC}_{\text{switchable}}$. As previously stated, the set $\mathcal{D}^{RHC}_{\text{switchable}}$ has a cardinality of d. We introduce the Boolean variable b_d for the dth dependency pair within $\mathcal{D}^{RHC}_{\text{switchable}}$. Using the big-M method, we can essentially *activate* or *deactivate* a constraint based on the binary value of b_d. Consider the dependency pair $(e_{\text{fwd}}, e_{\text{rev}}) \in \mathcal{D}^{RHC}_{\text{switchable}}$, with $e_{\text{fwd}} = (v^1_{\text{fwd}}, v^2_{\text{fwd}})$ and $e_{\text{rev}} = (v^1_{\text{rev}}, v^2_{\text{rev}})$. With a slight abuse of notation, where the optimization variable $t^1_{s,\text{fwd}}$ represents the value of $t_s(v^1_{\text{fwd}})$, the constraints can be written as

$$\begin{aligned} t^1_{s,\text{fwd}} &\geq t^2_{g,\text{fwd}} - Mb_d, \\ t^2_{s,\text{rev}} &\geq t^1_{g,\text{rev}} - (1-b_d)M \quad \forall \;\; (e_{\text{fwd}}, e_{\text{rev}}) \in \mathcal{D}^{RHC}_{\text{switchable}}, \end{aligned} \tag{5-4}$$

where M is a large, positive constant such that $M > \max_i t^{N_i}_{i,g}$. Note that $\max_i t^{N_i}_{i,g}$ 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. As the cost function, we consider the cumulative time taken for each AGV to reach its goal vertex. The cost function can therefore be written as

$$J(\cdot) = \sum_{i=1}^{N} t^{n_f}_{i,g}. \tag{5-5}$$

5-2 Different Cost Functions

We consider three different cost functions and explain the motivation behind them. Recall the basic optimization for an MILP as shown in (5-1).

5-2-1 Makespan optimization

As is often done in the MAPF formulations, the maximum route completion time of all the AGVs can be used as the objective function. The idea is to ensure that all AGVs reach their respective goals as quickly as possible.

$$J(\cdot) = \max\left(t_{1,g}^{n_f}, \ldots, t_{N,g}^{n_f}\right). \tag{5-6}$$

Such a maximization can be written by introducing a new optimization variable z, representing the upper bound of all the route completion times of each AGV. In this case, (5-6) can be written as

$$\begin{aligned} J(\cdot) &= \min z, \\ \text{s.t. } t_{i,g}^{n_f} &\leq z \ \ \forall \ \ i \ \ \in \ \{1, \ldots, N\}, \end{aligned} \tag{5-7}$$

which maintains the linear cost function specification as described in (5-1).

5-2-2 Penalty for switching

An alternative cost function could be the cumulative route completion time for each AGV, but with an additional penalty on the switching of dependencies. The concept here is to only switch a dependency in the case that it surpasses a certain performance increase threshold. It could happen that switching a dependency is only marginally better, and switching is not exactly necessary. Since the variables $b_d \in \{0, 1\}$, a penalty of K_b can be assigned to each switched dependency using then following cost function

$$J(\cdot) = \sum_{i=1}^{N} t_{i,g}^{n_f} + K_b \sum_{d=1}^{d_T} b_d. \tag{5-8}$$

5-2-3 Greedy Switching

Another cost function to be considered is allowing switching in a greedy fashion. The motivation behind this approach is founded on the unpredictable nature with which delays may occur in the future. Given impending delays, there is no guarantee that predicted trajectories at a given time-step will be executed in that exact way, due to the possibility of delays. For this reason, it may be reasonable to switch dependencies early on which yield improved performance in the short-term, since the long-term is difficult to predict. By assigning a cost

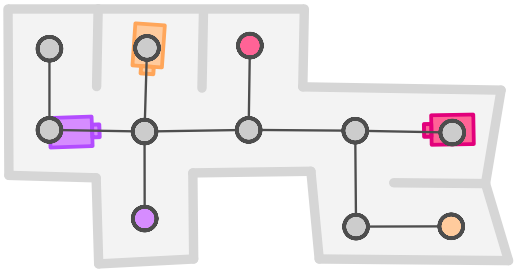
to the time it takes each AGV to complete the event defined by a vertex, early switching could potentially be incentivized, which would not be the case when using alternative cost functions. This cost function is written as

$$J = \sum_{i=1}^{N} \sum_{k=n_i}^{n_f} q_{i,k} t_{i,g}^{k}, \tag{5-9}$$

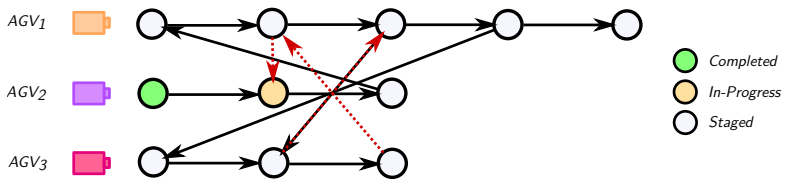
where the tuning parameter $q_{i,k} \geq 0$ can be used to define the cost of reaching the goal point of vertex v_i^k at time $t_{i,g}^k$.

5-3 Illustrative Example

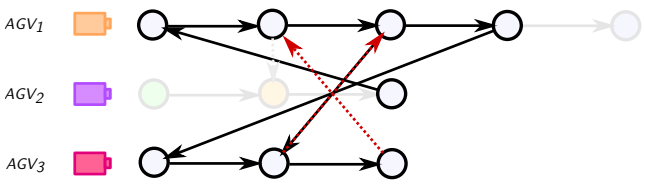
In this section, we present a simple example illustrating the how the MILP is formulated using Section 5-1. Consider the three-AGV example shown in Figure 5-1a. Based on the roadmap, the MAPF is solved, from which we can construct the SADG using Algorithm 3, yielding the SADG shown in Figure 5-1b. We now construct the MILP for this particular SADG based on the current AGV progress. Firstly, we use Algorithm 5 to determine an admissible SADG subset which we can use to define the constraints in the MILP. This subset is shown in Figure 5-1c.



(a) Unweighted roadmap occupied by three AGVs.



(b) Switchable Action Dependency Graph.



(c) Highlighted admissible SADG subset.

Figure 5-1: Illustrative example of the MILP formulation applied to a three AGV scenario.

Based on this SADG subset, we now use the OCP definition in (4-5) described in Section 4-4. Minimizing the cumulative route completion time cost function as in (5-5), the optimization problem using an MILP formulation can be written in its entirety as

$$\min_{\boldsymbol{b},\, \boldsymbol{t}_s,\, \boldsymbol{t}_g} \quad t^4_{1,g} + t^1_{2,g} + t^3_{3,g}$$

s.t.

Type 1 dependencies:

$$\begin{aligned}
&t^1_{1,s} \geq 0, \\
&t^1_{1,g} \geq t^1_{1,s} + 1.0, \\
&t^2_{1,s} \geq t^1_{1,g}, \\
&t^2_{1,g} \geq t^2_{1,s} + 1.0, \\
&t^3_{1,s} \geq t^2_{1,g}, \\
&t^3_{1,g} \geq t^2_{1,s} + 1.0, \\
&t^4_{1,s} \geq t^3_{1,g}, \\
&t^4_{1,g} \geq t^4_{1,s} + 1.0,
\end{aligned}
\qquad
\begin{aligned}
&t^1_{2,s} \geq 0.76, \\
&t^1_{2,g} \geq t^1_{2,s} + 1.0,
\end{aligned}
\qquad
\begin{aligned}
&t^1_{3,s} \geq 0, \\
&t^1_{3,g} \geq t^1_{3,s} + 1.0, \\
&t^2_{3,s} \geq t^1_{3,g}, \\
&t^2_{3,g} \geq t^2_{3,s} + 1.0, \\
&t^3_{3,s} \geq t^2_{3,g}, \\
&t^3_{3,g} \geq t^2_{3,s} + 1.0,
\end{aligned}$$

Static *Type 2* dependencies:

$$t^1_{1,s} \geq t^1_{2,g},$$

Switchable *Type 2* dependencies:

$$\begin{aligned}
&t^1_{3,s} \geq t^4_{1,g} - b_1 M, \\
&t^2_{3,s} \geq t^3_{1,g} - b_1 M, \\
&t^2_{1,s} \geq t^3_{3,g} - (1 - b_1) M, \\
&t^3_{1,s} \geq t^2_{3,g} - (1 - b_1) M,
\end{aligned}$$

where $\boldsymbol{b} = [b_1] : \{0, 1\}$ is the binary decision variable, $\boldsymbol{t}_s = [t^1_{1,s}, t^2_{1,s}, t^3_{1,s}, t^4_{1,s}, t^1_{2,s}, t^1_{3,s}, t^2_{3,s}, t^3_{3,s}]$, and $\boldsymbol{t}_g = [t^1_{1,g}, t^2_{1,g}, t^3_{1,g}, t^4_{1,g}, t^1_{2,g}, t^1_{3,g}, t^2_{3,g}, t^3_{3,g}]$. AGV_2 is assumed to still require 0.76 seconds to complete its the *in-progress* node. Note how the two switchable dependency pairs formed an *opposite* dependency group, permitting the use of only one binary variable.

5-4 Heuristics for MILP Solver

In this section, we propose two heuristic approaches aimed at reducing the computational time required to solve the MILP in (5-2) through (5-5). The SADG is a highly structured graph. However, this structure may not be immediately evident to the MILP solver simply based on the provided optimization formulation. By exploiting this structure and directly informing the solver hereof, we may potentially reduce computational time.

5-4-1 Preferential Switching Based on Dependency Time Difference

As shown in Section 4-4, each switchable dependency group or pair can be associated to a single binary variable within the MILP. Most MILP solvers use a branch-and-cut approach to solve the optimization problem. Essentially, this means fixing a subset of the binary variables, solving the relaxed sub-problem, and then identifying if the relaxed problem has a solution lower than the upper bound obtained thus far [34].

The order in which the binary variables are fixed before solving the relaxed problem can largely affect the optimization time, since this will influence how soon an upper bound to the optimization problem is obtained. Obtaining an upper bound sooner will allow the solver to eliminate combinations of binary variables just by considering the relaxed problem, potentially speeding-up the optimization process.

When a dependency pair is switchable, an idea could be to consider the immediate difference in route completion time if switching were to occur in order to determine if it makes sense to fix the associated binary variable. We could then order the dependency pairs in ascending order according to this route completion time difference to obtain the sequence in which binary variables should be fixed when solving the MILP. Intuitively, a large negative difference in route completion time implies that switching to the reverse dependency could greatly decrease route completion times for the AGVs, when only considering this *Type 2* dependency pair. Similarly, a large, positive difference in route completion time implies that switching will result in a large increase in route completion time when only considering these two AGVs.

We now show how a dependency pair's difference in route completion time can be determined. Consider a dependency pair $(e_{\text{fwd}}, e_{\text{rev}})$, where $e_{\text{fwd}} = (v^1_{\text{fwd}}, v^2_{\text{fwd}})$ and $e_{\text{rev}} = (v^1_{\text{rev}}, v^2_{\text{rev}})$. We now define an operator $t_{\text{pred}}(v)$ which denoted the time event $v \in \mathcal{V}_{\text{ADG}}$ is expected to start in the case that only *Type 1* dependencies are considered. A dependency pair's difference in route completion time can then be defined as

$$\Delta t = t_{\text{pred}}(v^2_{\text{fwd}}) - t_{\text{pred}}(v^2_{\text{rev}}). \tag{5-10}$$

In the case of a dependency group, the difference in route completion time can then be defined as

$$\Delta t = \min_{v^2_{\text{fwd}} \in \mathcal{V}_{\text{fwd}}} t_{\text{pred}}(v^2_{\text{fwd}}) - \min_{v^2_{\text{rev}} \in \mathcal{V}_{\text{rev}}} t_{\text{pred}}(v^2_{\text{rev}}), \tag{5-11}$$

where $\mathcal{V}_{\text{fwd}}$ is a set containing all the vertices to which the forward dependencies in $\mathcal{D}_{\text{switchable}}$ are pointing to. Similarly, $\mathcal{V}_{\text{rev}}$ is a set containing all the vertices to which the reverse dependencies in $\mathcal{D}_{\text{switchable}}$ are pointing to.

5-4-2 Providing an Initial Feasible Solution

Typical solution schemes for MILPs such as branch-and-bound or branch-and-cut solvers make use of lower- and upper-bounds to prune branches in the search tree of binary variable states. As such, finding an initial, feasible solution with a low cost will help eliminate multiple branches which the solver would otherwise need to consider. Since we know, by definition,

that each SADG has a feasible solution of $\boldsymbol{b} = \boldsymbol{0}$ at each time-step, this information can be provided to the solver explicitly.

5-5 Recursive Feasibility

An aspect of critical importance regarding optimal feedback control strategies is that of recursive feasibility. Recursive feasibility implies that the OCP remains feasible as long as the control law is applied [33]. In the context of this work, an infeasible optimization problem would imply that there is no way to switch the dependencies such that the resultant AGV ordering does not cause a deadlock. In order to maintain liveness of the resultant plans, it is necessary to ensure recursive feasibility. In this section, we address this challenge by proving recursive feasibility for both the shrinking and receding horizon control approaches presented in Chapters 3 and 4 respectively.

5-5-1 Shrinking Horizon Control Approach

Consider the shrinking horizon feedback scheme outlined in Algorithm 4. In Proposition 5.1, we show that this feedback scheme is guaranteed to remain recursively feasible based on the MILP formulation presented in this chapter.

Proposition 5.1 (Recursive Feasibility of Algorithm 4) *Consecutively solving the MILP in (5-2) through (5-5) based on the OCP defined in (3-6) when executing the control strategy defined in Algorithm 4 is guaranteed to ensure that the OCP remains recursively feasible.*

Proof. Proof by induction. Consider SADG($\boldsymbol{b}$) as defined in Definition 3.3, constructed from a MAPF plan $\mathcal{P}$ using Algorithm 3. Next, assume an acyclic ADG = SADG($\boldsymbol{0}$) at time t. The OCP in (3-6), formulated as an MILP in (5-2) through (5-5), always has the feasible solution $\boldsymbol{b} = \boldsymbol{0}$ if the initial ADG (from which the OCP's constraints in (3-3) through (3-4) are defined) is acyclic. Any improved solution of the OCP with $\boldsymbol{b} \neq \boldsymbol{0}$ is necessarily feasible, implying a resultant acyclic ADG = SADG($\boldsymbol{b}$). This implies that the resultant ADG = SADG($\boldsymbol{0}$) will always be acyclic if the ADG = SADG($\boldsymbol{b}$) before the OCP was solved, was acyclic. As per Assumption 2.1, the ADG = SADG($\boldsymbol{0}$) at time $t = 0$ is acyclic based on the MAPF solution, implying that it will remain acyclic for $t > 0$. Since SADG($\boldsymbol{b}$) acyclicity guarantees feasibility of the OCP, this guarantees recursive feasibility. □

5-5-2 Receding Horizon Control Approach

By similar argumentation, recursive feasibility for the receding horizon approach can also be proven, as shown in Proposition 5.2.

