OUKA
The University of Osaka
Institutional Knowledge Archive

Title	Planning and Acting in Multi-Environment Aerial, Surface, and Aquatic Missions with Multi-Robot Teams
Author(s)	De La Rochefoucauld, Virgile Georges Armand
Citation	大阪大学, 2026, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/105498
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

Planning and Acting in Multi-Environment Aerial, Surface, and Aquatic Missions with Multi-Robot Teams

Submitted to
Graduate School of Information Science and Technology
The University of Osaka

January 2026

Virgile DE LA ROCHEFOUCAULD

Thesis Committee

Prof. Yuki Uranishi (Osaka University)
Prof. Yasushi Sakurai (Osaka University)
Prof. Naoya Chiba (Osaka University)
Prof. Photchara Ratsamee (Kansai University)
Prof. Félix Ingrand (LAAS-CNRS, France)

List of Publications

Peer-Reviewed Journals

1. De La Rochefoucauld, V., Lacroix, S., Ratsamee, P., Ingrand, F., Uranishi, Y. (2025). A Modular Execution Architecture for Robust Multi-Robot Planning and Acting in Trans-Media Environments. *IEEE Access*.

Peer-Reviewed Conference Proceedings

1. De La Rochefoucauld, V., Lacroix, S., Ratsamee, P., Takemura, H. (2024). Solving Multi-Robot Task Allocation and Planning in Trans-Media Scenarios. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2024)*.

Acknowledgments

This thesis is the culmination of a journey that took me across borders and disciplines, and through many challenges and discoveries. None of this would have been possible without the generous support and encouragement of the people I met along the way.

First, I would like to thank my advisor, Photchara Ratsamee, who supported me for over four years and encouraged me to pursue my doctoral studies. Since my first internship, he has accompanied me in discovering Japan and adapting to life far from my home country, from everyone I knew. Thank for the many meals together, your help, your friendship and your support. I would also like to acknowledge my dear director, Simon Lacroix, who unfortunately passed away before I could present our work. This would not have been possible without him. More than an academic advisor, he was a mentor and a friend — always there to guide, counsel, and help me grow as a researcher. His loss affected me deeply. Simon was the first to believe in this dual Ph.D. project and worked tirelessly to make it possible. As I complete this long journey, I hope I have inherited some of the qualities that made him such a remarkable man. Without these two, none of this would have been possible.

I would also like to thank Félix Ingrand, who, despite his own feelings following our loss, took the time to help and encourage me to finish this thesis. You helped me continue and bring this work to completion — you became part of it. My gratitude also goes to my two Japanese directors: Professor Haruo Takemura, who first welcomed me into his lab and helped me with the administrative work of establishing the double Ph.D. program until his retirement, and Professor Yuki Uranishi, who continued the supervision and supported me in completing the thesis on the Japanese side amid the difficulties of my situation.

I would like to thank INSA Toulouse and Osaka University for creating this challenging double-degree Ph.D. program, drafting the

necessary agreements, and establishing a common ground between the two institutions. I am also grateful to all the researchers at LAAS-CNRS who helped me develop my understanding of automated planning — a discipline I knew nothing about before starting this Ph.D. My warmest thanks go to my wonderful colleagues and friends who welcomed me back and forth between Japan and France: Idriss, Antoine, Adrien, Smail, Simon, Valentin, Jérémy, Roland, Émile, William, Harn, Ming, David, Jason, Harold, Luc, Bastien, Stephany, Lou, and many others. I shared unforgettable moments with them, and they kindly tolerated (and enjoyed) my questionable sense of humor.

I would especially like to thank my family for always supporting me and my ambitions. Special thanks to my brother, Ambroise, for being a huge source of support, inspiration, and love, and for listening to my frustrations and motivating me; and to Semira, for being as supportive, kind and sweet as always. To my father, thank you for supporting me throughout my Ph.D., for being an inspiration, and for pushing me to become a better man. You are a model to me, and I hope to become someone who resembles you. Thank you, Naima, for being there whenever I needed you. Thank you, Mom, for always taking my calls — late or early — for listening, offering sharp advice, and providing a comforting ear. You always manage to make the world a better place with each conversation. Thank you, Bruno, for teaching me, supporting me throughout my life, and believing in me even when I was failing classes in middleschool. A heartfelt thank you to my uncle Charles, who came to visit me in Japan everytime and helped me on countless occasions. You taught me so much about work ethic, perseverance, and the importance of believing in myself.

Finally, I want to express my profound gratitude to my fiancée, Hitomi. Without you, none of this would have been possible. You were always there for me, and your support went far beyond anything I could ever write. You cared for me, believed in me, and never

doubted my success. You made the difficult choice to leave Japan and follow me. Thank you for bringing Nina, our dog, into my life. Our long walks helped me organize my thoughts and find new ideas for my work. Your love, patience, and care helped me get through the hardest times.

Abstract: Recent years have witnessed the emergence of increasingly complex robotic systems, operating in different environments while collaborating within heterogeneous teams. These advances have opened the way to new applications, such as the exploration of rough terrains, environmental monitoring, or, more simply, the development of autonomous logistics platforms. Nevertheless, planning and executing missions in such contexts remains a major challenge: coordinating robots with diverse capabilities, ensuring the synchronization of their actions, and guaranteeing the robustness of plans in the face of unforeseen events are central issues for the development of such robotic systems.

With the recent emergence of trans-media robots capable of operating across several domains (aerial, aquatic, and terrestrial), new challenges arise in both planning and execution. Their ability to adapt and operate in various environments introduces additional planning complexity, execution uncertainties during complex tasks, and a higher frequency of mechanical failures—particularly during transitions between environments.

To address these challenges, we present the Adaptive Modular Architecture (AMA), composed of two main components: AMA-Plan, a planner that restructures the problem to be solved, and AMA-Exec, a distributed execution system that integrates temporal reasoning and plan monitoring through an extended Simple Temporal Network (STN), combined with a multi-level failure detection and recovery mechanism.

- AMA-Plan decomposes and restructures complex problems into sequentially linked sub-problems: each team is responsible for solving one of these sub-problems, while the robots follow resolution paths in which they join different teams according to their capabilities, in order to satisfy the constraints specific to each sub-problem.

- AMA-Exec ensures the operational management of the plan through the coordination of the distributed real-time execution of plans, the monitoring of temporal constraints for each team's concurrent actions, and the maintenance of mission continuity in the event of desynchronization or the failure of one or more robots.

We evaluated AMA through a series of experiments in simulated environments involving homogeneous, heterogeneous, and transmedia robot teams. These experiments validated the system's ability to maintain temporal coherence and mission continuity despite action desynchronizations, various robot failures, and dynamic variations in the mission environment. The results highlight three key insights:

- modularity as a central factor of system robustness and scalability;

- the importance of temporal supervision to resynchronize teams and absorb temporal drifts;

- and the effectiveness of centralized planning combined with a distributed and reactive execution system, enabling multiple levels of adaptation.

Through this thesis, we sought to demonstrate that a clear articulation between centralized symbolic planning and reactive distributed execution provides an effective approach to strengthening the robustness and continuity of multi-robot missions in dynamic environments.

Keywords: Multi-robot systems, Multi-environment (transmedia) robotics, Task planning and execution, Autonomous missions

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *05/11/2025* **par :**

Virgile de La Rochefoucauld

Planning and Acting in Multi-Environment Aerial, Surface, and Aquatic Missions with Multi-Robot Teams

---

JURY

Vicente MATELLÁN	Catedratico de Universidad	Rapporteur
Andrea ORLANDINI	Primo Ricercatore	Rapporteur
Karen GODARY-DEJEAN	Directrice de recherche	Examinateur
Charles LESIRE	Directeur de recherche	Examinateur
Simon LACROIX	Directeur de recherche	Directeur de thèse
Yuki URANISHI	Full professor	Co-directeur de thèse
Photchara RATSAMEE	Professor	Co-encadrant de thèse
Félix INGRAND	Chargé de recherche	Co-directeur de thèse

École doctorale et spécialité :

EDSYS : Informatique 4200018

Unité de Recherche :

LAAS-CNRS

Directeur(s) de Thèse :

Simon LACROIX, *Yuki URANISHI*, *Photchara RATSAMEE* et

Contents

Acronyms

AMA Adaptive Modular Architecture. 149, 152, 153

BT Behavior Tree. 8, 50, 53, 54, 55, 56, 57, 58, 60, 61, 62, 86, 105, 121, 128, 139, 141, 150, 152

BT.CPP BehaviorTree.CPP library. 139, 140

DEM Distributed Execution Manager. 62, 151

HAAV Hybrid Aquatic–Aerial Vehicle. ix, 2, 11, 12, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28

MRS Multi-Robot System. 3

MRTA Multi-Robot Task Allocation. ii, iii, ix, 3, 4, 28, 31, 40, 41, 42, 43, 44, 45, 46, 47, 48, 63, 64, 65, 68, 71, 149

PDDL Planning Domain Definition Language. 33, 34, 35, 39, 66, 69, 86, 89, 92, 96, 101, 103, 107, 108, 110, 121, 128, 135, 139, 145, 149, 152

STN Simple Temporal Network. ix, 8, 55, 62, 64, 70, 71, 80, 85, 90, 96, 101, 102, 106, 113, 120, 121, 131, 133, 135, 145, 149, 152

STRIPS Stanford Research Institute Problem Solver. 33, 34, 35, 39

UAV Unmanned Aerial Vehicle. 1, 2, 3, 4, 13, 16, 17

UGV Unmanned Ground Vehicle. 1, 2, 4, 13

UUV Unmanned Underwater Vehicle. 1, 2, 4

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Context and Motivations

1.1.1 Multi-Medium Autonomous Multi-Robot Systems

The majority of robotic systems are designed for a single operational domain — air, land, or water — limiting their versatility. However, many real-world applications, including disaster response, environmental monitoring, and industrial inspections, require operation across multiple mediums. To address this limitation, multi-robot systems have been proposed, employing heterogeneous teams of specialized robots (e.g., Unmanned Aerial Vehicle (UAV), Unmanned Ground Vehicle (UGV), and Unmanned Underwater Vehicle (UUV): (Shkurti et al., 2012; Kumar and Padhy, 2015; Vasilijevic et al., 2017)). The implementation of these solutions gives rise to challenges related to coordination, resource efficiency, and task allocation. An alternative approach involves the utilization of trans-media robots, which possess the capacity to operate across multiple

mediums without necessitating the deployment of separate robots for each environment. The integration of cross-medium capabilities into a unified platform enables trans-media robots to offer enhanced flexibility and efficiency in multi-domain missions, thereby reducing the necessity for inter-agent coordination and minimizing operational constraints.

1.1.2 Trans-Media Robots: A New Solution

In this thesis, we use the term trans-media robots to denote systems capable of operating across multiple mediums (air, land, and water). We use multi-medium as a synonym when emphasizing the operational context.

Trans-media robots represent a novel category of autonomous systems designed to transition between different operational environments, thereby significantly expanding mission versatility (e.g., an aerial-aquatic robot that can both fly and navigate in water (Alzu'bi et al., 2018; Pedroso et al., 2022), an amphibious robot capable of crossing rivers (Ijspeert et al., 2007; Shi et al., 2013), or a legged drone that can walk on ground and fly (K. Kim et al., 2021; Shin et al., 2024; Martynov et al., 2024)). In contrast to conventional UAVs, UGVs, and UUVs, which are constrained to a single domain, trans-media robots exhibit the capacity to transition between different mediums, reducing the necessity for numerous specialized agents in a given mission. The capability to operate across multiple domains confers several advantages, including enhanced adaptability, scalability, and mission efficiency. A notable example of this innovation is HAAV, which have garnered significant attention from the research community. Some studies have provided comprehensive surveys and reviews of their technological advancements, design principles, and operational challenges (Yao et al., 2023; Yang et al., 2015; Tan and B. M. Chen, 2021). However, while trans-media robots enhance mission flexibility, their design introduces multiple challenges:

- **Energy Efficiency and Transition Costs:** Switching between mediums necessitates energy-intensive mechanisms, such as buoyancy control, propulsion system reconfiguration, or aerodynamic adjustments. Power management becomes a concern, as transitions often lead to sudden spikes in energy consumption.
- **Structural Trade-offs:** Designs must balance multiple mobility requirements, increasing mechanical complexity. Propulsion systems for aerial and aquatic mobility add weight and structural constraints that impact performance.

- **Medium Transition:** Medium transitions require preparatory actions like velocity control, reorientation, and stabilization before executing the transition phase. Poor management of transitions can lead to mission failure due to environmental interactions like turbulence, landing impact forces, or incorrect buoyancy adjustments.

- **Action Reconfiguration and Task Adaptation:** Unlike fixed-medium robots, the action set of trans-media robots are modified, depending on their current medium. For instance, a UAV transitioning into water must deactivate aerial rotors, adjust its control mechanisms for buoyancy, and switch to underwater propulsion.

- **Necessity of System Redundancy:** Although redundancy is a common Multi-Robot System (MRS) design principle, trans-media robots require greater robustness due to the increased variability in environmental conditions, designs, adaptation, and potential agent failure. Resilient error compensation mechanisms are needed in trans-media operations, where external disturbances across multiple mediums introduce more frequent and unpredictable failures.

While trans-media robots extend mission versatility, their complexity directly impacts how planning and execution frameworks must be designed. The next section explores these challenges, focusing on how existing multi-robot task allocation and execution models must evolve to accommodate cross-medium constraints.

1.2 Planning and Execution Constraints

MRTA planning and execution frameworks operate with well-defined agent capabilities and action definitions (Gerkey and Matarić, 2003; Gerkey and Matarić, 2004). However, in trans-media missions, robot actions must be planned with access to the full action domain, while feasibility depends on the current medium. This results in a more complex problem formulation. Planners must account for state-dependent action constraints, cross-medium dependencies, and dynamic action feasibility checks. This section examines how trans-media robots confront classical MRTA models, introducing new challenges in task allocation, execution adaptability, and failure handling.

1.2.1 Challenges in Planning for Trans-Media Robots

MRTA frameworks generally classify teams based on their agent distribution, task complexity and allocation mechanisms. In homogeneous teams of robots, agents possess identical capabilities, and the planning process emphasizes the optimization of complex task distribution. That is to say, agents share similar capabilities but must coordinate to complete a broad set of tasks (e.g., UAV fleets (Fu et al., 2019)). In heterogeneous teams, agents have predefined roles and capabilities. Tasks are allocated based on static constraints. For instance, a team of UAVs, UGVs, and UUVs is composed of robots that are each specialized for distinct tasks in the domain (Carreno et al., 2020; Shkurti et al., 2012). Trans-media robots are not categorizable as either of these types of teams. They exhibit capabilities akin to heterogeneous teams, but these depend dynamically on their current state. This leads to additional planning complexity, the feasibility of actions depending on medium transitions, environmental conditions, and coordination requirements.

This results in increased state-space complexity, as planners must reason over medium dependent actions, transition constraints, and trans-medium action dependencies. Additionally, trans-media planning introduces new interdependencies between tasks, as certain operations—such as underwater communication—may require multiple agents working together to ensure continuity.

1.2.2 Challenges in Executing Tasks for Trans-Media Robots

Trans-media missions present specific challenges during execution, including frequent failures (Yao et al., 2023), mission complexity, and the scarce distribution of robots across independent goals. These challenges require an execution system capable of handling unexpected failures while maintaining operational efficiency. In contrast to conventional execution models, which assume a robot's presence within a single medium, trans-media robots must actively manage their transitions, accounting for both mechanical stability and environmental conditions. It is imperative to actively monitor critical transition phases to ensure the successful execution of trans-media missions. The execution of actions such as submerging, landing, or takeoff on water necessitates precise control and preparatory steps. Furthermore, the execution robustness is a pivotal concern, as failures may stem from both internal malfunctions (e.g., transition mechanism failure) and external disturbances (e.g., turbulence or water currents). Due to their increased vulnerability to execution failures, trans-media robot teams must be able to persist in operation

without disrupting the entire mission. A system that enables local failure recovery is therefore more essential in trans-media execution than in traditional multi-robot systems.

1.3 Problem Statement

Figure 1.1: Artistic view of trans-media robots evolving in a aquatic-aerial environment

The problem addressed in this thesis is the design of a scalable planning and acting framework for complex multi-robot missions that can handle the challenges mentioned previously. Throughout the thesis we demonstrate the system with an environment pond pollution sampling mission (Kumar and Padhy, 2015) illustrated artistically in Figure 1.1. A team of aerial-aquatic robots is tasked with sampling water quality points across multiple sampling sites (fig: 1.2). The mission entails the navigation to designated sampling points using aerial mobility, followed by a transition into water mode, subsequent stabilization and collaboration to collect underwater samples. During the sampling process, real-time communication is essential, often requiring at least one robot to remain at the surface while others operate underwater. Furthermore, the system must demonstrate resilience to unexpected failures, such as blocked transition points, ensuring that the mission can adapt dynamically without compromising overall objectives.

This scenario highlights challenges in planning, execution, and

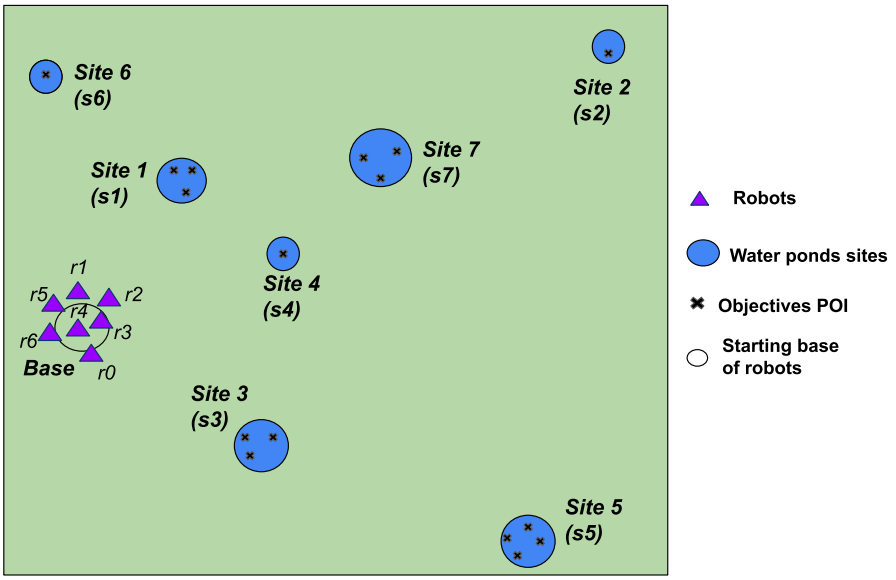


Figure 1.2: A typical representation of a trans-media environmental sampling mission with 7 robots and 7 sites.

failure handling, which this thesis aims to address:

- **Scalability and Complexity in Planning:** Multi-robot missions require efficient planning across teams with diverse capabilities. The system must support complex task dependencies, task allocation, and dynamic coalition formation while ensuring computational feasibility for large-scale missions.
- **Multi-Medium Transitions in Task Allocation and Execution:** Trans-media robots operate across different mediums, requiring task allocation and execution strategies that integrate transition actions while optimizing execution efficiency, environmental dependencies and balancing workload distribution in multi-robot operations.
- **Robust Execution Strategies for Failure Handling:** Real-world missions necessitate real-time failure handling, transition monitoring, and recovery strategies to adapt to execution failures, transition errors, and changing conditions.
- **Independent and Parallel Execution Capabilities:** In planning-to-acting frameworks, coordinating task execution across agents often relies on centralized coordination. In trans-media missions, the distribution of robots across sites and their

limited direct interaction present challenges. Specifically, failures affecting one agent should not disrupt the execution of others. To ensure mission continuity, the execution framework must support independent task execution and adaptive replanning. These mechanisms allow for the management of local failures without compromising the overall mission.

- **Generalizability to Various Multi-Robot Teams:** The proposed framework integrates with established planning and execution methodologies, ensuring adaptability across different missions and robotic teams.

1.4 Contributions and Thesis Outline

This thesis presents two major contributions aimed at improving the planning and execution capabilities of multi-robot teams, for complex trans-media missions. The proposed AMA framework (Figure 1.3) aims at enhancing both scalability in task allocation and execution robustness, addressing the challenges outlined in the problem statement.

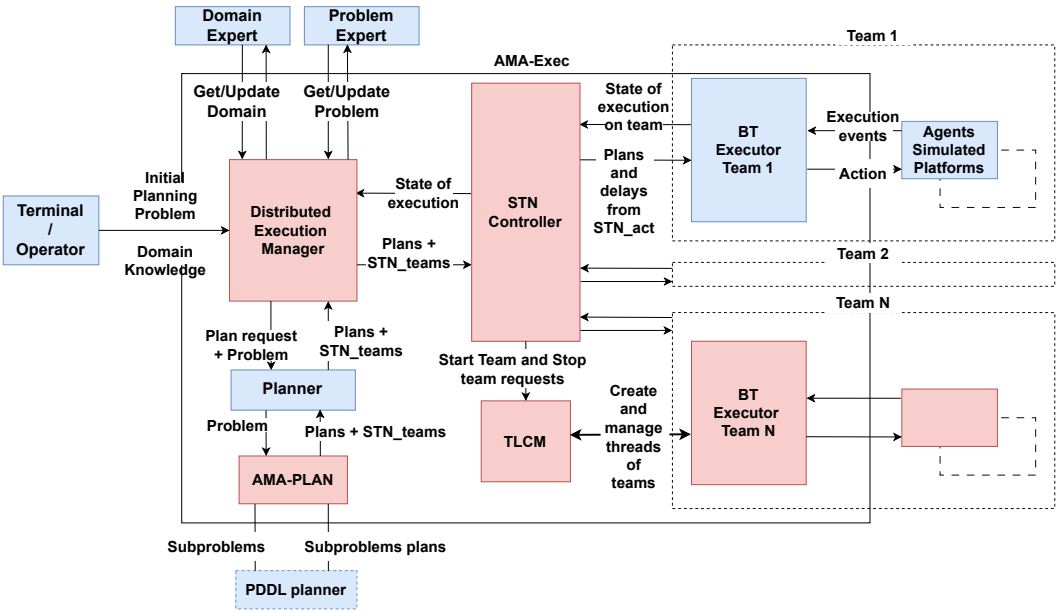


Figure 1.3: Diagram representing the structure of the AMA planning to acting system

1.4.1 Contribution 1: AMA-Plan – A Scalable Multi-Robot Planner through Modular Decomposition

The **Adaptive and Modular Architecture for Planning** (AMA-Plan) framework introduces a structured approach to multi-robot mission planning, focusing on reducing computational complexity with negligible loss in mission optimality. An initial version of this framework was presented in our IROS 2024 paper (De La Rochefoucauld et al., 2024). AMA-Plan achieves this by:

- **Decomposing the planning problem** into sub-problems, including coalition formation, optimal mission path estimation, and cost-effective agent allocation.
- **Improving computational efficiency** by avoiding monolithic optimization, allowing for scalable planning in large and complex scenarios.
- **Enhancing adaptability** by enabling local replanning for specific sub-problems.

1.4.2 Contribution 2: AMA-Exec – A Robust and Adaptive Execution Framework for Distributed Multi-Robot Teams

The **Adaptive and Modular Architecture for Execution (AMA-Exec)** ensures robust and adaptive execution by dynamically managing plan sequences in real-time. . It is designed to:

- **Handle complex and large-scale multi-robot scenarios efficiently** AMA-Exec integrates with the **AMA-Plan** planner to identify the near-optimal decomposition of the problem into structured sub-problems, ensuring efficient task distribution and enabling parallel, independent execution.
- **Leverage a two-stage execution model for adaptive coordination** AMA-Exec combines Behavior Tree (BT) for local, reactive decision-making with STN for higher-level temporal coordination, allowing for flexible execution adjustments and smooth plan adaptation.
- **Enable distributed and parallel execution** The framework minimizes synchronization bottlenecks by allowing agents in distributed teams to execute their tasks independently, preventing system-wide downtime when local adjustments or failure recoveries are required.

- **Enhance execution robustness and recovery** AMA-Exec ensures robust mission execution by incorporating real-time monitoring, modular replanning, and local failure recovery strategies, preventing unnecessary disruptions and improving overall system efficiency.

1.4.3 Thesis Structure

The thesis is organized as follows:

- **Chapter 2** introduces the context and challenges of trans-media robotics, including mission characteristics and operational requirements.
- **Chapter 3** reviews the state of the art in multi-robot planning and execution, highlighting approaches relevant to complex missions.
- **Chapter 4** introduces **AMA-Plan**, detailing its formalism, modular problem decomposition approach, and comparison with classical solvers.
- **Chapter 5** presents **AMA-Exec**, explaining its dynamic modular execution model and demonstrating its ability to handle failures and optimize real-time execution success.
- **Chapter 6** evaluates **AMA-Plan** and **AMA-Exec** through various case studies, demonstrating their effectiveness in homogeneous, heterogeneous, and trans-media multi-robot teams.
- **Chapter 7** concludes the thesis by summarizing key contributions, discussing broader implications, and outlining future research directions and potential applications.

Chapter 2

Context and Challenges of Trans-Media Robots

Contents

This chapter presents a comprehensive introduction to trans-media robotics, conceptualizing it as a novel extension of multi-robot systems. It delves into the capabilities, challenges, and ramifications of trans-media robotics on planning complexity. We introduce the concept and state-of-the-art in trans-media robotics, with a focus on HAAVs. It presents a taxonomy of system architectures, reviews transition strategies and mechanical implementations, and concludes with an analysis of challenges, application domains, and implications for autonomous multi-robot planning.

2.1 Definition and Scope

The concept of hybrid systems, which are defined as the integration of disparate technological components, has been a subject of exploration since the early 20^{th} century. A notable example of this exploration is the development of hybrid automobiles. However, the notion of integrating distinct components or modalities to leverage complementary capabilities has been observed in a variety of engineering disciplines. In the domain of robotics, for instance, the integration of disparate forms of locomotion, sensing, control, and energy systems within a cohesive architecture is a hallmark of hybrid systems. This integration is driven by the objective of enhancing performance, enabling new capabilities, or allowing systems to adapt to variable tasks or environments. This section introduces hybrid robotic systems operating across multiple domains, provides a definition and taxonomy for their use, and demonstrates their specificity and the various solutions that exist, with a heavy focus on HAAVs systems.

2.1.1 What is a Hybrid Robot?

In order to provide a more precise definition of "hybrid robots," it is necessary to specify the particular aspect or aspects on which the focus will be directed. The term "hybrid" is inherently vague and can be interpreted in a variety of ways. For the purpose of this discussion, the term can be defined as follows:

> *A system that integrates multiple technologies, methodologies, or approaches to achieve a specific objective.*

This definition encompasses multiple facets:

Hybrid Energy Sources: Systems integrating diverse energy sources (e.g., battery + solar, electric + combustion) to optimize balance between endurance and power demand.

Hybrid Control and Autonomy: Systems that may operate across a spectrum of autonomy, ranging from fully teleoperated to self-directed. Dynamic adaptation or human-robot teaming is often required.

Hybrid Perception: Systems that employ a combination of different sensing modalities (e.g., sonar, LiDAR, cameras) to address challenges related to perception across varied mediums.

Hybrid Locomotion: Such systems possess the capability to traverse varied terrains and environments. This signifies the ability

to navigate using disparate driving systems within a single environment (e.g., wheels, legs, etc.) as well as between distinct physical environments (e.g., land to air, water to air), thereby introducing the concept of trans-media capability.

In such systems, components with disparate characteristics, behaviors, or functionalities are integrated to work synergistically. The system is designed to leverage the strengths of each component to enhance overall performance and mitigate limitations inherent to individual modalities. The term "multimodal" is also frequently employed in the literature to describe such systems, reflecting their capacity to operate across multiple domains or modes of function.

2.1.2 From Hybrid to Trans-Media Systems

A particularly relevant and new subcategory of hybrid systems in robotics is that of robots with **trans-media capability**, the ability to move between environments with different physical constraints, such as transitions from air to water or land to air and perceptual categories. These transitions bring changes in locomotion, sensing, actuation, and often energy management. In contrast to conventional heterogeneous robot teams, which are constrained to a single medium (e.g., UAVs for air, UGVs for ground), trans-media robots integrate multiple modalities within a unified platform, thereby facilitating highly adaptable mission execution.

However, the notion of "medium" necessitates further elucidation. While the categories of "air," "land," and "water" are often considered intuitive, finer-grained distinctions, such as surface versus subsurface aquatic environments or indoor versus subterranean terrestrial zones, may necessitate additional abstraction. To address these nuances, we propose the following working definition:

> *A medium is a spatial area whose boundary is uncrossable to unimodal systems and whose traversal requires a specific action and behavioral change from hybrid systems.*

According to the aforementioned definition, three primary trans-media robotic categories are identified, as illustrated in Figure 2.1:

- Terrestrial–Aquatic
- Aerial–Terrestrial
- Aerial–Aquatic

While the primary focus of this thesis is on aerial–aquatic systems, it is important to acknowledge the significant contributions of

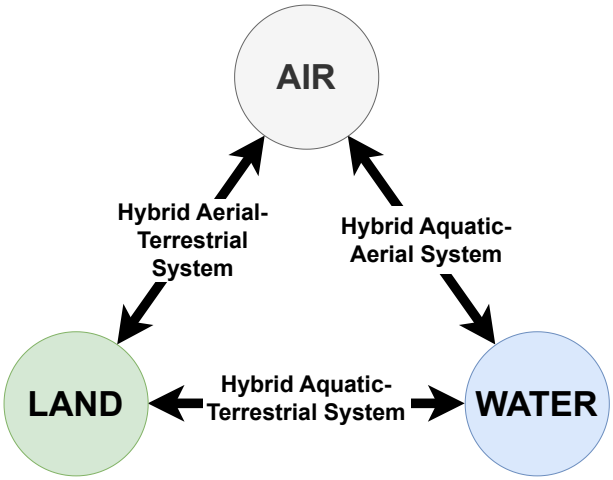


Figure 2.1: The three main trans-media ability relations, adapted from (Yao et al., 2023), Figure 22

two other trans-media categories to robotics research: **terrestrial–aquatic** and **aerial–terrestrial** systems.

Robots capable of operating across both land and water, terrestrial–aquatic vehicles, have been extensively explored in ecological monitoring, infrastructure inspection, and amphibious search-and-rescue missions. A pioneering example is AmphiBot II (Ijspeert et al., 2007; Crespi and Ijspeert, 2009), a modular serpentine robot developed at EPFL, designed to investigate coordinated swimming and crawling using central pattern generators. This system is well-suited for navigation through wetlands and flooded pipelines, thanks to its bio-inspired control and morphology. Similarly, ART (Amphibious Robotic Turtle) (Baines et al., 2022), developed by Yale University, demonstrates adaptive morphogenesis to switch between land and aquatic propulsion, enabling robust traversal in coastal or estuarine terrain where transitions between mud, rocks, and shallow waters are frequent.

Conversely, aerial–terrestrial robots provide a complementary set of capabilities, enabling aerial access and ground interaction in complex environments. For instance, DALER (Deployable Air–Land Exploration Robot (Daler et al., 2015)) employs foldable wings that function as leg-like appendages for crawling after aerial deployment. Initially designed for search-and-rescue and planetary exploration, DALER exemplifies mobility flexibility in cluttered and uneven terrain. Another innovative system is FlyCroTug, a micro aerial vehicle equipped with anchoring systems and winches. Inspired by the load-handling abilities of wasps, FlyCroTug (Estrada et al., 2018)

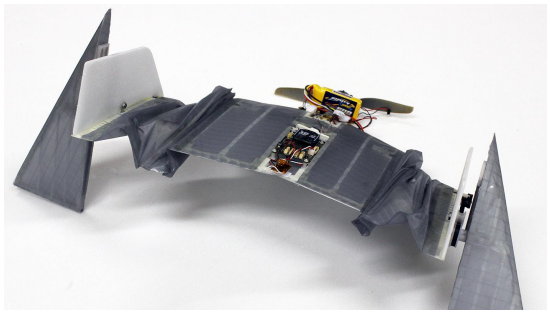
(a) AmphiBot II navigating through wetland terrain. (Crespi and Ijspeert, 2009)

(b) ART (Amphibious Robotic Turtle) on land and water (Baines et al., 2022).

Figure 2.2: Examples of terrestrial–aquatic robots designed for hybrid environments.

can latch onto vertical surfaces and manipulate objects up to 40 times its weight, enabling actions like lifting sensors into collapsed buildings or operating door handles during disaster response.

(a) FlyCroTug using tethered manipulation in cluttered indoor environments (Estrada et al., 2018).



(b) DALER robot transitioning from flight to ground locomotion using morphing wings (Daler et al., 2015).

Figure 2.3: Examples of terrestrial-aerial robots designed for hybrid exploration in complex environments.