Proposition 5.2 (Recursive Feasibility of Algorithm 6) *Consecutively solving the MILP in (5-2) through (5-5) based on the OCP defined in (4-5) when executing the control strategy defined in Algorithm 6 is guaranteed to ensure that the OCP remains recursively feasible.*

Proof. Proof by induction. Consider SADG($\boldsymbol{b}$) as defined in Definition 3.3, constructed from a MAPF plan $\mathcal{P}$ using Algorithm 3. Next, assume an acyclic ADG = SADG($\boldsymbol{0}$) at time t.

Consider the OCP in (4-5), which is based on $\bar{S}(\bar{\mathbf{b}})$, where $\bar{S}(\bar{\mathbf{b}})$ is obtained from SADG($\mathbf{b}$) using Algorithm 5. As per Proposition 4.1, a feasible solution for (4-5) is guaranteed to be feasible for the entire SADG($\mathbf{b}$). The OCP in (4-5), formulated as an MILP in (5-2) through (5-5), always has the feasible solution $\mathbf{b} = \mathbf{0}$ if the initial ADG (from which the OCP's constraints in (3-3) through (3-4) are defined) is acyclic. Any improved solution of the OCP with $\mathbf{b} \neq \mathbf{0}$ is necessarily feasible, implying a resultant acyclic ADG = SADG($\mathbf{b}$). This implies that the resultant ADG = SADG($\mathbf{0}$) will always be acyclic if the ADG = SADG($\mathbf{b}$) before the OCP was solved, was acyclic. As per Assumption 2.1, the ADG = SADG($\mathbf{0}$) at time $t = 0$ is acyclic based on the MAPF solution, implying that it will remain acyclic for $t > 0$. Since SADG($\mathbf{b}$) acyclicity guarantees feasibility of the OCP, this guarantees recursive feasibility. □

5-6 Summary

In this chapter, we considered the OCPs presented in Chapter 3 and Chapter 4 and showed how these can be written in the form of an MILP. Specifically, we used the big-M approach to model the switching of dependencies using binary variables. We then introduce three additional cost functions which can be used to enforce different behavior of the resultant control strategies. To illustrate the MILP in a more intuitive manner, we also introduced a simple three-AGV example and presented the resultant MILP which should be solved to obtain the desired switching based on the current AGV progress.

Next, we considered the structure of the SADG and introduced two possible heuristics which can be used to decrease the computational time required to solve the MILP by explicitly defining binary variable combinations which guarantee feasible solutions. Finally, we analyzed the recursive feasibility of the shrinking horizon and receding horizon feedback schemes presented in Chapter 3 and Chapter 4 respectively, and proved that these strategies will ensure the resultant OCP will remain feasible regardless of the disturbances applied to the system, as long as the initial ADG is acyclic.

Chapter 6

Statistical Evaluation

This is the first of two chapters aimed at presenting simulation results used to evaluate the performance and limitations of the control strategies presented in Chapter 3 and Chapter 4. In this chapter, we focus on gaining insight into the statistical performance properties of the proposed methods. To this end, a series of Monte-Carlo simulations are designed to determine how the shrinking and receding horizon control strategies perform for different AGV fleet size, map topologies, delay duration and varying start- and goal-positions. We present the results and discuss them in detail.

6-1 Overview

In this section, we present the general evaluation framework used throughout this chapter. The simulation consists of a discrete-time evaluation, akin to the movement behavior assumed by the MAPF problem defined in Problem 2.1. This means that AGVs are simulated to move in discrete steps along the edges of the roadmap. At each time-step, AGVs can either stay at their current vertex in the roadmap or travel to the next vertex as dictated by the MAPF plan and the ADG. This iterative scheme is illustrated in Figure 6-1. The benefit of this approach is that it allows us to experiment with different simulation parameters such as prediction horizon lengths, OCP cost functions for different AGV fleet sizes with different random start and goal positions, different map topologies, etc. in a systematic and independent manner.

We refer the reader to Chapter 7 for an analysis of the real-time applicability of these control strategies, where we consider optimization times, communication delays, and other real-time issues simultaneously in an extensive simulation environment to validate the proposed approach as a whole.

Simulations are performed the roadmap shown in Figure 6-2. The simulation considers AGV fleets of up to 70 AGVs. Each AGV is initialized with a random starting position and assigned a random, unique goal position on the roadmap. ECBS [22] is then used with varying sub-optimality bounds to solve the MAPF. Specifically, the sub-optimality bound w is chosen between 1.3 and 2.5, such that the MAPF can be solved in approximately 120 seconds.

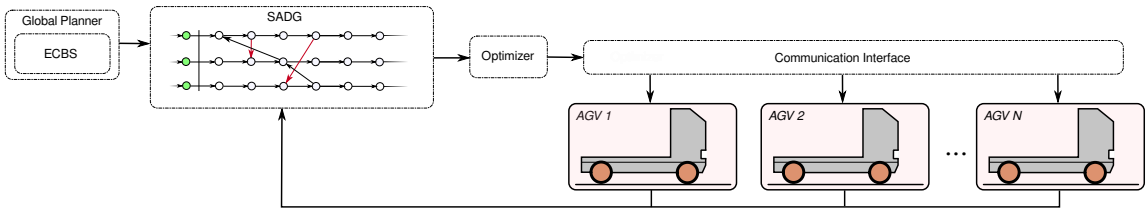
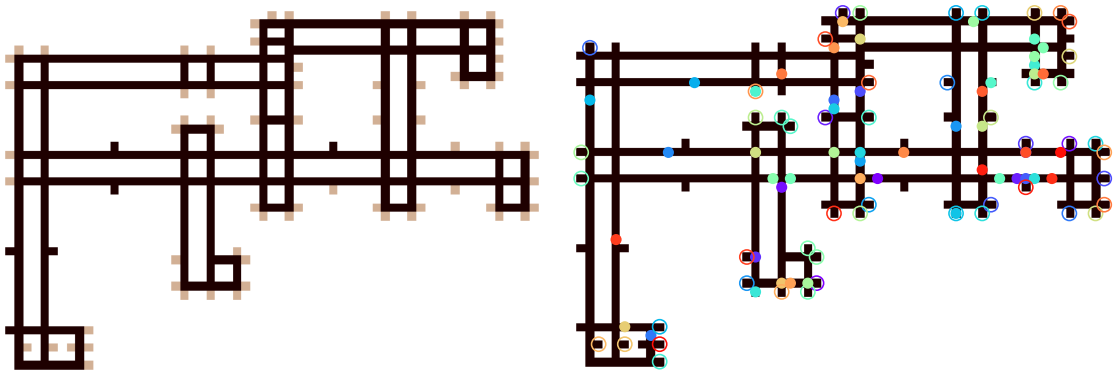


Figure 6-1: Diagram showing the iterative simulation loop which is executed at each time-step.

This simulation is implemented such that AGVs move in discrete time-steps, much like the basic assumption of the MAPF problem definition. 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 is stopped for a length of k time-steps. The MILP in (5-5) 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.

To minimize the number of binary variables in the MILP, the concept of dependency groups is used as described in Section 3-5. The receding horizon approach proposed in Section 4-5 is used for finite-horizon simulations, where Algorithm 5 is used to determine an admissible SADG subset at each iteration. Delays are simulated by selecting a random subset of the AGVs at each iteration and not allowing them to continue with their plan. The simplicity of this framework allows us to run a high number of simulations to gain a good understanding of the method from a statistical point of view.

Our approach is compared to the original ADG approach introduced by Hönig *et al.* [9] presented in Section 2-2-3. All simulations were conducted on a Lenovo Thinkstation with an Intel® Xeon E5-1620 3.5GHz processor and 64 GB of RAM.



(a) Roadmap (black) with possible goals (orange). **(b)** Random initialization of 50 AGVs.

Figure 6-2: Schematic of the roadmap used for the evaluations discussed in this chapter. Random goal locations are selected from the yellow regions in (a). AGVs are represented by colored dots, their goals are represented by the corresponding colored ring as shown in (b).

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 [9]. Specifically, the *improvement* is defined as

$$\text{improvement} = \frac{\sum t_{\text{baseline}} - \sum t_{\text{switching}}}{\sum t_{\text{baseline}}} \cdot 100\%, \tag{6-1}$$

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 (5-5) to $\boldsymbol{b} = \mathbf{0}$ at every time-step. Another important consideration is the time it takes to solve the MILP in (5-5) at each time-step. For our simulations, the MILP was solved using the academically orientated Coin-Or Branch-and-Cut (CBC) solver [35].

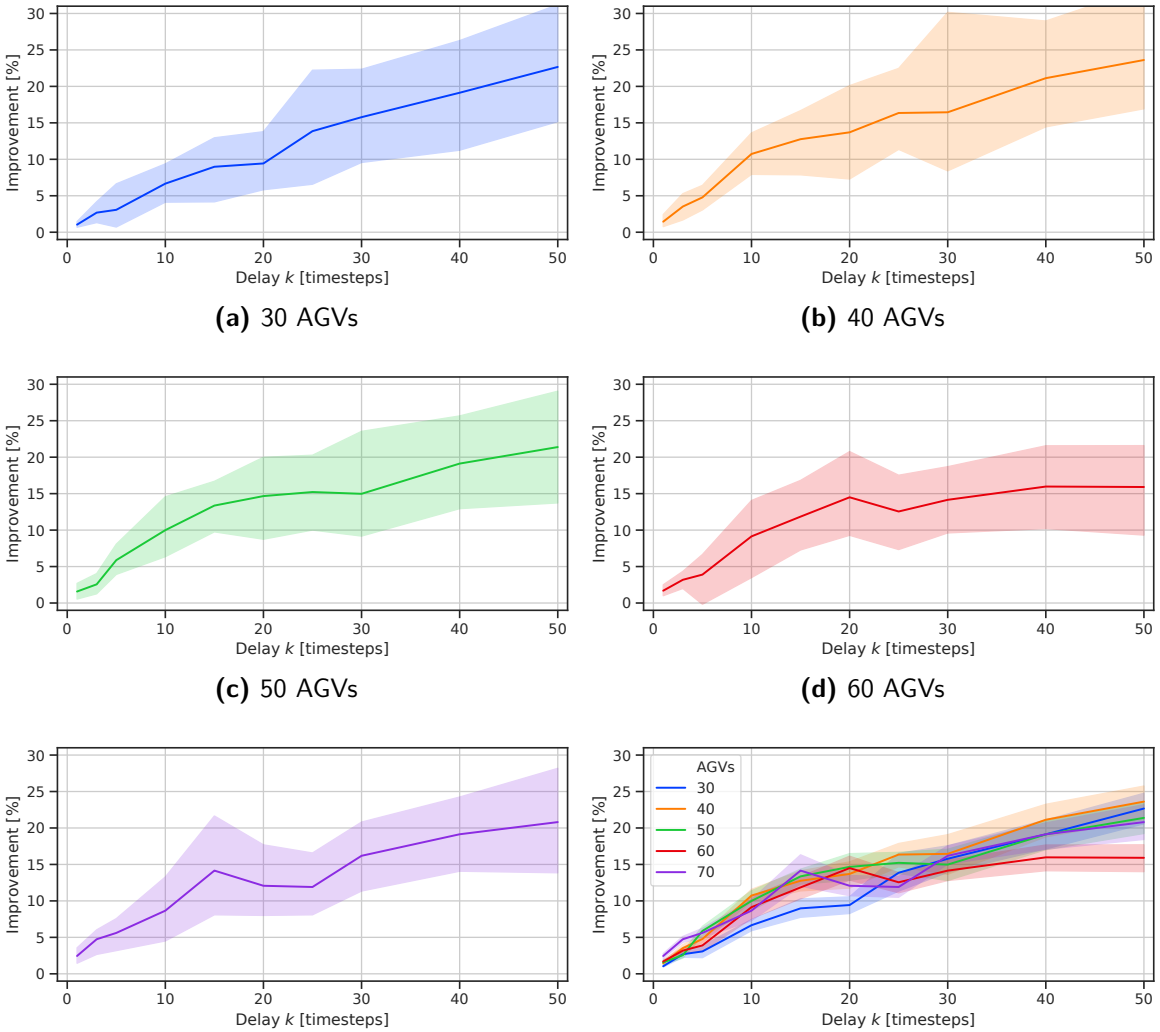
6-2 Simulations

6-2-1 Improvement for Various Delays and AGV Fleet Sizes

As a first step, we evaluate the average improvement for different delay lengths as well as AGV fleet sizes. For each AGV fleet size of {30,40,50,60,70} AGVs, we evaluate delays of $k = \{1, 2, 3, 5, 10, 15, 20, 25, 30, 40, 50\}$ time-steps. The improvement for each permutation is determined for 100 random start/goal assignments. The receding horizon approach in Section 4-5 is used, with a horizon length of 5 time-steps. This horizon length was chosen after determining that it yielded a similar performance to longer horizon lengths but at significantly faster optimization times. For an extensive analysis on horizon length, we refer to the reader to Section 6-2-2.

Figure 6-3 shows the results of this simulation. Figure 6-3a through 6-3e show the improvement of each AGV fleet size for various delays. The dark lines indicate the average improvement, and the light regions indicate the minimum and maximum increase in performance of the 100 simulations performed for each delay duration. Figure 6-3f superimposes the average improvement of the different AGV fleet sizes to facilitate comparison. In this figure, the envelopes represent the improvement within one standard deviation of the average improvement.

From these results, it is clear that the switching of dependencies can improve overall performance when AGVs are subjected to delays. We also note an almost linear relationship between the improvement and the delay duration. However, occasionally, larger delays actually caused a decrease in improvement, for example as seen in Figure 6-3d when comparing delays of $k = 20$ and $k = 25$ time-steps. Another observation is the large variability in the improvement. The envelopes in Figure 6-3a through 6-3e are relatively wide. For example, in Figure 6-3a, delays of $k = 25$ time-steps yielded improvements between 6.6% and 22.4%. This leads the author to believe that certain start/goal positions and random delay inputs can affect the *switchability* of the SADG. This implies that certain MAPF solutions could be more prone to switching than others, yielding significantly varying results. This effect will be explored from a different perspective in Section 6-2-5, where the effect of different map topologies on average improvement is considered.



(a) 30 AGVs **(b)** 40 AGVs

(c) 50 AGVs **(d)** 60 AGVs

(e) 70 AGVs **(f)** All AGV fleet sizes

Figure 6-3: 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.