These systems demonstrate the practical potential of hybrid terrestrial–aquatic and aerial–ground robots in complex, mission-critical environments. Their design strategies, whether focused on morphological adaptation or multimodal control, highlight the engineering innovation driving multi-domain robotic mobility. Increased stability further improves task feasibility.

In contrast, HAAVs present substantial physical challenges stemming from the transition between two highly dissimilar media: air and water. In contrast to the more gradual land–water transitions or the stable base of air–ground operations, the water–air interface

introduces abrupt variations in drag, lift, buoyancy, and environmental resistance. These demands necessitate complex structural design, adaptive propulsion systems, and medium-aware control strategies. This section provides a comprehensive technical synthesis of HAAVs, building upon the more detailed survey by (Yao et al., 2023) and other relevant references (Tan and B. M. Chen, 2021; Yang et al., 2015). The discussion is structured to facilitate a comprehensive understanding of the topic, its applications, and the challenges it faces.

The utilization of HAAVs in the domain of environmental robotics has recently witnessed a notable surge in prominence, particularly in the context of applications such as marine monitoring, pollution sampling, coastal mapping, and underwater infrastructure inspection. Their capacity to access remote or hazardous water bodies and perform tasks across the surface, aerial, and submerged domains renders them suitable candidates for real-world deployment in disaster response or ecological surveying.

2.1.3 Hybrid Aerial-Aquatic Vehicles: A Taxonomy

The concept of such vehicles is not a novel one. Literary works, such as Jules Verne's "Master of the World", have long envisioned the concept of "flying submarines," while historical attempts can be traced back to Longobardi's 1918 patent. During the Cold War era, both the United States and the USSR investigated the feasibility of submersible aircraft, culminating in projects like CONVAIR. However, due to limitations in propulsion and control, this project ultimately failed to progress to operational deployment. This period marked a significant hiatus in the development of water–air hybrid technologies.

However, with the advent of the 21st century, particularly marked by DARPA's 2008 research proposal on submersible UAVs, there was a resurgence of interest in the development of HAAVs. This resurgence has led to a proliferation of innovative designs, encompassing a diverse spectrum of morphologies, propulsion mechanisms, and conceptual inspirations.

In order to adequately characterize the diversity of emerging designs in this rapidly evolving field, a taxonomy of HAAVs is proposed. This classification is organized along four complementary axes: The first is structural design, the second is propulsion type, the third is navigational domain, and the fourth is operational duration. This structured approach enables the elucidation of how diverse platforms address the dual-environment challenge, thereby unveiling

the trade-offs between energy efficiency, mechanical complexity, and mission requirements.

The operational duration dimension is further defined as follows, in line with the UAV taxonomy proposed by (Hassanalian and Abdelkefi, 2017):

- **Tactical HAAVs:** Designed for short-term, reactive missions where agility and deployment speed are prioritized (e.g., inspection, close-range sampling). These systems generally feature limited endurance but excel in responsiveness.
- **Strategic HAAVs:** In contrast, strategic HAAVs are designed for long-term, proactive missions where endurance and deployment reliability are prioritized (e.g., persistent monitoring, wide-area mapping).

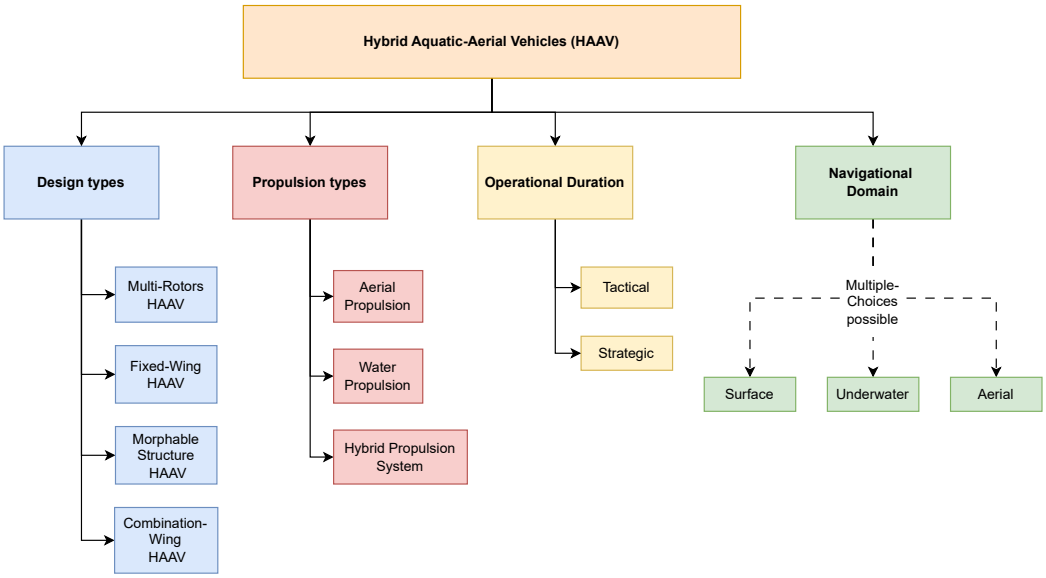


Figure 2.4: Taxonomic framework for HAAVs structured across four axes: design type, propulsion type, operational duration, and navigational domain. While the first three dimensions are exclusive, navigational domains may include multiple environments per platform. This classification enables structured analysis and comparison of HAAV architectures for mission-specific deployments.

Figure 2.4 offers a summary of key architectural and operational dimensions used to categorize existing HAAV systems. The diagram not only facilitates a structured reading of the literature but also guides the exploration of the design space for future systems. A selection of representative systems and their classification across the taxonomy is provided in Table 2.1.

Category	**Subcategory**	**Representative References**
Design Type	Multi-Rotor HAAV	(Pedroso et al., 2022; Alzu'bi et al., 2018)
	Fixed-Wing HAAV	(Weisler et al., 2018)
	Morphable Structure HAAV	(Tan and B. M. Chen, 2019; Li et al., 2022)
	Combination-Wing HAAV	(Li et al., 2022; Y. Chen et al., 2017)
Propulsion Type	Aerial Propulsion (Propeller / Jet)	(Alzu'bi et al., 2018; Pedroso et al., 2022; Jin, Bi, et al., 2024)
	Water Propulsion (Oscillatory / Fin)	(Y. Chen et al., 2017)
	Hybrid Propulsion System	(Li et al., 2022; Jin, Bi, et al., 2024)
Operational Duration	Tactical	(Li et al., 2022)
	Strategic	(Weisler et al., 2018)
Navigational Domain	Aerial/Surface	(Alzu'bi et al., 2018)
	Aerial/Underwater	(Li et al., 2022)
	Aerial/Surface/Underwater	(Bi et al., 2022; Jin, Bi, et al., 2024)

Table 2.1: Table of HAAV categories with representative publications across design type, propulsion type, operational duration, and navigational domains (see Figure 2.4)

In the following section, we examine the three primary HAAV design families: fixed-wing, multi-rotor, and morphable architectures, and analyze their respective advantages, limitations, and implementation strategies as well as examples of transition strategies.

2.2 Architectural Designs

The architecture of HAAVs is profoundly influenced by the challenge of operating across air and water, two environments with drastically different physical constraints. The extant landscape of HAAV designs can be categorized into four primary mechanical families: fixed-wing, multi-rotor, sweptback/morphable, and combination-wing or hybrid systems. Each category exhibits distinct trade-offs in terms of propulsion efficiency, transition feasibility, structural complexity, and mission adaptability.

2.2.1 Fixed-Wing Designs

These designs are derived from conventional aircraft and are designed for speed, endurance, and energy efficiency. Early designs, such as those from Auburn University and the US Naval Surface Warfare Center, adopted fighter-inspired wing-fuselage fusion structures, incorporating turbofans and pod-mounted propellers for both aerial and underwater locomotion. More recent prototypes, such as the EagleRay (Weisler et al., 2018) work, exemplify practical implementations with hollow wings, ballast tanks, and passive drainage systems to manage buoyancy during transitions. Despite their energy efficiency, these designs often require calm surfaces for hydroplaning takeoff and face maneuverability limitations underwater.

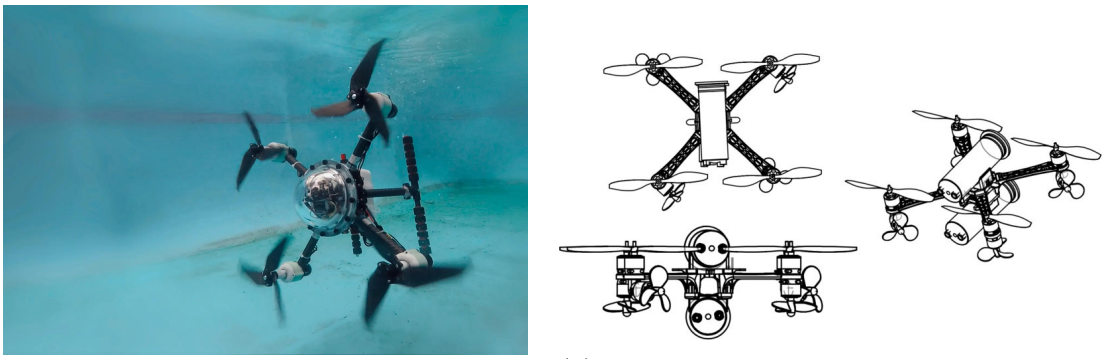
(a) EagleRay fixed-wing HAAV using fuselage-integrated ballast and drainage for transitions (Weisler et al., 2018).

(b) Nezha SeaDart fixed-wing platform with pneumatic buoyancy and passive hydroplaning takeoff (Jin, Zeng, et al., 2025).

Figure 2.5: Examples of fixed-wing HAAV configurations.

2.2.2 Multi-Rotor Designs

Multi-rotor HAAVs dominate current development due to their VTOL capabilities, low complexity, and stable control (Drews et al., 2014) first demonstrated a system with separate aerial and aquatic propellers. This dual propulsion design ensures performance in both media but increases system weight and reduces endurance. The Loon Copter (Alzu'bi et al., 2018) tackled this by using variable-speed aerial propellers for both domains. Later innovations include the Naviator's coaxial dual-rotor architecture for seamless transitions (Pedroso et al., 2022), and deformable or vector-thrust mechanisms (Tan and B. M. Chen, 2019; Li et al., 2022) to improve maneuverability. These designs are ideal for reactive missions but suffer from limited payload and short mission duration.



(a) TJ-FlyingFish with tail-sitter VTOL structure and integrated buoyancy (Liu et al., 2023).

(b) Naviator coaxial multi-rotor design enabling seamless transitions (Pedroso et al., 2022).

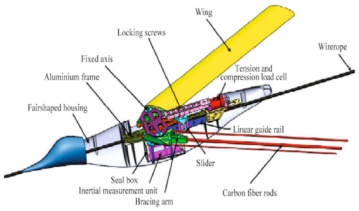
Figure 2.6: Examples of multi-rotor HAAV platforms.

2.2.3 Sweptback and Morphable Structures

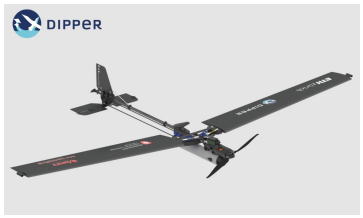
Inspired by the plunge-diving behavior of gannets, these systems emphasize medium-crossing efficiency. The Bionic Gannet and Aqua-MAV prototypes introduced morphable wings that reduce water-entry drag during vertical dives. Upon impact, the wings retract within milliseconds to minimize resistance, followed by buoyancy control and propulsion reconfiguration for underwater navigation (Liang et al., 2013; Rockenbauer et al., 2021). While plunge-diving enables rapid and efficient transitions, these systems require advanced mechanical design and structural robustness to withstand high-impact forces and complex kinematics.

2.2.4 Combination-Wing and Hybrid Designs

While these systems span distinct principles, plunge-diving, morphing wings, and multimodal flapping, their shared emphasis on dynamic structural adaptation for medium transitions justifies grouping them together. Hybrid systems are usually a combination of the features of fixed-wing and rotary-wing platforms, sometimes incorporating flapping-wing or tail-sitter mechanisms. Notable examples include the Nezha series, which integrate VTOL, gliding, and cruising capabilities with pneumatic buoyancy systems and morphable motor arms. These designs achieve a high degree of versatility but often necessitate compromises in terms of underwater control or structural integrity. RoboBee (Y. Chen et al., 2017; X. Chen et al., 2019), a micro-scale flapping-wing hybrid-air-vehicle, employs gas-assisted propulsion for water exit, though it remains in the early stages of development.



(a) Bionic Gannet I plunge-diving entry with morphing wings (Liang et al., 2013).



(b) Dipper with dual-propulsion and sweptback configuration (Rockenbauer et al., 2021).

(c) RoboBee with gas-assisted water exit and flapping-wing locomotion (Y. Chen et al., 2017).

Figure 2.7: Examples of sweptback-wing, morphable, and hybrid HAAV architectures.

Overall, the architectural diversity of HAAVs reflects ongoing trade-offs between control complexity, mission scope, and structural resilience, each design family optimized for specific transition strategies and operational contexts.

2.3 Transition Strategies and Mechanical Reconfiguration

Medium transitions represent a significant challenge in the context of HAAV operation due to the demands they place on mechanical and energetic resources. Each maneuver that involves crossing a body of water must address hydrodynamic forces, environmental variability, and system morphology to ensure safe and efficient transfer between the aerial and aquatic domains. In contemporary practice, three predominant strategies are observed in the design of HAAVs for water-entry and water-exit. Each of these strategies is associated with a particular architectural configuration and performance characteristics. These strategies function as paired entry/exit methods:

1. Skid Landing — Skid Takeoff
2. Soft Landing — Lift Takeoff
3. Plunge Landing — Launch Takeoff

2.3.1 Skid-Based Transitions

This method, typically employed by fixed-wing HAAV, emulates the behaviors of seaplanes and submarines. During the "skid landing" phase, the vehicle approaches at a shallow angle and makes contact with the water using its hull or floats, skimming along the surface to

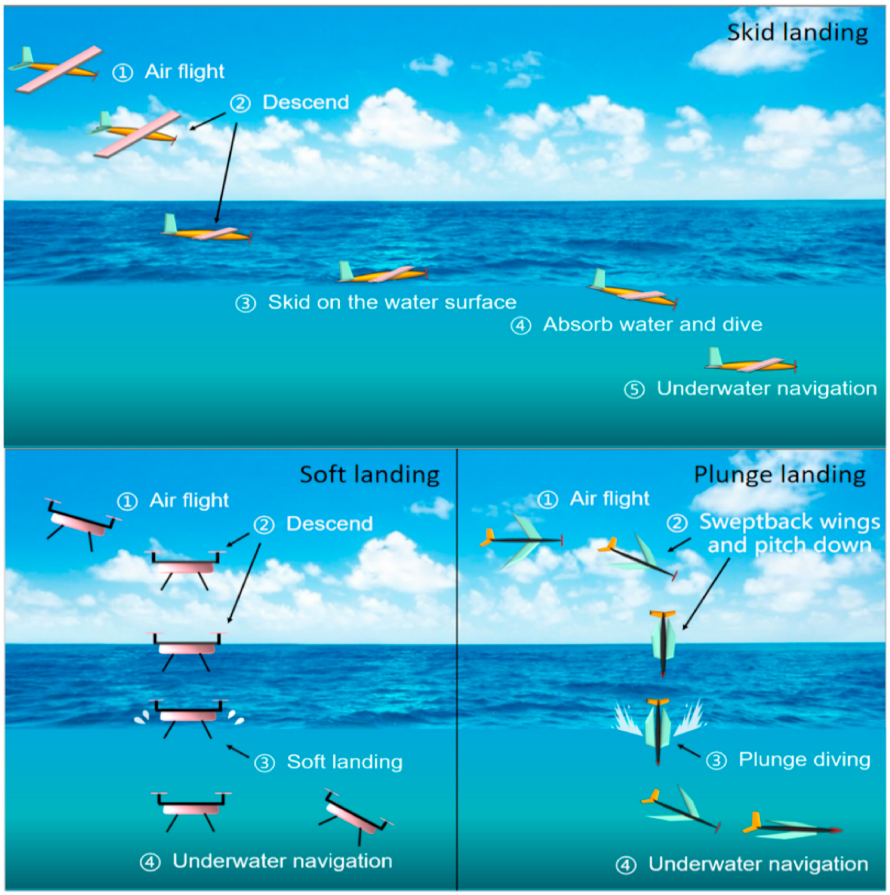


Figure 2.8: examples of entry motion and trajectory depending on hybrid design, Figure 20 of (Yao et al., 2023)

dissipate speed. Subsequent to deceleration, onboard ballast systems fill internal chambers with water, thereby increasing the vehicle's density and enabling submergence. The reverse process, termed "skid takeoff," involves the drainage of these chambers and the subsequent acceleration across the surface to generate aerodynamic lift.

This pair offers moderate control simplicity but suffers from low water-surface adaptability, being highly sensitive to waves and debris. The prolonged contact time during landing increases vulnerability to disturbance. Furthermore, the efficiency of this system is constrained by the time required for both deceleration and lift-off, resulting in a medium energy consumption due to the propulsion load necessary to overcome drag.

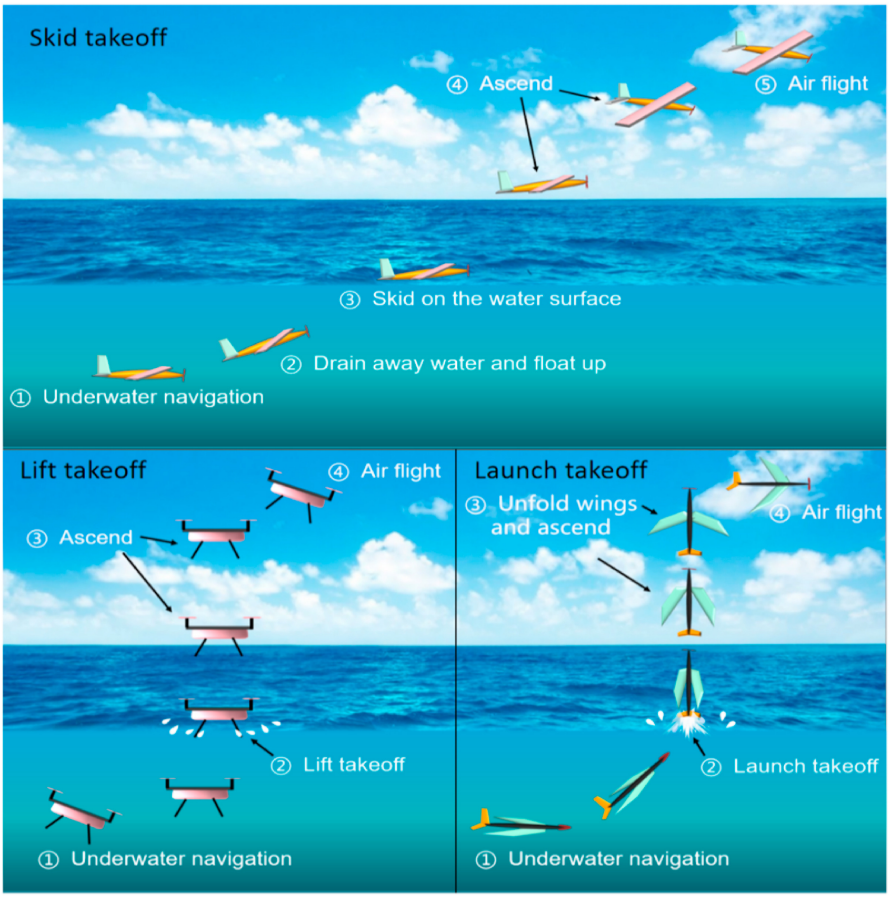


Figure 2.9: examples of exit motion and trajectory depending on hybrid design, Figure 21 of (Yao et al., 2023)

2.3.2 Soft Landing and Lift Takeoff

Adopted by multi-rotor and combination-wing HAAVs, this strategy utilizes rotor thrust modulation for controlled descent. During the soft landing phase, the robot delicately suspends or descends to the water surface and stabilizes vertically. Sinking is achieved either by gravity or by reducing rotor thrust further. The lift-takeoff phase, which initiates the process of re-emergence, commences with an increase in rotor speed until the HAAV disrupts the surface tension and ascends.

This transition type exhibits medium water-surface adaptability, as the vertical contact point is minimal and the system can be stabilized on calm water. However, this transition type exhibits a significant drawback in terms of energy consumption, as it necessitates the provision of full thrust without the assistance of inertial motion, which is a prerequisite for vertical takeoff. Despite its mechanical

impact being low, the method is generally favored for its ease of control and integration into VTOL-capable platforms.

2.3.3 Plunge Landing and Launch Takeoff

Primarily used by Sweptback-wing HAAVs and other bio-inspired designs as of today, this strategy mimics the plunge-diving behavior of gannets. During the *plunge landing* phase, the system adopts a steep descent trajectory, sweeps its wings back to minimize drag, and enters the water at high speed. To re-emerge, the *launch takeoff* uses mechanisms such as gas ejection, pressurized CO_2, or chemical blasts to achieve ballistic ejection from the water.

This technique offers the **highest water-surface adaptability** due to its minimal contact area and rapid transition, which reduces sensitivity to waves and debris. It also provides **high efficiency and low energy consumption** by converting gravitational potential into kinetic entry energy and using impulsive thrust for ejection. However, it generates the **largest mechanical impact forces** upon entry and demands highly durable structures and complex onboard propulsion or gas systems. As novel designs emerge, this category will encompass morphable structures and unconventional methods, such as jet-jump propulsion and other approaches to overcoming surface tension with a stronger, yet expeditious force.

2.3.4 Comparative Summary

These transition strategies are not interchangeable. They emerge from distinct design goals and entail trade-offs in terms of:

- **Water-Surface Adaptability:** Plunge landing > Soft landing > Skid landing
- **Efficiency:** Plunge landing > Soft landing > Skid landing
- **Energy Consumption:** Plunge landing (low) < Skid (medium) < Soft landing (high)
- **Mechanical Stress / Impact Force:** Plunge (high) > Skid (medium) > Soft (low)

While **plunge-landing and launch takeoff** offer superior performance, they remain technically challenging and are less common in operational platforms. **Soft-landing designs dominate current research** due to their stability and mechanical simplicity, despite their energy cost. The **skid-based strategy**, while viable in

calm, infrastructure-supported scenarios, is limited in hostile or unstructured environments.

Each method must be matched to both platform capability and mission requirements. For example, high-endurance environmental monitoring may tolerate the low efficiency of soft transitions, while rapid sampling or disaster response favors plunge-launch strategies with minimal exposure time on the surface. A diagram comparing these transitions is shown in Figs. 2.8 and 2.9.

2.3.5 Drive System Considerations

The transitions in question are underpinned by a set of propulsion schemes that can be categorized into two primary classifications: The first category is referred to as "water–air independent drives", which includes dual-propulsion systems utilized in Drews et al.'s Naviator. The second category is designated as "water–air integrated drives" and it encompasses shared propulsion systems with integrated speed and adaptation mechanisms, a notable example being the Loon Copter and the folding propellers developed by Li et al. The former ensures performance guarantees, albeit at the expense of weight and complexity, while the latter pushes the limits of actuator design, transitioning control logic, and power management.

These designs and strategies form the mechanical backbone of current HAAV systems. In the subsequent section, an exploration is conducted into the manner in which these designs and strategies influence the planning, perception, and coordination requirements for autonomous multi-robot operations in multi-medium environments.

2.4 Mechanical Challenges, Planning Implications, and Application Domains

HAAVs operate under highly divergent physical regimes, air and water, which impose substantial engineering and control challenges on both design and autonomy. This section synthesizes the mechanical difficulties, their impact on mission-level planning, and the wide array of potential applications that make such effort worthwhile.

2.4.1 Mechanical Design Challenges

Three primary mechanical hurdles constrain HAAV design and performance:

- **Water–Air Compatibility:** Reconciling aerodynamic lift and underwater thrust requires trade-offs in body shape, materials, and weight. Morphing propellers, morphable wings, and variable-density mechanisms (e.g., ballast tanks, inflatable chambers) are typical solutions.
- **Water Entry and Exit Stress:** Entry strategies like plunge-diving induce high impact forces, requiring damping systems (elastic or viscous). Soft-landing designs face instability on wavy surfaces and high power demands, while skid-based strategies require large surface runways, unsuitable for most field environments.
- **Environmental Adaptation:** Water droplets can impair sensors or motors. Passive (hydrophobic coatings) and active (shaking, vibration, spin) drainage methods are required to ensure successful aerial re-takeoff after submersion.

These constraints drive the evolution of increasingly complex HAAV morphologies and onboard systems, each with cascading effects on mission planning and robot coordination.

2.4.2 Mission Planning and Control Implications

From a mission planning perspective, the mechanical and environmental constraints translate into critical execution-level considerations:

- **Mode-Dependent Capabilities:** Sensors, actuators, and communication links are medium-specific (e.g., sonar vs. GPS, optical vs. inertial), communication might not be feasible with the same method in air and underwater (Radio frequency typically does not work well underwater). This could typically lead to cross-medium teamwork as a requirement for online missions.
- **Sequential Constraints:** Some tasks and requirements will be media-specific, with air and water bringing different risks for robots. Transitions must be explicitly scheduled with attention to timing, environmental and mechanical preconditions.
- **Energy and Structural Costs:** Each locomotion and transition mode has a distinct cost and wear impact on the robot's structure, which must be taken into account when planning a mission and various assignments.

- **Failure and Uncertainty Management:** Transition stages are still error-prone. Robust execution requires reactivity, fault-tolerant plans, and sometimes real-time coordination among aerial and aquatic agents to make sure the critical phases are observed.

These issues necessitate planning and execution frameworks capable of representing mode-switching, resource trade-offs, and contingency-aware strategies while creating a robust system due to the numerous possible failures, an objective that will be pursued in the following chapters.

2.4.3 Operational Relevance and Applications

The rationale for investing in HAAV technologies lies in their unique potential to streamline missions that would traditionally require heterogeneous robot teams or specialized agents. Their ability to act as "multi-role agents" across spatial domains unlocks a variety of civil and military applications:

- **Environmental Monitoring and Sampling:** HAAVs enable localized data collection in remote or sensitive areas such as coral reefs, polluted estuaries, and microplastic hotspots, offering dual-surface scanning and submerged measurement.
- **Disaster Response and Rescue:** Flood zones or tsunamis can render traditional infrastructure useless. HAAVs can fly to impacted zones, submerge to assess underwater damage, or deliver payloads without requiring a landing pad.
- **Infrastructure Inspection:** Bridges, offshore platforms, or ship hulls often span air and water zones. HAAVs can inspect both domains with a single vehicle.
- **Military and Surveillance Missions:** Tactical scenarios benefit from HAAVs' ability to silently approach from air, submerge to avoid detection, then resurface for extraction or transmission. Launch-based takeoffs further reduce exposure time.

The architectural and operational diversity of HAAVs reviewed in this section underscores the still maturing state and strategic significance of trans-media robotics. While cutting-edge prototypes demonstrate the potential of morphable and plunge-diving systems, the majority of real-world deployments currently favor multi-rotor and soft-landing designs due to their VTOL stability and integration

simplicity. These platforms offer a practical compromise between control complexity and mission versatility, particularly in applications such as ecological sampling, infrastructure inspection, and disaster response. However, the emerging capabilities of HAAVs introduce significant operational challenges, including the susceptibility to failure during transitions, the dependency on medium characteristics, and the variability in communication availability. Consequently, there is an imperative for novel models of coordination, robust planning methodologies, and mode-aware task allocation strategies. The subsequent chapter will address these challenges through the lens of multi-robot planning, proposing a framework tailored to the unique demands of trans-media operations.

2.4.4 Design-to-Planning/Execution Traceability.

The mechanical and operational constraints surveyed above (medium-specific sensing/actuation, explicit transition maneuvers, and mode-dependent costs) directly determine the structure of our planning and execution models. In Chapter 4 (AMA-Plan), we formalize MRTA with an explicit multimodal tuple and site partitioning, and we introduce a decomposition pipeline that leverages spatial structure and mode-aware costs for scalability (spatio-cost clustering, team/site allocation, and path sequencing). This mapping from physical constraints to problem abstractions is the rationale for AMA-Plan's hierarchical restructuring. *Concretely:* (i) transitions become explicit actions with preconditions/costs; (ii) sites become first-class units for abstraction, assignment and sequencing; and (iii) robot capabilities are conditioned by the current medium.[1]

In Chapter 5 (AMA-Exec), we address the execution-time consequences of these constraints: transition phases create synchronization points, uncertainty, and clustered failure modes. AMA-Exec employs a Simple Temporal Network (STN) controller to (i) monitor temporal consistency under drift, (ii) isolate failures locally when possible, and (iii) trigger graded recovery up to full reallocation. The supervision structure (temporal guards, synchronization events, and recovery budgets) is thus a direct response to transition-induced variability and inter-team coupling observed in trans-media missions.

2.4.5 Modeling Assumptions Made Explicit.

To guide the reader, we state the assumptions that carry forward:

[1] See the MRTA tuple and mode-aware action sets formalized at the start of AMA-Plan, and the decomposition pipeline that follows.

1. **Mode-aware feasibility:** an action's applicability and cost depend on the robot's current medium and on transition preconditions (e.g., surface points).
2. **Site-grained abstraction:** points of interest are partitioned into sites used for clustering, allocation, and sequencing.
3. **Transition as first-class actions:** crossing media incurs explicit duration, cost, and synchronization constraints.
4. **Communication asymmetry:** we assume degraded/heterogeneous communication across media; observation or relay roles may be required at the surface.
5. **Execution uncertainty concentrated at transitions:** delays and failures are more likely around entry/exit phases; supervision focuses on these windows.

These assumptions underpin the formalization and the modular solver in AMA-Plan (Chapter 4) and motivate the STN-based monitoring and recovery mechanisms in AMA-Exec (Chapter 5). The evaluation (Chapter 6) therefore emphasizes where multimodal constraints most affect planning time, makespan, and recovery behavior in homogeneous, heterogeneous, and trans-media scenarios.

Chapter 3

State of the Art in Multi-Robot Planning and Acting: A Structured Perspective

Contents

This chapter is divided in two main sections: the first section examines existant approaches in the multi-robot task allocation and planning literature, highlighting their limitations when applied to trans-media systems. Furthermore, it underscores the necessity for a novel framework that addresses these existing gaps. The second section reviews the current state of the art in executing symbolic plans, with the objective of identifying the challenges that arise during the transition from planning to execution. We adopt a structured comparative perspective, classifying MRTA approaches by problem formulation, algorithmic strategy, and temporal integration.

3.1 Planning and Task Allocation in Multi-Robot Systems

As robots become more autonomous and operate in teams, organizing and coordinating their actions to complete complex missions has become a central challenge in robotics. This challenge has two core components: planning and task allocation. Planning determines the sequence of actions that must be performed, while task allocation decides how these actions are distributed among the robots.

Planning and task allocation are interdependent processes. The way tasks are assigned influences which plans are feasible. Conversely, a plan's structure may impose constraints on which robots can perform which actions. Together, these processes form the backbone of decision-making in multi-robot systems.

This section explores the foundational models used to represent and solve these problems. First, we examine classical symbolic representations for single- and multi-agent planning. Then, we explore strategies for assigning tasks within robot teams. Finally, we identify shared assumptions that limit the applicability of these models to more dynamic and heterogeneous settings, such as trans-media robotic systems.

3.1.1 Planning Models for Robotic Systems

In robotics, many missions can be viewed as achieving one or more goals through a structured sequence of actions. Whether the task is navigating to a destination, collecting environmental samples, or manipulating an object, the fundamental question remains: How can an autonomous system determine the next actions required to fulfill its mission?

3.1.1.1 From Tasks to Planning Problems

In this context, planning involves organizing actions over time to achieve a desired goal, beginning with a known initial situation. The robot must consider its own capabilities, the rules that govern the execution of actions, and environmental constraints that could affect the outcome. This process is especially critical when robots operate in unstructured or dynamic environments, where predefined scripts are insufficient.

To support automated reasoning, these problems are converted into formal models that symbolically define the world, the agent's capabilities, and the desired goals. These models eliminate low-level control and sensor details, focusing instead on high-level representations of actions and world changes. A typical planning problem includes three core components (Figure 3.1).

- **An initial state**, describing the world at the beginning of the mission;
- **A goal condition**, specifying what the robot (or system) must achieve;

- **A set of actions**, each defined by the conditions/constraints under which it can be applied (preconditions) and how it changes the world (effects).

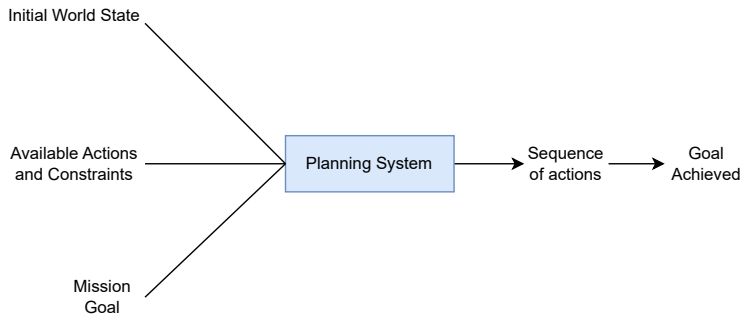


Figure 3.1: Simple representation of inputs and outputs of a planning system

The task of planning then becomes finding a sequence (or, in more complex cases, a partial order) of actions that transforms the initial state into one in which the goal is satisfied. This transformation must adhere to the logical and temporal constraints inherent in the action definitions.

Though this abstraction may seem simplistic, it provides a powerful basis for many robotic applications. By focusing on the symbolic effects of actions rather than their physical implementation, planners can simulate the consequences of decisions, generate alternative courses of action, and produce interpretable, executable plans. The following sections explore how this formalization is implemented in practice through widely used models, such as Stanford Research Institute Problem Solver (STRIPS) and Planning Domain Definition Language (PDDL).

3.1.1.2 Symbolic Models: STRIPS and PDDL

The STRIPS (Fikes and Nilsson, 1971) formalism, which was introduced in the early 1970s, is considered to be among the most influential models for representing planning problems. STRIPS employs a logical framework to model the environment and the robot's state, utilizing a set of discrete predicates to describe the environment's properties and the robot's state. STRIPS delineates actions by three elements: a set of preconditions that must be met for an action to be applicable, a list of positive effects, and a list of negative effects that describe how the state of the world changes after an action is executed.

For instance, an action such as `move(robot, A, B)` might precondition that the robot is at location A and that the path to B is clear. The effects would be the removal of the predicate `at(robot, A)` and the addition of `at(robot, B)`. Planning, therefore, entails the orchestration of successive actions, initiated from a designated initial state, with the objective of attaining the desired outcome.

As the field of planning matured, the need arose for a common, extensible language. To standardize and generalize this model, the PDDL was introduced in the late 1990s (McDermott et al., 1998). PDDL provides a syntax for expressing planning problems in a structured way and supports various extensions beyond classical STRIPS planning. A PDDL problem generally comprises two components:

- **The domain file**, which defines the types, predicates, and actions available to the planner;
- **The problem file**, which specifies the objects in the current instance, the initial state, and the goal condition.

PDDL has since become the de facto standard for symbolic planning research and has been widely adopted in robotic frameworks. Initially designed for classical symbolic planning, PDDL has evolved through several extensions to support a broader range of representations, including:

- **Durative actions**, allowing actions to span over time with defined start and end points (introduced in PDDL2.1) (Fox and Long, 2003);
- **Numeric fluents**, enabling updates and reasoning over quantities such as battery level, fuel, or resources (PDDL2.1) (Fox and Long, 2003);
- **Temporal and metric planning**, planning paradigms that use durative actions and numeric constructs to reason about deadlines, concurrency, and metric optimization (PDDL2.1) (Fox and Long, 2003);
- **Soft goals and Partial Satisfaction Planning (PSP)**, introduced in PDDL3, allowing planners to trade off between mandatory and preferred goals (Gerevini, Haslum, et al., 2009);
- **Dynamic exogenous events**, enabling the modeling of uncontrollable or environment-triggered changes during execution (PDDL+) (Fox and Long, 2006);

- **MAPL (Multi-Agent Planning Language)**, an extension of PDDL supporting agent-specific actions, internal/external state distinction, and joint plans in multi-agent domains (Brenner and Nebel, 2006);
- **HDDL (Hierarchical Domain Definition Language)**, a modern domain description language for hierarchical description and planning inspired by PDDL, used in IPC HTN tracks (Höller et al., 2020).

PDDL's flexibility makes it suitable for modeling a wide range of robotic applications, from high-level mission planning to reactive decision-making. In practice, PDDL-based systems have been used in autonomous underwater vehicles, planetary rovers, warehouse logistics, and, more recently, in multi-robot planners integrated into ROS ecosystems, such as ROSPlan and more recently PlanSys2. The symbolic abstraction allows robots to plan and replan based on their current knowledge, rather than following rigid behavior scripts.

Nevertheless, symbolic planning models also face significant challenges, particularly when applied to complex or dynamic domains. The formal structure provided by STRIPS and PDDL serves as the input to a variety of planning algorithms, ranging from heuristic search to constraint-based solvers. We now explore how these planning problems are solved computationally, and what trade-offs different solving strategies entail.

3.1.1.3 Planning Complexity and Solving Approaches

While symbolic models such as STRIPS and PDDL provide a robust framework for describing planning problems, solving them efficiently is computationally challenging. Classical planning, wherein actions are executed instantaneously and the world is characterized as discrete and fully observable, has been shown to be PSPACE-complete (Bylander, 1994). When temporal constraints, numeric fluents, or continuous processes are introduced (as in PDDL2.1 or PDDL+), the complexity increases substantially, sometimes reaching EXPSPACE or even undecidability depending on the expressiveness of the model (Rintanen, 2004; Fox and Long, 2006). This theoretical difficulty motivates the development of a wide variety of solving strategies, each with its own trade-offs in scalability, optimality, and domain compatibility.

The core of a planning system is the solver, an algorithm that takes a symbolic planning problem as input and searches for a valid

(and ideally optimal but not always) sequence of actions that transforms the initial state into a goal state. Solvers diverge in their internal representation of planning problems and the algorithms employed to navigate the search space. These solving paradigms are classified into distinct categories based on their representations and techniques. The most prominent categories are described below.

Heuristic State-Space Planners.
One of the most widely used families of planners relies on *heuristic forward search*, where the planner incrementally constructs a solution by selecting actions that appear to bring the system closer to the goal. Planners such as `Fast-Forward (FF)` (Hoffmann and Nebel, 2001), `Fast-Downward` (Helmert, 2006), and `LAMA` (Richter and Westphal, 2010) use domain-independent heuristics derived from relaxed versions of the planning problem, typically by ignoring delete effects, to estimate the cost-to-go from a state.

These heuristics (e.g., h_{FF}, h_{add}, h_{max}) guide the search process efficiently through vast state spaces. While these planners are scalable and well-suited to a wide range of domains, they often do not guarantee optimality and typically assume static and fully observable environments.

Plan-Space Planners.
In addition to state-based search, another family of approaches reasons directly in the space of plans. Plan-space planning, (often referred to as partial-order planning, POP), is a methodology that delineates a plan as a series of actions interconnected by causal constraints, together with ordering relations that are only specified when necessary. Rather than enumerating world states, these planners incrementally refine partially ordered plans by resolving flaws, e.g., open preconditions and causal threats (Penberthy and Weld, 1992; Ghallab and Laruelle, 1994; Younes and Simmons, 2003).

The key advantage of this paradigm is that it produces flexible plans: actions need not be totally ordered, and concurrency is preserved whenever possible. This makes plan-space methods appealing in robotics, where execution must accommodate uncertainty in action durations/success and where multiple agents may act asynchronously. Recent work such as FAPE (Bit-Monnot et al., 2020) illustrates this well: by combining plan-space refinement with temporal and resource reasoning, it supports continuous interaction between planning and execution. However, plan-space approaches rely on more complex heuristics than forward-search planners, which have limited their adoption in large-scale benchmarks and in the commu-

nity. Nonetheless, they remain particularly relevant when the structure of plans and their causal robustness are as important as raw scalability.

SAT and CSP-Based Planners.
A notable category of planners reformulates planning problems as Boolean satisfiability (SAT) or Constraint Satisfaction Problems (CSP). In the context of SAT-based planning, such as in `SATPlan` (Kautz and Selman, 1992), the plan is encoded as a propositional formula across discrete time steps, and a SAT solver searches for a satisfying assignment that corresponds to a valid action sequence. CSP-based approaches similarly encode planning variables, action choices, and temporal constraints into a constraint network solved using domain filtering or propagation techniques (Gazen and C. L. Knoblock, 1999; Do and Kambhampati, 2003).

These methods are capable of finding optimal or shortest-length plans (bounded planning), and they offer clear completeness guarantees. However, the scalability of these models can be constrained by the exponential growth in the number of variables and constraints associated with longer planning horizons or larger domains.

Temporal and Numeric Planners.
In numerous real-world robotic domains, actions are characterized by duration and constraints imposed by quantitative variables, including energy, distance, and deadlines. Temporal and numeric planners explicitly model these aspects, typically using extensions of PDDL (e.g., PDDL2.1). Planners such as `POPF` (Coles et al., 2010) and `OPTIC` (Benton et al., 2012) extend classical planning frameworks to support temporal synchronization, continuous effects, and numeric reasoning. These systems often combine heuristic forward search with constraint-based techniques, such as temporal constraint propagation or linear programming, to enforce consistency over time and resources.

While powerful, these planners encounter difficulties with scalability and completeness when confronted with large or temporally constrained problems. Nevertheless, these systems have gained widespread adoption in domains requiring precise temporal control, such as mission planning, multi-robot coordination, and task scheduling.

Hierarchical and Task Network Planners.
Hierarchical Task Network (HTN) planners adopt a decomposition-based approach to planning, wherein high-level tasks are system-

atically broken down into more concrete sub-tasks until executable primitive actions are obtained. In contrast to classical planners, which compute flat sequences of actions, HTN planners utilize user-defined methods to guide this decomposition process. Systems such as *SHOP2* (Nau et al., 2003) and more recent formalizations like *HDDL* (Höller et al., 2020) embody this approach.

The alignment of HTN planning with human reasoning facilitates modular plan construction, often resulting in smaller search spaces and semantically richer plans. However, this approach necessitates the development of intricate domain models, incorporating decomposition hierarchies, a process that can be arduous and restricts the generalizability across different domains. Despite this, HTN planning remains highly effective in applications such as robotics task coordination, service robotics, and planning with structured procedural knowledge (Erol et al., 1994).

The diversity of planning models and solvers reflects the diversity of robotic environments and tasks. Certain applications necessitate meticulously designed, universally coherent plans, while others demand swift, local decisions that require minimal computational effort. In all cases, the formal structure of symbolic planning provides a foundation for scalable and interpretable decision-making, especially when extended to support multi-robot coordination, which will be explored in the next sections.

3.1.1.4 Toward Multi-Robot Planning

The models and solvers presented thus far provide the conceptual and algorithmic foundation for automated planning in robotics. These tools empower agents to reason about action sequences, temporal constraints, and goal satisfaction. However, they were originally designed for single-agent systems, where decision-making is centralized and the scope of actions is confined to one robot's capabilities and perspective.

When planning must be extended to multi-robot systems, new challenges arise. Plans must now encompass inter-agent coordination, resource contention, spatial distribution, and communication constraints. Furthermore, planning may occur under partial observability or be distributed across agents.

One strategy for addressing these challenges is to encode allocation constraints directly into symbolic planning models. For instance, a domain might include fluents such as `(assigned ?r ?t)` or action preconditions like `(can-sample ?r)`. In this way,

Planner Type	**Model**	**Core Technique**	**Strengths**	**Limitations**
Plan-Space / Partial-Order (e.g., UCPOP, VHPOP, IxTeT, FAPE)	Partial-order plans with causal links	Refinement of partially ordered plans by resolving open conditions and threats	Produces flexible, concurrent plans; preserves causal structure; tight integration with execution possible	More complex heuristics; weaker scalability on large benchmarks
Heuristic (e.g., FF, FD, LAMA)	STRIPS / PDDL	Forward search with domain-independent heuristics (e.g., h_{FF}, h_{add}, h_{max})	Scalable and efficient; widely used in planning competitions	May miss optimal plans; assumes static and fully observable environments
SAT-based (e.g., SatPlan)	Boolean encoding (time steps)	Propositional logic with SAT/SMT solving	Finds short or optimal plans; well-suited for logic-heavy problems	Poor scalability; encoding complexity increases rapidly
CSP / LP-based (e.g., CP-SAT, LP-RPG)	Constraint-based models	Constraint propagation and linear/integer programming	Handles numeric and temporal constraints well; precise scheduling	Memory-intensive; sensitive to model encoding and size
Temporal Numeric (e.g., OPTIC, POPF, TFD)	PDDL with durative actions and numeric fluents	Temporal constraint reasoning and heuristic search	Supports deadlines, time windows, and continuous resource management	Slower execution; incomplete in complex or concurrent domains
Hierarchical (e.g., SHOP2, HDDL/HTNs)	Hierarchical Task Networks	Recursive task decomposition via methods	Matches human reasoning; reduces search space with structure	Requires expert-crafted domain models; low generalization

Table 3.1: Comparison of symbolic planning techniques used in the planning community.

symbolic planners such as POPF or OPTIC can implicitly perform task allocation by selecting grounded actions that respect robot capabilities, cost structures, and temporal constraints. The allocation is not optimized explicitly, but emerges from symbolic reasoning guided by metric fluents and search heuristics.

This approach effectively blends planning and allocation when allocation constraints are well-modeled, and forms the basis for hybrid architectures that integrate symbolic planning with MRTA. The next subsection expands on these ideas by examining how planning representations and algorithmic strategies have been adapted to handle task allocation across multi-robot teams.

3.1.2 Task Allocation in Multi-Robot Planning

To systematically analyze how tasks are distributed across multi-robot teams, several taxonomies have been proposed to classify MRTA problems based on their structural properties. Among the most influential is the three-dimensional framework by (Gerkey and Matarić, 2004), which categorizes problems according to robot capabilities, task requirements, and assignment timing. This classification not only defines problem space complexity but also guides the choice of coordination and planning strategies.

Beyond classification, MRTA problems can be represented using a variety of formal models, from optimization formulations to symbolic planning languages. These representations shape the capabilities and limitations of MRTA solvers, particularly when applied to complex domains such as dynamic environments or multi-modal robot platforms.

3.1.2.1 MRTA Taxonomy and Representations

As robotic systems shift from autonomous agents to collaborative multi-robot teams, the *MRTA* problem becomes a central concern. MRTA refers to the process of deciding *which robot(s)* should execute *which task(s)*, a decision that directly impacts coordination, efficiency, and mission success in team-based operations.

Foundational MRTA Taxonomy.

A foundational taxonomy was introduced by (Gerkey and Matarić, 2004), who classified MRTA problems along three dimensions:

- **ST vs. MT (Single-task vs. Multi-task robots):** Whether robots can handle only one task at a time or multiple concurrently.

- **SR vs. MR (Single-robot vs. Multi-robot tasks):** Whether tasks require a single robot or collaborative execution.
- **IA vs. TA (Instantaneous vs. Time-extended assignment):** Whether task allocation is performed once at the beginning or re-evaluated over time.

These axes define eight canonical MRTA classes (e.g., ST-SR-IA, MT-MR-TA), each representing different levels of coordination complexity and domain assumptions. For example, ST-MR-TA reflects tightly coordinated team behaviors (e.g., cooperative manipulation), while MT-SR-IA reflects loosely coupled planning (e.g., multi-drone surveillance). A summary of example cases is presented in Table 3.2.

Robot Type	**Task Type**	**Assignment Type**	**Example Application**
ST	SR	IA	Pre-assigned parcel delivery by single robots
ST	SR	TA	Dynamic patrolling with task insertion
ST	MR	IA	Cooperative object transport with fixed robot teams
ST	MR	TA	Adaptive infrastructure inspection with dynamic reallocation
MT	SR	IA	Multi-tasking drones for observation missions
MT	SR	TA	Long-term service robots handling cleaning and delivery
MT	MR	IA	Static team assignment for hybrid robot cooperative sampling
MT	MR	TA	Adaptive cross-modal teaming for complex tasks

Table 3.2: Examples of MRTA problem classes based on Gerkey and Mataric's taxonomy.

Understanding the MRTA class helps guide algorithm design: ST-MR problems often require synchronization protocols, while MT-SR scenarios allow more reactive, distributed solutions. These distinctions will be important as we explore solution paradigms in the next section.

Extended Taxonomies and Problem Dimensions.

Later works expanded this taxonomy to include additional dimensions:

- **Task structure:** loosely vs. tightly coupled tasks;
- **Task arrival model:** online vs. offline;

- **Inter-task relationships:** precedence, deadlines, dependencies;
- **Environmental and system assumptions:** observability, communication, dynamics.

(Korsah et al., 2013) emphasized these extensions in a comprehensive taxonomy. (Khamis et al., 2015) highlighted the need for hybrid solutions in real deployments, where assumptions like full observability and deterministic outcomes no longer hold.

Together, these perspectives offer a nuanced lens for analyzing MRTA complexity and selecting appropriate solving strategies.

Formal Problem Representations.

A general MRTA problem can be represented as a tuple:

$$\mathcal{M} = (R, T, \mathcal{U}, \mathcal{C}, \mathcal{A})$$

where:

- $R = \{r_1, ..., r_n\}$: set of robots,
- $T = \{t_1, ..., t_m\}$: set of tasks,
- $\mathcal{U}$: utility or cost matrix (e.g., distance, time),
- $\mathcal{C}$: constraints (e.g., deadlines, capabilities),
- $\mathcal{A}$: allocation function mapping tasks to robots.

Multiple modeling formalisms exist:

- **Cost-based models:** classical assignment/matching problems;
- **Graph-based models:** encoding compatibilities or dependencies;
- **Symbolic planning models:** e.g., PDDL-based domains;
- **Logic-based models:** e.g., MAPL or epistemic logic planning.

In symbolic planners, MRTA can be encoded using domain-independent constructs. Task assignment may be represented through fluents such as `(assigned ?r ?t)` or through preconditions tied to robot roles or capabilities (e.g., `(can-perform ?r ?task-type)`). The planner implicitly determines which robot executes which task based on initial facts and cost structures, often guided by metric fluents or heuristic estimates.

Alternatively, operational systems may model MRTA explicitly using behavior trees, finite state machines, or auction protocols that assign tasks at runtime based on local estimates or bids.

Centralized vs. Decentralized Perspectives.
Planning architectures differ depending on centralization:

- **Centralized approaches:** assume global knowledge, enabling optimal plans but suffering from scalability and robustness limitations;
- **Decentralized systems:** rely on local views and negotiation protocols, offering higher scalability and resilience at the cost of optimality.

These architectural choices influence algorithm design and are often aligned with the MRTA class, centralized solutions tend to suit tightly coupled MR tasks, while decentralized ones are better for loosely coupled, dynamic domains.

The next subsection builds on this foundation to examine the main families of MRTA solution strategies, from economic frameworks to mathematical optimization and heuristic-based methods.

3.1.2.2 Algorithmic Strategies for Multi-Robot Task Allocation

A variety of algorithmic strategies have been proposed to solve the MRTA problem, each grounded in different assumptions about task structure, robot communication, and environmental dynamics. This subsection surveys the major families of MRTA solvers, providing both conceptual insights and links to use cases that will be revisited in later sections.

Market-Based Methods.
Inspired by economic theory, market-based methods treat task assignment as a decentralized auction process, where robots bid on tasks based on local utility or cost estimations. The Contract Net Protocol (CNP) and variants have been widely applied in dynamic, distributed settings where central coordination is impractical (Dias et al., 2006). Systems like MURDOCH exemplify real-time bidding frameworks used in multi-agent robotics.

These methods are robust to partial observability and agent failures, but rely on:

- Predefined and accurate utility/cost models,
- Limited task interdependence,
- Reliable communication for task announcements and bid responses.

They often fail to handle task dependencies, team-level coordination, or domain-specific feasibility constraints and modeling.

Typical use cases: loosely coupled tasks, dynamic arrivals, unreliable communication.
MRTA fit: MT-SR-TA, MT-MR-IA

Optimization-Based Approaches.

Formulations such as Mixed-Integer Programming (MIP), Integer Linear Programming (ILP), and Constraint Satisfaction Problems (CSP) offer centralized, globally optimal task assignments under strict constraints (Zheng and Koenig, 2009; Khamis et al., 2015). These approaches are particularly powerful when task precedence, synchronization windows, or joint execution are required. These formulations are effective in structured environments and can incorporate constraints such as:

- Task precedence,
- Resource limitations,
- Team composition,
- Synchronization windows

However, their complexity grows rapidly to NP-hard with the number of robots and tasks, and they are often unsuitable for time-critical or dynamic scenarios. Several frameworks (e.g., OR-tools, OptaPlanner) allow declarative constraint programming for MRTA that integrate these constraint-based optimization with symbolic reasoning, and formal solvers like OPTIC (Benton et al., 2012) enable temporal and numeric constraint handling within symbolic planning, bridging expressiveness and control. While such planners do not treat allocation as an explicit optimization variable, they can produce effective task distributions through action cost modeling, especially in domains where roles and capabilities are encoded as preconditions.

These symbolic planners thus offer an implicit resolution of MRTA, provided that task constraints are accurately modeled through action schemas, robot-specific predicates, and metric fluents.

Typical use cases: centralized mission planning with known constraints and low dynamics.
MRTA fit: ST-SR-TA, ST-MR-TA

Heuristic and Greedy Approaches.

For scalable, real-time systems, many MRTA frameworks rely on heuristics or greedy policies, such as assigning the closest available

robot to each task, using task priority queues, or evaluating custom scoring functions that balance cost, distance, availability, or urgency (Khamis et al., 2015). These methods are lightweight, computationally efficient, and particularly suited to systems where fast adaptation to environmental changes or new task arrivals is critical.

Unlike optimization-based approaches, which aim for global optimality, heuristic methods prioritize responsiveness and scalability. In many practical deployments, especially those involving streaming tasks or uncertain environments, global reallocation is either infeasible or unnecessary. Instead, local decisions guided by domain-aware heuristics offer sufficient performance at a fraction of the computational cost.

Moreover, heuristic reasoning plays a central role in symbolic planning as well: most planners, including Fast Downward, POPF, and OPTIC, use domain-independent or domain-specific heuristics (e.g., relaxed plan estimates, cost-to-go functions) to guide the search for action sequences. In these contexts, task allocation is often achieved not by explicit matching but by heuristic-driven selection of grounded actions under encoded preconditions and fluents.

In this thesis, heuristic strategies are particularly important as they are embedded at multiple levels of the planning process: from problem structuring (e.g., clustering sites, filtering goals) to guiding symbolic resolution and execution-time decisions.

Typical use cases: fast response, streaming tasks, loosely coupled environments, execution under uncertainty. *MRTA fit:* MT-SR-IA, MT-SR-TA

Hybrid and Hierarchical Methods.

Hybrid and hierarchical MRTA strategies integrate symbolic planning, constraint-based optimization, and reactive execution to overcome the limitations of isolated approaches. Instead of relying on a single algorithmic paradigm, they orchestrate multiple reasoning layers to balance global structure and local flexibility.

In such systems, symbolic planners (e.g., POPF, OPTIC) are used to structure missions and goals using domain models that embed robot roles, task dependencies, or team requirements. These planners solve MRTA implicitly by grounding actions based on cost heuristics and logical constraints. Where symbolic reasoning lacks expressiveness or fine-grained control, optimization modules, such as ILP or CSP solvers mentioned earlier, are introduced to enforce tighter constraints over allocation, synchronization, or resource usage. Finally, reactive control layers (e.g., behavior trees, finite state machines) enable fast adaptation during execution.

This layered integration is increasingly common in robotics. (Car-

reno et al., 2020) propose a framework that decomposes mission-level goals and solves each subproblem using symbolic temporal planning. (S. Kim et al., 2016) embed PDDL-based planning into an optimization-driven MRTA architecture, while (Kaelbling and Lozano-Perez, 2011) combine symbolic planning and reactive execution in an end-to-end hybrid control system.

Such architectures are particularly suited to heterogeneous or dynamic domains, where expressiveness, responsiveness, and mission feasibility must all be addressed. However, they often demand careful interface design between planning layers and may incur consistency or timing issues if integration is not properly managed.

Typical use cases: tasks with temporal or role coupling, multi-robot coordination, transitions between symbolic and reactive control, high complexity of domain less accessible to generalisable solvers.
MRTA fit: MT-MR-TA, ST-MR-TA

Architectural Perspectives.

MRTA Class	**Typical Features**	**Best-Fit Algorithmic Approaches**
ST-SR-IA	Static, single-agent tasks	Heuristic, Greedy
ST-SR-TA	Sequential tasks over time	Optimization, Heuristic
MT-SR-IA	Parallel independent tasks per robot	Market-Based, Heuristic
MT-SR-TA	Time-extended parallel tasking	Hybrid with local reallocation
ST-MR-IA	Static collaborative tasks	Centralized Optimization
ST-MR-TA	Dynamic team-based tasks	Optimization, Hybrid
MT-MR-IA	Static multi-task teams	Market-based + symbolic planning
MT-MR-TA	Dynamic, interdependent multi-team tasks	Hybrid, Hierarchical

Table 3.3: Mapping between MRTA problem classes and suitable algorithmic strategies.

While these algorithmic strategies offer powerful tools for solving MRTA problems, they often rely on simplifying assumptions, such as static robot capabilities, full observability, or reliable communication, that do not hold in many real-world missions. In the next subsection, we examine these shared assumptions, their limitations, and how they influence the applicability of MRTA models to complex domains, such as multi-medium and trans-media robotic missions.

3.1.3 Limits of Current MRTA Models: Assumptions and Structural Gaps

While a wide range of strategies have been developed to address MRTA problems, ranging from market-based and optimization-based methods to heuristic and hybrid frameworks, these approaches often rely on simplifying assumptions that are increasingly misaligned with the requirements of real-world multi-robot systems, especially those operating in heterogeneous or trans-media environments.

3.1.3.1 Common Assumptions Across MRTA Models

Across most existing MRTA formulations and planning strategies, several foundational assumptions are frequently observed:

- **Static capabilities:** Robots are often assumed to have predefined roles and capabilities that remain fixed during execution.
- **Well-defined tasks:** Tasks are fully specified, atomic, and decomposable; their feasibility is assumed to be independent of dynamic context.
- **Reliable communication:** Communication is typically modeled as reliable and low-latency, enabling coordination and auction-based allocation.
- **Complete observability:** The environment and robot state are assumed to be fully known or rapidly updated.
- **Stable environment:** Environments are either static or evolve in predictable ways, allowing replanning from scratch when changes occur.

These modeling simplifications enable computational tractability, but can lead to brittle behaviors when deployed in uncertain or cross-modal environments.

3.1.3.2 Limitations of Assumptions in Trans-Media Missions

In complex, multi-medium scenarios, such as those involving hybrid aerial-aquatic vehicles (HAAVs), the foundational assumptions of many MRTA models begin to show their limitations. These scenarios introduce new dimensions of variability and interdependence that are difficult to capture using static or monolithic formulations:

- **Action feasibility is mode-dependent:** A robot's ability to perform actions depends on its current operational medium (e.g., air or water), which invalidates static capability assumptions and requires mode-aware modeling.
- **Transitions introduce uncertainty:** Medium changes are non-instantaneous and may be subject to feasibility constraints based on environmental factors such as visibility, turbulence, or depth.
- **Task success requires cross-medium coordination:** Many operations require synchronization between heterogeneous robots operating in different physical layers (e.g., underwater sampling followed by aerial data relay), posing challenges for task decomposition and timing.
- **Execution may be decentralized and partially observable:** Robots often operate asynchronously, with limited or delayed communication and incomplete knowledge of the system state, making centralized or fully precomputed plans fragile and inefficient.

These characteristics highlight the need for new abstractions that extend beyond classical MRTA taxonomies and solver paradigms. In particular, action feasibility must be redefined not only in terms of geometric or temporal reachability, but also with respect to mode compatibility, transition readiness, and team-based synchronization across heterogeneous agents.on synchronization, and team-based execution structure.

3.1.3.3 Toward Structured Problem Decomposition

As highlighted in recent literature (Carreno et al., 2020; S. Kim et al., 2016; Kaelbling and Lozano-Perez, 2011), emerging MRTA systems increasingly blend symbolic reasoning, optimization constraints, and reactive control to balance mission structure with execution adaptability. However, few of these systems offer explicit mechanisms to factor in structural discontinuities introduced by environmental dynamics (e.g., medium transitions, capability evolution, or multi-stage goals), rather than modeling them solely as robot capability constraints.

This motivates the need for a modular approach to MRTA that explicitly models:

1. Multi-stage missions composed of mode-specific subproblems,

2. Capability transitions and their preconditions,
3. Inter-agent synchronization in heterogeneous teams,
4. And domain-specific or learned heuristics that adapt allocation and planning decisions to evolving mission context.

In response to these gaps, the next chapter introduces the **AMA-plan framework**, a generalizable model for decomposing complex multi-robot missions into structured subproblems, enabling scalable and adaptive planning under complex physical and temporal constraints. However, scalable planning alone is insufficient: once a plan is generated, its successful enactment depends on an execution system capable of handling temporal dynamics, failures, and unforeseen events in the real world.

3.2 Introduction and State of the Art in Execution for Robotic Systems

The problem of execution in robotics has long been recognized as the dual of planning: rather than generating a plan from a symbolic model, an execution system must reliably enact a given plan in the physical world. Early work like Reactive Action Package (RAP) (Firby, 1987) introduced reactive execution, focusing on adapting to the current state rather than projecting only future plans. IxTeT (Ghallab and Laruelle, 1994) initiated the incorporation of temporal reasoning at the execution level, while Remote Agent (Muscettola et al., 1998) demonstrated autonomous temporal supervision and failure recovery in space missions. Other systems, including UM-PRS (Lee et al., 1994) (an extension of Procedural Reasoning System (PRS) (Ingrand, Georgeff, et al., 1992)), CIRCA (Musliner and Durfee, 1993), and Propice-Plan (Despouys and Ingrand, 1999), have expanded upon this vision by incorporating continuous supervision and plan repair. Collectively, these efforts established that execution involves more than action dispatch alone; it requires dynamic supervision of temporal, causal, and resource constraints.

Subsequent research has explored different paradigms to address the link between planning and acting in robotics. Dispatcher-based frameworks such as ROSPlan (Cashmore et al., 2015) and its ROS2 successor PlanSys2 (Martı´n, Clavero, et al., 2021) provide practical pipelines from PDDL to ROS actions. Other systems such as the Refinement Acting Engine (RAE) (Patra, Ghallab, et al., 2019; Patra, Mason, et al., 2021) and OMPAS (Turi and Bit-Monnot, 2022)

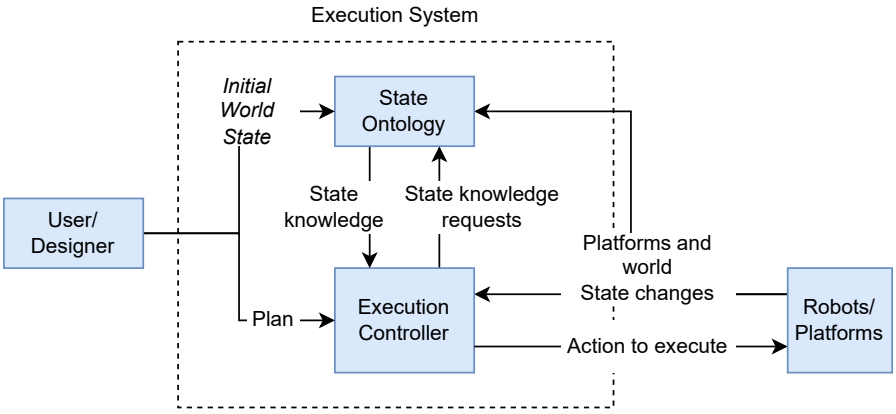


Figure 3.2: Simplistic representation of an execution system: Plans are translated into robot actions to execute and continuously supervised under temporal and resource constraints.

introduce hierarchical acting models that support deliberative reasoning, reactive adaptation, and skill-level abstraction. Timeline- and constraint-based execution (e.g., PLATINUm (Umbrico, Cesta, Cialdea Mayer, et al., 2017)) emphasizes temporal flexibility and resource coordination. Skill-grounded acting (Lesire, Doose, et al., 2020; Ingrand, 2023; Pelletier et al., 2025) explicitly models robot skills and their controllers, clarifying when actions are applicable and safe in the continuous domain. Finally, BTs (Colledanchise and Ögren, 2018), initially popularized in the game industry and now adopted in robotics (Jeon et al., 2022; Ingrand, 2024), offer a modular execution control formalism for sequencing, monitoring, retries, and fallbacks, and are used in conjunction with other systems, including at the core of ROS2 PlanSys2.