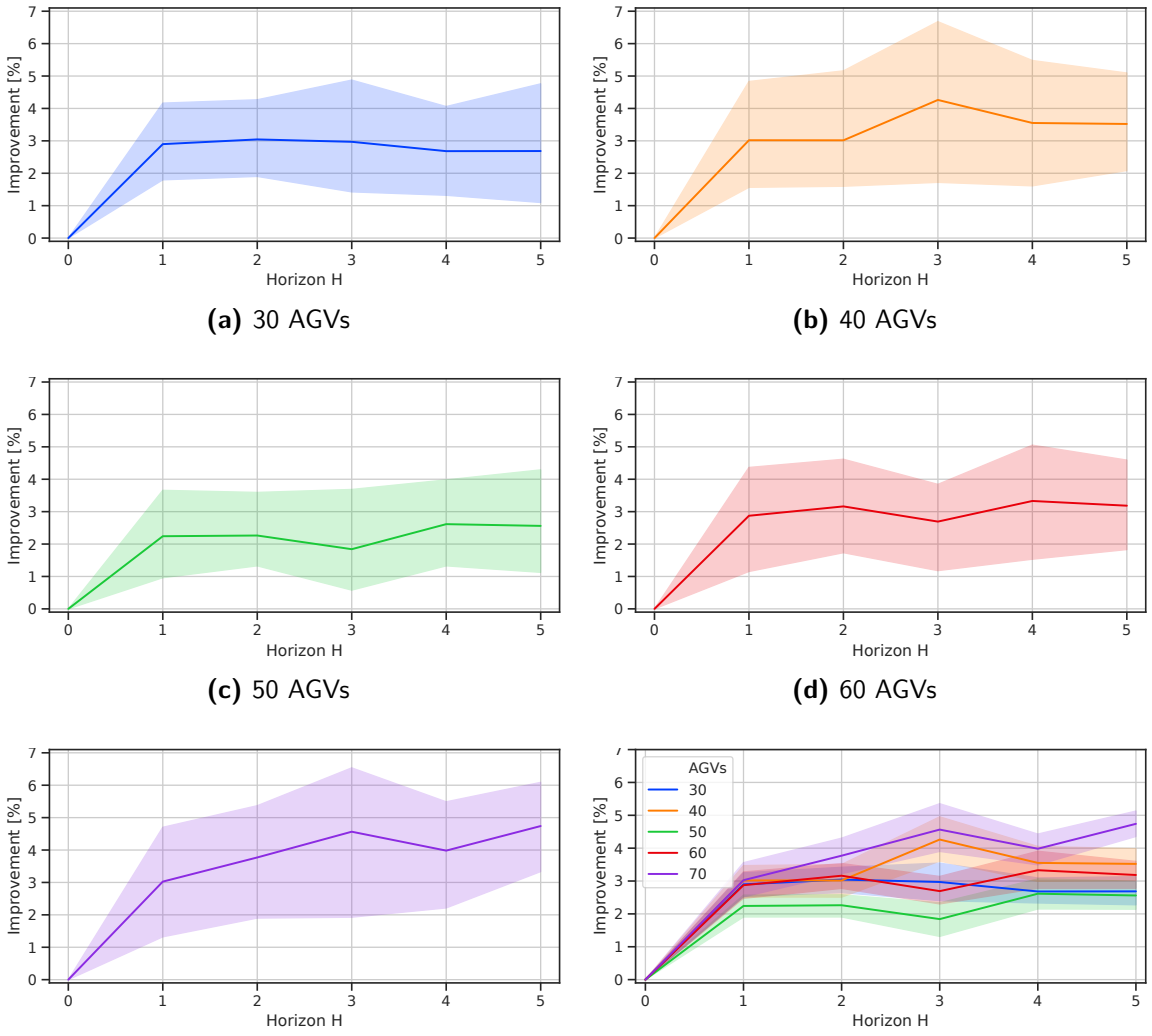
Another interesting observation is that negative improvement can occur, albeit rarely. This was typically observed for small delay durations. An example can be seen in Figure 6-3d at a delay of $k = 3$, where the envelope has a minimum of 0.2%. The reason is that the optimization problem solves the switching 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.

Finally, we observe that improvement does not always increase with delay length. The improvement for an AGV fleet size of 60 seems to plateau at a delay of $k = 40$ time-steps. One possible explanation is that evaluating 100 randomized start/goal locations is not sufficient to accurately represent this trend in a statistical manner.

6-2-2 Performance for Different Horizon Lengths

We now evaluate the effect the horizon length has on the cumulative route completion time. Once again, for various AGV fleet sizes and horizon lengths, we consider a relatively short and long delay length of $k = 3$ and $k = 25$ respectively. For each AGV fleet size and horizon length permutation, we ran 100 simulations with different randomly selected start/goal positions. Based on the observations in Section 6-2-1, we only consider a horizon of $H = \{1, 2, 3, 4, 5\}$ time-steps for delays of $k = 3$. For the delays of $k = 25$ we considered horizons of length $H = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15\}$. The results for delays $k = 3$ and $k = 25$ time-steps are shown in Figure 6-4 and Figure 6-5 respectively.

A first observation is that a horizon of $H = 1$ already yields a significant improvement for every delay and AGV fleet size permutation considered. This is an interesting result which is not

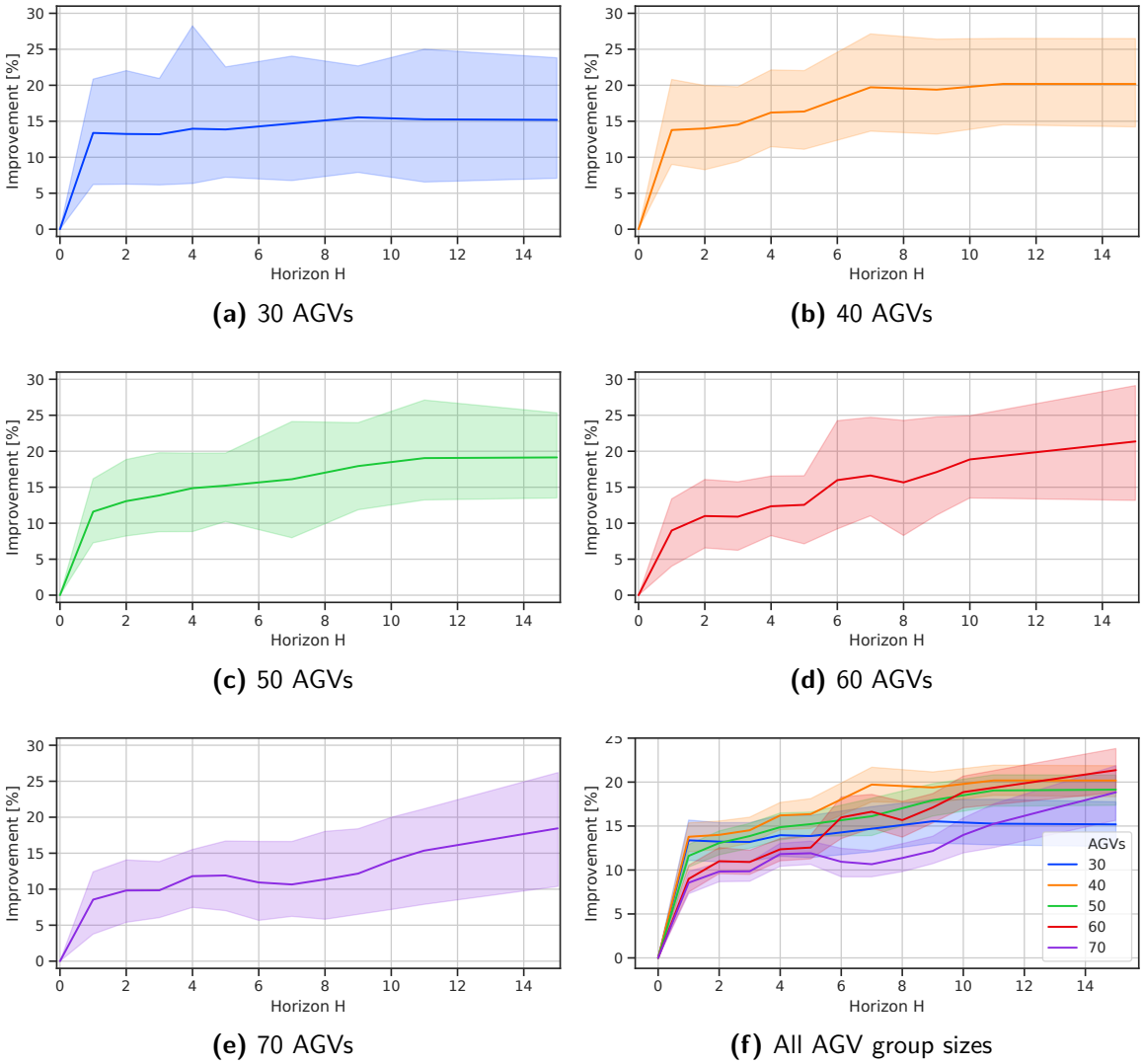


(a) 30 AGVs **(b)** 40 AGVs

(c) 50 AGVs **(d)** 60 AGVs

(e) 70 AGVs **(f)** All AGV group sizes

Figure 6-4: 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. $k = 3$.



(a) 30 AGVs **(b)** 40 AGVs

(c) 50 AGVs **(d)** 60 AGVs

(e) 70 AGVs **(f)** All AGV group sizes

Figure 6-5: 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. $k = 25$.

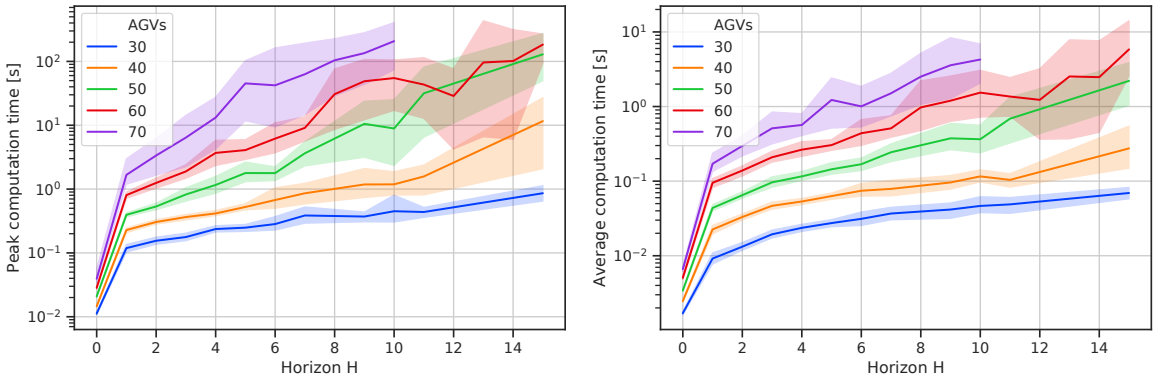
typically seen with continuous-time systems controlled using MPC strategies. Additionally, it is worth mentioning that, unlike continuous-time systems, there is no notion of stability for this SADG approach, meaning that any horizon length can be used to control the system without needing to consider regions-of-attraction or invariant-sets to maintain stability as is the case in typical MPC literature [33].

Another observation which might seem counter-intuitive is that larger horizon lengths do not always yield increased improvements, as can clearly be seen for all the AGV fleet sizes related to the delay of $k = 3$. The author believes that the main reason for this phenomenon is the fact that the OCP formulation considers the progress of the AGVs at the current time-step, and assumes that there will be no delays in the future. This assumption occasionally results in dependency switching that may be optimal assuming no future delays, but actually results in switching which may be detrimental to performance when delays inevitably occur.

As in Section 6-2-1, we once again note a considerable variability in the improvement for the same horizon length and AGV fleet size for different random start/goal locations, indicated by the large lightly colored envelopes.

6-2-3 Computational Time and Horizon Length

As is the case for all optimal feedback control policies, the computational time required to solve the OCP at each iteration will greatly determine whether the methods can be implemented in practice. To this end, we evaluate the time required to solve the MILP at each iteration by considering various horizon lengths and AGV fleet sizes. We consider 100 randomly initialized start/goal positions for various AGV fleet sizes, each for delays of duration $k = \{1, 3, 5, 10, 15, 20, 25\}$ and horizon lengths $H = \{1, \dots, 15\}$. The computation times for different permutations of the aforementioned simulation parameters are shown in Figure 6-6. Specifically, Figure 6-6a shows the peak computation time for an entire simulation, whereas Figure 6-6b shows the average computation time.



(a) Maximum computation time **(b)** Average computation time

Figure 6-6: The computation time to solve the MILP in (5-5) for different AGV team sizes and considered dependency horizon lengths. Solid lines depict the average, lighter regions encapsulate the min-max values.

We note that the computation time is approximately log-linear, where the slope is determined by the AGV fleet size. This can be expected since the MILP typically takes exponential time with regards to the number of binary variables in the optimization problem, which is approximately a linear function of the number of AGVs for a given horizon length.

A key factor in keeping the computation times relatively low is the concept of dependency grouping, as discussed in Section 3-5-2. This greatly reduced computation time and resulted in an MILP with roughly one quarter of the binary variables necessary when no dependency grouping was performed.

Finally, we consider the preferential switching heuristic discussed in Section 5-4-1 as well as explicitly providing an initial feasible solution in Section 5-4-2. Based on 20 simulations involving 50 AGVs and a horizon length of $H = 5$, the preferential switching heuristic was found to yield a computational time that was 21.3% less than the computational time required without using this heuristic. Providing an initial feasible solution, however, did not yield much

improvement, with only a 1.2% decrease in computational time compared with no heuristics and using the aforementioned simulation parameters.

6-2-4 Evaluating Different Cost Functions

As well as the cumulative route completion time, we introduced three additional cost functions in Section 5-2 which could be used when defining the MILP in (5-5). Setting the delay to $k = 10$ time-steps with a horizon of $H = 5$, each cost function was evaluated for 20 random start/goal positions and the improvement compared. The different cost functions and equations are shown in Table 6-1.

Name	Equation
cumulative	(5-5)
makespan	(5-7)
greedy	(5-9)
binary penalty (BP)	(5-8)

Table 6-1: Cost function names and equations

For the binary penalty, we consider four different penalty coefficients $K_b = \{1, 2, 5, 10\}$. In the case of the greedy cost function, we set $q_{i,k} = 1 \; \forall \; i, k$. The results are shown in Figure 6-7.

From the results it is clear that the most effective cost function is the cumulative cost function as well as the binary penalty with low K_b coefficients. Note that optimizing over the makespan or in a greedy fashion would almost not yield any improvement. The authors believe the reason to be that since the proposed strategy only switches the ordering of AGVs along their paths, the last AGV will always take roughly the same time to fulfill its task regardless of the performed switching. Considering the makespan as the cost function can therefore not be expected to yield an improved solution, since the objective value will remain largely the same regardless of the switching. We observed a similar phenomenon when considering the greedy cost function.