In modern multi-robot contexts, execution must therefore ensure temporal coherence under drift, enforce inter-agent dependencies, arbitrate shared resources, and support recovery from failures. While planning systems (cf. Section 3.1) provide symbolic problem-solving, the transition to acting still requires a dedicated execution layer. In this Section, we introduce the planning-to-acting problem, present PlanSys2 as a baseline, and outline the specific challenges that motivate our AMA-Exec system (Chapter 5),

3.2.1 From symbolic planning to acting systems

The execution problem can be understood as the dual of planning: rather than generating a plan from a symbolic model, the system

must consume a symbolic plan and guarantee its reliable enactment in the physical world (Figure 3.2). This process necessitates the maintenance of consistency between abstract representations and robot platforms, while accommodating delays, uncertainties, and resource conflicts.

A generic execution system integrates three essential elements:

- A plan to execute, i.e., a symbolic description of actions with temporal and causal dependencies;
- A dynamic world model, reflecting the current state of robots and environment, updated continuously during execution;
- An execution controller, which dispatches actions to robots, supervises their progress, and reacts to deviations.

This abstraction mirrors the earlier definition of planning problems (cf. Figure 3.1), but in reverse: instead of searching for an action sequence to satisfy a goal, execution frameworks must ensure that the prescribed sequence remains feasible in reality. This highlights three fundamental challenges in bridging planning and acting: the *symbolic-to-skill gap*, *temporal supervision*, and *resource concurrency*, discussed next.

3.2.1.1 Symbolic-to-skill gap

In robotics, plans derived from symbolic models (e.g., PDDL domains (Ghallab, C. Knoblock, et al., 1998)) specify actions via preconditions and effects. However, execution requires grounding these actions in concrete skills (navigation, sensing, manipulation) with controllers, monitors, and termination conditions. This *symbolic–skill interface* is well documented (Ghallab, Nau, et al., 2016): even when symbolic preconditions hold, continuous dynamics or sensing uncertainty can invalidate assumptions. BTs are often used to *organize* skill invocation and monitoring at runtime; they do not, by themselves, define the skills. Skill-grounded approaches (Lesire and Pommereau, 2018; Lesire, Doose, et al., 2020; Pelletier et al., 2025; Ingrand, 2023) make capabilities explicit (applicability envelopes, safety guards, recovery hooks), improving traceability between symbolic effects and low-level controllers. In summary, acting systems benefit from a clear symbol–skill contract plus feedback that supports adaptation under drift, a property that several existing frameworks already pursue in various forms.

3.2.1.2 Temporal supervision

Whether plans are produced by temporal *state-space* planners for PDDL (e.g., OPTIC (Benton et al., 2012), POPF (Coles et al., 2010), TFD (Röger et al., 2008)) or by *plan-space* (PSP) approaches with temporal annotations, execution in robotics must cope with variable durations, asynchronous progress, and synchronization drift. Without runtime temporal reasoning (Dechter et al., 1991; Stedl and Williams, 2005), small discrepancies can accumulate and jeopardize feasibility for multi-robot teams. Timeline and constraint-based execution (e.g., IxTeT-Exec (Lemai-Chenevier, 2004), PLATINUm (Umbrico, Cesta, Cialdea Mayer, et al., 2017)) maintain flexible temporal envelopes and check consistency online. Resource-aware extensions (Umbrico, Cesta, Mayer, et al., 2019) link time with shared resources among robots. In practice, temporal constraints should be treated as dynamic, continuously monitored, with incremental repair (delays, rescheduling) when needed. Many existing systems already implement parts of this capability; our emphasis is on making it explicit and central at execution time for multi-robot settings.

3.2.1.3 Concurrency & resources

Modern planning and acting frameworks frequently represent resources (capacities, mutual exclusion, shared tools/locations) and use them to guide search, verification, or scheduling (Carreno et al., 2020). In robotics execution, the practical challenge is *operational arbitration*: multiple robots contend for places or tools, communication bandwidth fluctuates, and charging docks become bottlenecks. If unmanaged, this yields deadlocks, priority inversions, or degraded performance. Safety-critical arbitration has long been studied (e.g., CIRCA (Musliner and Durfee, 1993); multi-agent planning (Ingrand, Chatilla, et al., 2001)); timeline-based acting (PLATINUm (Umbrico, Cesta, Cialdea Mayer, et al., 2017)) integrates resources with temporal flexibility, and hybrid symbolic-numeric models (Mayer et al., 2015; Umbrico, Cesta, Mayer, et al., 2019) express richer constraints. For acting systems, the key need is *runtime* scheduling, monitoring, and negotiation of scarce resources across distributed robots, complementing what is decided at planning time. Several existing systems address parts of this; here we state it as a first-class execution requirement.

3.2.1.4 Beyond dispatch

A baseline execution loop dispatches actions when preconditions hold and awaits completion. This works well in stable settings; handling drift, contention, and partial failures typically calls for additional mechanisms. ROSPlan (Cashmore et al., 2015) provides a dispatcher-centric pipeline tightly integrated with ROS (plan dispatch, knowledge base, action interfaces). In ROS2, PlanSys2 (Martı´n, Clavero, et al., 2021) extends this lineage and adds a BT-based executor to structure sequencing, monitoring, retries, and fallbacks. Hybrid acting engines (RAE (Patra, Ghallab, et al., 2019; Patra, Mason, et al., 2021), OMPAS (Turi and Bit-Monnot, 2022)) support online refinement and repair, while skill-grounded execution (Pelletier et al., 2025; Ingrand, 2023) ties symbolic decisions to adaptive controllers. The general trend, present across many systems, is to *combine* symbolic supervision, temporal reasoning, and resource awareness in a feedback loop rather than relying on dispatch alone.

3.2.1.5 Implications from AMA-Plan

AMA-Plan (Chapter 4) outputs execution-ready subplans embedded in a $\text{STN}_{\text{teams}}$, where symbolic, temporal, and resource relations are explicitly represented. AMA-Exec builds on this by enabling distributed multi-robot execution with runtime monitoring and localized recovery. Unlike prior approaches that primarily focused on centralized supervision or single-robot skill integration, AMA targets scalability and robustness in team-based missions where drift, failures, and resource contention are prevalent. In this way, AMA extends the bridges proposed by prior acting systems (Ghallab, Nau, et al., 2016; Umbrico, Cesta, Mayer, et al., 2019; Lesire, Doose, et al., 2020), scaling them to distributed, uncertainty-prone environments.

3.2.2 PlanSys2 as a Baseline for Execution

3.2.2.1 Architecture

PlanSys2 (Martı´n, Clavero, et al., 2021) is a ROS2-native framework that extends the lineage of ROSPlan (Cashmore et al., 2015) to the new ROS2 middleware. While ROSPlan pioneered the integration of symbolic planning with ROS action servers, PlanSys2 rebuilt this architecture around ROS2's topics, services, actions, real-time-friendly communication primitives, QoS policies and modular scalability. Its architecture (simplified in Figure 3.3) comprises a *Domain Expert* (managing domain models in PDDL), a *Problem Ex-*

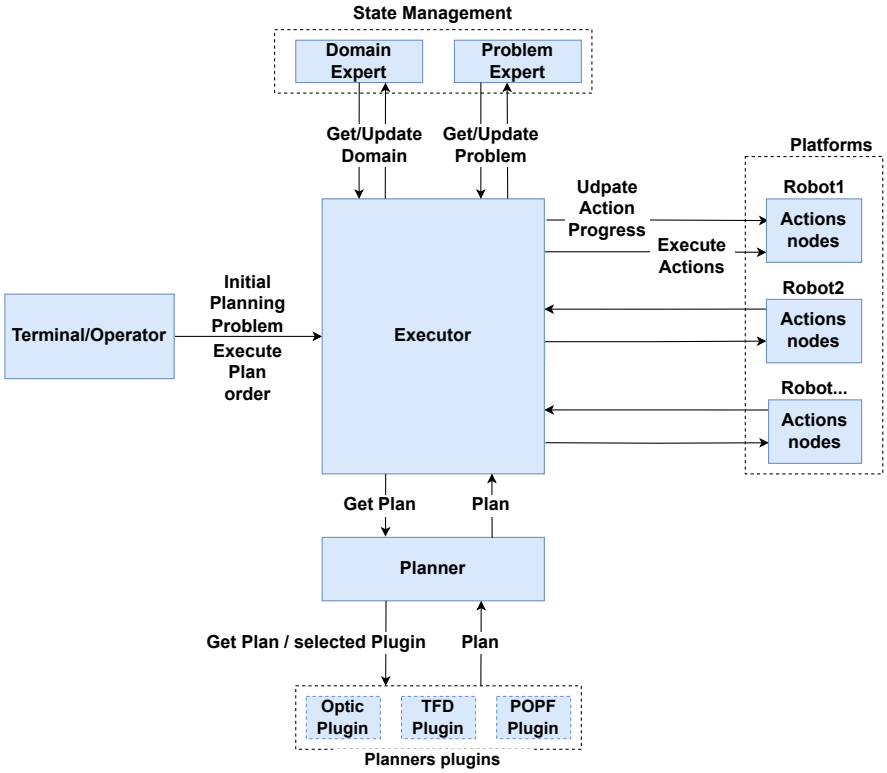


Figure 3.3: Simplified PlanSys2 pipeline. Symbolic plans are generated by external planners and executed by a centralized BT-based Executor.

pert (managing problem in PDDL and its objects, predicates, functions etc.), a planner node with several plugins (e.g., OPTIC, POPF, TFD), and a centralized *Executor*. The Executor dispatches actions through BTs (Martı´n, Morelli, et al., 2021) to action nodes for each robot, which encapsulate symbolic checks, control logic, and calls to robot action servers. This tight integration between planning, execution, and ROS2 infrastructure has popularized PlanSys2 adoption in the robotics community (also referenced in the AIPlan4EU project).

3.2.2.2 Strengths

PlanSys2 introduced several innovations compared to ROSPlan and other dispatcher-based systems:

- **ROS2-native integration.** Built directly on ROS2 communication primitives, PlanSys2 benefits from multi-robot support, deterministic QoS, and interoperability with the ROS2 ecosystem, making it a natural choice for modern robotic deployments.

- **BT-based execution.** By adopting Behavior Trees, PlanSys2 provides modular, hierarchical control structures with native support for action monitoring and reactivity. This improves transparency, debugging, and extensibility compared to finite-state-machine style dispatchers.
- **Planner modularity.** PlanSys2 provides clean APIs for connecting to multiple planners (OPTIC, POPF, TFD, etc.), allowing users to select the most appropriate solver depending on their domain.
- **Skill mapping.** Symbolic actions can be mapped directly to ROS2 actions or services, enabling seamless connection between high-level planning and low-level robot skills.
- **Community adoption.** Due to its clean design and integration in ROS2, PlanSys2 has been adopted in numerous academic and industrial projects, serving as the de facto reference framework for PDDL-based execution in ROS2.
- **Temporally informed BT synthesis.** Recent work has explored generating BTs from temporal networks (e.g., constructing BTs after a PDDL $\rightarrow$ STN transformation) (Zapf et al., 2024). This can strengthen time-aware checks within BT execution while preserving plan structure, partially offsetting native temporal limitations of plain BT dispatchers.

Together, these strengths make PlanSys2 not only a successor to ROSPlan, but a significant step forward in bringing symbolic planning and execution into practical robotic applications.

3.2.2.3 Automatic PDDL $\rightarrow$ BT synthesis in PlanSys2

A distinctive feature of PlanSys2 is the automatic synthesis of BTs from planner output. Starting from a PDDL plan, PlanSys2 builds a planning graph whose nodes are action instances annotated with their requirements and effects; it then transforms this graph into a BT that preserves causal precedences and enables parallel execution where safe. The synthesis introduces two key mechanisms *wait nodes* (to delay an action until a required effect produced elsewhere becomes available) and *action singletons* (so the same action instance, if referenced from multiple branches, executes once and returns SUCCESS thereafter).

Concretely, the algorithm extracts an execution graph from the PDDL plan, identifies execution flows and convergences, maps linear

segments to `Sequence` nodes, fan-outs/fan-ins to `Parallel` nodes, and inserts `Wait` nodes and singletons to satisfy inter-branch dependencies without duplicating execution. Each action node is expanded into an "action subtree" (e.g Figure 3.4) that checks `at_start`, `over_all`, and `at_end` conditions and applies effects at the right phase during runtime.

This approach also bring some advantages when considering implementation as `ready-to-use` planning-to-acting system:

- **Ease of use.** The automatic PDDL $\rightarrow$ BT pipeline removes a large amount of manual BT authoring while keeping the PDDL plan's causal structure visible and understandable.
- **Parallelism out of the box.** By analyzing the execution graph, the synthesized BT can tick independent actions in parallel, reducing idle time and improving makespan in multi-robot scenarios where robots acts in parallel.
- **ROS2 integration.** The BT executor plugs into PlanSys2's ROS2 action protocol and multi-robot execution support (action hubs, bidding/selection), making it practical to deploy on real robots. It also blends really well with other recent efforts to standardize navigation and control process in ROS2, such as NAV2 (Macenski et al., 2024)

A typical representation of a planning graph for a flow of actions can be found in Figure 3.5[1]. Furthermore, an example of one our team's PDDL plan from AMA-Plan transformed into a BT with PlanSys2 BT executor planning graph can be found in appendix in Figure A.1.

3.2.2.4 Understanding Behavior Trees in Robotic Control

Behavior Trees (BTs) are hierarchical, reactive control structures widely used in robotics and game AI (**Colledanchise2018**). Unlike traditional imperative programming (if-else, switch-case), BTs provide modular composition, runtime reactivity, and explicit failure handling—properties essential for complex multi-robot missions.

Core Structure and Node Types A BT is a directed tree evaluated ("ticked") at regular intervals. Each node returns `SUCCESS`, `FAILURE`, or `RUNNING`. Standard node types include:

[1]Figure 3.4 and 3.5 come from https://plansys2.github.io/ BT example

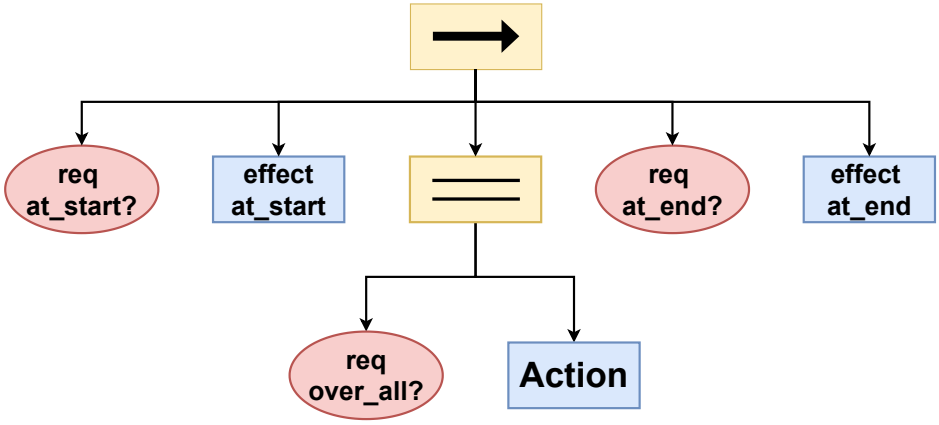


Figure 3.4: **Action subtree.** BT expansion that checks `at_start`, `over_all`, *at_end* requirements and applies effects at the correct phases with the PDDL action.

- **Sequence (→)**: Executes children left-to-right until one fails
- **Fallback (?)**: Tries children until one succeeds (provides recovery)
- **Parallel**: Executes multiple children concurrently
- **Condition (✓)**: Checks state (e.g., preconditions)
- **Action**: Executes behaviors (e.g., navigation, sampling)

Why BT Instead of If-Else/Switch-Case? Traditional control structures have critical limitations:

If-else/switch-case problems: Deep nesting, no modularity, implicit control flow, no action monitoring, difficult to extend.

BT advantages: Visual clarity, modular subtrees, runtime reactivity, built-in recovery, compositional design.

Example: Sampling Action with Retry **If-else approach (rigid, nested):**

Listing 3.1: Imperative sampling with retry

```
def execute_sample(robot, site, retry_count=0):
    if not robot.at_start_condition(site):
        return FAILURE

    robot.start_sampling(site)
```

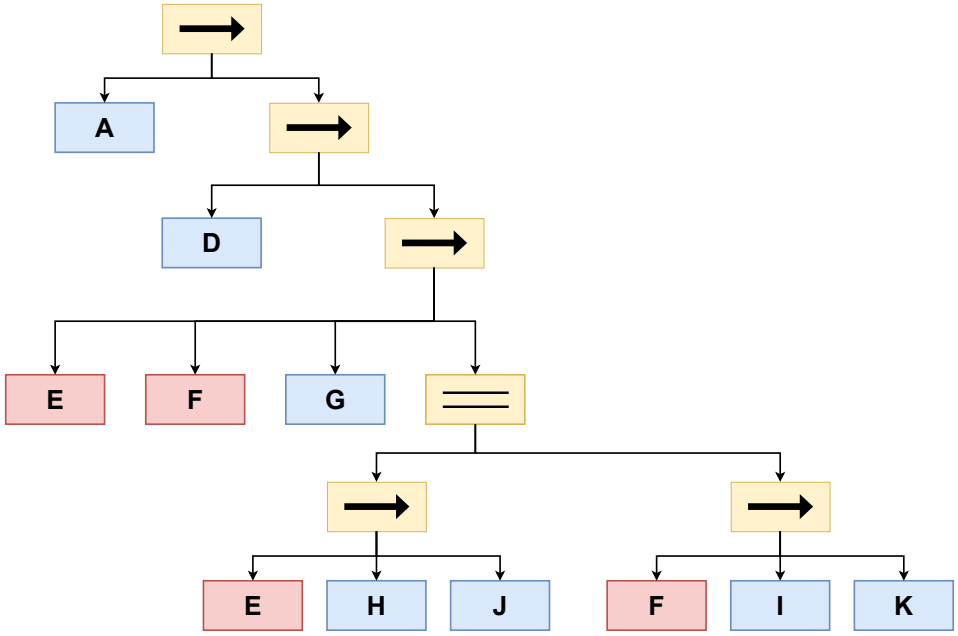


Figure 3.5: **From planning graph to BT.** Execution flows from the planning graph mapped to a BT with `Sequence`/`Parallel` control nodes, using `Wait` nodes (shown in red) to ensure that dependencies on actions executed by other robots are satisfied before starting.

```
if robot.action_failed():
    if retry_count < 3:
        return execute_sample(robot, site, retry_count+1
    else:
        if alternative_available():
            return try_alternative(robot, site)
        else:
            return FAILURE

if robot.at_end_condition():
    return SUCCESS
return FAILURE
```

BT approach (modular, composable):
The BT version separates main execution from recovery, makes failure handling explicit, reuses nodes, and is easily extended.

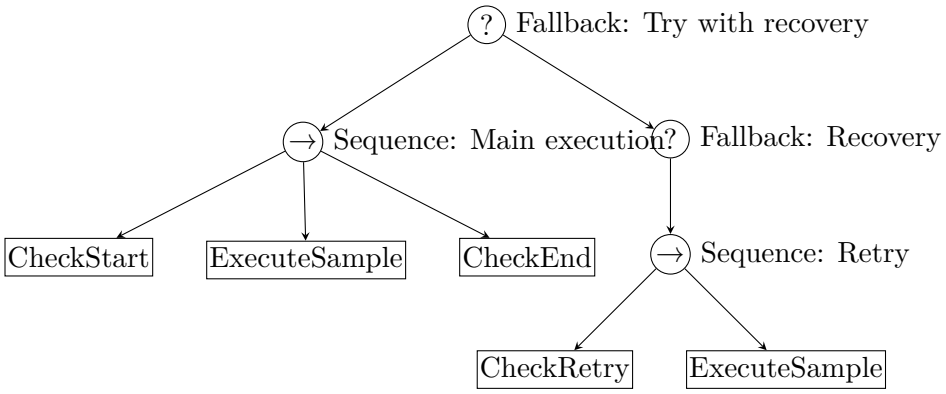


Figure 3.6: BT structure for sampling with recovery. Circles are control nodes (? = Fallback, → = Sequence), rectangles are action/condition nodes.

Why BTs for Robot Planning? BTs bridge symbolic planning and low-level control:

1. **Plan execution**: Naturally represent PDDL action sequences with temporal constraints
2. **Concurrency**: Parallel nodes enable simultaneous actions (essential for multi-robot)
3. **Monitoring**: Built-in progress tracking (`RUNNING` state)
4. **Recovery**: Hierarchical fallbacks from local retry to mission replanning

BTs in AMA-Exec In our framework, BTs serve three roles:

1. **Automatic synthesis**: PlanSys2 converts PDDL plans to BTs (Section 3.2.2.3), preserving causal structure
2. **Action-level supervision**: Each action wrapped in BT that checks preconditions, monitors progress, verifies postconditions, handles timeouts
3. **Distributed execution**: Each team executes its own BT independently, with STN-based temporal synchronization

This hierarchical structure (detailed in Section 6.3.1.1) enables robust execution without over-conservative mission aborts, essential for trans-media missions with multiple failure modes (mechanical issues, medium transitions, communication loss).

3.2.2.5 Limitations

Like most centralized execution frameworks, PlanSys2 faces certain limitations from its centralized execution model. The Executor remains a single point of failure and a bottleneck in large-scale multi-robot deployments. Temporal constraints are represented statically in the generated plan or in BT checks, but struggle to natively adapt dynamically to timing drift or asynchronous progress. Failure handling, while supported via BT constructs (e.g., retries, fallbacks), typically requires mission-level replanning when major deviations occur. Recent extensions, such as BTs synthesized from temporal networks (Zapf et al., 2024), improve temporal expressiveness but still encode fixed checks in the BT structure. These limitations do not diminish the value of PlanSys2, but they motivate complementary research into distributed, temporally adaptive execution architectures such as AMA-Exec.

3.2.3 Challenges in Executing Distributed Multi-Robot Plans

Multi-robot plans often involve complex dependencies (e.g., coalition formation, tightly constrained concurrency, and phased submissions within plan with cross-coalitions precedence). Execution systems must enforce these symbolic-temporal dependencies across asynchronous teams (Joyeux et al., 2009; Brenner and Nebel, 2009). Centralized executors typically struggle to capture these distributed constraints at runtime. While frameworks such as PlanSys2 provide strong baselines for symbolic execution, distributed missions often introduce additional challenges that go beyond centralized dispatching.

3.2.3.1 Temporal coherence under drift

In multi-robot settings, execution times vary due to heterogeneous platforms, environments, and communication latencies, weak repeatability of actions. Without a live temporal model to propagate delays, synchronization points break down and coordination can collapse (Stedl and Williams, 2005; Umbrico, Cesta, Mayer, et al., 2019). Timeline-based approaches explicitly model temporal flexibility, but few execution systems in ROS2 support this natively.

3.2.3.2 Failure isolation and recovery

Robust execution requires detecting, classifying, and localizing failures such as robot breakdowns, aborted actions, or communication

losses (Di Rocco et al., 2013; Belbachir et al., 2012). While BTs provide local recovery constructs, they do not scale to complex replanning of higher level plans due to their inherent execution-related complexity. Distributed execution demands mechanisms that can identify the impact of a failure and guide the system toward different strategies or mechanisms for recovery, ensuring that the idle time is minimized. In line with established principles in acting systems (Ghallab, Nau, et al., 2016), escalation should be staged: attempt local recovery and reallocation first, resorting to partial or global replanning only when necessary.

3.2.3.3 Resource arbitration

Shared resources, such as charging docks, communication relays, or task-specific tools, must be dynamically allocated without deadlock or starvation. Timeline-based frameworks like PLATINUm (Umbrico, Cesta, Cialdea Mayer, et al., 2017) demonstrate how time and resources can be treated jointly as constraints. Extending such reasoning to multi-robot ROS2 execution remains an open challenge.

3.2.3.4 Scalability and observability

Centralized execution engines, while simple to manage, become bottlenecks in large robot teams and obscure system status from operators. Recent work on multi-robot BT execution (Jeon et al., 2022; Ghzouli et al., 2023) shows how decentralized execution improves scalability and robustness. While decentralized execution improves scalability and robustness, it also introduces challenges of coordination and state synchronization, and must still provide operators with adequate observability for supervision and safety.

3.2.3.5 Toward the AMA system

These challenges do not undermine the contributions of frameworks such as PlanSys2. Rather, they reflect the increasing complexity of distributed missions in heterogeneous, uncertain environments. As presented in Chapter 2 and Section 3.1 this complexity rises significantly when treating with multi-medium domains and trans-media robots. To address the limitations in execution and building time reduction our AMA-Exec incorporates several improvements with regard to the execution monitoring of symbolic plans. First, we incorporate a dedicated *STNController* that supervises symbolic plans at runtime by maintaining a live precedence graph of symbolic and

temporal dependencies. In contrast to previous studies that incorporated temporal structure in a static manner into BTs, our system employs active monitoring to assess the progression of actions in the STN and identify discrepancies in timing expectations at the next constrained bottleneck.

Second, AMA-Exec decentralizes plan execution by decomposing problems into constrained sub-problems through AMA-Plan and execute subproblems through modular, per-team BT, enabling distributed robot groups to operate asynchronously under unified symbolic constraints. This modularization preserves the generality of symbolic PDDL plans while embedding real-time supervision and constraint propagation directly into execution, bridging the gap between centralized planning and dynamic, distributed, failure-prone environments recovery.

Finally, our Distributed Execution Manager (DEM) coordinates global recovery mechanism through constraint-aware reallocation and failure handling mechanisms. The combination of these modules establishes a flexible, runtime-supervised execution pipeline with the capacity to plan and manage execution of complex multi-robot missions. Instead of replacing existing paradigms, our approach integrates symbolic plan with decomposition, the modular robustness of BT execution, and the temporal responsiveness of STN-based delay propagation. This synergy facilitates reliable and adaptive execution in asynchronous, failure-prone environments without compromising generality or modularity.

Chapter 4

Adaptive and Modular Planning (AMA-Plan)

Contents

This chapter presents AMA-plan, a structured planning framework for MRTA in complex, multi-medium environments. It recalls the challenges in existing planning methodologies, highlights the limitations of traditional solvers, and presents AMA-Plan as a modular solution for efficient decomposition and resolution of complex problems.

4.1 Principles of the Approach

The AMA-Plan framework is designed to improve the scalability and adaptability of MRTA by introducing hierarchical abstraction and cost-based optimization techniques. The foundational principle of AMA-Plan is to decompose complex planning problems into smaller, independent subproblems that can be solved efficiently without direct execution modeling. To address the high complexity of multi-robot mission planning, AMA-Plan employs a structured problem restructuring of the mission planning pipeline into four distinct stages that exploit domain knowledge to improve scalability and structure: A Spatio-cost clustering of sites and goals (in 4.2.1), a

Robot allocation with resolution cost estimation (in 4.2.2), with the resulting Path assignment and sequencing for each robot (in 4.2.2.3) and finally the Path Sequential Solving in Teams in a STN-based graph (in 4.2.3).

The AMA-Plan approach does not natively perform dynamic reassignments during execution, but is invoked by AMA-Exec. Instead, it precalculates the mission structure and ensures that an efficient plan is provided before its execution begins, providing the execution system with execution-ready plan delivered at finite times. This section introduces the key components of AMA-Plan and illustrates how it enables efficient, scalable, and structured multi-robot mission planning.

4.1.1 MRTA and Planning Problem Modeling

As presented in Chapter 1, our study builds upon a pollution sampling mission involving multiple water bodies separated by land, as depicted in Figure 4.1. The mission requires the collection of underwater measurements, which require operator supervision and real-time communication with a base station. This communication is facilitated by a surface robot that converts underwater acoustic signals into radio transmissions.

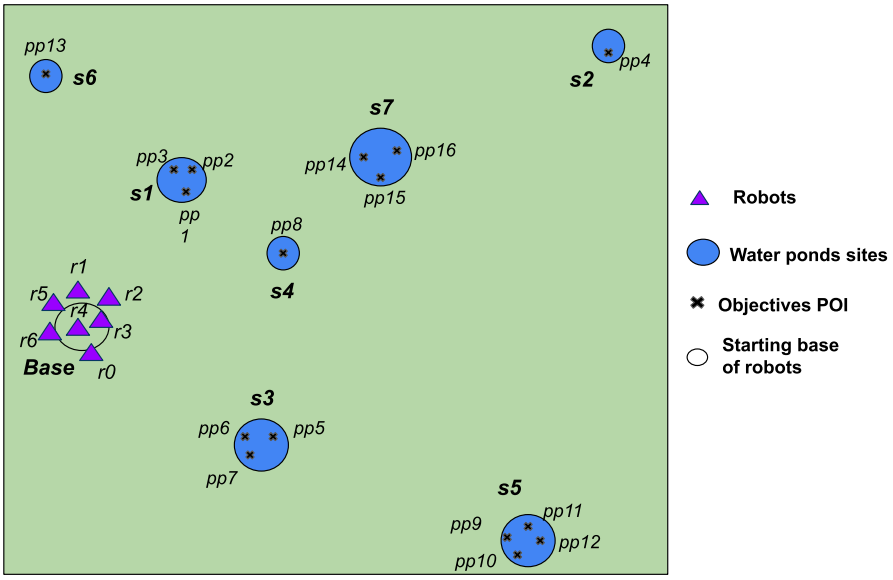


Figure 4.1: A detailed representation of a trans-media environmental sampling mission with 7 robots, 7 sites with base and 15 pollution points to sample.

4.1.1.1 Problem Formalization

We base our formulation on the classical MRTA problem structure (Gerkey and Matarić, 2004), which models the allocation of a set of robots to a set of tasks under resource and temporal constraints. In this work, we extend that formalization to explicitly incorporate *spatial partitioning into medium-based sites* and *agents' medium-dependent capabilities*, in order to represent the multimodal and multi-medium nature of the considered missions. This extension highlights the medium-specific characteristics of multi-environment mission parameters.

Although this work illustrates the formulation through a trans-media mission scenario for complexity analysis purposes, the modeling remains applicable to other homogeneous and heterogeneous multi-robot settings, as demonstrated in Chatper 6. The proposed predicates, temporal constraints, and cost structures are generic and can represent a wide range of cooperative multi-robot problems involving spatially distributed tasks and dynamic agent capabilities—such as inspection, logistics, or environmental monitoring.

Formally, we define our MRTA problem as a tuple:

$$\mathcal{P} = \langle \mathcal{R}, A, P, \mathcal{S}, M, I, G \rangle$$

where:

- $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ the set of robots
- A the set of robot actions
- $P = \{p_1, p_2, \ldots, p_m\}$ the set of points of interest (PoIs)
- $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$ the set of water bodies (or "sites")
- $M = \{aerial, terrestrial, aquatic\}$ the set of media
- I and G are respectively the initial and goal states.

PoIs are divided spatially, with every PoI being associated to a site: the set $P = \bigcup_{j=1}^{K} P_{s_j}$ is partitioned into sites. Furthermore, we classify the PoIs into three categories:

- **Sampling points** (P_{spl}) locations where underwater robots collect mission-critical data; in our example we call them *pp* for pollution points.

- **Transition points** (P_{tr}) positions enabling a change of medium, such as switching between aerial and aquatic configuration; we call them *sp* for surface points.
- **Observation points** (P_{obs}) aerial vantage points used to observe a site, typically to "verify the state of transition points; we call them *cp* for central points.

Each site contains a combination of sampling and transition points, and a single observation point above the site center.

A base point p_b denotes the terrestrial position at which all robots are located in the initial state giving the representation of fig 4.1. Given the initial state I, the goal is to compute the best near-optimal multi-robot plan (makespan-wise) that allocates robots to sites while respecting concurrency constraints, minimizes overall mission time (makespan) and ensures all sites are successfully visited and sampled (e.g fig 1.2).

4.1.1.2 Action Formalization and Constraints

This planning problem is modeled in PDDL 2.1, wherein actions are defined as preconditions and effects with associated cost functions (domain example available in appendix A.3). The key domain actions are:

Navigate: These actions model the robot motions between two PoIs p_i, p_j *belonging to the same medium.* The set of navigation actions is:

$$A_{\text{nav}} = \{a_{\text{nav}}(r, p_i, p_j) \mid r \in \mathcal{R},\, p_i, p_j \in P\}$$

with

$$r \in \mathcal{R}, p_i, p_j \in P, p_i \neq p_j, M_{p_i} = M_{p_j}$$

e.g., moving from surface point sp1 to sampling point pp1 in the same medium.

Observe: These actions model the observation of all the transition points of a site, from the unique observation PoI associated to the site. The set of observation actions is:

$$A_{\text{obs}} = \{a_{\text{obs}}(r, p, s) \mid r \in \mathcal{R},\, p \in P_{obs} \cap P_s\}$$

Switch: This action represents a physical medium transition from m_k to m_l for a robot r, that can only be executed at one of the transition points of a site. The set of switch actions is:

$$A_{sw} = \{a_{sw}(r, p, m_k, m_l, s) \mid r \in \mathcal{R},\, p \in P_{tr} \cap P_s, \{m_k \neq m_l\} \in M_p\}$$

This action can only be executed after the transition points of the site s have been observed:

$$\begin{gathered} t_{\text{start}}(a_{\text{sw}}(r, p_i, m_k, m_l, s)) \geq t_{\text{end}}(a_{\text{obs}}(r, p_j, s)) \\ p_i \in P_{tr} \cap P_s,\ p_j \in P_{obs} \cap P_s \end{gathered} \tag{4.1}$$

This ensures that the environment is verified before a medium transition occurs.

Takeoff and Land: these actions only occur at the base point p_b:

$$A_{\text{ld}} = \{a_{\text{ld}}(r, p_b) \mid r \in \mathcal{R},\ p_b \in P, m_{r_i} = aerial\}$$
$$A_{\text{toff}} = \{a_{\text{toff}}(r, p_b) \mid r \in \mathcal{R},\ p_b \in P, m_{r_i} = terrestrial\}$$

Relay Data: This action enables the online transmission of underwater data to base operators through a robot located at the site surface, converting acoustic transmissions into radio transmissions. The set of data conversion actions is:

$$A_{\text{trsl}} = \{a_{\text{trsl}}(r, p, s) \mid r \in \mathcal{R},\ p \in P_{tr} \cap P_s, m_r = aquatic\}$$

Sample: These actions model underwater data collection, and constitute the mission goal. Their set is:

$$A_{\text{spl}} = \{a_{\text{spl}}(r, p, s) \mid r \in \mathcal{R},\ p \in P_{spl} \cap P_s, m_r = aquatic\}$$

Sampling actions can only be executed when another robot is executing a data conversion on the same site s:

$$\begin{gathered} \forall r_u, r_v \in \mathcal{R},\ \forall p_i \in P_{spl} \cap P_s,\ \forall p_j \in P_{tr} \cap P_s, \\ m_{r_u} = m_{r_v} = \text{aquatic}, \\ t_{\text{start}}(a_{\text{spl}}(r_u, p_i, s)) \geq t_{\text{start}}(a_{\text{trsl}}(r_v, p_j, s)) \\ \wedge\ t_{\text{end}}(a_{\text{spl}}(r_u, p_i, s)) \leq t_{\text{end}}(a_{\text{trsl}}(r_v, p_j, s)) \end{gathered} \tag{4.2}$$

This ensures real-time relay observations of the pollution data at the base.

The goal state G is reached when all sampling points P_{spl} in P have been sampled and the robots are back at the base point p_b, the planning problem consists in finding sequences of actions from the initial state I to the goal state G, satisfying communication and observation constraints 4.2 and 4.1.

4.1.1.3 Cost Model

Each action is associated with a cost, defined by its execution duration:

- Actions with fixed cost: Observe, Sample, Relay Data, Takeoff, Land. Switch actions have two fixed costs, defined by direction of the transition (Aquatic↔Aerial)

$$T_a = \text{const}$$

- Navigate actions have a cost defined by their duration, the navigation cost depending on the distances $D(p_u, p_v)$ between the PoIs (p_u, p_v) and the medium m_{r_i} in which they are executed:

$$C_{\text{nav}}(r_i, p_u, p_v) = \frac{D(p_u, p_v)}{V(m_{r_i})}. \tag{4.3}$$

  where $V(m_{r_i})$ is the velocity of the robot r_i in the medium m_{r_i}.

The fixed action costs (T_a) have been selected in accordance with the literature on the trans-media robot operation and physical challenges, satisfying the following constraints:

- Switch from water to air takes longer than the opposite.
- Relay Data duration is longer than Sampling action.
- The observe action's duration scales down when two robots share the task, its cost depends on the size of the site and the number of robots allocated to it.

4.1.2 Abstracting Related Sub-Problems

AMA-Plan applies a sequence of pre-allocation algorithms and constraints to determine near-optimal robot–sites distributions before each subproblem is solved by a local planner.

4.1.2.1 Problem Challenges and Decomposition

MRTA planning in multi-medium environments presents three key challenges that influence mission planning and execution:

1. Task allocation must minimize mission makespan: The mission duration is driven by the minimization of the longest execution path for all robots, requiring a balanced workload distribution among them.

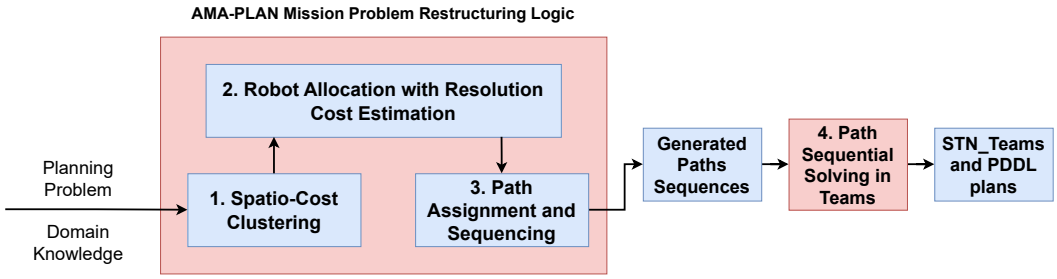


Figure 4.2: Structure of the decomposed pre-allocation problem in AMA-plan. The global MRTA mission is restructured into four stages presented in: (**1.**) in Section 4.2.1, (**2.**) in Section 4.2.2, (**3.**) Section in 4.2.2.3, and (**4.**) in Section 4.2.3 and Figure 4.3. These steps generate execution-ready team subplans, encoded as $\text{STN}_{\text{teams}}$ and PDDL plans.

2. Medium-dependent navigation significantly impacts execution time: Movement across different mediums (air, land, and water) introduces speed variations, meaning navigation must be cost-efficient to reduce travel time.
3. Multi-robot collaboration requirements: Certain tasks, such as sampling, require coordinated execution by multiple robots, adding constraints on task order and synchronization.

To address these challenges, AMA-Plan structures the problem into three levels of abstraction:

1. **Spatio-Cost Clustering**: Sites are grouped into clusters based on spatial proximity and task cost heuristics, using a weighted K-Means approach. Each cluster is used to separate sites and facilitate the greedy allocation assignment.
2. **Robot Allocation with Resolution Cost Estimation**: A cost-aware greedy algorithm assigns robots to site clusters by minimizing inter-site travel and estimating resolution costs (e.g., sampling, relay, navigation) of sites, with diminishing-return penalties for redundant assignments.
3. **Path Assignment and Sequencing**: Robots are assigned to paths of sites based on cost-optimal coverage. Each site is converted into symbolic subproblems, with inter-path dependencies extracted to guide execution coordination. Teams are defined as coalitions of robots whose paths converge to jointly operate on one or more shared sites.

This decomposition reduces the computational complexity of planning by introducing a structured pre-allocation stage that as-

signs site sequences (paths) to robots before the full planning process begins. Partitioning the mission into smaller, independent subproblems reduces the solver's search space and enables parallel resolution of unrelated site groups.

Figure 4.2 illustrates the general problem decomposition process into robot paths, and Figure 4.3 shows the sequential solving of these subproblems by teams. In AMA-plan, a path is the ordered sequence of sites allocated to a single robot, while a team is a coalition of robots that jointly resolve a site. Teams emerge directly from overlaps in the site assignments of different robot paths and may operate together across multiple sites when the same team is assigned consecutively, thus forming a higher-level planning unit that preserves site-level coalitions while respecting individual robot assignments.

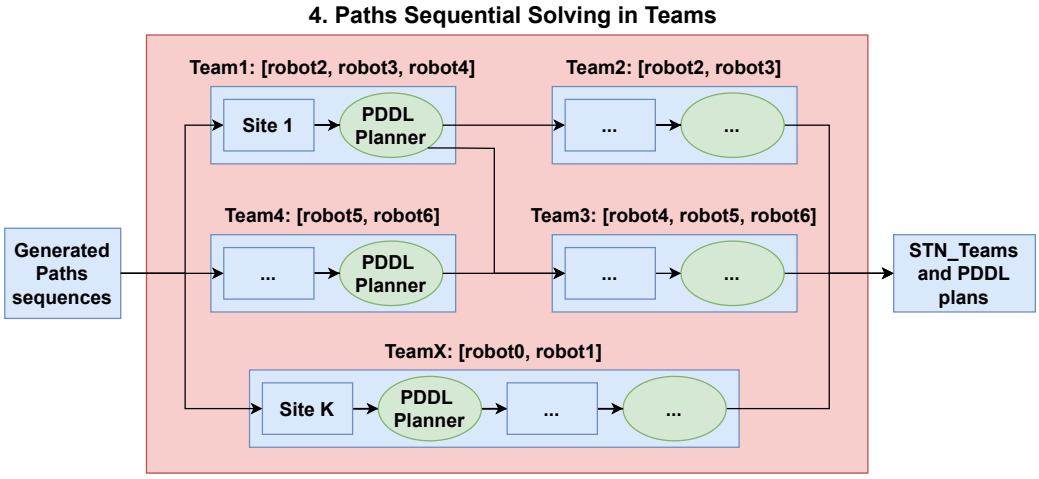


Figure 4.3: Sequential resolution of clustered subproblems by robot teams derived from pre-allocated paths. Each colored block represents a site assigned to a team of robots, resolved in the order determined by the pre-allocation strategy.

4.1.2.2 Plan Representation and Subproblems Sequencing

A team is defined as the set of robots assigned to jointly resolve one or more sites. Each team is associated with a sequence of sites forming its sub-mission. The initial assignment phase ensures that each site receives an assigned team of robots based on task requirements, travel costs, and robot capabilities. Each site resolution subproblem consists of a sequence of tasks that the team must complete.

The structure of these subproblems and their interconnections within the overall plan can be represented as a structured STN

(Dechter et al., 1991), which we extend into a team-based representation called $\text{STN}_{\text{teams}}$.

Classical STN formalism. An STN encodes temporal dependencies using:

- **Nodes:** Timepoints corresponding to the start or end of actions.
- **Edges:** Temporal constraints $(x, y, [\min, \max])$ indicating that the time difference between two timepoints x and y must lie between min and max.

Our enriched $\text{STN}_{\text{teams}}$ representation. We enrich this formalism in our abstraction by grouping timepoints into execution teams:

- **Team nodes:** Each execution team $team_i$ is represented by two timepoints $T_{\text{start}}(team_i)$ and $T_{\text{end}}(team_i)$, capturing the temporal span of its plan.
- **Edges:** Dependencies between teams endpoints and startpoints, encoding precedence relations due to shared robots, resource contention, or inter-site ordering. These edges may include minimum and maximum delays.

This hierarchical enrichment preserves the semantics of an STN while structuring the network around our team-based decomposition. By structuring the plan in this way, AMA-Plan ensures that inter-site and intra-site dependencies are captured explicitly before execution. This structured plan can be directly interpreted by an execution system but is computed entirely within AMA-plan, based on cost-efficient robot assignments and cost-efficient path formation to reduce the idle time during team transition.

This approach allows AMA-Plan to bridge high-level planning and real-world execution constraints, ensuring that plans remain computationally efficient while respecting multi-robot collaboration requirements.

4.2 Abstracting and Solving Multi-Robot Planning

Building on the problem decomposition principles, we now detail the sequential steps by which AMA-Plan transforms a global MRTA mission into a set of smaller, tractable and constrained subproblems.

4.2.1 Spatio-Cost Clustering

The clustering phase groups mission sites into compact, cost-balanced clusters, forming the building blocks for the execution paths introduced in Section 4.1.2. Each cluster is designed to minimize long-distance travel while distributing workload evenly across teams. To achieve this, AMA-Plan first estimates the resolution cost of each site, considering domain's constraints: sampling dependencies, navigation time, and collaboration constraints, and then applies a cost-augmented K-Means algorithm. This approach integrates both spatial proximity and workload complexity into the clustering process, ensuring that each execution group is well-balanced in terms of location and operational effort, a typical representation is presented in Figure 4.4. This step takes the mission parameters and generates possible clusterings of sites, based on the number of robots, teaming constraints, and the distribution of goals.

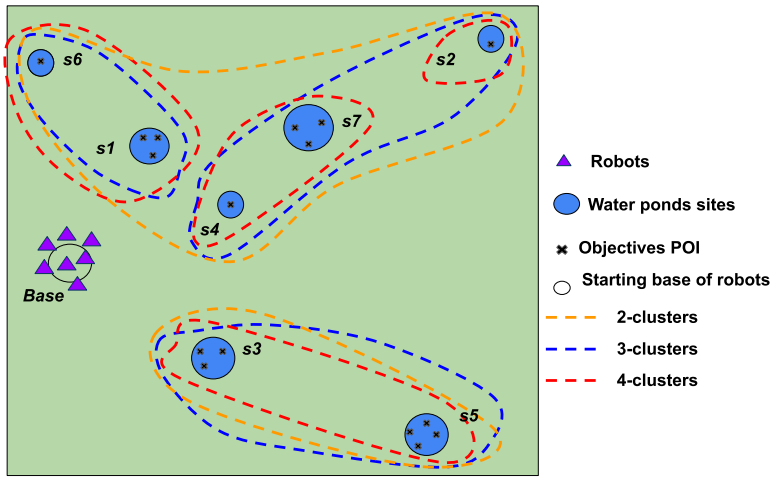


Figure 4.4: Spatio-cost clustering of sites (4.2.1). Water pond sites are grouped into potential clusters (2, 3, or 4 clusters), balancing both spatial proximity and workload complexity.

4.2.1.1 Cost Estimation for Site Resolution

Each site $s_j \in \mathcal{S}$ consists of a set of tasks, such as sampling, data transmission, and navigation, which must be completed by a team of robots. The estimated resolution cost $C(s_j, \mathcal{R}_{s_j})$ is defined as the total time required for a team of robots $\mathcal{R}_{s_j} \subset \mathcal{R}$ assigned to site s_j,while considering task parallelization, relay constraints, and intra-site navigation.

Factors Affecting Site Resolution Cost

The time required to resolve a site depends on multiple factors:

- The number of robots assigned ($|\mathcal{R}_{s_j}|$) and their specific roles (sampling, relaying, etc.).
- The operational complexity of the site, i.e., number and type of points of interest to be processed.
- The total intra-site navigation distance between tasks.
- Task concurrency requirements, e.g., sampling that must occur simultaneously with data relay.

We define the estimated resolution cost $C(s_j, \mathcal{R}_{s_j})$ as the time needed to fully complete all tasks at site s_j when assigned team of robots $\mathcal{R}_{s_j} \subset \mathcal{R}$. The goal is to ensure that the heuristic models the domain's constraints while ensuring that additional robots effectively decrease the estimated cost.

Heuristic Model for Site Resolution Cost

We approximate the cost of resolving a site s_j for a certain number of robots $\mathcal{R}_{s_j}$ using a decomposition of its contributing factors:

$$C(s_j, \mathcal{R}_{s_j}) = \max\left(C_{\text{sample}}(s_j, \mathcal{R}_{s_j}), C_{\text{trsl}}(s_j, \mathcal{R}_{s_j})\right) + C_{\text{transit}}(s_j, \mathcal{R}_{s_j}) \tag{4.4}$$

where:

1. $C_{\text{sample}}(s_j, \mathcal{R}_{s_j})$ is the estimated time to complete all sampling tasks at the site.
2. $C_{\text{trsl}}(s_j, \mathcal{R}_{s_j})$ is the time for data relaying and translation (when required).
3. $C_{\text{transit}}(s_j, \mathcal{R}_{s_j})$ is the intra-site navigation time required for completing tasks.

Modeling Task Execution Cost

1. Sampling Cost Estimation: Sampling tasks are the primary objective at each site. Since sampling requires at least two cooperating robots (one for sampling, one for relaying data), we model it as:

$$C_{\text{sample}}(s_j, \mathcal{R}_{s_j}) = \frac{T_{\text{sample}}(s_j)}{\max(1, |\mathcal{R}_{s_j}| - 1)} \tag{4.5}$$

where $T_{\text{sample}}(s_j)$ is the total sampling duration for site s_j, and the denominator ensures parallelism scaling (since at least one robot is needed for relay). This enforces diminishing returns on adding robots.

2. Data Relay and Transmission Cost: For sites requiring data relay:

$$C_{\text{trsl}}(s_j, \mathcal{R}_{s_j}) = T_{\text{trsl}}(s_j) \cdot \left(1 - \frac{\gamma}{|\mathcal{R}_{s_j}|}\right) \tag{4.6}$$

where $T_{\text{trsl}}(s_j)$ is the base relay time and γ models relay efficiency gain with additional robots.

3. Transit Cost for Intra-Site Navigation: Robots must navigate between tasks within a site:

$$C_{\text{transit}}(s_j, \mathcal{R}_{s_j}) = D_{\text{site}}(s_j) \cdot \frac{1}{V_{\text{eff}}(|\mathcal{R}_{s_j}|)} \tag{4.7}$$

where $D_{\text{site}}(s_j)$ is the total intra-site travel distance, estimated using a Traveling Salesman Problem (TSP) solver, and $V_{\text{eff}}(r_i)\ \forall r_i \in \mathcal{R}_{s_j}$ is the scaling effective velocity, which improves with more robots.

Dynamic Scaling Factors for Cost Estimation

To reflect realistic execution dynamics in the estimated resolution cost $C(s_j, \mathcal{R}_{s_j})$, AMA-Plan applies scaling factors derived from the composition of each site, capturing diminishing returns, parallel execution efficiency, and relay effectiveness.

$$C(s_j, \mathcal{R}_{s_j}) = \left[\max\left(\gamma \cdot C_{\text{relay}}(s_j),\ \frac{C_{\text{sample}}(s_j)}{|\mathcal{R}_{s_j}| - 1} \cdot \beta\right) + C_{\text{trans}}(s_j)\right] \cdot \frac{1}{1 + k(|\mathcal{R}_{s_j}| - 2)^{\beta}} + \frac{D_{\text{sit}}}{|\mathcal{R}} \tag{4.8}$$

where:

- $C_{\text{sample}}(s_j)$ is the cumulative sampling effort within site s_j.
- $C_{\text{relay}}(s_j)$ is the cumulative relay overhead at site s_j.
- $C_{\text{trans}}(s_j)$ is the cumulative transition cost at site s_j.

- $D_{\text{site}}(s_j)$ is the intra-site navigation distance.
- k is the *diminishing returns factor*, defined as:

$$k = \frac{|P_{\text{spl}}(s_j)|}{|P_{\text{spl}}(s_j)| + |P_{\text{tr}}(s_j)| + 1} \tag{4.9}$$

- β is the *parallel execution factor*, defined as:

$$\beta = \frac{|P_{\text{spl}}(s_j)|}{|P_{\text{spl}}(s_j)| + |\mathcal{R}_{s_j}|} \tag{4.10}$$

- γ is the *relay efficiency factor*, defined as:

$$\gamma = 1 - \frac{|P_{\text{tr}}(s_j)|}{|P_{\text{spl}}(s_j)| + |P_{\text{tr}}(s_j)| + 1} \tag{4.11}$$

Implementation in AMA-plan

The cost estimation is implemented in AMA-Plan as follows:

- **Base Costs:** The functions `CostEstimOftour` compute $C_{\text{transit}}(s_j, \mathcal{R}_{s_j})$ using a TSP-based heuristic.
- **Scaling Factors:** The function `compute_scaling_factors` dynamically computes k, β, and γ based on mission parameters.
- **Cost Computation:** The function `get_weights_of_sites` integrates all factors into the final site cost estimation for all possible $\mathcal{R}_{s_j}$ attributions.

We define the resolution cost matrix C_{mat} as follows:

$$C_{\text{mat}} : \mathcal{S} \times \mathbb{N}^+ \to \mathbb{R}$$

where $C_{\text{mat}}(s_j, \rho)$ represents the estimated resolution cost for site $s_j \in \mathcal{S}$ when assigned a team of ρ robots. The value ρ corresponds to the cardinality $|\mathcal{R}_{s_j}|$ of any feasible robot team assigned to s_j.

This heuristic generates the resolution cost matrix $C_{\text{mat}}(s_j, \rho)$ $\forall s_j \in \mathcal{S}$, providing a fast yet accurate estimate of site resolution times for any feasible team size. By precomputing these values, AMA-Plan reduces the computational burden during clustering and enables more informed partitioning decisions. The matrix acts as a weighted input to the cost-aware clustering stage, ensuring that resulting site groups are balanced not only in geographic proximity but also in operational complexity.

4.2.1.2 Clustering Sites Based on Cost & Distance

Problem Definition Given a set of sites $\mathcal{S} = \{s_1, s_2, ..., s_n\}$ and their estimated resolution costs matrix $C_{\text{mat}}(s_j, \rho)$ computed in the previous step, our goal is to partition them into clusters $Cl_{set} = \{c_1, c_2, ..., c_k\}$ that minimize unnecessary travel while ensuring balanced workloads. Each cluster should group geographically close sites while also ensuring that resolution costs are distributed evenly across clusters. This prevents some teams from being overloaded while others remain underutilized.

To achieve this, we encode each site s_j into a cost-weighted feature space:

$$F(s_j) = [x_i, y_i, C_{\text{mat}}(s_j, \rho)] \tag{4.12}$$

where:

- (x_i, y_i) are the spatial coordinates of the site.
- $C_{\text{mat}}(s_j, \rho)$ is the estimated resolution cost for a representative team size ρ, capturing the expected effort required to process the site depending on various attributions.

Weighted K-Means Clustering for Site Partitioning

To integrate both spatial and workload information into the partitioning process, we apply K-Means clustering in this weighted feature space, ensuring that the resulting clusters account for both spatial proximity and workload similarity. The clustering process follows these steps:

1. **Extract site coordinates and resolution costs**. Normalize cost values to balance spatial and workload influence.
2. **Construct feature vectors** $F(s_j)$ by combining site locations and normalized costs.
3. **Apply K-Means clustering** to partition sites into execution groups based on Euclidean distances in the feature space.
4. **Ordering intra-cluster execution order**:
   - Solve a Traveling Salesman Problem (TSP) within each cluster to determine the shortest task execution path.
   - Reorder tasks using a zigzag assignment strategy to prevent early overcommitment of robots to a single sequence.

K-Means clustering minimizes:

$$\sum_{c_j \in C} \sum_{s_j \in c_j} \|F(s_j) - F(c_j)\|^2 \tag{4.13}$$

where $F(c_j)$ is the centroid of cluster c_j, ensuring that sites within each cluster are both spatially compact and workload-balanced.

Ordering Execution Order

After clustering, the execution sequence inside each cluster is ordered more optimally. A Traveling Salesman Problem (TSP) heuristic determines the shortest execution path within the cluster. However, executing sites strictly in TSP order can cause some robots to be committed early to long single-robot sequences, reducing concurrency. To avoid that, a zigzag reordering strategy is applied. Instead of processing sites strictly in TSP order, this strategy alternates site assignments from both ends of the optimal path, balancing execution across teams and reducing bottlenecks.

Algorithm: Cost-Weighted K-Means for Site Clustering

Algorithm 1 AMA-Plan Cost- and Distance-Aware Site Clustering

Input: Sites $\mathcal{S}$, robots $\mathcal{R}$, cost matrix $C_{\text{mat}}(s_j, \rho),\ \rho \in N_{\text{Team}}$, number of clusters k
Output: Clusters $Cl_{set} = \{c_1, \ldots, c_k\}$
Normalize: $C(s_j, \rho) \leftarrow \frac{C_{\text{mat}}(s_j, \rho)}{\max_{s \in \mathcal{S}} C_{\text{mat}}(s, \rho)}$
Build feature vectors: $F(s_j) \leftarrow [x_{s_j}, y_{s_j}, C(s_j, \rho)]$
Apply K-Means on $F(S)$ with k clusters
for each $c_j \in Cl_{set}$ **do**
    Solve intra-cluster TSP for site order
    Apply zigzag reordering to sequence
return Cl_{set}

Clustering sites based on both spatial proximity and estimated resolution cost helps prevent execution bottlenecks where some robot teams are overloaded while others are underutilized. Traditional K-Means clustering only considers geometric distance, which may group together sites of vastly different complexity. By augmenting the site coordinates with a normalized resolution cost $C(s_j, \rho)$, AMA-Plan ensures that both geographic proximity and workload distribution are taken into account during clustering. This cost-aware clustering leads to execution groups that are more balanced in both location and effort. Within each cluster, task execution is further refined using a Traveling Salesman Problem (TSP) heuristic

to determine an efficient site visitation order. However, executing this order sequentially can lead to early overcommitment of certain robots. To address this, AMA-Plan applies a zigzag reordering strategy, interleaving site assignments from both ends of the TSP path. This maintains a more evenly distributed execution load and improves concurrency. This produces compact, cost-balanced clusters that serve as the input for the next stage: robot-team allocation and path formation within each cluster.

By structuring the mission into compact and cost-balanced clusters, AMA-Plan minimizes long-distance navigation inefficiencies while improving team-level workload equilibrium. The next section details how robots are assigned to these clusters and how optimal execution paths are generated within each group.

4.2.2 Robot Allocation with Resolution Cost Estimation

We now detail how AMA-Plan assigns robots to execution paths within each cluster obtained in Section 4.2.1, selecting allocations that minimize the mission makespan while satisfying operational constraints. Figure 4.5 represents this step for the 2-cluster clustering of our typical scenario and finding the best allocations scenarios. From the candidate clusterings, AMA-Plan evaluates feasible robot-to-cluster and robot-to-site allocations, estimating these scenarios resolution costs.

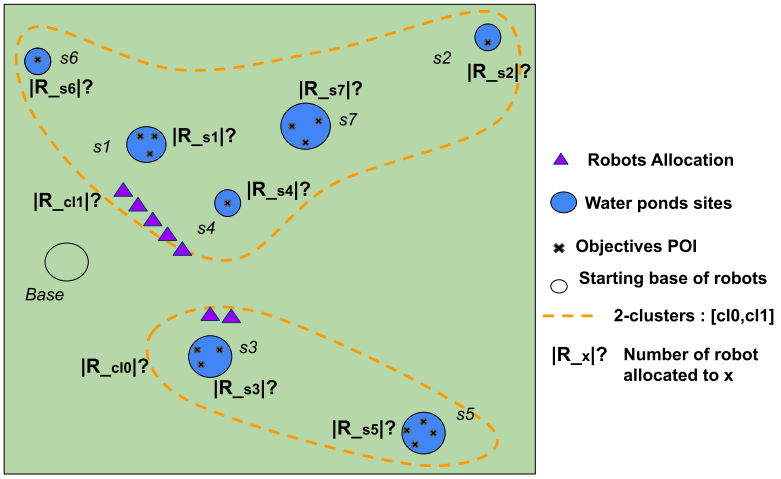


Figure 4.5: Robot allocation with resolution cost estimation (4.2.2). Each cluster cl_i and site s_j is annotated with the number of robots $|\mathcal{R}_{cl_i}|, |\mathcal{R}_{s_j}|$ allocated for resolution,allowing the evaluation of different allocation scenarios and their associated costs.

4.2.2.1 Defining the Robot Assignment Problem and Execution Path Formation

Once sites have been clustered, the next step in AMA-Plan is to determine the best robot teams and execution paths for each cluster. The goal is to identify the optimal allocation of robots to paths such that:

1. the total mission duration (makespan) is minimized,
2. site-level constraints (e.g., minimum required robots per site) are respected,
3. workloads are balanced to avoid excessive delays.

Given a mission scenario with:

- A set of robots $\mathcal{R} = \{r_1, r_2, ..., r_n\}$.
- A set of clusters $Cl_{set} = \{cl_1, cl_2, ..., cl_y\}$, where each cluster cl contains a set of sites $\mathcal{S}_{cl}$.
- A set of possible team sizes $N_{\text{Team}} = \{\rho \in \mathbb{N} \mid 2 \leq \rho < n\}$, where ρ denotes the number of robots assigned to a given site. The size $n-1$ is excluded to avoid forming a single-robot team in the remaining set.
- A matrix of estimated site resolution costs $C_{\text{mat}}(s_j, \rho)$, where $\rho = |\mathcal{R}_{s_j}| \in N_{\text{Team}}$, representing the expected cost for resolving site s_j with a team of size ρ, as defined in Section 4.2.1.1.

The objective is to determine the optimal assignment of robots to sites within each cluster by generating all relevant robot allocation scenarios for each cluster, evaluating their total cost using a cost-based heuristic and finally selecting the optimal assignment that minimizes the mission estimation of the makespan (plan's duration).

Objective and Cost Model

To determine the most cost-efficient allocation in term of makespan, we explore all possible combinations of team and site assignments. We define $Al(s_j, \mathcal{R}_{s_j})$ as the allocation of team $\mathcal{R}_{s_j}$ to site s_j. For a cluster of sites $cl \in Cl_{set}$, the allocation is expressed as $Al(\mathcal{S}_{cl}, \mathcal{R}_{cl})$, where $Al(Cl_{set}, \mathcal{R})$ represents the allocation of all robots $\mathcal{R}$ across all clusters. Each team $\mathcal{R}_{s_j} \subseteq \mathcal{R}$ is a subset of robots assigned to site s_j, with $|\mathcal{R}_{s_j}| = \rho \in N_{\text{Team}}$.

The cost function $C_F(s_j, \mathcal{R}_{s_j})$ captures the cost $Al(s_j, \mathcal{R}_{s_j})$, taking into account previous assignments of the robots in $\mathcal{R}_{s_j}$. Given a site s_j, the assigned robots $\mathcal{R}_{s_j}$, and the previously visited sites $\mathcal{S}_{prev}(r) = \{s_1, \ldots, s_\sigma\}$ by robot r with cost C_{prev}, $C_F(s_j, \mathcal{R}_{s_j})$ is defined as:

$$C_F(s_j, \mathcal{R}_{s_j}) = C(s_j, \mathcal{R}_{s_j}) + C_{prev} \tag{4.14}$$

$$C_{prev} = \max_{r \in R_s \cap \mathcal{R}_{s_j}} \Big(\sum_{s \in \mathcal{S}_{prev}(r)} C_F(s, R_s) + C_{\text{nav}}(r, p_{s_\sigma}, p_{s_j}) \Big) \tag{4.15}$$

where:

- $\sum_{s \in \mathcal{S}_{prev}(r)} C_F(s, R_s)$ is the sum of the cost of the previous site assessment allocated to the robots.
- $C_{\text{nav}}(r, p_{s_\sigma}, p_{s_j})$ is the distance cost between the last and current sites for a robot $r \in R_s \cap \mathcal{R}_{s_j}$.

The total cost of an allocation scenario $Al(\mathcal{S}_{cl}, \mathcal{R}_{cl})$ for a cluster cl is defined as the maximum cost across all team-site assignments, ensuring that the mission makespan is minimized:

$$C_F(\mathcal{S}_{cl}, \mathcal{R}_{cl}) = \max_{s_j \in \mathcal{S}_{cl}, \mathcal{R}_{s_j} \in \mathcal{R}_{cl}} C_F(s_j, \mathcal{R}_{s_j}) \tag{4.16}$$

where:

- $\mathcal{R}_{cl} = \{\mathcal{R}_{s_j} \mid s_j \in \mathcal{S}_{cl}\}$, the set of assigned robot teams per site.
- $Al(\mathcal{S}_{cl}, \mathcal{R}_{cl}) = \{Al(s_j, \mathcal{R}_{s_j}) \mid s_j \in \mathcal{S}_{cl}\}$, the full allocation scenario.

Optimization Goal

We aim to minimize the mission makespan completion time by reducing the cost of the longest path of site, defined as a greedy min–max makespan optimization. Since the total execution time is dictated by the longest site resolution time in any given allocation scenario, the objective function follows a min–max heuristic, minimizing the maximum site resolution cost.

$$\text{argmin}_{Al(\mathcal{S}_{cl}, \mathcal{R}_{cl})} \max_{s_j \in \mathcal{S}_{cl}, \mathcal{R}_{s_j} \in \mathcal{R}_{cl}} C_F(s_j, \mathcal{R}_{s_j}). \tag{4.17}$$

This yields a balanced allocation that reduces bottlenecks before the full STN-based scheduling phase.

4.2.2.2 Generating and Evaluating Team-Site Allocation Scenarios

Once the problem has been formally defined (Section 4.2.2.1), AMA-Plan proceeds to generate and evaluate all possible robot allocation scenarios for each cluster. It systematically explores different team sizes, site orders, and robot assignments to construct a set of execution paths that can be evaluated using the cost function C_F. Each scenario consists of a team formation $\mathcal{R}_{s_j} \subseteq \mathcal{R}$ for each site s_j, and a corresponding execution cost based on the size $\rho = |\mathcal{R}_{s_j}|$, accounting for task duration, navigation, and inter-team dependencies. To ensure feasible execution, all team-site assignments must satisfy task constraints such as a minimum of two robots per site.