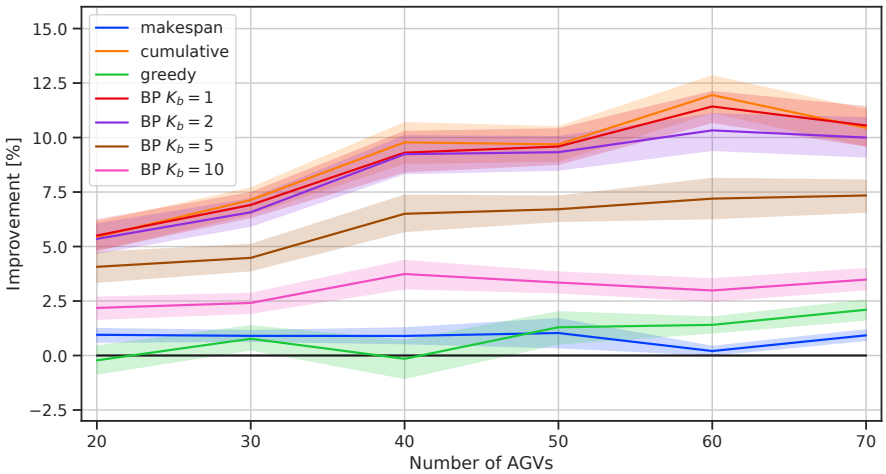
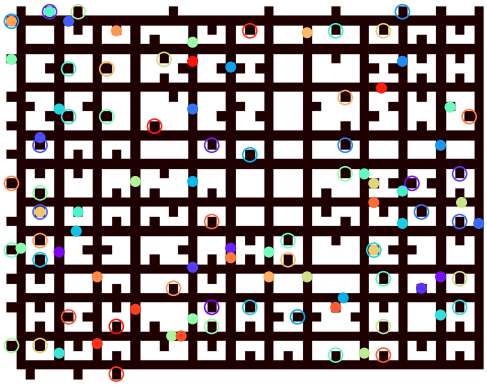


Figure 6-7: Comparison of cost functions for different AGV fleet sizes. Solid lines depict the average improvement and lighter regions encapsulate the min-max values.

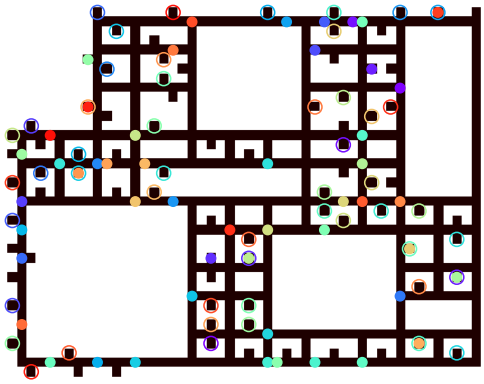
6-2-5 Different Map Topologies

Since the proposed control strategies are based on the SADG derived for various MAPF plans and the number of switchable dependency pairs within the SADG is a function of the interweaving of plans over time, we pose the question of whether map topology affects the *switchability* of the resultant MAPF plans. By *switchability*, we mean the degree of inclination of plans to allow switching when considering the resultant SADG. Additionally, it would be desirable to identify properties of map topologies which influence this *switchability*.

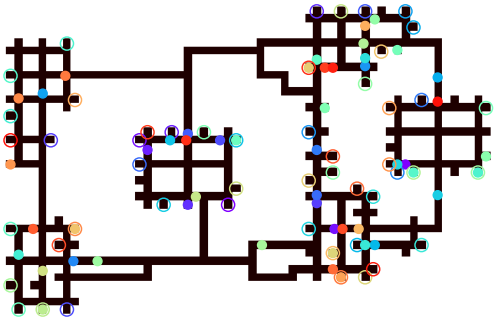
To answer these questions, we consider four fundamentally different roadmap topologies, shown in Figure 6-8, and evaluate the differences in plan improvement. The roadmaps are inspired by various topologies found in real warehouses. The map in Figure 6-8a shows a large, grid-shaped warehouse largely based on the inventory shelving solution presented in [6]. Figure 6-8b shows a similar map, but with large gaps in between which represent portions of a factory floor reserved for items other than inventory. The roadmap in Figure 6-8c is aimed at replicating an unstructured warehouse with multiple rooms and long, single-lane corridors connecting them. Finally, Figure 6-8d shows a map topology similar to the one in Figure 6-8c, but with two-lane corridors.



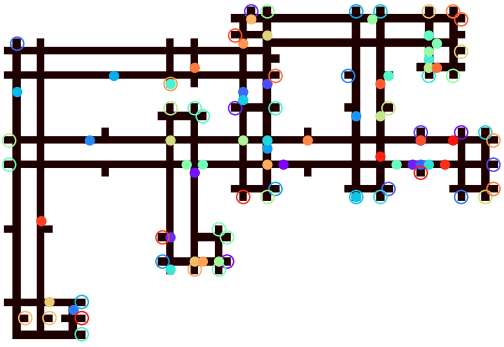
(a) General: grid-shaped map.



(b) Semi: partitioned grid-shaped map.



(c) Islands: single-lane corridors.



(d) Warehouse: two-lane corridors.

Figure 6-8: The four roadmaps used for the statistical evaluations in this section. Each map emphasizes a different topology often seen in real warehouse maps.

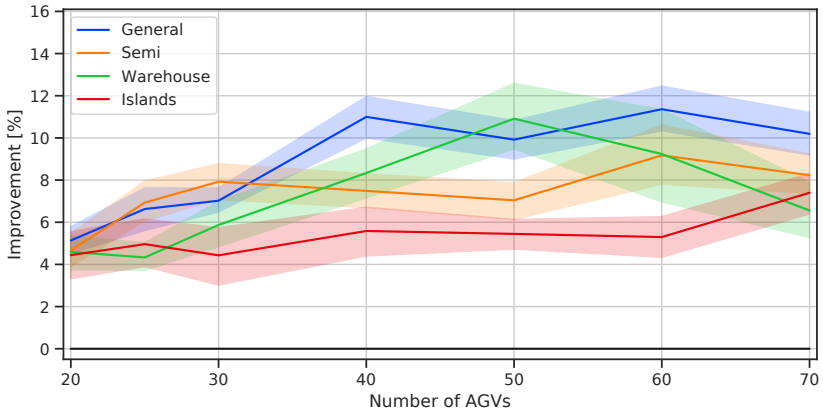


Figure 6-9: Performance comparison for different map topologies for different AGV fleet sizes. The shaded region indicates the results within one standard deviation of the mean, indicated by the solid, colored line.

To evaluate these map topologies, we consider 20 different start/goal positions for AGV fleet sizes of $\{20, 25, 30, 40, 50, 60, 70\}$ and a horizon of $H = 3$. The results are shown in Figure 6-9.

We note that the most favorable map topology was the general layout, shown in Figure 6-8a. The map topology with the lowest improvement was the island map shown in Figure 6-8c. Based on these results, as well as considering the SADG structure for the different maps, we noticed that a the general map layout had far more dependency groups, with fewer dependencies per group, as opposed to the island map which had a smaller number of dependency groups, each with a large number of dependencies.

This result was to be expected, since small dependency groups indicate that AGVs only share a small portion of the roadmap with another in succession, whereas large dependency groups indicate that a large portion of the roadmap is shared (such as a whole corridor). For reference, the SADG for the general map had an average of 1435 dependency groups for the 20 scenarios, whereas the island map only had average of 832.

With the OCP formulation of (4-5) in mind, consider the fact that a smaller dependency group typically requires a smaller delay of the dependent AGV for switching to be beneficial. If switching occurs more easily for shorter delays, as is the case in the General map, we can expect higher improvement percentages compared to topologies such as the Islands map.

6-3 General Discussion

From the simulation results presented in the previous section, we observed that the receding horizon strategy presented in Section 4-5 generally yields lower cumulative route completion times for AGVs when subjected to random delays than when using the originally proposed ADG approach. We noticed that average improvement is approximately linearly correlated to the delay duration experienced by the AGVs.

Considering Figure 6-3f, 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 author to believe there is an optimal AGV group size for a given roadmap, which ensures the workspace is both:

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, Figure 6-4 and Figure 6-5 show 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. Figure 6-6 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.

One of the key concepts used to keep computational times when solving the OCP at each time-step was the use of dependency groups. We found that dependency grouping typically resulted in an MILP with one fifth of the binary variables for the same AGV fleet size and horizon length. Since MILPs typically take exponential time to solve with respect to the number of binary variables, this clearly yields a great increase in performance.

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 Figure 6-3f. 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, further delaying the AGV it passed. We believe a robust optimization approach could potentially resolve this.

In conclusion, we observed that the receding horizon control scheme would reduce the cumulative route completion time of the AGVs based on arbitrary delays. A noteworthy observation is that this approach does not affect the performance negatively when only small delays occur, making it well suited to multi-AGV scenarios where delays are not frequent. Finally, we note that this simulation environment is relatively simple. This simplicity was chosen to allow for a large number of simulations, testing multiple design parameters such as horizon length and AGV fleet size in a timely manner. Naturally, a more extensive simulation or practical experiment is necessary to fully validate the approach. Chapter 7 presents a realistic simulation environment and implementation used to evaluate these real-time challenges.

6-4 Summary

In this chapter, we evaluated the statistical properties of the proposed methods of this thesis. We showed that the proposed methods yield a decrease in route completion times for AGVs subject to delays of various lengths. The receding horizon approach was also evaluated for various horizon lengths, and the computation time to solve the MILP at each iteration was considered. We also identified important features in map topologies which benefit more from our proposed approach than others. As a next step, to further validate this approach, it is desirable to move towards system-level tests in more realistic environments, such as by simulations in Gazebo. This analysis is shown in the next chapter.

Chapter 7

Gazebo Simulation

The penultimate chapter of this thesis is dedicated to the implementation of the control strategies presented in Chapters 3 and 4 into a realistic simulation environment. The motivation is to identify and address the challenges associated with implementing the developed methods on multiple AGVs in a real warehouse. To this end, we use the Robot Operating System (ROS) to facilitate the interaction of the coordinator with the AGVs. The AGVs are simulated in Gazebo, a realistic physics engine and simulation environment. The results show lower cumulative route completion times for the AGV fleet, confirming the results obtained in Chapter 6.

7-1 Overview

In Chapter 6, we conducted a statistical evaluation of the methods presented in this thesis. This evaluation considered simulations with up to 70 AGVs and showed that the proposed methods can greatly decrease cumulative route completion time for the AGVs when AGVs are subjected to delays as the plans are executed. The simulations in Chapter 6, however, were done using a synchronous communication scheme without considering complications such as communication delays. This means that each AGV moved to its neighboring position on the roadmap in one, discrete time-step, following the same assumptions made by MAPF planners such as ECBS. In a similarly discrete fashion, when an AGV was delayed, it would simply pause at its current position for that discrete time-step. This simulation framework was sufficient to showcase the theoretical guarantees of the proposed methods such as collision- and deadlock-free execution, but further analysis is required to ensure the control strategies yield the same improvements when applied to an actual warehouse scenario.

In this chapter we present a more realistic simulation framework which we use to identify the challenges associated with implementing the proposed feedback control strategies with asynchronous communication and naturally occurring AGV delays. The term *asynchronous* communication is used to refer to event-based communication between AGVs and the coordinator occurring at arbitrary times throughout the control loop, as opposed to the communication

protocol used in Chapter 6 which only allowed communication after each OCP instance was solved. To this end, we make use of the ROS software package which uses an asynchronous communication protocol in the form of an all-to-all subscriber-publisher framework. To ensure the model of the AGVs is accurate and sufficiently represents the dynamics and limitations of a real AGV, we model each AGV in Gazebo [36]. A screenshot of the Gazebo environment with four AGVs is shown in Figure 7-1.

The three main objectives of this chapter are to:

1. Identify the challenges of implementing and simulating the control strategies of Chapters 3 and 4 in a realistic simulation environment.
2. Address these challenges by translating the feedback control scheme in to the ROS framework and simulating the AGVs in the Gazebo simulation environment.
3. Identify the limitations of this approach when applying it to actual AGVs in a warehouse as well as gaining insight into which future work would still need to be performed to address these limitations.

To this end, Section 7-2 presents the general framework used to implement the proposed control strategies using the ROS framework and Gazebo simulation environment. Section 7-3 describes the simulation setup and a discussion of the results, followed by a summary in Section 7-4.

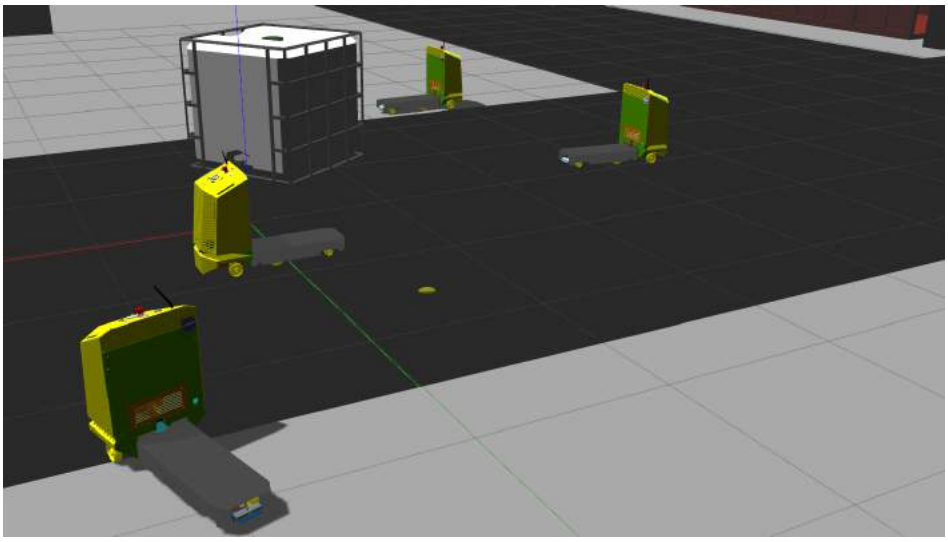


Figure 7-1: Four simulated AGVs in the Gazebo simulation environment with a static obstacle (white crate) lying in the middle of the workspace.

7-2 Proposed Simulation Framework

In this section, we describe the architecture and components of the proposed framework in more detail. Section 7-2-1 presents the communication network implicitly contained within

the ROS framework, showing the inter-connection of all the components required by the feedback strategy. An important sub-component within this is the local planner and localization scheme used by each AGV, which is also presented in this subsection. The fact that communication occurs in an asynchronous manner requires us to slightly adapt previously presented feasible plan execution. This algorithm adaptation is presented in Section 7-2-2. Finally, an augmented simulation setup is proposed in Section 7-2-3 which permits the simulation of large numbers of AGVs by including AGVs simulated in Gazebo as well as a simple holonomic motion models, allowing near real-time simulations of large numbers of AGVs despite limited computing resources.

7-2-1 Architecture and ROS Network Description

The control framework is implemented using ROS, which is a code-agnostic middelware facilitating the communication and interaction between processes as typically seen in robotics applications [37]. Each process is called a *node*, and communication is represented as an *edge* connecting two *nodes*. As such, the interaction between software components implemented in ROS can be seen as a graph representing such a communication network.