For a given cluster cl containing sites $\mathcal{S}_{cl}$, AMA-Plan systematically generates all valid robot allocation scenarios. Each robot team $\mathcal{R}_{s_j}$ must satisfy the following constraints::

1. Teams must contain at least two robots to support concurrent operations,
2. All robots must be assigned to at least one site.
3. the total number of robots assigned across sites must not exceed the available robots.

Each team-site allocation $Al(s_j, \mathcal{R}_{s_j})$ is generated by considering all possible teams from N_{Team}, resulting in a combinatorial set of possible assignments per site. Formally, we define the set of all possible robot allocations across a cluster cl as:

$$\mathcal{A}_{cl} = \left\{ Al(\mathcal{S}_{cl}, \mathcal{R}_{cl}) \mid \forall s_j \in \mathcal{S}_{cl}, \mathcal{R}_{s_j} \subseteq \mathcal{R}, |\mathcal{R}_{s_j}| \geq 2 \right\} \tag{4.18}$$

where $\mathcal{A}_{cl}$ represents the set of all valid allocation scenarios within cluster cl, ensuring that each site s_j is assigned a valid robot team $\mathcal{R}_{s_j}$.

Since the execution order of sites within a cluster impacts navigation efficiency, AMA-Plan leverages the selected site ordering and zigzag allocation strategy introduced in Section 4.2.1. The zigzag allocation strategy ensures that robots do not fully commit to a single linear sequence but instead distribute themselves across different sections of the execution path, improving load balancing and reducing bottlenecks.

Once all feasible allocations are generated, each scenario is evaluated using the cost function C_F, which integrates execution duration, travel time, and synchronization overhead. The optimal scenario is

selected based on a min–max makespan heuristic, ensuring that mission execution remains balanced.

Algorithm: Generating and Evaluating Allocation Scenarios

The following algorithm outlines the structured approach to generate and evaluate allocation scenarios for a given cluster cl:

Algorithm 2 AMA-Plan Allocation Scenario Generation and Evaluation

Input: Cluster cl with sites $\mathcal{S}_{cl}$, robots $\mathcal{R}$, possible team sizes N_{Team}, cost function C_F
Output: Optimal allocation $Al^*(\mathcal{S}_{cl}, \mathcal{R}_{cl})$ minimizing mission makespan
Initialize $\mathcal{A}_{cl} \leftarrow \emptyset$
for $s_j \in \mathcal{S}_{cl}$ **do**
    for $\rho \in N_{\text{Team}}$ **do**
        Generate all $\mathcal{R}_{s_j} \subseteq \mathcal{R}$ with $|\mathcal{R}_{s_j}| = \rho$
        Add $Al(s_j, \mathcal{R}_{s_j})$ to $\mathcal{A}_{cl}$
for $Al(\mathcal{S}_{cl}, \mathcal{R}_{cl}) \in \mathcal{A}_{cl}$ **do**
    Compute total cost using:

$$C_F(\mathcal{S}_{cl}, \mathcal{R}_{cl}) = \max \left(C_F(s_j, \mathcal{R}_{s_j}) \right)_{s_j \in \mathcal{S}_{cl}, \mathcal{R}_{s_j} \in \mathcal{R}_{cl}}$$

return $Al^*(\mathcal{S}_{cl}, \mathcal{R}_{cl}) = \arg\min_{Al} C_F(\mathcal{S}_{cl}, \mathcal{R}_{cl})$

where:

- $C_F(s_j, \mathcal{R}_{s_j})$ is computed using Equation 4.14.
- The cost is propagated from prior site assignments for all assigned robots, ensuring dependency-aware sequencing.

Algorithm 2 systematically explores all feasible team-site assignments while enforcing execution constraints. By evaluating each scenario based on its execution cost, AMA-Plan ensures that local robot allocations are optimized for efficiency within each cluster.

4.2.2.3 Path Assignment and Sequencing

After generating and evaluating all potential assignments, the final step is to determine the optimal allocation scenario across all clusters. This process selects the configuration that minimizes the mission makespan while ensuring a balanced distribution of workload. Each scenario consists of a structured execution plan, where robot teams $\mathcal{R}_{s_j} \subseteq \mathcal{R}$ are assigned to sites $s_j \in \mathcal{S}_{cl}$, and costs are evaluated according to team size $\rho = |\mathcal{R}_{s_j}|$.In the final restructuring step, the

system selects the most cost-efficient allocation scenario and assigns robots to concrete execution paths accordingly. Figure 4.6 illustrates a typical representation of the resulting robot path assignments in our illustrative scenario.

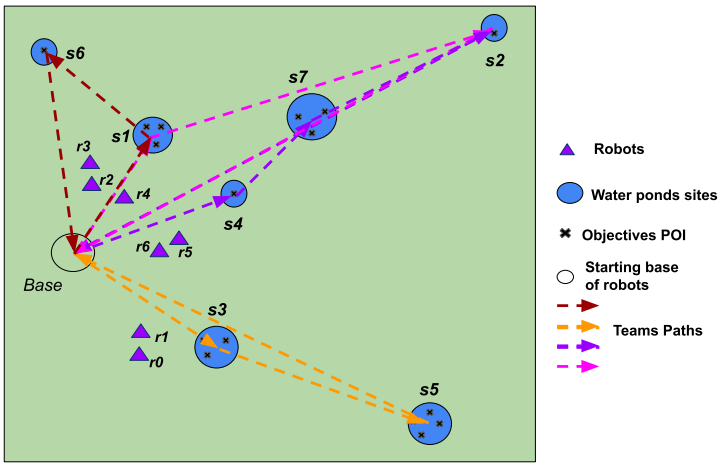


Figure 4.6: Path assignment and sequencing (4.2.2.3). Robots are assigned to sequential site visits (paths), starting from the base and traversing through assigned sites according to the optimal allocation scenario. These structured paths serve as input for the $\text{STN}_{\text{teams}}$ construction.

The selection follows a hierarchical decision-making process:

1. Compute execution costs for all site assignments within a cluster.
2. Identify the best team-site allocation that minimizes the execution cost.
3. Evaluate the best assignments across all clusters.
4. Select the scenario with the lowest mission makespan, ensuring minimal delays and load balancing.

To formalize this selection process, we define the final optimization step as follows:

$$Al(Cl_{set}, \mathcal{R}) = \arg \min_{cl \in Cl_{set}} \max_{s_j \in \mathcal{S}_{cl}} C_F(s_j, \mathcal{R}_{s_j}) \tag{4.19}$$

This ensures that the final allocation scenario minimizes the overall mission duration by selecting the best-performing robot-site assignments from all evaluated scenarios, noted $Al^*(Cl_{set}, \mathcal{R})$.

Algorithm: Optimal Pre-Allocation of Teams Across Clusters

The final global pre-allocation is obtained by merging the optimal allocation noted Al^*_{Cl} from each cluster.

Algorithm 3 Optimal Pre-Allocation of Robot Teams Across Clusters

Input: Clusters Cl_{set} with site sets $\{\mathcal{S}_{cl}\}$, robots $\mathcal{R}$, team sizes N_{Team}, site-cost function $C_F(\cdot)$
Output: Global pre-allocation $Al^*(Cl_{set}, \mathcal{R})$ minimizing makespan
for $cl \in Cl_{set}$ **do**
    Build candidate allocations $\mathcal{A}_{cl} = \{Al(\mathcal{S}_{cl}, \mathcal{R}_{cl}) \mid \forall s_j \in \mathcal{S}_{cl}, \mathcal{R}_{s_j} \subseteq \mathcal{R}, |\mathcal{R}_{s_j}| \in N_{\text{Team}}\}$
    for $Al(\mathcal{S}_{cl}, \mathcal{R}_{cl}) \in \mathcal{A}_{cl}$ **do**
        Compute cluster cost:

$$C_F(\mathcal{S}_{cl}, \mathcal{R}_{cl}) \leftarrow \max_{s_j \in \mathcal{S}_{cl}} C_F(s_j, \mathcal{R}_{s_j})$$

    $Al^*_{cl} \leftarrow \arg\min_{Al \in \mathcal{A}_{cl}} C_F(\mathcal{S}_{cl}, \mathcal{R}_{cl})$
return $Al^*(Cl_{set}, \mathcal{R}) = \bigcup_{cl \in Cl_{set}} Al^*_{cl}$, with makespan $\max_{cl \in Cl_{set}} C_F(\mathcal{S}_{cl}, \mathcal{R}^*_{cl})$

After evaluating all possible robot-team allocations and execution sequences, AMA-Plan outputs a structured mission plan that ensures efficient and balanced execution. The resulting plan consists of near-optimal execution paths, where robots are assigned to clusters and sites in a way that minimizes makespan and prevents team overloading. The pre-allocation strategy distributes workloads evenly and optimizes execution time using a greedy min–max heuristic. Additionally, AMA-Plan enables flexible resource reassignment across clusters, improving utilization. This structured pre-allocation phase produces a scalable and execution-ready plan.

The next section detail how these execution paths are transformed into structured mission plans and team-based representation.

4.2.3 Path Sequential Solving in Teams

We now dive into how the optimal robot assignment $Al^*(Cl_{set}, \mathcal{R})$ is transformed into structured execution teams graph: $\text{STN}_{\text{teams}}$, giving the execution ready structure of the plan.

4.2.3.1 Teams Formation and Sequential Execution Links

Given a mission scenario with a set of robots $\mathcal{R} = \{r_1, ..., r_n\}$, a set of sites $\mathcal{S} = \{s_1, ..., s_k\}$, and the optimal robot-to-site allocations $Al^*(Cl_{set}, \mathcal{R})$ from Section 4.2.2, where each $Al(s_j, \mathcal{R}_{s_j})$ specifies the

robots assigned to a given site, the next step is to transform these allocations into execution teams $\mathcal{T}$ and structured paths that respect temporal dependencies.

To achieve this, AMA-Plan constructs execution paths by:

1. Grouping sites with same assigned robots into teams: Each assigned team follows a path that optimizes its execution order while respecting its assigned sites.
2. Establishing sequential dependencies: Teams that share robots or require inter-team coordination introduce precedence constraints.
3. Structuring execution into an STN: The final execution plan is formalized as a temporal network that schedules task execution while preserving concurrency constraints.

By structuring the mission into execution teams before runtime, AMA-Plan ensures that the problem is decomposed into subproblems, allowing for efficient scheduling and execution adaptation.

Once execution teams are formed, AMA-Plan encodes their relationships within a $\text{STN}_{\text{teams}}$. This $\text{STN}_{\text{teams}}$ represents execution teams as nodes and sequential dependencies as directed edges, ensuring that:

- Independent execution teams operate in parallel.
- Shared robot dependencies enforce sequential constraints.
- Synchronization nodes regulate execution timing when necessary.

Formally, let $\mathcal{T} = \{\text{team}_1, \ldots, \text{team}_k\}$ be the set of execution teams. Each path $team_i$ consists of a sequence of sites assigned to a team of robots. The $\text{STN}_{\text{teams}}$ is defined as a directed graph $\mathcal{S} = (V, E)$ where:

- V contains nodes representing execution teams $\mathcal{T}$.
- E contains directed edges that enforce execution order constraints. $\{(\text{team}_i, \text{team}_j) \mid \text{team}_i \prec \text{team}_j\}$

Each team $team_i$ has a start time $T_{\text{start}}(team_i)$ and an end time $T_{\text{end}}(team_i)$, subject to:

$$T_{\text{start}}(team_j) \geq T_{\text{end}}(team_i), \quad \forall (team_i, team_j) \in E.$$

Written as:

$\text{team}_i \prec \text{team}_j$ denotes a symbolic precedence constraint induced by various dependencies like shared robot participation and temporal constraints.

This ensures that robots operating across multiple teams execute their tasks in the correct order.

4.2.3.2 Execution Strategy and Synchronization Handling

During execution, teams move asynchronously, with task durations varying according to site complexity and navigation distances. When robots must coordinate at a shared site, asynchronous arrivals can cause idle periods. To reduce such inefficiencies, AMA-Plan introduces synchronization nodes, ensuring that robots align their arrival times at shared sites whenever this reduces overall waiting time.

In order to address execution timing mismatches, AMA-Plan introduces synchronization nodes in the mission execution plan. The synchronization strategy falls into two possible categories:

- No synchronization: Robots proceed independently without modifying their schedules, waiting if necessary at shared sites.
- Adaptive synchronization: One or more robots adjust their execution timing dynamically to minimize overall idle time. The integration of a transition phase at the future team's location immediately following the conclusion of the current team's activities serves to mitigate periods of inactivity preceding the initiation of subsequent inter-team collaborations.
- Dividing teams: The implementation of a team with a more comprehensive plan would inevitably lead to the emergence of significant delays during the process of transforming the PDDL plan into BT. To circumvent this issue, a limitation is imposed on the number of actions and the duration of the plan. Upon reaching this limit, AMA-PLAN divides the team into subteams. Each sub-team comprises the same robots team and the same sites to solve. These sites are solved sequentially, linked in several instances.

Algorithmic Implementation:

This approach ensures that synchronization is applied only when beneficial, preventing unnecessary delays while maintaining execution efficiency.

For each execution path $team_i$, AMA-Plan follows these steps:

1. Compute the **earliest arrival time** for all robots assigned to the path.

2. Evaluate **different synchronization scenarios** to determine the one that minimizes idle time.

3. Dynamically modify the execution structure by **introducing synchronization nodes** when necessary.

Each sync node represents an enforced waiting period, ensuring that dependent tasks begin simultaneously and execution inefficiencies are minimized. AMA-Plan continuously monitors execution progress and dynamically adapts synchronization points.

4.2.3.3 Synchronization Algorithm for Multi-Robot Execution

In cases where robots traverse teams with shared execution constraints, AMA-Plan dynamically introduces synchronization nodes to regulate execution timing. The algorithm follows these steps:

Algorithm 4 Insert Synchronization Nodes for Coordinated Execution

Input: Execution teams $\mathcal{T} = \{\text{team}_1, \ldots, \text{team}_k\}$, $\text{STN}_{\text{teams}}$ graph $\mathcal{S} = (V, E)$, travel times T_{travel}
Output: Augmented $\text{STN}_{\text{teams}}$ $\mathcal{S}'$ with synchronization nodes
for all teams $team_i \in \mathcal{T}$ with multiple predecessors **do**
    $R_i \leftarrow$ assigned robots to $team_i$
    $T_{\text{arr}} \leftarrow$ estimated arrival times for each $r \in R_i$
    $T_{\text{idle}}^{\text{base}} \leftarrow \max T_{\text{arr}} - \min T_{\text{arr}}$
    $T_{\text{best}} \leftarrow T_{\text{idle}}^{\text{base}}$
    $C_{\text{best}} \leftarrow \emptyset$
    for all subsets $C \subseteq \text{Predecessors}(team_i)$ **do**
        $T_{\text{arr}}^{C} \leftarrow$ recomputed arrival times with sync delay from C
        $\Delta T \leftarrow \max T_{\text{arr}}^{C} - \min T_{\text{arr}}^{C}$
        if $\Delta T < T_{\text{best}}$ **then**
            $T_{\text{best}} \leftarrow \Delta T$
            $C_{\text{best}} \leftarrow C$
    if $C_{\text{best}} \neq \emptyset$ **then**
        for all $team_j \in C_{\text{best}}$ **do**
            Create sync node $sync_{i,j}$ for shared robots
            $V \leftarrow V \cup \{sync_{i,j}\}$
            $E \leftarrow (E \setminus \{(team_j, team_i)\}) \cup \{(team_j, sync_{i,j}), (sync_{i,j}, team_i)\}$
return Augmented $\text{STN}_{\text{teams}}$ $\mathcal{S}' = (V, E)$

Unlike the cost terms defined in Section 4.2.2, which evaluate allocation scenarios, the $\text{STN}_{\text{teams}}$ encodes the temporal resolution of already-selected team plans. It includes additional vertices for synchronization nodes (Section 4.2.3.2) and edges weighted by navi-

gation or waiting time, reflecting inter-team temporal dependencies rather than allocation costs.

4.2.3.4 Final Execution Model

Let's now review the whole structure from getting the robots path of sites allocation to making a logical and constrained execution-ready structure for the subproblems plans.

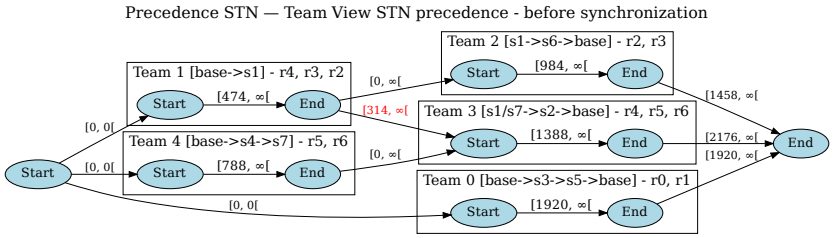
1. Team Abstraction and Canonical sequential link Generation Robots are first grouped into teams based on identical site assignments. However, now each team retains sequential connections to its predecessors and successors, and is annotated with its robots, assigned sites, initial state and final state.

2. Temporal $\text{STN}_{\text{teams}}$ Construction over Paths. The high-level $\text{STN}_{\text{teams}}$ is built over these sequential connections. Each node represents a symbolic plan assigned to a team, and edges encode precedence constraints due to shared robot participation or resource dependencies. Let $\mathcal{T} = \{\text{team}_1, \ldots, \text{team}_k\}$ be the set of symbolic team plans. The $\text{STN}_{\text{teams}}$ $\mathcal{S} = (V, E)$ is defined as seen previously

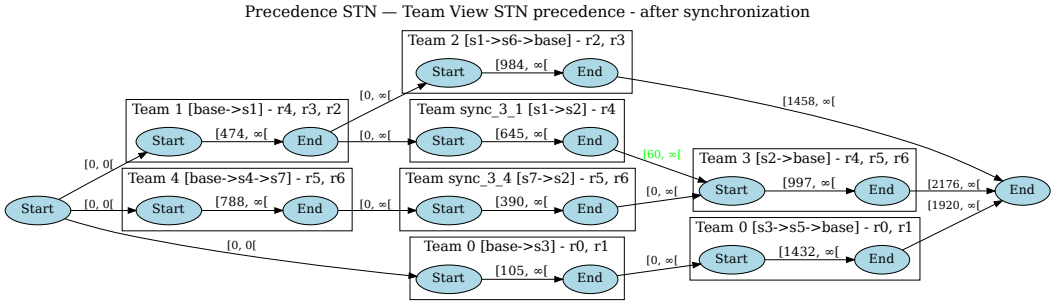
3. Synchronization Node Insertion. When a team has multiple predecessors and requires robot coordination at the first site, synchronization nodes are evaluated. For each team $team_i$, the system:

1. Computes earliest arrival times for all robots from predecessor teams.
2. Evaluates all subsets of predecessors that could participate in a sync.
3. Computes idle time for each sync scenario and compares it to a no-sync baseline.
4. Inserts one or more synchronization nodes if idle time is reduced.

This mechanism enforces robot arrival alignment only when beneficial. The process is summarized in Algorithm 4, and Figure 4.7 compares the pre-synchronization STNteams graph $\mathcal{S}$ with the execution-ready STNteams graph $\mathcal{S}'$. In this example following our illustrative scenario of Figure 1.2 in page 6, synchronization between `Team 1` and `Team 4` toward `Team 3` reduces the overall idle gap from $314s$ to $60s$, yielding a net saving of $254s$ in mission time preserving temporal feasibility.



(a) Initial $\text{STN}_{\text{teams}}$ structure, unsolved and with no synchronization adjustment.



(b) Improved $\text{STN}_{\text{teams}}$ with synchronized path nodes and durations from PDDL plans.

Figure 4.7: Comparison of the $\text{STN}_{\text{teams}}$ structures before and after synchronization optimization. The enhanced version includes execution intervals for each path and introduces synchronization nodes with refined temporal links.

4. Execution-Time Estimation and Path Splitting. Symbolic execution teams are composed of sequences of sites and are incrementally transformed into a chain of subproblems. Each subproblem corresponds to a site (or set of sites) and is generated using the robots's updated states resulting from their previous subproblem's plans, ensuring continuity of execution context across the $\text{STN}_{\text{teams}}$. These subproblems are solved using a PDDL planner, and the resulting plans are annotated with their estimated durations. If the cumulative duration or action count of a plan exceeds a predefined threshold (e.g., temporal or certain number of actions), the path is automatically split into multiple synchronized subteams covering all sites in the team's allocation. This strategy enhances execution tractability and isolates failures to localized portions of the mission plan. An example of problem and plans from team_0 of $\text{STN}_{\text{teams}}$ in Figure 4.7a illustrative scenario are available in appendix A.2

5. Subproblem Generation and Validation Structure. Each execution team is converted into a directory of symbolic subproblems, domain

files, plans, and merged solutions. These files are used for both execution and post-failure revalidation. The planning sequence ensures continuity of robot states across subproblems.

6. Final $\text{STN}_{\text{teams}}$ Graph and Execution-Ready Output. The resulting $\text{STN}_{\text{teams}}$ preserves the standard STN formalism of timepoints and temporal constraints, but organizes them according to the team-based decomposition:

- **Nodes:** Timepoints for the start and end of each team's plan, plus synchronization nodes when required.
- **Edges:** Temporal constraints and sequential dependencies with minimum gaps, ensuring precedence relations are respected.
- **Metadata:** Estimated execution times, robot participants, site locations, and allocation details.

This enriched $\text{STN}_{\text{teams}}$ structure forms the backbone of the execution schedule. It is provided to the execution system AMA-Exec, which monitors timing, handles drift, and enforces synchronization and dependency constraints at runtime. Plans are now considered *execution-ready*: they are temporally annotated, robot states remain consistent across subproblems, and they can be directly handed off to the execution layer without additional processing.

4.2.4 Overall Complexity of AMA-PLAN

Keeping only dominant terms across stages, the overall complexity is

$$T_{\text{AMA-PLAN}} = \mathcal{O}\big(|\mathcal{S}|^2\, 2^{|\mathcal{S}|/2}\big) + \mathcal{O}\big(|\mathcal{S}|\,|\mathcal{R}|^2\, \mathcal{B}^{\,|\mathcal{S}|/2}\big) + \sum_i T_{\text{OPTIC}}(L_i). \tag{4.20}$$

where $\mathcal{S}$ denotes the set of non-base sites, $\mathcal{R}$ the set of robots (with cardinalities $|\mathcal{S}|$ and $|\mathcal{R}|$), $\mathcal{B}$ is the effective post-pruning per-site branching factor induced by Stage 2 (*Allocation resolution cost estimation*)—i.e., the size of the small retained set of near-optimal coalitions per site (data-dependent; worst case equals the unpruned option count), and L_i the size of early-split subproblems for OPTIC. The first term comes from exact intra-cluster TSP ordering during spatio–cost clustering; the second from pruned cross-site composition with greedy cost estimation ; the last term is the sum of OPTIC solver times over bounded subproblems produced by $\text{STN}_{\text{teams}}$

synchronization and early-split. All remaining components (fixed-iteration k-means, weight construction, greedy picks, STN updates) are lower-order in our implementation.

4.2.4.1 Scope and Limitations

Because Stage 1 uses weighted Euclidean k-means and Stage 2 employs greedy allocation over a pruned band, AMA-PLAN is heuristic: elongated/anisotropic layouts, severe per-site workload imbalance or interleaved clusters can degrade solution quality or enlarge $\mathcal{B}$. We mitigate these effects with travel-time weighting and multiple restarts (Stage 1), band-pruning (Stage 2, controlling $\mathcal{B}$), and early-split caps in $\mathrm{STN}_{\mathrm{teams}}$.

4.2.4.2 Bottleneck Implication

As $|\mathcal{S}|$ grows, the scenario-construction/assignment term $\mathcal{O}(|\mathcal{S}|\,|\mathcal{R}|^2\,\mathcal{B}^{|\mathcal{S}|/2})$ becomes dominant; increasing $|\mathcal{R}|$ at fixed $|\mathcal{S}|$ primarily inflates the $|\mathcal{R}|^2$ factor, while STN consistency checks remain lightweight.

4.2.4.3 Empirical Validation

Experimental results (chapter 6) confirm these theoretical predictions:

- **Moderate scales** ($|\mathcal{S}| \leq 12$, $|\mathcal{R}| \leq 8$): Planning completes in 5-30s, with symbolic solving (the $\sum_i T_{\mathrm{OPTIC}}(L_i)$ term) consuming 70-80% of total time.
- **Larger scales** ($|\mathcal{S}| > 15$): The scenario construction term becomes dominant, leading to timeouts (> 300s) especially in trans-media scenarios where $\mathcal{B}$ increases due to mode-switching complexity.
- **Comparison with monolithic solving**: AMA-PLAN achieves 11× speedup over direct OPTIC calls on full problems (Table 4.1), validating the decomposition benefit despite introducing coordination overhead.

The next section presents example of illustratives scenarios and comparison with classical solvers.

4.2.5 Comparison with Classical Solver and Illustrative Scenarios

This section evaluates AMA-Plan against traditional PDDL solvers on easy-to-solve problems and provides results and visualization of the AMA-Plan output with robot and site allocation during a mission.

In preparation for this comparison, we experimented with several temporal–numeric planners, including those already integrated in the Unified-Planning (UP) framework (Micheli et al., 2025) (e.g., Tamer (Valentini et al., 2020), LPG (Gerevini and Serina, 2002), and Aries (Bit-Monnot, 2023) in its current version), as well as widely used external solvers such as OPTIC (Benton et al., 2012), POPF (Coles et al., 2010), and TFD (Röger et al., 2008). Empirically, OPTIC demonstrated the best balance between expressiveness (e.g., support for *:timed-initial-literals* in PDDL) and computational stability across our representative mission scenarios. It consistently solved the largest number of problems involving complex temporal dependencies and metric costs, whereas other solvers either failed to parse extended durative constructs or exhibited significant runtime variance. In addition to its robust handling of temporal–numeric constraints, OPTIC's hybrid heuristic search strategy, temporal task decomposition, and flexible constraint propagation made it particularly well suited to the structured, concurrent, multi-goal nature of AMA-Plan's subproblems. The following results are therefore reported using OPTIC as the reference solver.

4.2.5.1 Context and Comparison

We report results established on averages defined by 20 different scenarios, each with 15 randomly generated instances. The mission area is $1\,\mathrm{km}^2$, and contains randomly placed water sites of size varying from 100 to $2500\,\mathrm{m}^2$, the number of sampling and transition points for sites being directly related to their surface. Table 4.1 compares results for scenarios with 4 robots, and a number of sites varying from 1 to 18 sites, for the monolithic planning OPTIC approach (MP) and AMA-plan. Both approaches are stopped when either a 360 s timeout or a 32GB memory allocation is attained. OPTIC can only solve the smaller problems, rapidly reaching the maximum time or memory allowed, whereas our approach solves the most complex problems in a few seconds.

Given that OPTIC does not scale with problem complexity, we can not statistically assess to what extent our approach loses solution optimality (mission plan makespan). However, on a small scale

Metric	MP	AMA-Plan	Number of sites
Elapsed Time (s)	0.08	0.03	1
	0.2	0.05	2
	1.15	0.13	3
	Timeout	0.17	4
	...	...	...
	Timeout	7.16	18
Coverage of found solutions in %	95	100	1
	35	100	2
	10	100	3
	0	100	4
	...	...	...
	0	100	18

Table 4.1: Elapsed time and solution percentages over 20 scenarios with 4 robots, varying site numbers (Intel E5-2695 v3, 2.3GHz 32GB RAM), monolithic planning with OPTIC (MP) and AMA-Plan

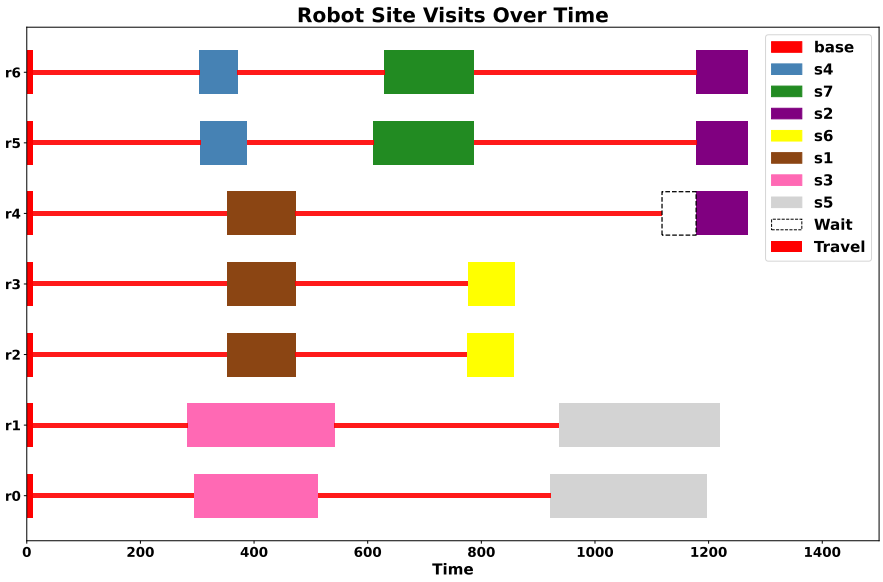
scenario (4 sites from 100 to 200m^2 on a 25000m^2 terrain), the first solution found by OPTIC in 40.6 s has a makespan of 2530 s, and the optimal solution (makespan of 2391 s) is found in 48 s, whereas our approach finds a solution in 0.11 s, with a plan makespan of 2425 s. The loss of optimality is negligible with respect to the reduction of time to find a solution.

4.2.5.2 Trans-media Illustrative Scenario

To illustrate the outcome of the AMA-Plan allocation and resolution process, we use the aforementioned mission scenario involving 7 robots and 7 sites, representative of our typical multi-robot deployments. This scenario showcases how a symbolic planning problem is decomposed into team-based subproblems with assigned paths of sites and scheduled resolution.

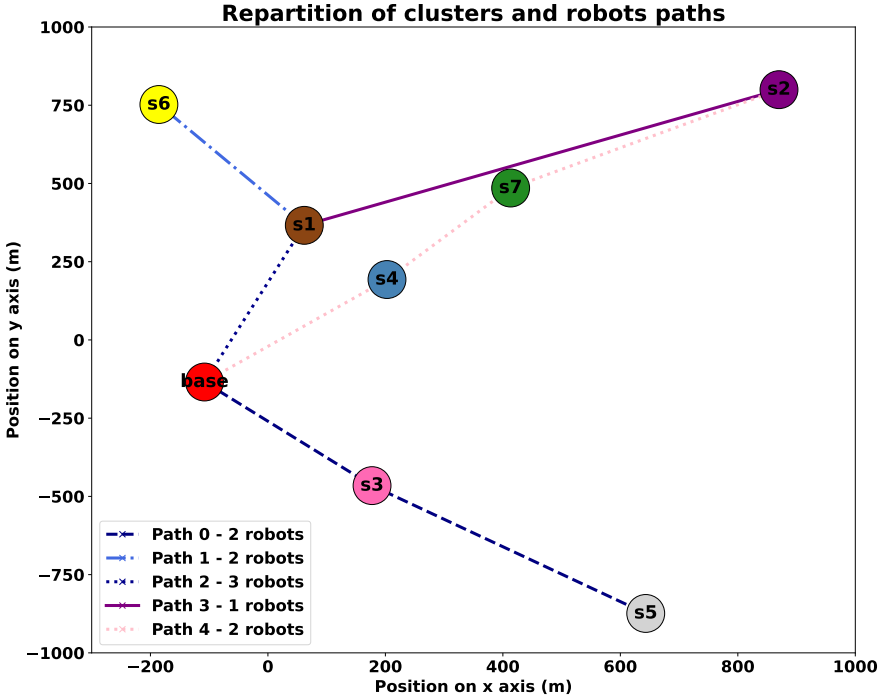
As illustrated in Figure 4.8, the AMA-Plan output is presented in three dimensions. First, the execution timeline in Figure 4.8a reflects the sequencing of site visits per robot, including planned idle durations that arise when a robot temporarily switches teams, for example, to support complex operations at another site.

In Figure 4.8b, the geographical proximity of sites is reflected in the formation of teams responsible for visiting these locations. Robots allocated to a common path traverse a shared trajectory across the sites within a given cluster, a path being determined by both spatial proximity, task requirements, and the cost estimation



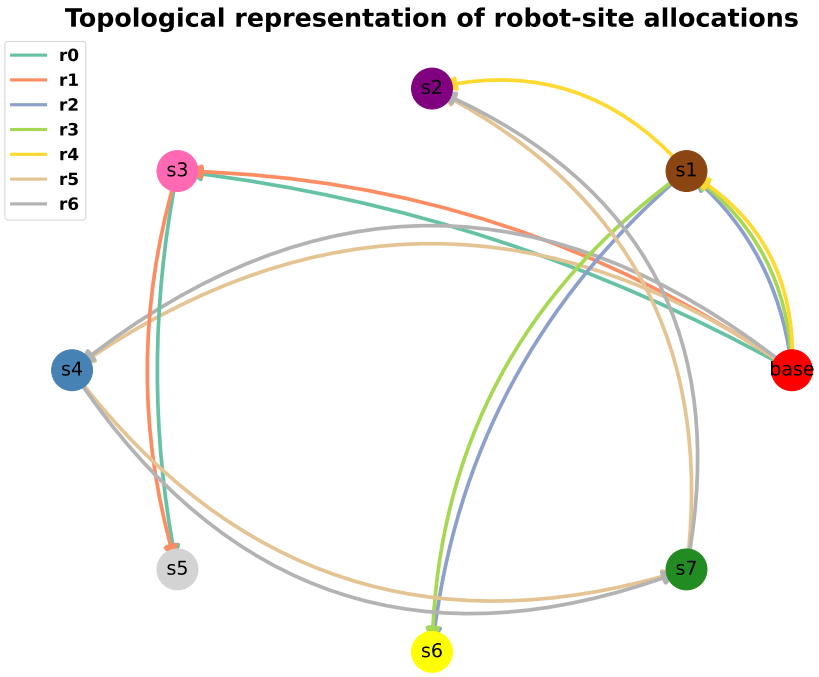
(a) Execution timeline showing site visits and wait durations due to team switches.

Figure 4.8: Subfigure a): AMA-Plan decomposition and allocation result for a mission involving 7 robots and 7 sites including the base.



(b) Cluster decomposition and robot paths, grouped by teams.

Figure 4.8: Subfigure b): AMA-Plan decomposition and allocation result (continued).



(c) Topological dependency graph of robot–site visits.

Figure 4.8: Subfigure c):AMA-Plan decomposition and allocation result (continued).

of each the sites. Finally, Figure 4.8c presents a topological graph of robot-to-site transitions. It help emphasizes the shared trajectories of robots and the dynamic reallocation of robot 4 with color-coded transitions by robots.

The collective analysis of these figures underscores the adaptability and organizational framework inherent within the AMA-Plan planning layer. Robots can dynamically change teams across clusters based on role requirements and action cost estimation, leading to improved resource utilization and mission efficiency. This modular abstraction is essential for the downstream integration of temporal reasoning, thereby enabling the STN-based execution engine to supervise synchronization and address delays without compromising the integrity of the mission plan.

The resulting output comprises a series of temporally annotated, team-specific symbolic subplans, accompanied by interdependencies. This modular representation functions as a direct input to the execution system presented in chapter 5, thereby facilitating scalable and robust plan execution across distributed trans-media robot teams.

Despite its effectiveness in producing feasible subplans, AMA-Plan in its first version lacked a formal mechanism for managing global temporal dependencies or synchronization constraints between distributed teams. The symbolic paths are resolved sequentially (see Figure 4.3), and while they are valid locally, external integration is required to ensure consistency across teams, especially in missions requiring robots to coordinate across clusters or operate jointly on shared sites.

4.3 Conclusion

The AMA-Plan framework is designed to address the inherent complexity of multi-robot, trans-media mission planning through a modular, hierarchical approach. The AMA-Plan methodology integrates cost-aware clustering, greedy yet constraint-respecting team assignment, and STN-based sequencing. This integrated approach produces execution-ready mission plans that are both scalable and adaptable. Its structured decomposition has been shown to dramatically reduce the search space of solvers. This, in turn, enables solutions for scenarios in which classical PDDL planners are unable to scale. Moreover, the method maintains near-optimal makespan performance in solvable cases. The modularity of the AMA-Plan has been demonstrated to enhance computational efficiency and facilitate partial replanning and integration with runtime supervision layers, such as AMA-Exec. The delineation of responsibilities be-

tween global allocation and local temporal adaptation renders AMA-Plan a pragmatic foundation for complex, distributed robotic operations. Quantitative performance studies, including large-scale statistical evaluations and failure-recovery benchmarks, are presented in Chapter 6, further demonstrating the benefits and trade-offs of the proposed planning methodology.

Chapter 5

AMA-Exec: Adaptive Modular Architecture for Distributed Execution

Contents

This chapter presents an overview of AMA-Exec, a modular distributed execution framework designed to address the discrepancy between high-level mission planning and real-time execution. While AMA-Plan structures complex multi-robot missions into execution-ready subproblems, AMA-Exec ensures that these plans are carried out efficiently in dynamic, uncertain environments. We present the architecture of the AMA-Exec system, delving into its internal execution pipeline. Conclusively, we offer illustrative scenarios that demonstrate the system's fundamental mechanisms.

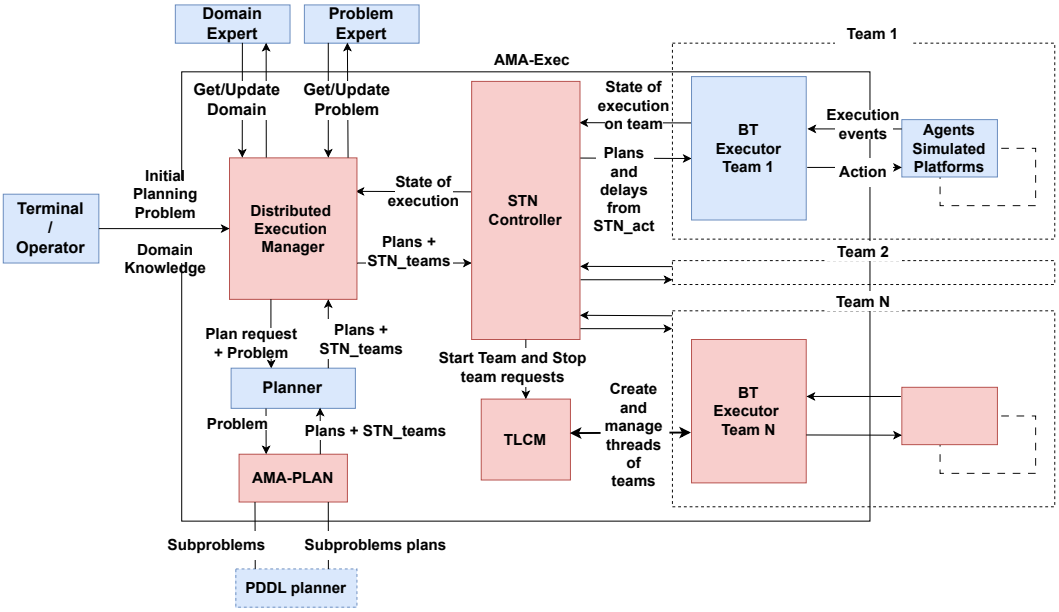


Figure 5.1: High-level overview of AMA-Exec execution components (in red) and PlanSys2 original components (in blue).

AMA-Plan → AMA-Exec Building upon the temporally annotated and team-synchronized execution-ready plans produced by AMA-Plan, this chapter focuses on their distributed and failure-resilient execution with temporal monitoring within AMA-Exec.

5.1 AMA-Exec System Architecture Overview

This section presents the architectural foundations of AMA-Exec. It provides a high-level view of the system's modular components, their roles, and the overall mission execution pipeline. While PlanSys2 provides a solid baseline, and the previous section highlighted open challenges, AMA-Exec was designed to explicitly address distributed execution with non-deterministic actions (delays, non-boolean failure effects, etc.). In this section, we present its architecture and justify the key design principles guiding its development. The section emphasizes modularity, scalability, and temporal robustness in the context of distributed trans-media robotic missions.

5.1.1 Motivation and Architecture Overview

In Figure 5.1 we present an overview of the AMA-Exec framework, incorporating the original PlanSys2 architecture components (in blue

boxes) with the extensions presented in this paper (in light red). Although PlanSys2 offers PDDL-based planning and BT-based execution, it lacks support for temporal supervision, distributed team management, and adaptive failure recovery presented in AMA-Exec.

The **AMA-Exec** execution framework is grounded in symbolic plans produced by AMA-Plan presented in Chapter 4. The symbolic plans produced by AMA-Plan are considered *execution-ready* as they are temporally annotated, robot-state consistent across subproblems, and embedded in a structured two-layer temporal model. This includes a team-level STN ($\text{STN}_{\text{teams}}$), which encodes inter-team precedence and synchronization constraints, and individual action-level STNs (STN_{act}) for each robot team, which capture intra-team PDDL plan temporal dependencies. These plans result from spatio-temporal clustering, task allocation, and mission decomposition performed by AMA-Plan. Once generated, AMA-Exec ensures that these high-level plans are executed in real time, even in the case of temporal drift, action delays, or failures.

To address some of the limitations of PlanSys2, we developped AMA-Exec with additional modules, each responsible for a specific layer in the pipeline. Each component manages its responsibilities independently, from symbolic coordination to reactive control, with ROS 2 ensuring safe concurrency, threading and isolation across teams. AMA-Exec is designed to support the robust execution of complex multi-robot missions.

5.1.2 Modular Component Responsibilities

The AMA-Exec framework is designed around five core modules responsible for different layers of the mission execution and monitoring:

- **AMA Plan:** Presented in Chapter 4 this restructuring toolbox decomposes the mission into symbolic subproblems, generates a $\text{STN}_{\text{teams}}$ precedence graph, and solve each subproblem plan through a chosen PDDL solver, creating a manageable structure of distributed execution plans.
- ***STNController***: This component enforces temporal constraints between and within teams, by monitoring action-level execution and injecting delays when necessary through the proper STN_{act}. It ensures that dependent actions (e.g., concurrent actions, etc.) are executed/finished in the correct order and with appropriate synchronization margins. It also determines when new teams can be launched based on progression of each team in the $\text{STN}_{\text{teams}}$.

Module	Role	Inputs	Outputs
AMA-Plan	Mission problem decomposition and symbolic plan generation	World data, domain, problem	Team-wise symbolic plans + $\text{STN}_{\text{teams}}$
STNController	Temporal supervision, execution control	Team plans, $\text{STN}_{\text{teams}}$, execution feedback	Start requests, delays, temporal adjustments and controls
DEM	Parsing, \textit{STNController} \init, failure analysis via VAL	AMA-Plan plans, VAL feedback	Parsed team plans, STN requests, replanning triggers
TLCM	Team lifecycle control: robot/team node spawning	Start/stop requests from \textit{STNController}	BT Executor nodes, simulated robot threads
BT Executors	Team execution Action-level control	Team-assigned symbolic plan	Action status, feedback topics, mission completion flags

Table 5.1: AMA-Exec modules, roles, inputs, and outputs.

- **Distributed Execution Manager (DEM):** Acting as the mission-level controller, the DEM receives symbolic plans and $\text{STN}_{\text{teams}}$ from AMA-Plan, parses the result into executable structures, starts and monitors *STNController* and its execution feedback, and handles partial replanning or recovery in case of failure.
- **Team Lifecycle Manager (TLCM):** The TLCM is responsible for the lifecycle management of team execution, their BT Executors and simulated robots. It handles the launch, supervision, and termination of team's environment, allowing dynamic team creation and removal during runtime.
- **Multi-Executors:** These BT-based execution nodes are derived from PlanSys2 and extended to operate in their own ROS2 namespaces and threads, allowing for parallel execution. Each executor controls the actions of a specific team, reacting to feedback and executing symbolic instructions while remaining decoupled from other teams' execution flows.

This modular organization enables AMA-Exec to run distributed missions with asynchronous team behaviors, localized failure response, and high concurrency while limiting bottlenecks. These modules operate in coordination across planning, temporal control, and distributed execution. Their roles, inputs, and outputs are summarized in Table 5.1, which provides a modular overview of the execution pipeline.

5.1.3 Execution Flow and Module Interactions

This subsection details the information flow between AMA-Exec components, from symbolic mission planning to distributed execution. It reflects the modular startup, sequential dispatching, and coordination process enabled by ROS 2 threading and PlanSys2 integration.

AMA-Exec initializes, monitors, and adapts execution across robot teams using its modular design. The execution flow is as follows:

1. At launch time, core components are brought online:
   - The **PlanSys2 core**, including the domain/problem experts and planner node;
   - The **DEM**, which monitors and orchestrates planning and execution;
   - The **TLCM**, responsible for spawning and managing the team's robot environments.
2. Once the problem and domain definitions are available (either from launch or through terminal requests), the DEM issues a planning request via PlanSys2's standard interface.
3. AMA-Plan module receives this request and performs decomposition of the mission, generating symbolic plans per robot team through the PDDL planner along with a set of sequential dependency links between them.
4. Upon receiving the team plans and their temporal dependencies, the DEM initializes the *STNController* thread. Team plans and dependency links in $\text{STN}_{\text{teams}}$ are passed to the controller, which constructs the STN_{act} for each team, capturing intra-team execution dependencies.
5. From this point, the ***STNController*** supervises execution using the $\text{STN}_{\text{teams}}$ graph:
   - It launches teams as their dependencies are satisfied, coordinating with the TLCM to start BT executors;
   - It publishes symbolic plans to executors and monitors progress through execution feedback;
   - Execution proceeds asynchronously across teams, with the controller ensuring symbolic-temporal constraints are respected.

When a team succeeds, the *STNController* removes its executor and action nodes through a request to TLCM, freeing resources. It then activates the next plan in the $\text{STN}_{\text{teams}}$ whose dependencies are now satisfied. This process repeats until all symbolic plans are completed or a failure prevents continuation.

Throughout execution, the DEM remains active in the background, receiving execution status from the *STNController*. In the

event of a symbolic failure or plan-level inconsistency, the DEM evaluates impact across teams and provides a failure report to the AMA-Plan which will determine which level of replanning is necessary.

This execution flow establishes a tightly coordinated pipeline that connects the symbolic planning and modular execution layers. Each module activates in a defined sequence, allowing multi-robot teams to work together simultaneously while maintaining global synchronization. A summary of these interactions and responsibilities can be found in the final subsection of this section.

5.1.4 Plan Execution Infrastructure: BT Executors and PlanSys2 Integration

At a lower level each robot team executes its symbolic plan using a dedicated BT executor node. AMA-Exec preserves PlanSys2's BT-based execution core while extending its deployment for modular use, specifically:

- **Action namespace scoping:** Each BT executor is operated within a dedicated ROS 2 namespace corresponding to its team and robot's actions, preventing cross-team topic interference;
- **Dynamic team lifecycle management:** Execution environments, including BT Executors, robots and action nodes are instantiated and managed by the TLCM dynamically based on *STNController* requests;
- **Feedback multiplexing:** Each team environment contains various multiplexed feedbacks on its BT execution of plan and state of its robots and actions nodes.

Together, these design choices make AMA-Exec suitable for real-world deployment in multi-agent missions with dynamic environmental constraints, failure modes, and inter-team dependencies. Its structured separation of responsibilities allows:

- Independent initialization, execution, and shutdown of robot teams;
- Flexible replanning in response to failure, based on localized impact;
- Real-time enforcement of temporal constraints without centralized blocking;
- Parallel team execution using without resource contention.

BTs have become a widely adopted framework for reactive plan execution in robotics, due to their modular structure, explicit local success/failure propagation, and ease of composition compared to finite-state machines. They have been increasingly used as an intermediate layer bridging symbolic planning and low-level control, enabling plans to remain reactive to changing execution conditions.

PlanSys2 leverages this paradigm by automatically translating symbolic PDDL plans into BTs and executing them through modular ROS 2 action nodes. Each node encapsulates a symbolic action, its preconditions, and its monitoring logic, allowing PlanSys2 to supervise plan execution reactively. However, its default implementation is centralized and lacks mechanisms for distributed temporal synchronization and inter-team recovery.

AMA-Exec builds directly upon this foundation. It preserves PlanSys2's PDDL $\rightarrow$ BT-based execution core, but extends it for distributed, temporally supervised, and failure-resilient operation. AMA-Exec also preserves PlanSys2 BT-for-action local leaves structure ability, where actions execution can be modelised as leave BT of the plan. Further details on how AMA-Exec ensures temporal alignment, delay injection, and symbolic-temporal consistency enforcement during execution are provided in the next Section 5.2.

5.2 Inside AMA-Exec: Temporal Reasoning and Failure-Resilient Execution

This section presents the modeling, temporal reasoning, and failure-resilient mechanism and algorithms integrated in AMA-Exec. Building on the symbolic plans and synchronized structures produced by AMA-Plan, AMA-Exec enables robust real-time execution in environment with non-deterministic actions. It introduces runtime models that monitor temporal constraints, handle drifts and delays, and detect and analyze failures to identify the impacted parts of the $\text{STN}_{\text{teams}}$ plans.

5.2.1 Execution-Time Challenges in Real-World Deployment

Symbolic plans generated by AMA-Plan are typically temporally annotated and logically coherent. However, they rely on idealized assumptions: perfectly synchronized execution across agents, fixed action durations, and failure-free environments. These assumptions rarely hold in practice, especially in trans-media, multi-robot scenarios that involve complex vehicle capabilities behaviors, medium

transitions, and partially observable conditions.

During execution, agents may experience delays, asynchronous progress, or misaligned temporal dependencies, particularly when they are operating in separate teams. For instance, sampling tasks may begin prematurely before prerequisite data translation actions have completed, leading to invalid behavior. Furthermore, failures such as robot crashes or the inability to reach a site can impact mission feasibility in localized ways, often without invalidating the entire mission.

As discussed before, PlanSys2 traditional BT-based execution engines and ROS2-based planning frameworks do not monitor temporal coherence at runtime, nor do they provide fine-grained control over how failures are diagnosed or mitigated. As a result, minor execution drifts or localized robot failures may lead to over-conservative responses such as full replanning or mission failure.

We designed AMA-Exec to addresses these limitations through a dual-layered execution framework. The *STNController* performs continuous supervision of action timing and dependencies, ensuring that symbolic-temporal constraints are upheld during distributed execution. In parallel, the DEM handles failure detection and recovery, including constraint-aware reallocation and replanning decisions. Together, these modules enable robust and adaptive execution across multi-agent trans-media teams.

5.2.2 Temporal-Aware Distributed Execution: The *STNController*

5.2.2.1 Formal Temporal Model and Dependency Graph Structure

The *STNController* supervises the execution of symbolic plans of the $\text{STN}_{\text{teams}}$ using a multi-layered dependency graph that extends the classical STN. This extended model, referred to as STN_{act}, captures both the temporal structure and semantic dependencies of individual robot actions during execution. This structure is used by the *STNController* to supervise runtime execution, ensure correct action monitoring and enforce synchronization constraints.

Temporal Annotation of STN_{act}

Building on the classical formulation of STN presented in Subsubsection 4.1.2.2 and propose a direct structuration of this formalism with our constraints

A classical STN encodes temporal dependencies using:

- **Nodes:** Timepoints corresponding to the start or end of actions.

- **Edges:** Temporal constraints $(x, y, [\min, \max])$ indicating that the time difference between two timepoints x and y must lie between min and max.

We structure this formalism in our abstraction as follows:
We define a symbolic plan $\mathcal{P}$ as a set of actions $\{a_1, ..., a_n\}$, where each action a_i is annotated with:

- s_i: planned start time,
- d_i: expected duration,
- $e_i = s_i + d_i$: planned end time.

The core temporal structure of STN_{act} is defined as:

$$\mathcal{G}_{\text{act}} = (V, E), \quad V = \{s_i, e_i\}, \quad E = \{(e_i + \varepsilon \leq s_j)\}$$

where ε is a buffer margin to absorb uncertainty or enforce loose synchronization.

To better capture execution semantics and mission logic, we extend this structure into a multi-layered dependency graph derived from the PDDL plan and domain constraints file:

$$\mathcal{G} = (V, E_T, E_C, E_R) \tag{5.1}$$

where:

- E_T: *Sequential (temporal) dependencies* Represents strict ordering between actions executed by the same robot (e.g., navigation must finish before sampling), (this is similar to the leads_to in Allen's logic (Allen, 1981)),
- E_C: *Concurrency exclusions* Links actions across robots that must be executed concurrently or with minimal offset (e.g., `sample` must be contained_by (in Allen logic) a `relay_data` action),
- E_R: *Repeated action equivalences* Denotes semantically equivalent actions (with same parameters) or refer to similar operations, often used to group and track similar sub-tasks across time,

Illustrative Example.

We consider the following sequence from a plan involving two robots visiting a site, where 'robot0' `relay_data` (for relays) and 'robot1' performs `sample`, both actions being constrained such that

a `sample`(a_{spl}) is always contained in another robot's `relay_data` (a_{trd}) action on the same site s:

$$\begin{aligned} &\forall r_u, r_v \in \mathcal{R},\ \forall p_i \in P_{spl} \cap P_s,\ \forall p_j \in P_{tr} \cap P_s, \\ &m_{r_u} = m_{r_v} = \text{aquatic}, \\ &t_{\text{start}}(a_{\text{spl}}(r_u,p_i,s)) \geq t_{\text{start}}(a_{\text{trd}}(r_v,p_j,s)) \\ &\wedge\ t_{\text{end}}(a_{\text{spl}}(r_u,p_i,s)) \leq t_{\text{end}}(a_{\text{trd}}(r_v,p_j,s)) \end{aligned} \tag{5.2}$$

The following action sequence (with symbolic temporal annotations) illustrates our extended STN_{act} dependency structure from the PDDL plan:

- a_1: `(relay_data robot0 sp9 site3 A)` $s_1 = 334$, $d_1 = 45$, $e_1 = 379$
- a_2: `(sample robot1 pp7 site3)` $s_2 = 349$, $d_2 = 30$, $e_2 = 379$
- a_3: `(navigation_water robot1 pp7 pp6 site3)` $s_3 = 379$, $d_3 = 17.9$, $e_3 = 397$
- a_4: `(relay_data robot0 sp9 site3 B)` $s_4 = 382$, $d_4 = 45$, $e_4 = 427$
- a_5: `(sample robot1 pp6 site3)` $s_5 = 397$, $d_5 = 30$, $e_5 = 427$

From this, the dependency graph $\mathcal{G} = (V, E_T, E_C, E_R)$ includes:

- **Sequential dependencies (E_T)**
  - $(a_1, a_4) \in E_T$, robot0 sequence
  - $(a_2, a_3) \in E_T, (a_3, a_5) \in E_T$, robot1 sequence
- **Concurrency constraints (E_C)**
  - $(a_1, a_2) \in E_C$, first concurrent actions
  - $(a_4, a_5) \in E_C$, second concurrent actions
- **Repeated action Constraint (E_R)**
  - $(a_1, a_4) \in E_R$, e.g: same `relay_data` action parameters but with different `sample` action related at a different time. An index representing their order (A $\rightarrow$

Z,) is added to the action automatically when a constraint is added. This can be seen in the example above with `(relay_data robot0 sp9 site3 A)` and `(relay_data robot0 sp9 site3 B)`.

Algorithm 5 Extraction of Multi-Layered Dependencies from a Plan

Input: Symbolic plan $\mathcal{P} = \{a_1, \ldots, a_n\}$ with timing and parameters
Output: $\mathcal{G} = (V, E_T, E_C, E_R)$

$V \leftarrow \mathcal{P}$ ▷ Actions as nodes
$E_T, E_C, E_R \leftarrow \emptyset$
for each pair $(a_i, a_j) \in \mathcal{P}^2$, with $i \neq j$ **do**
  if a_i and a_j share a robot and a_j starts after a_i within tolerance **then**
    $E_T \leftarrow E_T \cup \{(a_i, a_j)\}$ ▷ Temporal order
  else if a_i and a_j operate on same site or PoI and roles are complementary **then**
    $E_C \leftarrow E_C \cup \{(a_i, a_j)\}$ ▷ Coordinated concurrency
  else if a_j is a repeated instance of a_i (e.g., repeated `relay_data` on same site) **then**
    $E_R \leftarrow E_R \cup \{(a_i, a_j)\}$ ▷ Semantic repetition
return $\mathcal{G} = (V, E_T, E_C, E_R)$

Algorithm 5 details the construction of the multi-layered dependency graph $\mathcal{G}$ from the symbolic plan $\mathcal{P}$. It examines inter-action relationships using semantic information, temporal annotations, and shared roles or PoI involvement as shown in Figure 5.2. This structure serves as the basis for execution-time synchronization and enforcement of constraints in the *STNController*.

Although planners typically explore and encode these dependencies during search, our architecture treats planners as interchangeable plugins and does not rely on their internal search state or solver-specific APIs. To remain planner-agnostic and broadly applicable, we therefore reconstruct the necessary dependencies from the produced plan.

This layered structure allows mission controllers to track execution with well established semantics, enabling constraint-based adaptation, delay diagnosis, and real-time synchronization enforcement.

5.2.2.2 Architecture and Execution-Time Monitoring

The ***STNController*** is the centralized supervision node responsible for maintaining the temporal consistency of distributed symbolic plans executed in parallel by robots teams. It interfaces with both the TLCM and per-robot BT executors, and serves five key purposes:

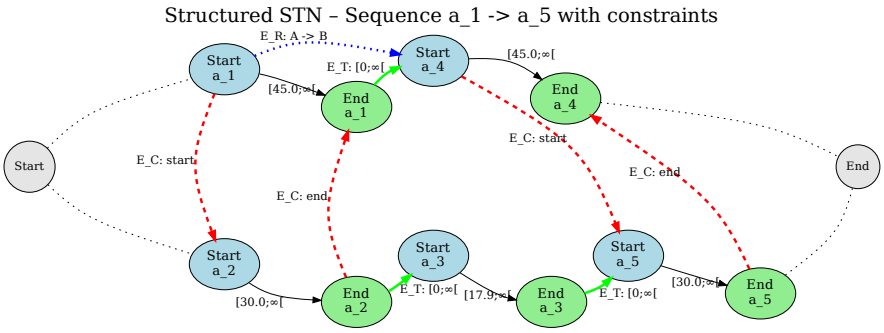


Figure 5.2: Multi-layered STN_{act} dependency graph derived from symbolic plan. Edges represent: Temporal dependencies (green line), concurrency constraints (red dashed line), and redundancy (blue dotted line) constraints are shown between annotated start and end nodes.

- Creating the STN_{act} by building dependency graphs $\mathcal{G}$ from the PDDL plans.
- To maintain a dedicated STN_{act} instance and action tracking table for each active team.
- Continuously monitoring real-time feedback from execution to assess a robot's action and plan progression in its team's STN_{act} .
- Enforcing the symbolic and temporal constraints as introduced in STN_{act}: temporal (E_T), concurrency (E_C), and semantic equivalence (E_R).
- Propagating delay commands to robots and logging failure contexts if violations are detected.

Each team is supervised through an independent timer loop initiated by the `startTeamSyncTimer()` routine. This loop evaluates constraint satisfaction during execution using up-to-date action states published by each robot. Internally, each STN_{act} instance includes expected durations between start and end nodes for actions, as well as inter-action dependencies derived from plan parsing.

During execution, if a violation of temporal constraints is detected, such as an action finishing before its dependent peer (e.g., `relay_data` ending before `sample`) the controller reacts by computing and issuing a delay signal. The affected robot is instructed to extend execution of the ongoing action before sending success to its BT executor (typically representing a wait) to realign the team's

plan temporally. This mechanism ensures robust and synchronized execution when dealing we execution delays and drift.

Algorithm 6 Concurrency Constraints (E_C) Synchronization in the *STNController*

Input: Team T, current actions $\mathcal{A}_T$ with expected start/end times $(\hat{s}_i, \hat{e}_i)$
while team T is active **do**
  for nexts $(a_i, a_j) \in E_C$ **do**
    if a_j not started **then**
      if $\hat{s}_j < \hat{s}_i$ **then**
        Delay a_j to start after a_i
      if $\hat{s}_j + d_j > \hat{e}_i$ **then**
        Delay a_i to extend containment
    else if a_j running and $\hat{e}_j > \hat{e}_i$ **then**
      Delay a_i to preserve containment

Algorithm 6 illustrates the containment-based synchronization process used by the *STNController* to enforce concurrency constraints (E_C) at execution time. For each team, the controller iterates over active action pairs and monitors whether the upcoming E_C constraints of the team are still satisfied. Introducing delays to the appropriate robots to realign their execution and satisfy containment of the sampling actions when needed.

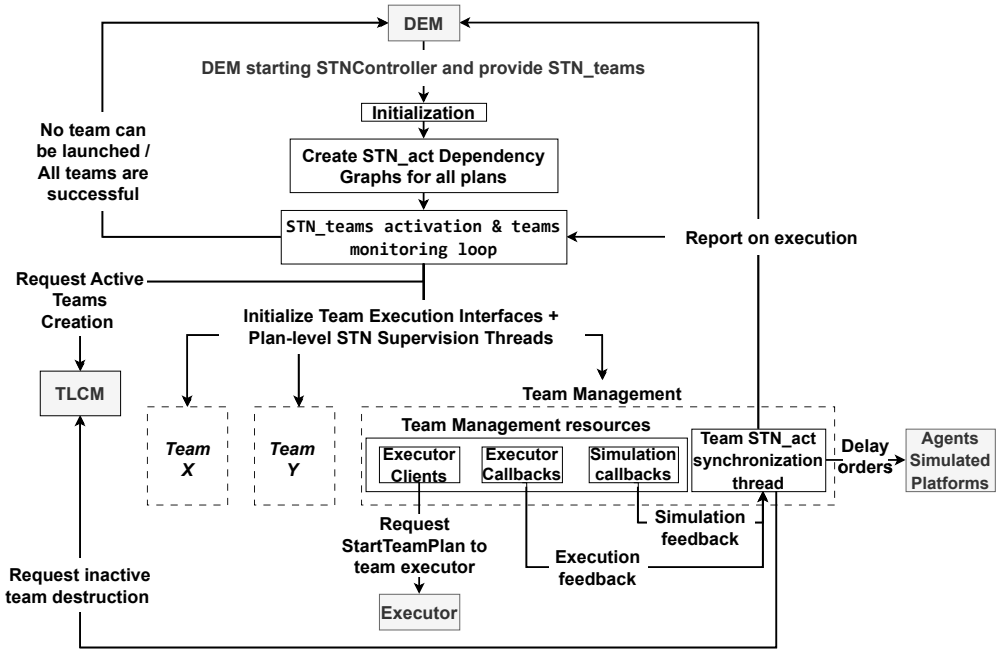


Figure 5.3: *STNController* execution architecture. Teams are supervised in parallel via dedicated synchronization threads enforcing the constraints from the team's STN_{act}.

Figure 5.3 illustrates the expanded runtime execution architecture of the *STNController*. After initialization by the DEM, it prepares STN_{act} graphs for all teams of the $\text{STN}_{\text{teams}}$ and launches a centralized team monitoring loop. This loop activates robot teams based on readiness and assigns each one a dedicated supervision thread.

Each thread interfaces with its team's action executor and robots simulations, continuously comparing live feedback with STN_{act} constraints. As seen in algorithm 6, this allows the *STNController* to maintain alignment of actions and execution without resorting to replanning.

The teams lifecycles (creation and removal) are coordinated with the TLCM, and updated execution statuses are reported upstream to the DEM for potential diagnostic or replanning decisions in case of failure. This modular structure allows for plan-level temporal supervision, enabling scalable and resilient execution of distributed multi-robot missions.

The modularity of this design allows for parallel monitoring of multiple teams, each with its own plan, STN_{act} instance, and feedback loop. This structure scales to heterogeneous execution speeds, inter-robot variability, and enables smooth transitions across symbolic-temporal plans.

5.2.2.3 Failure-Aware Execution Reporting and Time Budget Estimation

When execution significantly violates STN_{act} constraints or an action failure is detected, the *STNController* logs structured failure data in $\mathcal{F}$ and performs a forward estimation of the remaining duration of the failing team's plan, along with those of others currently executing teams. This estimation is performed using the current state of the STN_{act} dependency graph, which includes action progress, expected end times, and active constraints. The process includes:

- Identifying the failing team, robot, site, and point of interest,
- Structuring the failure context (including causal and temporal metadata),
- Estimating remaining execution time for the affected team,
- Identifying parallel teams currently executing and their estimated completion times,
- Transmitting this information to the DEM to support constraint propagation and reallocation strategies.

This mechanism allows the DEM to make informed decisions on whether to reassign idle teams, wait for execution completion, or trigger partial replanning. The failure report $\mathcal{F}$ thus acts as a bridge between runtime monitoring (*STNController*) and strategic failure recovery (DEM).

These supervision and delay injection mechanisms enable AMA-Exec to maintain symbolic and temporal coherence across distributed teams, adapting to execution drift and action misalignment without centralized blocking or unnecessary replanning. They serve as the runtime backbone for robust multi-team coordination.