Each *node* is a software component that has the ability to publish and subscribe to a so-called *topic*. Publications and subscriptions can be seen as directed edges connecting nodes in the ROS network. A node is able to publish information it has locally to a topic, such that other nodes which are subscribed to this topic can have access to this information. A topic is a message with a specific datatype which provides the means of structured communication between nodes within the ROS network. Consider the receding horizon control scheme presented in Section 4-5. When implemented in ROS, this scheme was translated into the ROS network as shown in Figure 7-2.

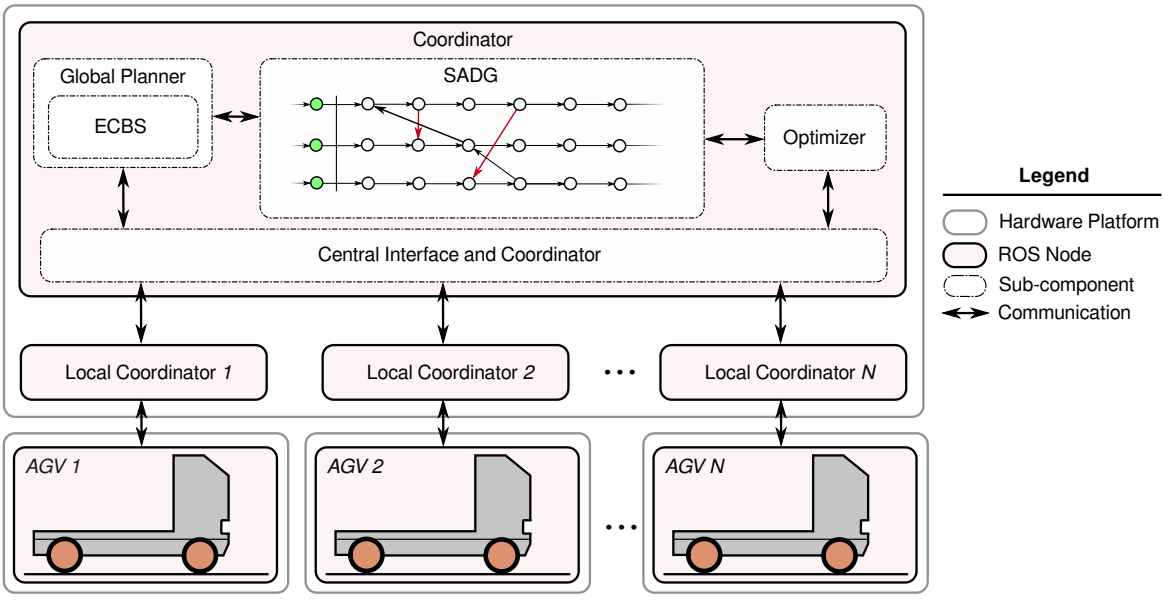


Figure 7-2: Illustration of the communication and physical architecture for the ROS implementation of the receding horizon solution in Section 4-5. See Figure 7-3 for more details on the sub-components of *AGV i* $\forall\, i \in \{1, \ldots, N\}$.

The central coordinator interface is connected to the MAPF planner (in this case, we use ECBS), the SADG data-structure and the optimizer. These components are all contained within a single ROS node. To enable communication between the coordinator and each AGV, we included a *Local Coordinator* i $\forall\, i \in \{1, \ldots, N\}$. This intermediate ROS node was necessary since coordination was implemented in Python 2.7.12, whereas the coordinator was implemented in Python 3.6.7 (since the Python-MIP package used to construct and solve the MILP is only available in Python 3.5 or higher [38]). This intermediate node need not be independent from the coordinator node if this were not the case, e.g. if the code above were implemented using ROS2, which officially supports Python 3 [39].

Finally, *Local Coordinator* i is connected to *AGV* i $\forall\, i \in \{1, \ldots, N\}$. For simplicity, the details of the control architecture of each AGV is omitted in Figure 7-2. Instead, the most relevant details thereof are shown in Figure 7-3. The navigation stack of each AGV consists of the motion planner and the localization scheme. In this case, we use move_base and a localization scheme called AMCL respectively. Both these nodes interact with a simulated instance of the AGV in Gazebo via control inputs to the simulated hardware and sensor readings. The details pertaining to move_base and AMCL are discussed next. However, we only discuss the details relevant to our simulations. The reader is referred to [40] for a comprehensive reference on all the components within the navigation stack.

Motion Planner using move_base

As shown in Figure 7-3, the move_base motion planner consists of an interconnection of five key components. In the context of this thesis work, move_base is a local planner. Sticking to the move_base nomenclature, we note that move_base itself consists of both a global and local planner. It is important to differentiate between the ECBS global planner and the global planner for each AGV's navigation stack. Move_base's global planner uses Dijkstra's

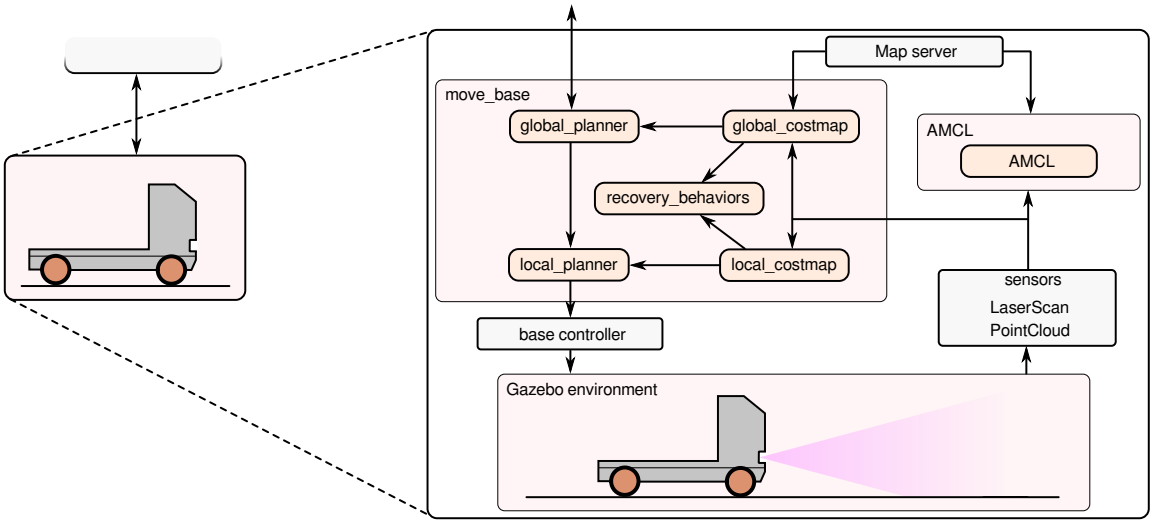
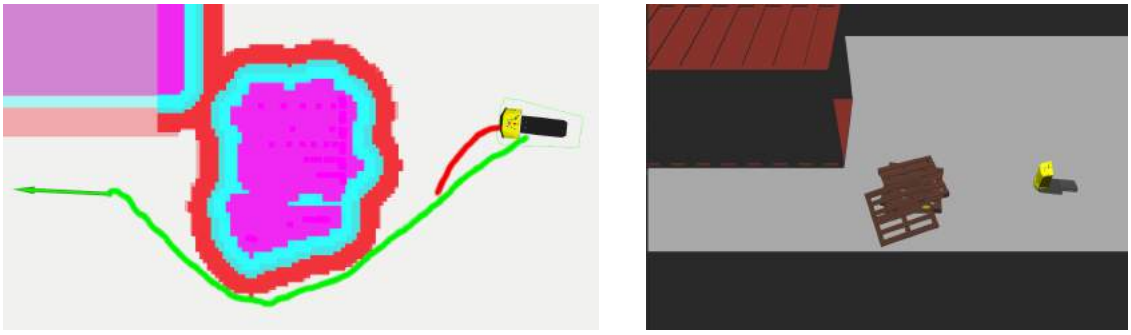


Figure 7-3: Diagram illustrating the components within the navigation stack. The three main components are the *move_base* and *AMCL* ROS nodes and the simulated AGV in the Gazebo environment adapted from [4].

algorithm to determine the fastest route to the a goal based on a roadmap defined by the global costmap. The local planner is then invoked to follow this global trajectory. In our simulations, we use the DWB local planner [41], an improved implementation of the original DWA local planner [42] which uses a dynamic window approach as detailed in [43]. This planner was chosen because it proved to be the most robust of the available open-source local planners.

Figure 7-4 shows an example of the costmaps and planners in RViz. Rviz is a tool used to visualize information contained within ROS nodes and provides a perspective into an AGV's sensor readings and planned control actions. In Figure 7-4a, the global and local costmaps are depicted by a lighter and darker red-blue-magenta colors respectively. The local costmap changes dynamically based on the AGV's sensor data, and the global costmap is updated based on these local observations. In this example, consider the obstacles in-front of the AGV, as seen in the Gazebo simulation environment in Figure 7-4b, and the corresponding costmaps in RViz. When a new goal position is specified, as depicted by the green vector, the global planner determines the fastest route to this goal by performing a graph search, as shown by the green path. Once determined, the local planner determines the control inputs required by the AGV to follow this global path while taking the AGV's kinematic constraints into account. In Figure 7-4a, this local trajectory corresponds to the red path. For the simulations, the local planner is run at 5.0 Hz, and the global planner at 2.0 Hz. Note how only the local planner considers the AGV's kinematics within its planned trajectory.

The final key component of the move_base system is the recovery behavior protocol, which can be depicted as a flow chart as shown in Figure 7-5. This protocol defines the evasive maneuvers which an AGV should follow in the case that it cannot proceed towards its goal. This could be caused by an obstacle which the AGV can no longer avoid by following the global trajectory in a collision-free manner. Referring to Figure 7-5, the default operating state is *navigating*. If the AGV can no longer navigate to its goal, it will clear obstacles outside a 2-meter radius and re-plan, corresponding to the *conservative reset*. If this fails, it will (if possible) rotate in-place in order to re-localize and try find a valid collision-free trajectory that permits it to resume its navigation to the goal. If this fails, it will clear all perceived obstacles outside the space in which it can rotate, followed by another in-place rotation. If all this fails, the AGV will go into an *aborted* state, meaning it has abandoned its navigation



(a) Global (green) and local (red) planner trajectories. **(b)** Gazebo simulation of the AGV

Figure 7-4: Illustration of the global and local costmaps and trajectories for an AGV navigating through an environment littered with static obstacles as seen in RViz and in the Gazebo simulation environment.

towards the goal. In the context of this work, this recovery behavior is important because it was observed that AGVs can potentially take a considerable amount of time to return to the *navigating* state, potentially delaying other AGVs as dictated by the currently active ADG.

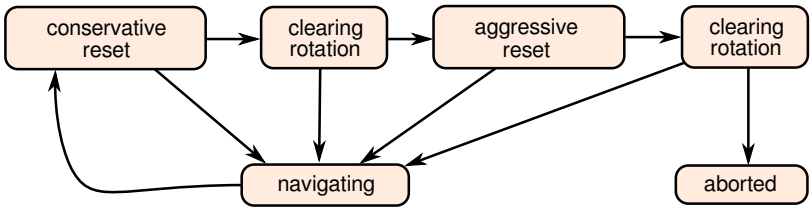


Figure 7-5: Recovery behavior flow diagram illustrating the state-transitions of the *move_base* ROS node as adapted from [4].

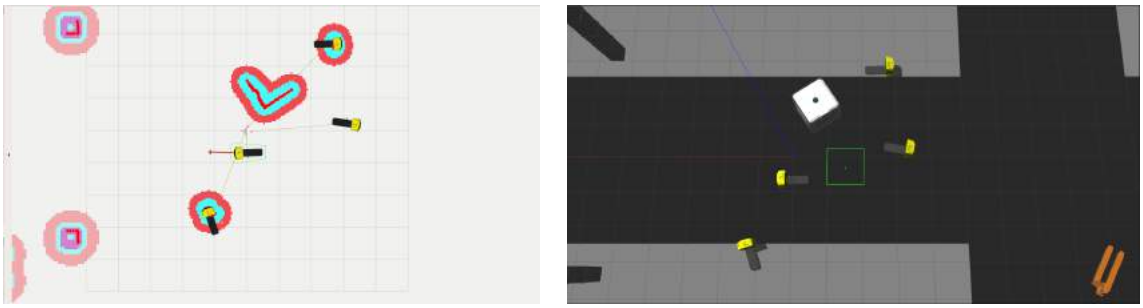
Localization using an Adaptive Monte Carlo Localization Scheme

For localization, the AGV uses an adaptive Monte Carlo localization (AMCL) scheme. This is a probabilistic localization method that is based on a particle filter which uses a predetermined static map input, combined with sensor data obtained from the laser scanner to determine a best estimate of the AGV's pose in the environment [44]. In the simulations, the AMCL node runs at 10 Hz, with a minimum of 500 and maximum of 5000 particles. These parameters were found to yield a good balance between computational efficiency, localization accuracy and overall performance.

Figure 7-6 shows an example of an AGV which is localizing itself in the workspace. Figure 7-6a shows a visual interpretation of the sensor data used by an AGV to localize itself within the workspace in RViz, corresponding to the simulated scenario shown in Figure 7-6b. In order to visualize what an AGV can see around it, based on its sensor input, the RViz visualization tool is used. This illustration shows how an AGV's perception of its environment can be seen from its own perspective. Figure 7-6b shows four AGVs within a Gazebo environment as well as a previously unknown obstacle. The visualization in Figure 7-6a, shows the laser scan data as seen by the AGV in the center. Note how the neighboring AGVs in its field-of-view are detected by the ego AGV. Additionally, two edges of the obstacle are visible to the AGV as well. Despite these additional, unforeseen obstacles in its field-of-view, the AMCL localization scheme was found to return adequate pose estimates allowing for each AGV to navigate the workspace without the localization affecting the plan execution speed in a significant manner.

7-2-2 Guaranteeing Feasibility Despite Asynchronous Event Completion

In order to minimize communication overhead between AGVs and the coordinator, we let communication take place in an event-based fashion. This means that instead of continuously publishing an AGVs location, and letting the coordinator determine what its next goal pose should be based on the currently active ADG, AGVs only publish messages to the coordinator when it has completed an event as dictated by the ADG. However, it is important to ensure that the AGV does not start an event within the ADG which the coordinator is unaware of, because this might cause the resultant OCP solution to become infeasible. To illustrate this point, consider an inactive reverse dependency pointing to an ADG vertex, which is initially



(a) RViz simulation seen by the ego robot **(b)** Gazebo Simulation

Figure 7-6: Four AGVs in the Gazebo simulation environment and as seen in the RViz environment from the viewpoint of one of the AGVs.

not constrained by another dependency. Based on the previous solution, the ADG dictates that the AGV can start executing this event since no dependencies are stopping it from doing so. However, since the OCP is being solved assuming this vertex is still *staged*, the optimal solution might require the reverse dependency to now be active, resulting in infeasible plan execution, despite an optimal (and therefore feasible) solution to the OCP.

To ensure this does not happen, we adapt the algorithm of the receding horizon control scheme in Section 4-5 such that AGVs can only execute new vertices within the ADG before the OCP is being solved. In the case that the OCP does not yet have a solution, the AGV is forced to wait. The adapted algorithm is shown in Algorithm 7. Here, a boolean state variable *canGetNewEvent* is used to define whether or not an AGV can request a new vertex within the ADG plan to continue executing its plan.

Algorithm 7 Switching ADG RHC Asynchronous Feedback Scheme

```
Get goals and locations
Solve MAPF to obtain 𝒫
Construct SADG(b) using Algorithm 3
AGVs reserve vertices in ADG
while Plans not done do
    positions ← currentPosition(AGV)
    canGetNewEvent = False
    Extract S̄(b̄) using Algorithm 5 and positions
    Construct OCP from S̄(b̄) as in Section 4-4
    b ← solve OCP in (4-5)
    ADG ← SADG(b)
    canGetNewEvent = True
    AGVs reserve vertices in ADG
```

7-2-3 Augmented Simulation Setup

Simulating an AGV with move_base and AMCL in Gazebo requires a non-negligible amount of computing resources. Although this is rarely an issue with smaller fleet sizes (less than

five AGVs), this becomes a challenge when simulating large amounts of AGVs, as is required in this case. The increase in computational complexity is reflected in the real-time factor, a metric in Gazebo which represents the ratio of simulation time to the estimated real time. Ideally, the real-time factor should be 1.0. However, when simulating more than five AGVs, the real-time factor would go as low as 0.1. One of the goals of these simulations, however, is to validate the efficiency of the control strategy despite challenges such as communication delays and occasionally longer optimization times. This can only be done if the real-time factor is not too low.

To this end, we introduce an alternative AGV simulation to the one in Gazebo, allowing a so-called *augmented* simulation setup. The idea is to split the AGV fleet into two subsets: the first is still simulated in Gazebo, and the second is simulated using a far simpler (and less computationally expensive) simulation model. This allows the simulation of large AGV fleets, with a higher real-time factor, all the while considering the complications associated of the complex navigation stack mentioned in previous subsections.

The simplified simulation setup is shown in Figure 7-7 which replaces the navigation stack in Figure 7-3. This simulation component uses the same move_base server as the Gazebo variant, allowing direct compatibility with the already developed ROS-based coordinator, but without a localization scheme, and a holonomic movement model which can be steered towards a target.

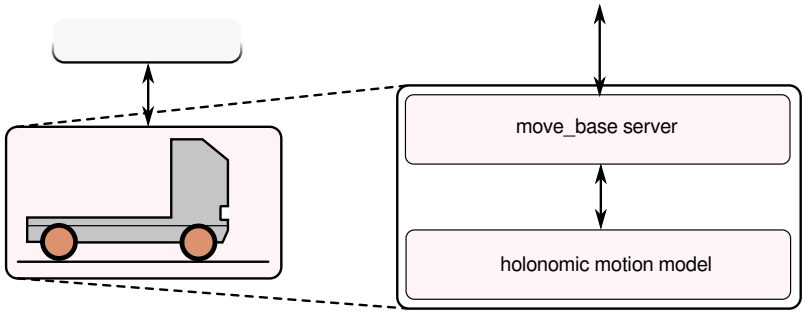
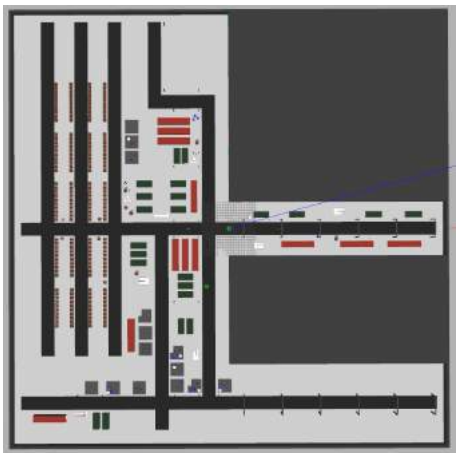


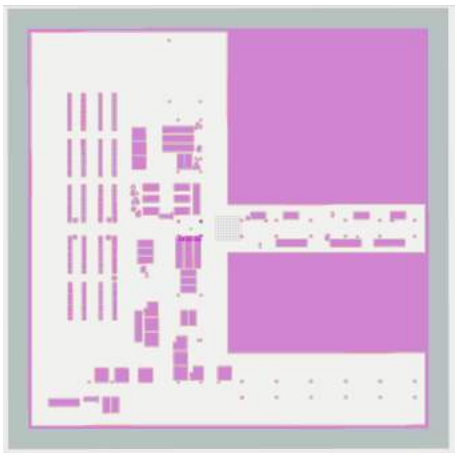
Figure 7-7: Diagram illustrating the components within the simplified navigation stack used in the augmented simulation setup.

7-3 Simulations

Having outlined the basic ROS framework and relevant components, we introduce the simulations used to evaluate the proposed method. All simulations were conducted on a Lenovo Thinkstation with an Intel® Xeon E5-1620 3.5GHz processor and 64 GB of RAM. The AGVs are simulated to lie in the workspace shown in Figure 7-8a, which is a model of a full-sized factory warehouse of dimensions 79.5m $\times$ 78.9m. From this warehouse model, a global costmap is determined as shown in Figure 7-8b in RViz, as well as a roadmap which defines the traverseable space which can be used in the MAPF problem definition. The roadmap is shown in Figure 7-9a.

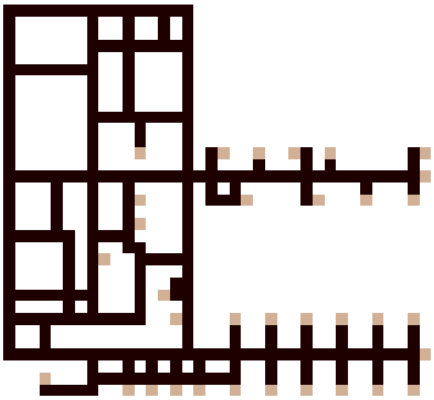


(a) Workspace visualized in Gazebo

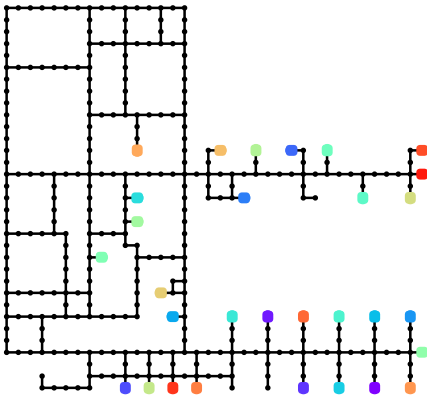


(b) Workspace visualized in RViz

Figure 7-8: Workspace visualized in Gazebo and RViz



(a) Roadmap in black with goal positions in orange



(b) Instantiated with 30 AGVs as colored dots

Figure 7-9: Workspace and the associated roadmap used in the Gazebo simulations.

Consider the orange locations in Figure 7-9a. These signify potential goal locations for each AGV. 30 AGVs are given a random starting location, as well as a unique goal location which is randomly selected from these orange locations. With these parameters defined, the receding horizon scheme as defined in Algorithm 7 is used to coordinate the AGVs towards their goal. The horizon is set to $H = 10$ seconds and the feedback loop is run at a frequency of 1 Hz. This simulation is run 20 times for different random start and goal locations using the persistent framework defined in Section 4-6. This means that the AGVs are given new goals after the tasks are completed. The AGVs then navigate to these new goals based on their current positions.

Furthermore, as described in Section 7-2-3, a randomly generated subset of the AGVs is selected to be simulated in Gazebo in order to keep the simulation real-time factor above 0.5. This means that for each simulation, 5 AGVs are simulated in Gazebo, and the remaining 25 are simulated using the simplified holonomic model described in Section 7-2-3.

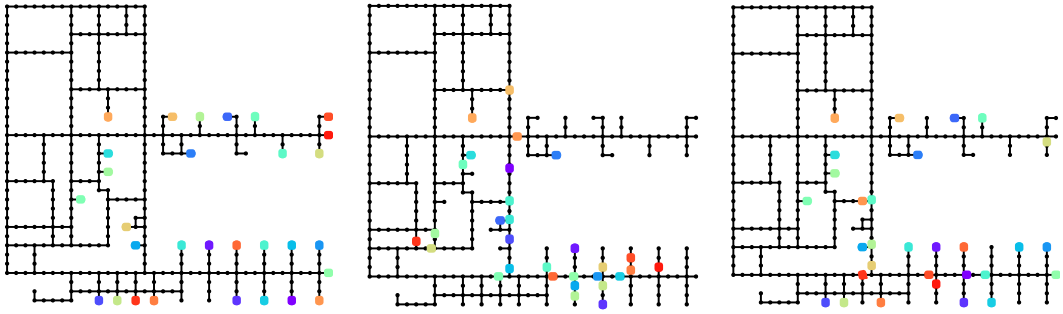
As in Chapter 6, we compare the cumulative route completion time for the AGVs with and

without switching. The improvement, as defined in Section 6-1, can be calculated as

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

Referring to Algorithm 7, running the original ADG approach corresponds to running the proposed receding horizon control strategy with a horizon of length 0. Furthermore, the *canGetNewEvent* variable remains *True* throughout the loop, since the ADG does not change dynamically as with the SADG-based approach. Theoretically, this should allow the original ADG approach to be marginally faster than the receding horizon approach when it does not perform any switching. Naturally, we expect the switching of dependencies to compensate for this small reduction in efficiency.

As shown in Figure 7-4 and Figure 7-6, we include obstacles in the workspace. For each simulation, we randomly placed 40 obstacles such as crates or pallets in different locations in the warehouse. The location of these obstacles is not known to the MAPF planner. The AGVs are not aware of these obstacles beforehand, and only encounter them via sensor measurements as they execute their plans. Obstacles are also placed such that the AGVs are not permanently blocked, but are able to bypass them using the previously introduced local planner. An illustrative graph showing the AGV locations for different times in a single simulation is shown in Figure 7-10.



(a) Start at $t = 0$s **(b)** At $t = 100$s **(c)** At $t = 200$s

Figure 7-10: Simulation of 30 AGVs using the receding horizon implemented in ROS. Colored dots represent the location of AGVs simulated in the Gazebo simulation environment.

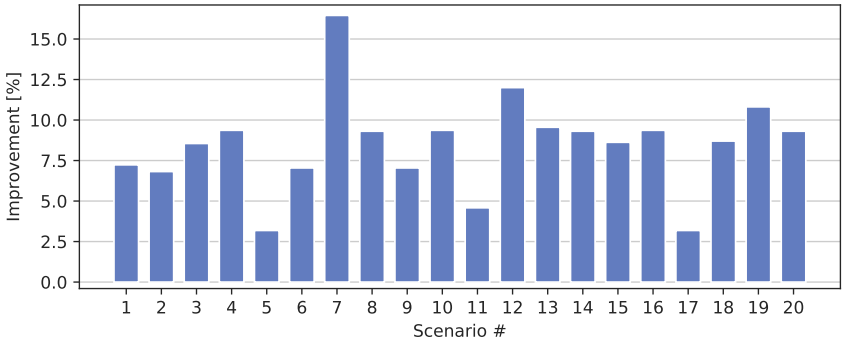


Figure 7-11: Improvement for 20 randomly generated start/goal and obstacle locations.

Figure 7-11 shows the improvement of route completion times for the 20 scenarios, showing that our approach reduces the route completion time for the AGVs in each of the 20 simulations. The average improvement is 8.49%. Recall that, unlike in Chapter 6, we did not artificially create delays by simply stopping AGVs. Instead, delays occurred naturally due to AGVs taking longer to fulfill certain parts of their plans. Specifically, the following phenomena were observed to contribute towards delays:

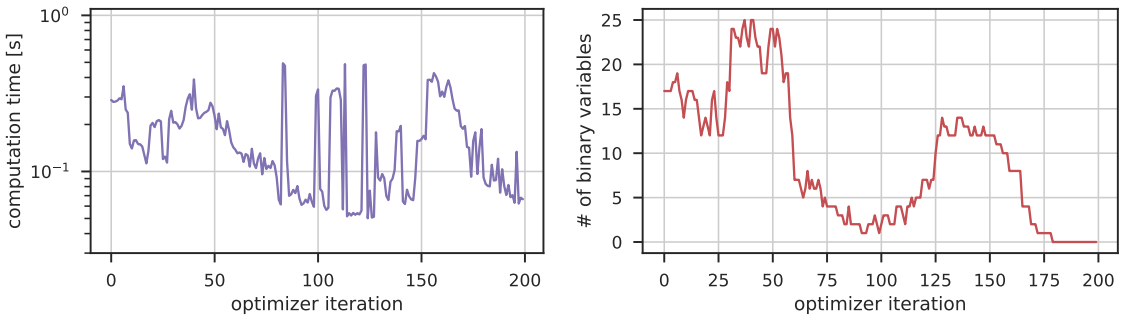
1. **Unpredictable event completion times**
   As previously stated, the MAPF solution considers neither AGV kinematics nor dynamics. However, an AGV turning around an obstacle can take significantly longer than an AGV simply following a straight line trajectory.

2. **Mismatch in predicted velocities**
   AGVs are expected to execute plans at a constant velocity. However, the move_base planner would often cause the AGV to accelerate and decelerate such as when it had to change direction.

3. **Recovery behaviors**
   We observed that AGVs would sometimes loop through their prescribed recovery behaviors due to the proximity of other AGVs causing the costmap to be occupied and temporarily blocking the AGV from advancing towards its goal.

4. **Errors in localization**
   Although localization is mostly stable and consistent, we noticed that, occasionally, an AGV's pose estimate would be very inaccurate. This was mostly caused by other AGVs and obstacles in proximity of the ego AGV. When badly localized, the AGV would temporarily not proceed to its goal in an efficient manner, resulting in a delay.

An important factor to ensure the efficacy of this approach is the computational time required to solve the MILP at each time-step. Consider the computational time shown in Figure 7-12a, which remained below 500ms. This was achieved by ensuring the horizon is low enough such that the amount of binary variables is limited to below 25, as shown in Figure 7-12b. This was also facilitated by the use of dependency groups. On average, the initial SADG derived from the MAPF plan would have 5500 *Type 2* dependencies which were then grouped into dependency groups, resulting in an average of only 1146 dependency groups. This implies



(a) Computation time **(b)** Number of binary variables

Figure 7-12: The computation time to solve the optimization problem

that, on average, the MILP only requires around 20% of the binary variables for the same AGV fleet size and horizon length. The result is that the feedback scheme could be run at a conservatively fast rate of 1 Hz for a group of 30 AGVs, which proved frequently enough to yield an improvement in the overall route completion times.

As a final comment, we note that this approach can accommodate for inaccuracies in the various models used to approximate the real world. For example, the accuracy of roadmaps and knowledge of obstacles is no longer as important as it is in a scenario which does not permit online switching such as the original ADG approach. This is because the re-ordering can account for these inaccuracies by adjusting the ordering as the inaccuracies influence the progress of AGVs along their plans.

In conclusion, the extensive simulation presented in this chapter showcased the real-time applicability of our approach on real AGVs. A next would naturally be to test these scheme on a real warehouse. This is left for future work.

7-4 Summary

In this chapter, we evaluated the receding horizon control strategy proposed in this thesis by implementing the control strategy using the ROS software framework and the Gazebo simulation environment. This allowed for a system-level evaluation of our approach since the efficiency of our approach could be evaluated while considering important factors such as vehicle dynamics, imperfect AGV localization and asynchronous communication. The results showed that the receding horizon approach allows for reduced route completion times for the AGVs with an improvement average of 8.49%. The simulation confirms the applicability of the proposed method to a real-world use-case.

Chapter 8

Conclusions & Outlook

In the final chapter of this thesis, we present the reader with a summary of the work performed during this thesis, a discussion of the obtained results, as well as a short description of recommendations for future work.

8-1 Summary

The principal objective of this thesis was to address the need for the dynamic adjustment of multi-agent plans based on unpredictable delays. Multi-agent planning for AGVs is typically formulated as some version of the MAPF problem, which is known to be NP-hard. Unpredictable delays in dynamic environments can cause the original plan schedule to become highly inefficient. Furthermore, executing the plans defined by a MAPF solution is also a challenging task since the assumptions of the MAPF typically assume perfect synchronization among the AGVs: a property which cannot easily be guaranteed in practice. Some recent work has considered delays within the MAPF planning stage by defining the expected delay properties in an *a-priori* manner. However, in Chapter 2, we go on to motivate the need for an online solution to account for delays as they occur using a feedback control strategy. That being said, online solutions to plan re-adjustment without re-solving the original MAPF problem are scarce.

The work in this thesis is aimed at addressing this scarcity by presenting a new re-ordering scheme in the form of an SADG which allows the re-ordering of AGVs while executing MAPF plans, minimizing the affects delayed AGVs might have on the AGV fleet. To this end, we presented both a shrinking horizon and receding horizon feedback scheme in Chapters 3 and 4 respectively. Both these feedback schemes utilize the SADG to re-order AGVs while maintaining the collision- and deadlock-free guarantees of the original MAPF solution. We go on to formulate an OCP for both of the control strategies which can be solved in an iterative scheme. This OCP is formulated as an MILP in Chapter 5. We also prove recursive feasibility of both control strategies, implying that the optimization problem will remain feasible as long as the underlying assumptions, which we present, are met.

In Chapter 6, the proposed methods were evaluated in an extensive statistical simulation framework to showcase the expected increase in performance, while showing online re-ordering for AGV fleet sizes up to 70 AGVs. This method was compared to the original ADG approach which represents any MAPF execution strategy where the re-ordering of AGVs is not performed. Initial results showed a significant decrease in cumulative route completion times for the AGV fleet. Finally, the proposed control strategies were implemented in ROS and the AGVs simulated in the Gazebo simulation environment. We show that the method translates well to a realistic environment which includes complications such as asynchronous communication and inaccurate route descriptions. These results illustrate the proposed method's viability in real robotics applications.

8-2 Conclusions & Discussion

The work in this thesis contributes a small portion to a much larger problem being faced by the control and artificial intelligence (AI) communities at this point in time: how can the control schemes be used to dynamically update decisions inAI related problems in a natural manner. In short, the control domain is largely focused on dynamic systems using the well-known feedback-loop paradigm, as well as typically looking for mathematical guarantees for solving problems. On the other hand, the AI domain has always typically considered discrete, large-scale problems, but in a static setting, meaning that disturbances are difficult to account for in a natural manner.

This was showcased in this thesis, where we considered the MAPF problem, a typical NP-hard challenge in the AI domains, and attempted to tackle it from a control perspective by developing a receding horizon control strategy capable of adjusting MAPF plans in an online fashion. Finding natural combination of continuous and discrete decision spaces will allow us to solve a multitude of problems in the future. Parallel challenges could include a traveling salesman problem with dynamically changing destination availability.

Referring back to the three research objectives laid out in Section 1-1, we have addressed all three these objectives by considering (1) adjusting the ordering of AGVs based on their measured delays while maintaining collision- and deadlock-free execution guarantees of the original plan; (2) extending this notion for use in a persistent planning architecture by translating the method into a receding horizon approach; and (3) validating the proposed method in an extensive simulation environment.

8-3 Recommendations & Future Work

We list a few of these ideas below which could be addressed in future work.

Re-planning within a Sub-graph

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. An idea could be to consider the ADG structure and use this abstraction as the boundary constraints of a new MAPF problem which is only defined within the confines of a sub-graph within the entire workspace.

Planner Optimizing the Possibility of Re-Ordering

One of the weaknesses of the proposed method is the fact that it is highly dependent on the overlap of AGV routes which depends on the MAPF planner. If an area is known to be highly unpredictable, considering an alternative objective in the global planner could help improve real-time performance: namely, if the MAPF plan objective is to maximize the amount of potential re-ordering able among AGVs, if coupled with the framework in this thesis, could allow a great increase in efficiency.

Avoiding Occasional Worse Performance

We observed that the receding horizon control scheme would occasionally yield worse performance for a random fleet size and delay duration. The reason for this is that future AGV delays are not considered in the OCP of either control strategy. In fact, the proposed methods assume that the AGVs will no longer be delayed in the future, therefore only considering delays which have occurred in the past to adjust the ordering of AGVs in the present. On occasion, this assumption would prove to be invalid, since the receding horizon scheme would yield worse results than when no re-ordering was performed. A possible solution to this is the use of a robust optimization approach which can consider the potential delays in the future and derive a switching strategy which would account for this.

Evaluation on a Real Robotic Platform

Having simulated the AGVs in an extensive simulation environment, it would still be beneficial to validate the proposed approaches on real robotic hardware, subject to the natural delays which occur in warehouses with dynamic obstacles in the way. The developed ROS framework could start as a baseline for an eventual hardware implementation of the proposed methods.

Receding Horizon Control using Max-Plus Algebra

Another interesting approach to disturbance rejection in a receding horizon fashion is by formulating the problem using max-plus algebra [45, 46]. Applications include real-time train-scheduling and printer optimization. Max-plus algebra is powerful because it can be used to represent inherently non-linear schedules in a linear manner, where well-known linear control methods can be applied to design receding horizon controllers. For example, Kersbergen *et al.* [45] use max-plus linear algebra to switch the ordering of delayed trains using a receding horizon Model Predictive Control (MPC) framework. The MPC framework is used to determine if delayed train connections should be broken or not. The use of max-plus algebra, however, requires cyclic or semi-cyclic systems for analysis, a commodity which is lacking in

the MAPF problem structure. However, this method would be highly applicable to a problem where multiple AGVs perform tasks defined by a cyclic schedule.

Appendix A

Paper Submission to ICAPS 2020

The work presented in this thesis has lead to a workshop paper submitted to the International Conference on Automated Planning and Scheduling (ICAPS) 2020 in Nancy, France [1].

This work presents the initial concepts and ideas of this thesis, specifically the work presented in Chapter 3. However, the reader should be aware of a slight deviation in the nomenclature: the concept of a modified ADG in the paper is equivalent to the SADG presented in this thesis.

[1] ICAPS 2020 information: https://icaps20.icaps-conference.org/workshops/dmap/

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

Alexander Berndt[1], Niels van Duijkeren[2], Luigi Palmieri[2] and Tamás Keviczky[1]

***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*

I. Introduction

Multiple Automated Guided Vehicles (AGVs) have shown to be capable of efficiently performing intra-logistics tasks such as moving inventory in distribution centers [1]. The coordination of AGVs in shared environments is typically formulated as the Multi-Agent Path Finding (MAPF) problem, which has been shown to be NP-Hard [2]. 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 [3], [4]. This is mostly due to the abundance of application domains, such as intralogistics, airport taxi scheduling [5] and computer games [6]. Solutions to the MAPF problem include Conflict-Based Search (CBS) [7], Prioritized Planning using Safe Interval Path Planning (SIPP) [8], declarative optimization approaches using answer set programming [9], heuristic-guided coordination [10] and graph-flow optimization approaches [11].

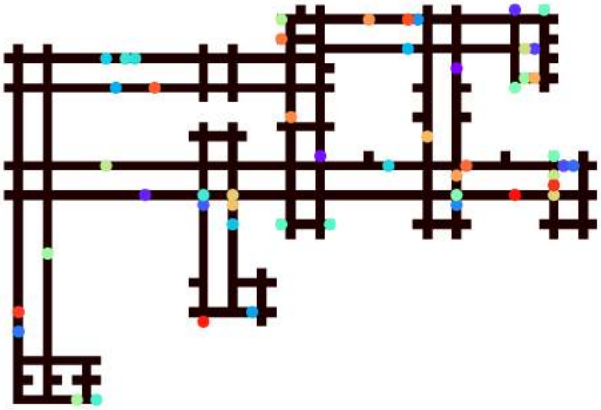


Fig. 1: A roadmap occupied by 50 AGVs (represented by colored dots). AGVs must efficiently navigate from a start to a goal position while avoiding collisions with one another, despite being subjected to delays.

Algorithms such as CBS have been improved by exploiting properties such as geometric symmetry [12], using purpose-built heuristics [13], or adopting a Mixed-Integer Linear Program (MILP) formulation where a branch-cut-and-price solver is used to yield significantly faster solution times [14].

Similarly, the development of bounded sub-optimal solvers such as Enhanced Conflict-Based Search (ECBS) [15] 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 [16]. 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 [17]. Stochastic AGV delay distributions are considered in [18], where the MAPF is solved by minimizing the expected overall delay. These robust MAPF formulations and solutions inevitably

[1]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

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

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 [19]. 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 [20].

The aforementioned plan execution solutions such as [20], [17], [18] 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 [19], [18], [17], [20] 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 II. Based on a modified version of this ADG, we introduce the concept of a reverse agent dependency in Section III. This will allow an alternative ordering of Automated Guided Vehicles, while maintaining collision avoidance constraints. In Section IV, 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 V.

II. 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 $g_i \neq g_j \ \forall \ i, j \in \{1, \dots, N\}, \ i \neq j$, an MAPF solution $\mathcal{P} = \{P_1, \dots, P_N\}$ 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 $t \in [0, \infty)$. $P_i = \{p_i^1, \dots, p_i^{N_i}\}$ 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 [16] 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.

A. Modified Action Dependency Graph

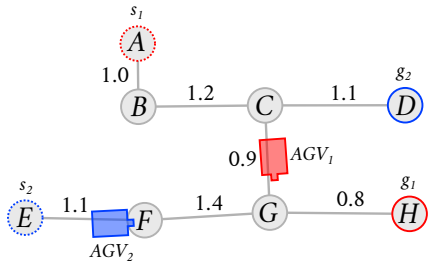
Based on a MAPF solution $\mathcal{P}$, we can construct a modified version of the original Action Dependency Graph (ADG), formally defined in Definition 2. This modified ADG encodes the sequencing of AGV movements to ensure the plans are executed as originally planned despite delays.

***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 $v_i^k = (\{p_1, \dots, p_q\}, status) \in \mathcal{V}_{ADG}$ 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 $\in \{staged, in\text{-}progress, completed\}$.*

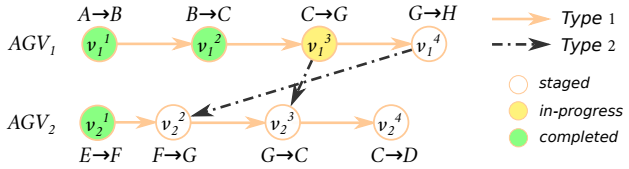
Initially, the *status* of v_i^k are *staged* $\forall \ i, k$. The directed edges in an ADG, from here on referred to as dependencies, define event-based constraints between two vertices. Formally, (v_i^k, v_j^l) implies that v_j^l cannot be *in-progress* or *completed* until $v_i^k =$ *completed*. A dependency $(v_i^k, v_j^l) \in \mathcal{E}_{\text{ADG}}$ 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 [20], 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,



(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.

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

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 $v_i^k \in \mathcal{V}_{\text{ADG}}$ 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 $v_i^k \in \mathcal{V}_{\text{ADG}}$ 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 $v \in \mathcal{V}_{\text{ADG}}$.

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 III-A. Referring to Algorithm 1, we introduce $plan(v_i^k)$ which returns the sequence of plan tuples $\{p_1, \ldots, p_q\}$ for $v_i^k \in \mathcal{V}_{\text{ADG}}$. 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.

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

Algorithm 1 Modified ADG construction based on [20]

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

// 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: $v_{\text{prev}} \leftarrow None$
5: $k = 2$
6: **for** $k = 2$ to N_i **do**
7: Append p_i^k to $plan(v)$
8: **if** $S(p) \oplus S_{\text{AGV}} \cap S(p_i^k) \oplus S_{\text{AGV}} = \emptyset$ **then**
9: Add v to $\mathcal{V}_{\text{ADG}}$
10: **if** v_{prev} not *None* **then**
11: Add edge (v_{prev}, v) to $\mathcal{E}_{\text{ADG}}$
12: $v_{\text{prev}} \leftarrow v$
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 $\mathcal{E}_{\text{ADG}}$
22: **return** $\mathcal{G}_{\text{ADG}}$

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*

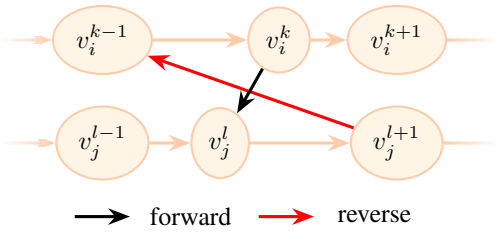


Fig. 3: A subset of an ADG with a dependency (black) and its reverse (red)

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. □

III. 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.

A. 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 $t_s(v_i^k) \geq t_g(v_j^l)$.*

***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.* □

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

B. 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 $\mathcal{G}_{ADG} = SADG(\boldsymbol{b})$ 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 $\boldsymbol{b}$ will guarantee collision-free plans, since at least one dependency of each forward-reverse dependency pair is selected, by Proposition 1. If $\boldsymbol{b}$ ensures $ADG = SADG(\boldsymbol{b})$ 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 $\boldsymbol{b}$ which ensures $ADG_{\boldsymbol{b}}$ is acyclic, while simultaneously minimizing the cumulative AGV route completion times. This is formulated as an optimization problem in Section IV.

IV. Optimization-Based Approach

Having introduced the SADG, we now formulate an optimization problem which can be used to determine $\boldsymbol{b}$ 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.

A. Translating a Switchable Action Dependency Graph to Temporal Constraints

1) 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 $v_i^k \in \mathcal{V}_{\text{ADG}}$ 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.

2) 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 $\mathcal{E}_{\text{ADG}}^{\text{Type 2}} = \{e \in \mathcal{E}_{\text{ADG}} | e \text{ is Type 2}\}$ which represents the sets of all *Type 2* dependencies. The aim here is to determine a set $\mathcal{E}_{\text{ADG}}^{\text{switchable}} \subset \mathcal{E}_{\text{ADG}}^{\text{Type 2}}$ containing the dependencies which could potentially be switched and form part of the MILP decision space, as required by Corollary 1. $\mathcal{E}_{\text{ADG}}^{\text{switchable}}$ is determined by considering the dependencies in $\mathcal{E}_{\text{ADG}}^{\text{Type 2}}$ 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 $\mathcal{E}_{\text{ADG}}^{\text{switchable}}$, the next step is to include the switched dependencies as temporal constraints. Directly referring to Section III-B, we assume m_T forward-reverse dependency pairs in $\mathcal{E}_{\text{ADG}}^{\text{switchable}}$, where the Boolean b_m is used to select the forward or reverse dependency of the mth forward-reverse dependency pair, $m \in \{1, \dots, m_T\}$. These temporal constraints can be written as

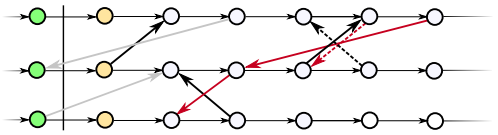


Fig. 4: Illustrative example of dependencies considered in the optimization problem. Considered dependencies are black (forward) and red (reverse). Grey dependencies are ignored.

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

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.

B. Optimization Problem Formulation

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

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

where $\boldsymbol{b} : \{0,1\}^{m_T}$ is a vector containing all the binary variables b_m and the vectors $\boldsymbol{t}_s$ and $\boldsymbol{t}_g$ contain all the variables $t_{i,s}^k$ and $t_{i,g}^k$ respectively $\forall\, k \in \{1, \dots, N_i\}, i \in \{1, \dots, N\}$.

C. 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.

***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)*

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: $ADG \leftarrow SADG(\boldsymbol{b})$

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 $t > 0$. □

D. 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 expenontially in the number 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.

1) 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.

2) 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

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

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.

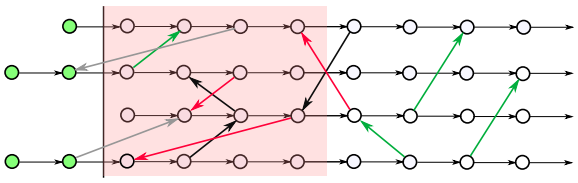
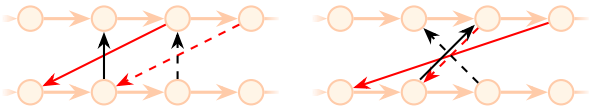


Fig. 5: Dependency selection for a horizon of 4 vertices. Switchable dependency pairs are shown in black (forward) and red (reverse). Regular dependencies considered in the MILP are green. Dependencies not considered are grey.



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

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

V. 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.* [20] 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.

A. 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 [15] 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 IV-D.

B. 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 [20]. 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 $\boldsymbol{b} = \mathbf{0}$ 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 [21]. However, based on preliminary tests, we did note better performance using the commercial solver Gurobi [22]. This yielded computational time improvements by a factor 1.1 up to 20.

C. 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:

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.

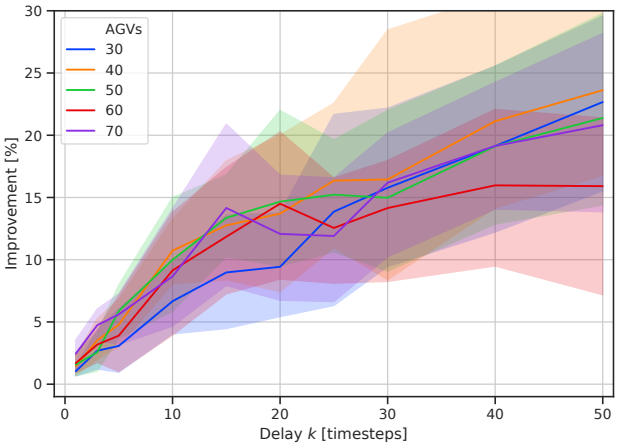
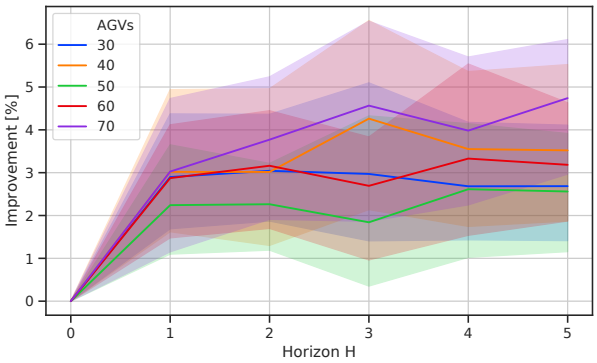
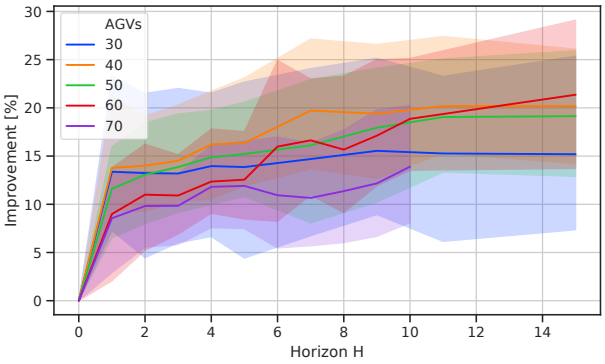


Fig. 7: Average improvement of 100 scenarios for various delay lengths and AGV group sizes. Each scenario refers to different randomly generated starts/goals and a randomly selected subset of delayed AGVs. Solid lines depict the average, lighter regions encapsulate the min-max values.



(a) Delay $k = 3$.



(b) Delay $k = 25$.

Fig. 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.

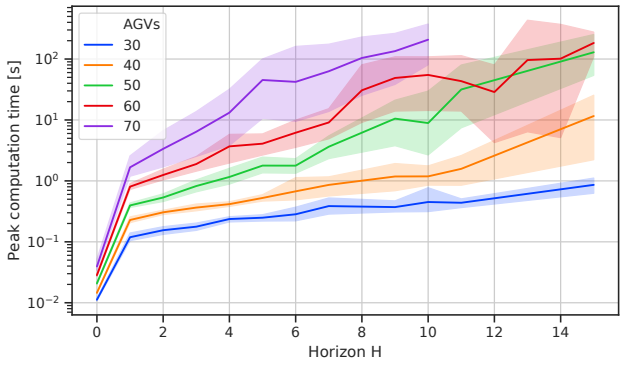


Fig. 9: The peak computation to solve the optimization problem for different AGV team sizes and considered dependency horizon lengths. This plot considers the average improvement for delays $k = \{1, 3, 5, 10, 15, 20, 25\}$. Solid lines depict the average, lighter regions encapsulate the min-max values.

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) [23], as long as the full plans are of finite length, since the optimization formulation in Eq. (4) must consider the final plan completion time.

VI. 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

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," vol. 29 of *AI Magazine*, (Menlo Park, CA), pp. 9 – 19, AAAI, 2008.

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

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

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

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

[6] 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 Transactions on Computational Intelligence and AI in games*, vol. 5, pp. 293–311, Dec 2013.

[7] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40 – 66, 2015.

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

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

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

[11] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3612–3617, May 2013.

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

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

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

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

[16] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 39–45, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

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

[18] H. Ma, S. Kumar, and S. Koenig, "Multi-agent path finding with delay probabilities," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3605–3612, 2017.

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

[20] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1125–1131, 2019.

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

[22] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2020.

[23] H. Ma, J. Li, S. Kumar, and S. Koenig, "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)*, pp. 837–845, 2017.

Bibliography

[1] M. Kessler, "Automatisiert in die Zukunft: Mit dem ActiveShuttle zur effizienten Intralogistik," *Bosch Rexroth AG, Fachpressekommunikation*, pp. 1–3, 2019.

[2] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee Using Relative Bernstein Polynomial," *IEEE International Conference on Robotics and Automation (ICRA) 2020*, p. 7, 2020.

[3] M. R. Ryan, "Exploiting Subgraph Structure in Multi-Robot Path Planning," *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, Mar. 2008.

[4] "ROS move_base Package: ROS interface for configuring, running, and interacting with the navigation stack on a robot.." http://wiki.ros.org/move_base. Accessed: 2020-05-17.

[5] D. K. Elms and P. Low, *Global Value Chains in a Changing World*, ch. 14, pp. 329–360. 154 rue de Lausanne, CH-1211 Geneva 21, Switzerland: WTO Publications, 2013. ISBN: 978-92-870-3882-1.

[6] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," vol. 29 of *AI Magazine*, (Menlo Park, CA), pp. 9 – 19, Association for the Advancement of Artificial Intelligence (AAAI), 2008.

[7] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic foundations of robotics X*, pp. 157–173, Springer, 2013.

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

[9] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1125–1131, 2019.

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

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

[12] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-robot cooperation in the martha project," *IEEE Robotics Automation Magazine*, vol. 5, pp. 36–47, March 1998.

[13] S. Alarie and M. Gamache, "Overview of solution strategies used in truck dispatching systems for open pit mines," *International Journal of Surface Mining, Reclamation and Environment*, vol. 16, no. 1, pp. 59–76, 2002.

[14] 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 Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[15] H. Ma, J. Li, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 837–845, 2017.

[16] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2018.

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

[18] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40 – 66, 2015.

[19] J. Li, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Symmetry-breaking constraints for grid-based multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 6087–6095, 2019.

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

[21] E. Lam, P. L. Bodic, D. Harabor, and P. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19), International Joint Conferences on Artificial Intelligence Organization*, pp. 1289–1296, 2019.

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

[23] K. S. Yakovlev and A. Andreychuk, "Any-angle pathfinding for multiple agents based on SIPP algorithm," in *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017* (L. Barbulescu, J. Frank, Mausam, and S. F. Smith, eds.), p. 586, AAAI Press, 2017.

[24] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller, "A General Formal Framework for Pathfinding Problems with Multiple Agents," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, p. 7, 2013.

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

[26] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3612–3617, May 2013.

[27] C. Wilt and A. Botea, "Spatially distributed multiagent path planning," in *Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling*, pp. 332–340, 2014.

[28] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 39–45, International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[29] H. Ma, W. Hönig, T. S. Kumar, N. Ayanian, and S. Koenig, "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 7651–7658, 2019.

[30] M. Čáp, J. Gregoire, and E. Frazzoli, "Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5113–5118, IEEE, 2016.

[31] W. Hönig, T. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," in *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, pp. 477–485, 2016.

[32] H. Ma, S. Kumar, and S. Koenig, "Multi-agent path finding with delay probabilities," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3605–3612, 2017.

[33] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[34] T. K. Ralphs, L. Ladányi, and M. J. Saltzman, "Parallel branch, cut, and price for large-scale discrete optimization," *Mathematical Programming*, vol. 98, no. 1-3, pp. 253–280, 2003.

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

[36] N. Koenig and J. Hsu, "The many faces of simulation: Use cases for a general purpose simulator," in *ICRA 2013. Workshop on Developments of Simulation Tools for Robotics & Biomechanics*, 2013.

[37] Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System (ROS)." https://www.ros.org, Version: Melodic Morenia, date: May, 2018.

[38] T. A. Toffolo and H. G. Santos, "Python MIP (Mixed-Integer Linear Programming) Tools: version 1.9.2," June 2020.

[39] Gerkey, Braian, "Robotic Operating System 2 (ROS2)." https://index.ros.org/doc/ros2/, Version: Foxy Fitzroy, date: June, 2020.

[40] Stanford Artificial Intelligence Laboratory et al., "Robotic Operating System (ROS): Navigation Stack." https://www.ros.org/navigation, Version: Melodic Morenia, date: May, 2020.

[41] D. Lu, "DWB Local Planner using the Dynamic Window Approach." https://github.com/locusrobotics/robot_navigation, Version: Melodic Morenia, date: May, 2020.

[42] D. Lu, M. Ferguson, and A. Hoy, "DWB Local Planner using the Dynamic Window Approach." http://wiki.ros.org/dwa_local_planner, Version: Melodic Morenia, date: June, 2020.

[43] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics Automation Magazine, IEEE*, vol. 4, pp. 23 –33, Mar. 1997.

[44] "ROS AMCL Package: A Probabilistic Localization System for a Robot Moving in 2D." http://wiki.ros.org/amcl. Accessed: 2020-05-18.

[45] B. Kersbergen, B. De Schutter, and T. van den Boom, "Distributed model predictive control for railway traffic management," *Transportation Research Part C: Emerging Technologies*, vol. 68, pp. 462 – 489, 2016.

[46] B. De Schutter and T. van den Boom, "Max-plus algebra and max-plus linear discrete event systems: An introduction," in *2008 9th International Workshop on Discrete Event Systems*, (Goteborg, Sweden), pp. 36–42, IEEE, 2008.

Glossary

List of Acronyms

ADG	Action Dependency Graph
AI	artificial intelligence
AGV	Automated Guided Vehicle
AGVs	Automated Guided Vehicles
AMCL	Adaptive Monte-Carl Localization
CBC	Coin-Or Branch-and-Cut
CBS	Conflict-Based Search
CCBS	Continuous Conflict-Based Search
ECBS	Enhanced Conflict-Based Search
ECBS	Bounded Sub-Optimal Conflict-Based Search
MAPF	Multi-Agent Path Finding
MAPD	Multi-Agent Pickup and Delivery
MILP	Mixed-Integer Linear Program
MAPF-DP	multi-agent path finding problem with delay probabilities
MPC	Model Predictive Control
OCP	Optimal Control Problem
RHC	Receding Horizon Control
ROS	Robot Operating System
SIPP	Safe Interval Path Planning

SADG Switchable Action Dependency Graph

UAVs Unmanned Aerial Vehicles

List of Symbols

$\hat{t}_g(v)$ Planned time a vertex v changes from *in-progress* to *completed*
$\hat{t}_s(v)$ Planned time a vertex v changes from *staged* to *in-progress*
$\mathcal{D}$ List of dependencies
$\mathcal{D}_{\text{static}}$ Subset of the list of dependencies that exclude switchable dependencies
$\mathcal{D}_{\text{switchable}}$ Subset of the list of switchable dependencies
$\mathcal{E}$ Set of edges of a graph
$\mathcal{G}$ Graph object consisting of vertices and edges
$\mathcal{P}$ Set of plans for each AGV
$\mathcal{P}_i$ List of plan tuples corresponding to AGV_i
$\mathcal{V}$ Set vertices of a graph
N Number of AGVs in a fleet
N_i Number of vertices in the SADG associated with AGV_i
$t_g(v)$ Actual time a vertex v changes from *in-progress* to *completed*
$t_s(v)$ Actual time a vertex v changes from *staged* to *in-progress*