5.2.3 Failure Diagnosis and Impact Assessment

Failure Diagnosis begins with the structured failure report issues by the *STNController*, which contains both causal context and a estimated execution budget for the affected team and other currently executed teams.

When a failure is reported by the *STNController*, the DEM initiates a structured recovery pipeline composed of three main stages:

1. **Failure Context Parsing:** Extracts key metadata: robot, team, point and site of failure, failed action, and timing.
2. **Propagation Across Dependencies:** Traverses the inter-team dependency graph to identify all mission segments potentially affected by the failure. This graph encodes not only causal relations but also symbolic links (e.g., shared roles or resources) and temporal constraints (from STN), enabling richer and more accurate failure impact propagation than a standard task graph.
3. **Constraint-Aware Diagnosis:** Evaluates the executability of each affected team, checking constraint satisfaction, plan validity, timing feasibility, and potential for reallocation.

Each team is independently evaluated along the following dimensions:

- **Constraint Satisfaction**: Are minimum robot counts, role requirements, capabilities and other domain constraints still met at every site?
- **Plan Validity**: Is the remaining plan structurally executable with the updated robot capabilities? This is assessed using the VAL (Howey et al., 2004) validator in plan continuation mode, which checks whether all remaining steps in the team's plan can

still be achieved given current conditions, without revalidating completed actions or full mission goals.

- **Timing Feasibility**: Are symbolic-temporal constraints (e.g., action orderings and bounds from the STN) still satisfiable?

- **Reallocatability**: Could a valid reassignment of available robots restore executability under constraints?

The results are aggregated into a diagnostic map $\mathcal{D}$, which annotates each affected team with flags (invalid, constraint violation, timing violation, reallocatable). This report does not categorize failures into predefined types but instead enables the planning module (AMA-Plan) to select the minimal replanning scope based on structural impact.

The DEM also computes a numerical *impact depth score*, representing how severely a failure compromises the original team's future actions. This provides a quantitative basis for AMA-Plan to weigh robot exclusion or reallocation.

The next subsection describes how reallocation decisions are made from the diagnostic report and how allocation candidates are scored.

5.2.4 Allocation Scoring and Failure Recovery Decision

Given the diagnostic map $\mathcal{D}$, the DEM first determines whether all teams' plans have been validated. If all invalid teams are affected but reallocatable, candidate robot assignments are generated and evaluated based on former paths structures.

5.2.4.1 Per-Robot Assignment Score

Let r be a robot and $team_i$ a candidate team. The cost of assigning r to $team_i$ is given by:

$$\begin{aligned}\text{score}(r, team_i) = \text{idle}(r, team_i) + \\ \text{dist}(r, team_i) + \text{penalty}(r, team_i)\end{aligned}$$

Where:

$$\begin{aligned}
\eta(r, team_i) &= t_r^{\text{avail}} + d\left(team_r^{\text{last}},\ team_{team_i}^{\text{start}}\right) \\
t_{team_i}^{\text{start}} &= \max_{r' \in \mathcal{R}_{team_i}} \eta(r', team_i) \\
\text{idle}(r, team_i) &= \max\left(0,\ t_{team_i}^{\text{start}} - \eta(r, team_i)\right) \\
\text{dist}(r, team_i) &= \left\| team_r^{\text{last}} - team_{team_i}^{\text{start}} \right\|_2
\end{aligned}$$

- t_r^{avail}: robot r's earliest availability time,
- $d()$: travel time cost function,
- $team_r^{\text{last}}$: last known site visited by r,
- $team_{team_i}^{\text{start}}$: first site of $team_i$,
- $\mathcal{R}_{team_i}$: set of robots assigned to team $team_i$,
- $\eta(r, team_i)$: estimated arrival time of r at $team_i$.

Note that a robot's availability time (t_r^{avail}) reflects both its current state and ongoing execution obligations, ensuring that idle time accounts for realistic timing, not just whether a robot is currently idle.

Proportional Reassignment Penalty. To discourage unnecessary disruption, we assign a penalty to each robot based on how much its reassignment modifies the original team and the severity of the impact:

$$\text{penalty}(r, team_i) = \begin{cases} \gamma \cdot \left(1 - \dfrac{|T_{\text{rem}}|}{|T_{\text{orig}}|}\right) \cdot \delta, & (i) \\ \gamma, & (ii) \\ 0, & (iii) \end{cases} \tag{5.3}$$

Here, T_{orig} is the set of robots originally assigned to the team, and T_{rem} the team after removing r. The factor $\delta = 1$ if the removal violates constraints (e.g., not enough robots remaining), and 0.5 otherwise. The scaling factor γ (e.g., $\gamma = 3.0$) controls the overall importance of reassignment penalties. This results in three typical cases:

- case(i): r is reassigned from a Team that is not executing, incurring a penalty scaled by future team disruption and constraint violation;
- case(ii): r was not initially assigned to any path (e.g., its previous path has finished), and receives a flat penalty γ;
- case(iii): r remains on its original path, resulting in no penalty.

This penalty guides the allocation toward minimal disruption of valid team distribution.

5.2.4.2 Valid Allocation Generation

GenerateValidAllocations (GVA) constructs candidate assignments by generating variations of the affected paths after applying the failure's effects. It considers only robots not involved in currently executing or fixed plans. For each candidate team, it verifies the absence of constraint violations such as insufficient team size, missing capabilities, or broken relay/switch requirements. Only allocations that restore feasibility are passed on for ranking.

5.2.4.3 Global Allocation Score

Once all candidate reallocation options identified, each is scored using the average cost over its constituent robot-to-team assignments:

$$\text{score}_{\text{alloc}} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \text{score}(r, team_{i_r}) \tag{5.4}$$

Where:

- $\mathcal{R}$ is the set of robots participating in the candidate allocation,
- $team_{i_r}$ is the team path to which robot r is assigned in that allocation

The candidate with the lowest average score is selected as the recovery plan in the `RankAllocs` function.

5.2.4.4 Recovery Algorithm

The recovery loop below builds the diagnostic map $\mathcal{D}$, identifying the state of the $\text{STN}_{\text{teams}}$ plans, and attaches a recovery allocation if applicable.

Algorithm 7 Progressive Diagnosis and Recovery Loop

Input: Failure event $\mathcal{F}$, Plan $\mathcal{P}$, Dependencies $\mathcal{G}$, Constraints $\mathcal{C}$, Robot states $\mathcal{R}$, World info W

Output: Diagnostic map $\mathcal{D}$

$\mathcal{D} \leftarrow \{\}$
$T_{\text{aff}} \leftarrow \texttt{PropagateFailure}(\mathcal{F}, \mathcal{G})$
for each $team_i \in T_{\text{aff}}$ **do**
    $\mathcal{D}[team_i] \leftarrow$ {invalid: False, constraints: False, timing: False, realloc: False}
    if not $\texttt{Validate}(team_i)$ **then**
        $\mathcal{D}[team_i].\texttt{invalid} \leftarrow$ True
        if not $\texttt{ConstraintsOK}(team_i, \mathcal{C})$ **then**
            $\mathcal{D}[team_i].\texttt{constraints} \leftarrow$ True
        if $\texttt{ValidReallocationExists}(team_i, \mathcal{R}, \text{W}, \mathcal{C})$ **then**
            $\mathcal{D}[team_i].\texttt{realloc} \leftarrow$ True
        if $\texttt{TimingDesync}(team_i, \mathcal{F})$ **then**
            $\mathcal{D}[team_i].\texttt{timing} \leftarrow$ True
if $\texttt{AllValidOrReallocatable}(\mathcal{D})$ **then return** $\mathcal{D}$
$A \leftarrow \texttt{GVA}(\mathcal{D}, \mathcal{R}, \mathcal{C}, W)$
if $A = \emptyset$ **then**
    $\mathcal{D}.\texttt{replan\_required} \leftarrow$ True
else
    $a^* \leftarrow \texttt{RankAllocs}(A, W, \mathcal{R})$
    $\mathcal{D}.\texttt{best\_allocation} \leftarrow a^*$
return $\mathcal{D}$

This model ensures a clear separation between failure interpretation and recovery decision-making. By generating a complete diagnostic map $\mathcal{D}$ that classifies each affected teams, it ensures that all possible reallocation opportunities are considered before any corrective action is triggered. When reallocation is feasible, the new optimal assignment is included directly in the report, avoiding unnecessary replanning in AMA-Plan. This structured approach improves robustness, supports explainability, and ensures consistent precise handling of both isolated failures and cascading execution disruptions.

5.2.5 Execution Overhead of AMA-EXEC

Execution supervision is event-driven and per-team. The STN-CONTROLLER runs at 1 Hz and performs guarded pairwise checks over the team's concurrently tracked actions, giving a worst-case $\mathcal{O}(\mathcal{A}_T^2)$ work per tick (where $\mathcal{A}_T$ is the number of actions currently executing in a team and is small in practice). On failures, DEM recovery is local: we propagate on the plan/action dependency graph in $\mathcal{O}(N+E)$, validate only affected subproblems with VAL, and, if needed, backtrack reallocation teams over the few dependent paths. These overheads are consistently dominated by AMA-PLAN costs and any additional OPTIC calls when DEM triggers replanning. In contrast to these planning costs, STN-CONTROLLER consistency checks and DEM propagation remain lightweight; observed timeouts at larger scales stem from scenario construction/assignment and occasional OPTIC calls—consistent with the $\mathcal{O}(|\mathcal{S}|\,|\mathcal{R}|^2\,\mathcal{B}^{|\mathcal{S}|/2})$ term in Section 4.2.4.

5.2.5.1 Quantitative Breakdown

For typical missions with $|\mathcal{R}| = 4$ robots, $|\mathcal{S}| = 12$ sites, and mission duration $D \approx 2000$s:

- **STN-CONTROLLER per-tick cost**: < 0.001s (with $\mathcal{A}_T \leq 8$ concurrent actions per team)
- **Total supervision overhead**: $D \times 0.001 \approx 2$s ($< 0.1\%$ of mission time)
- **DEM failure propagation**: < 0.1s per failure event
- **DEM constraint diagnosis**: 0.1-0.5s (VAL validation + constraint checking)

- **Reallocation decision**: 0.5-2s (when feasible)
- **Full replanning**: 5-50s (triggers AMA-PLAN, dominates recovery time)

5.2.5.2 Scalability Characteristics

The STN-CONTROLLER scales well with the number of teams T due to independent per-team supervision threads. The $\mathcal{O}(\mathcal{A}_T^2)$ per-tick cost remains negligible as long as $\mathcal{A}_T < 15$ (typical in practice due to sequential action dependencies within teams). DEM propagation complexity $\mathcal{O}(N+E)$ is lightweight because the $\text{STN}_{\text{teams}}$ dependency graph is sparse ($E \approx T$ in most scenarios), with dense inter-team dependencies occurring only in highly coupled missions.

Experimental results (Section 6.3.2.3) show no execution-time bottlenecks across 9,720 failure injection cases covering diverse failure types, injection timings, and mission scales. The primary computational bottleneck remains in planning (AMA-PLAN), not execution supervision, validating AMA-EXEC's design for real-time deployment.

5.2.5.3 Comparison: Planning vs. Execution Costs

To contextualize execution overhead:

Component	Typical Cost	Impact
Initial planning (AMA-PLAN)	10-200s	Mission start
Replanning (full)	5-50s	Rare (major failure)
Replanning (local)	0.5-2s	Common (reallocation)
STN-CONTROLLER (total)	$\sim$2s	Continuous ($< 0.1\%$)
DEM recovery (no replan)	< 0.6s	Per failure

This breakdown confirms that execution supervision introduces minimal overhead relative to planning costs, enabling robust runtime coordination without sacrificing responsiveness.

5.3 Illustrative Execution Scenarios

To demonstrate the runtime supervision and recovery capabilities of our architecture, we present four illustrative execution scenarios. The first scenario highlights how the *STNController* ensures

symbolic-temporal consistency on the STN_{act} through delay injection. The following three scenarios cover failure handling via the Execution Manager Node (DEM), with increasing severity and recovery cost: local reallocation, multi-path recovery, and full replanning.

5.3.0.1 STN_{act} -Only Delay Synchronization

During multi-robot execution, minor temporal desynchronizations may arise due to communication delays, BT scheduling inertia, or slight execution drift. In the example shown in Figure 5.4, a symbolic-temporal constraint (E_C) requires that the `sample` action completes before the corresponding `relay_data` action ends.

The *STNController* continuously monitors action progress against the STN_{act}. When it detects a potential constraint violation, such as `relay_data` ending prematurely while `sample` is still executing—it dynamically injects a runtime delay into the ongoing action.

This injected delay (highlighted as orange segments with dashed outlines in the top two timelines of Figure 5.4) allows the symbolic order to be preserved without halting the plan or raising a failure. The plan proceeds nominally after this adjustment, and the DEM is not involved in this case.

Comparison with Existing BT Execution:

In the default PlanSys2 implementation, plans are executed as strictly sequential BTs, without enforcing temporal or symbolic constraints during execution. Although a recent preprinted extension has proposed using STN structures to build BTs with temporal flexibility (Zapf et al., 2024), to our knowledge this approach is not yet fully integrated into PlanSys2 or publicly released. Furthermore, the technique employed in that work is used for creation of better temporally constraints BT, while this may address some on plan-inherent temporal constraints, it might fail to address execution-running temporal drifts and other non-planned variations. In contrast, our *STNController* directly supervises runtime execution and enforces temporal constraints dynamically, preserving inter-action dependencies across multiple robots.

Delay Injection Statistics Across Teams

Following this study, we observe how frequently the injected delays to preserve symbolic constraints in the different team configurations presented above and shown in Figure 4.7b. Table 5.2 summarizes the delay statistics observed across three representative team

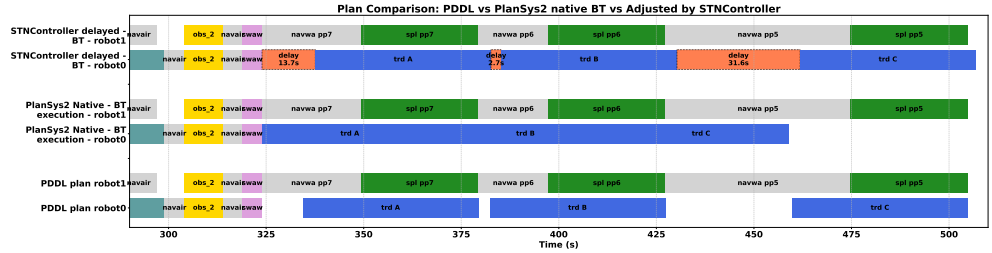


Figure 5.4: STN-only synchronization scenario. The top two timelines show how the *STNController* injects runtime delays into the `relay_data` actions (orange dashed boxes) to respect symbolic-temporal constraints with `sample` actions, without invoking the DEM. Comparisons with PlanSys2 BT and PDDL execution timelines are shown on the four other timelines.

plans executed in parallel, each varying in size and number of resulting E_C constraints bottlenecks.

Table 5.2: STN delay injection behavior across different teams.

Team	Duration (s)	# Delays	Avg Delay (s)	# Bottlenecks
Team 0 (2 robots)	504.80	4	16.36	3
Team 1 (3 robots)	473.85	1	7.34	2
Team 4 (2 robots)	787.97	3	5.97	4

This observation demonstrates that delay injection is not solely a function of team size and plan duration, but also of the temporal structure of the plan and action dependencies. The *STNController* effectively adapts its intervention strategy depending on the concurrency profile, injecting minimal and targeted delays to maintain symbolic order without triggering full replanning.

5.3.0.2 DEM Case 1: Localized Failure with recovery of a Single Team

Considering the plan presented in Figure 4.7b we introduce several levels of failure and observe how different failures will change the impact on the plan and its recovery mechanism, each case leads to a different recovery mode, which can be interpreted as local, reallocatable and full replanning.

Impact Score Computation

To provide a consistent measure of how failures affect mission viability, we compute a S_{global} score for each recovery case. This score captures symbolic degradation, execution disruption, and structural propagation across the plan. It is defined as:

$$S_{\text{global}} = k_1 \cdot S_{\text{val}} + k_2 \cdot S_{\text{prop}} + k_3 \cdot S_{\text{realloc}} \tag{5.5}$$

S_{val} corresponds to the ratio of remaining ($\text{STN}_{\text{teams}}$) elements (that are invalidated due to the failure, as assessed by the VAL validator). The value of k_1 was set to 0.5 to represent the increased overall replanning time that results from additional unvalidated plans. The S_{prop} score is a quantitative metric that quantifies the extent to which the constraints (through (STN_{act})) and actions of the failing plan were directly and indirectly affected by the failure after propagation. We determined $k_2 = 0.2$ as an appropriate setting, since the impact is primarily on the failing plan. Finally, S_{realloc}represent the number of robots-teams allocations that are modified after the failure is propagated through ($\text{STN}_{\text{teams}}$). This provides a useful model for the reallocation impact and total replanning necessity. It also allows for further reflection of the impact variation between the different recovery mechanisms. To increase this variation, we chose $k_3 = 0.3$. This metric facilitates quantitative comparison of localized versus cascading failures and guides the selection of recovery mechanisms.

Failure Scenario

A failure affected one robot on a non-critical team (see Figure 5.5). The failure was propagated, but all impacted teams were reallocatable. The DEM selected a new team from idle robots, validated constraint satisfaction, and resumed execution. AMA-Plan did not trigger a new plan.

In this scenario, a failure occurs in a single team (`Team 0`), while all other teams either remain valid or are not impacted by the failure propagation. AMA-Exec's execution monitoring system detects this localized failure and initiates a recovery process without triggering a full reallocation or global replanning.

This is made possible through:

- Constraint-based validation of all team subproblems.
- VAL-based plan verification to assess subplan viability and mission degradation.
- Targeted propagation of failure effects to dependent teams only.

The only directly failed team is `Team 0`, and a dependent synchronized team, `sync_0_a`, exhibits partial degradation. The rest of the mission structure remains unaffected, and all existing allocations can be preserved.

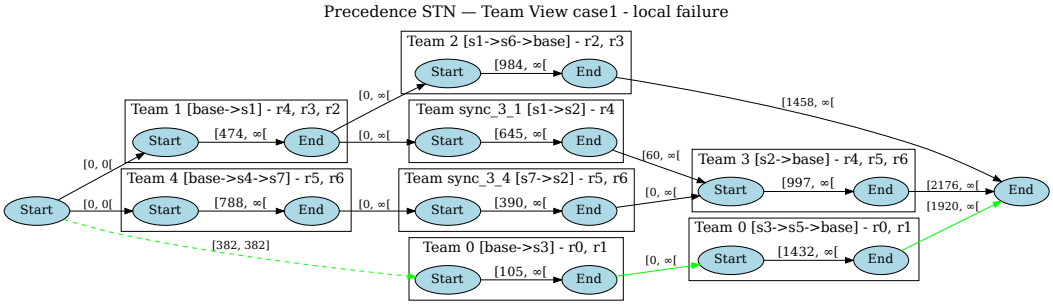


Figure 5.5: Reconstructed STN_{teams} for Case 1. Team 0 in Figure 4.7b is invalidated due to failure and is replanned (at 382s), All other teams remain unaffected and continue execution.

5.3.0.3 DEM Case 2: Localized Failure with Partial Reallocation

A critical failure occurred during execution of `sample robot1 pp6 site3`, invalidating the current team on `Team 0` (see Figure 5.6). Unlike Case 1, the failure could not be resolved by minor adjustments, requiring partial team reallocation.

- `robot4` was reassigned to assist `robot0` on `Team 0`.
- `Team 3`, previously executed by `robot4` with `robot5` and `robot6`, will be done by the two robots later.
- `Team 2` remains unaffected, and global replanning was avoided.

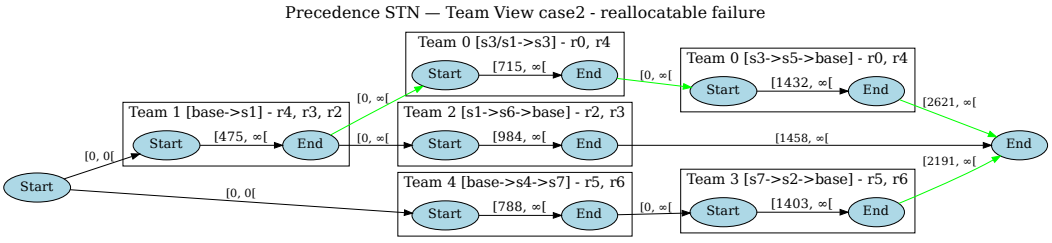


Figure 5.6: Reconstructed STN_{teams} for Case 2. Team 0 in Figure 4.7b failed. Robot4 was re-allocated from Team3 to Team0 after Team1 finish.

The higher VAL and global impact scores (0.67 and 0.5) indicate a broader degradation than in Case 1, though still manageable without a full AMA-Plan replan.

5.3.0.4 DEM Case 3: Escalated Multi-Team Failure Triggering Full Replanning

In the next case we are diving in the last interpretation of replanning possibilities. Following the partial reallocation successfully performed in Case 2, an additional failure occurred on an unrelated team, `Team 2` (see Figure 5.7). This new failure, combined with existing degraded paths, invalidated the current mission configuration and required a full replanning step.

This scenario illustrates the limits of localized recovery. While the AMA-Exec Execution Manager can absorb isolated failures through constraint-aware reallocation, compounded failures across disjoint teams trigger a complete mission reassessment and regeneration.

- The initial failure (`sample robot1 pp6 site3`) was recovered via partial reassignment on `Team 0`.
- A second failure on `Team 2` (`observe robot3 ...`) rendered its team invalid.
- Multiple teams became unrecoverable under current constraints, and AMA-Plan was called to generate a new plan.

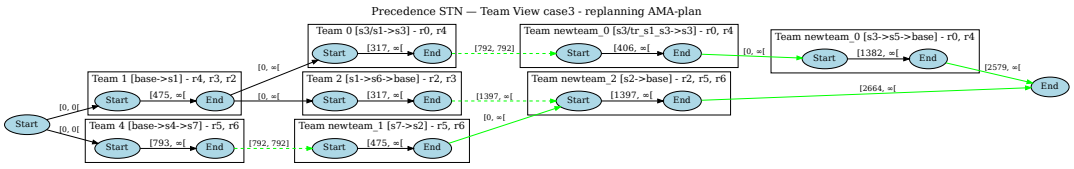


Figure 5.7: Reconstructed $\text{STN}_{\text{teams}}$ for Case 3. Teams 2 failed after Case2's in Figure 5.6 Teams 0 failure. The current teams and paths could not be continued, path replanning procedure.

Table 5.3 summarizes the 3 DEM recovery scenarios presented above.

5.4 Conclusion

This chapter presented AMA-Exec, the execution component of the Adaptive Modular Architecture (AMA) that execute mission plans generated by AMA-Plan. After discussing the difficulties of carrying out symbolic plans in dynamic environments, we presented AMA-Exec's modular architecture. This architecture consists of the Distributed Execution Manager (DEM), the STNController, the Team

Table 5.3: Comparative Summary of Failure Recovery Scenarios

Metric	**Case 1**	**Case 2**	**Case 3**
Failed Teams	0	0	2
Failed Action	relay_data robot0 sp9 site3 B	sample robot1 pp6 site3	observe robot3 cpsite6 site6
Failure Type	robot_cannot_relay	critical_robot_failure	2nd critical_robot_failure
Other Affected Teams	sync_0_a	sync_0_a, 3	2,3,4,0
Unaffected Teams	1, 2, 3, 4, sync_3_1, sync_3_4	1, 2, 4	1
Non-Required Teams	—	sync_3_1, sync_3_4	1
Global Replan Required	No	No	Yes
Preserved Allocation	Yes	Partial (Teams 0 and 3 reallocated)	None
VAL Global Score S_{val}	0.375	0.6667	1.0
Global Impact Score S_{global}	0.236	0.4959	0.9356
Replanning Time (s)	3.93	8.75	16.26

Lifecycle Manager, and per-team BT executors. These components work together to bridge the gap between symbolic planning and robust multi-robot execution, combining global temporal supervision with localized adaptability.

We explained how AMA-Exec consumes execution-ready subplans and the $\text{STN}_{\text{teams}}$ from AMA-Plan, transforming each of them into distributed execution pipelines with temporal controller in STN_{act} that enforces temporal constraints, manage concurrency, and react to failures through structured recovery strategies in the central DEM. The architecture's ability to integrate local decision-making with global coordination enables scalable, failure-resilient, multi-robot operation.

The next chapter builds on this foundation by examining illustrative execution scenarios and quantitatively evaluating AMA across large-scale problem instances. These studies demonstrate how the described mechanisms relay into measurable gains in robustness and efficiency in practice.

Chapter 6

Study Cases and Evaluation

Contents

This chapter evaluates the proposed AMA system across study cases of increasing difficulty. Our goal is to (1) establish feasibility in simpler and different settings, (2) isolate the impact of mode switching and cross-domain constraints, and (3) present robot's level execution structure, and validate the system on realistic trans-media mission. Complexity is increased along two axes: team composition (homogeneous → heterogeneous → trans-media) and spatial scope (single-site → multi-site).

It evaluates the performance of the AMA architecture across multiple simulated mission scenarios of different domains, quantifying planning efficiency, execution robustness, and recovery latency under various failure conditions.

We organize the evaluation into three cases:

1. **Homogeneous and heterogeneous feasibility with AMA.** We demonstrate AMA on single-medium homogeneous robots teams observing and deposing beacons on water sites, then on heterogeneous teams with varying capabilities and mediums

that needs to sample and check water sites in a water environment. These scenarios provide a clean baseline and allow comparison with classical PDDL solver outputs in domains with fewer execution constraints.

2. **Heterogeneous vs. trans-media in a single-site mission.** We quantify the additional complexity introduced by *switching actions* and cross-medium coupling by solving an identical single-site task with (i) a heterogeneous, single-medium team and (ii) a trans-media team. This isolates the impact of trans-media capability and the reason for chosing it as our use-case.

3. **Trans-media, multi-site mission.** We return to the thesis's illustrative scenario (Cf. Section 4.2.5 and Section 5.3) and evaluate AMA at scale across varying scenario parameters. We introduce the structure of AMA-Exec's BT action nodes (as instantiated via PlanSys2), then extend the illustrative results with a quantitative study over varied instances. We report both AMA-Plan planning-time breakdowns and AMA-Exec planning level recovery outcomes, providing an empirical validation of the architecture.

Across cases we track solution reachability, quality and computational cost (plan makespan, planning time decomposition, and scalability), and in the trans-media mission we additionally record runtime robustness using varying failures scenarios and Chapter 5 plan-level recovery mechanisms.

Unless stated otherwise, all experiments run on an Intel Core i7-11800H (8 cores, 16 threads, up to 4.6 GHz) with 32 GB RAM under Ubuntu. We denote by R the number of robots and by S the number of sites. In Case2 we also vary the number of objectives K (points of interest) at a single site. Wall-clock times are reported in seconds with non-breaking thin spaces (e.g., 12.3 s).

We evaluated multiple planners through the Unified-Planning (UP) framework (Micheli et al., 2025) developed in the AIPlan4EU project, including *LPG* (Gerevini and Serina, 2002), *Aries* (Bit-Monnot, 2023) (current version), and several temporal-numeric planners such as *POPF* (Coles et al., 2010), *TFD* (Röger et al., 2008), and *OPTIC* (Benton et al., 2012). Following these tests, *OPTIC* was selected as the main baseline (for monolithic solving) and also integrated within *AMA-Plan* as the sub-solver for decomposed sub-problems, we chose to use OPTIC's first solution in both case as planning time is a major issue in robotic missions.

Instances are generated from a common distribution throughout the thesis: sites are sampled uniformly within the operational area; travel times follow a medium-dependent kinematic model; and actions durations are either fixed by mechanical constraints or computed from medium's speed and distances. Each configuration is evaluated over multiple randomized batches (distinct seeds).

6.1 Case Study 1: Homogeneous and Heterogeneous Missions with AMA

6.1.1 Motivation and Scenario

This first case study establishes a baseline by evaluating AMA in missions involving purely homogeneous robots, and then extends to a heterogeneous configuration with multiple robot types. These scenarios are simpler than the trans-media case but are fundamental to demonstrate that AMA-Plan and AMA-Exec perform robustly across basic mission settings.

In the homogeneous setting, all robots share identical capabilities (navigation, sampling), and the environment is composed of multiple sampling sites of similar type. This setup allows us to compare AMA-Plan against classical temporal planners such as OPTIC under controlled conditions.

The heterogeneous setting introduces robots with differentiated roles (e.g., relay vs. sampler). This adds additional allocation constraints without the full complexity of mode-switching. It enables us to validate that AMA's decomposition and allocation logic generalize beyond homogeneous teams.

6.1.2 Evaluation Focus

In these cases, we analyze:

- Comparative planning performance between AMA-Plan and OPTIC in homogeneous missions;
- How heterogeneous constraints impact allocation and planning time;
- Success rates and solvability across configurations;
- Overhead introduced by AMA's decomposition and $\mathrm{STN}_{\mathrm{teams}}$ creation in simple missions.

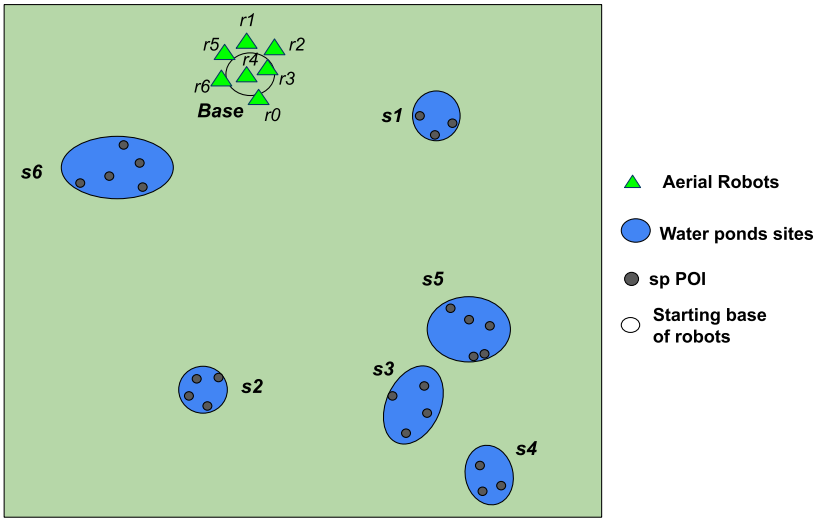


Figure 6.1: Mission example for the homogeneous case: multiple sites to be visited by robots with identical sampling capability.

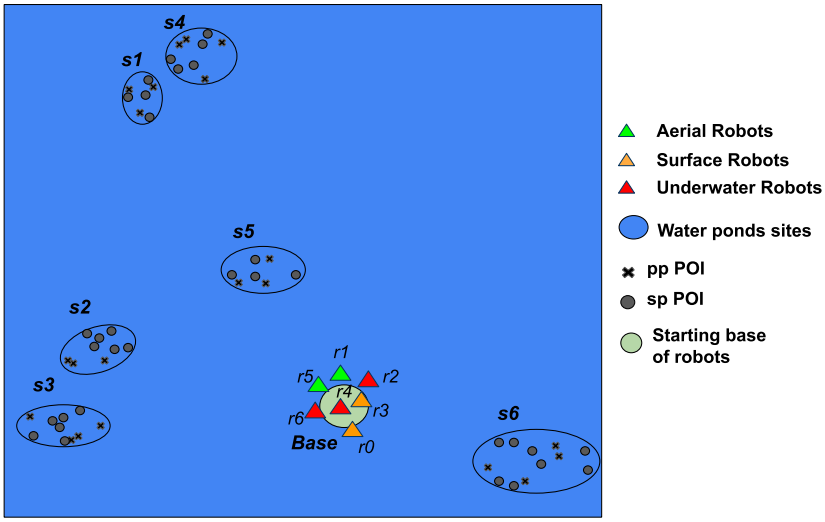


Figure 6.2: Mission example for the heterogeneous case: differentiated robot roles (e.g., relay vs sampler) must be allocated to distinct site tasks.

6.1.3 Results

Homogeneous scenario (D1). Presented in Figure 6.1 (domain available in appendix A.1), Figures 6.3–6.6 illustrate the homogeneous case. Robots are allocated to sites with identical roles, and AMA decomposes the mission into balanced teams. The Gantt chart (Fig. 6.3) shows robot visits over time, while the topological allocation

graph (Fig. 6.4) and path clustering (Fig. 6.5) provide insight into the distribution of robots across sites. The corresponding $\text{STN}_{\text{teams}}$ (Fig. 6.6) highlights precedence relations between subteams.

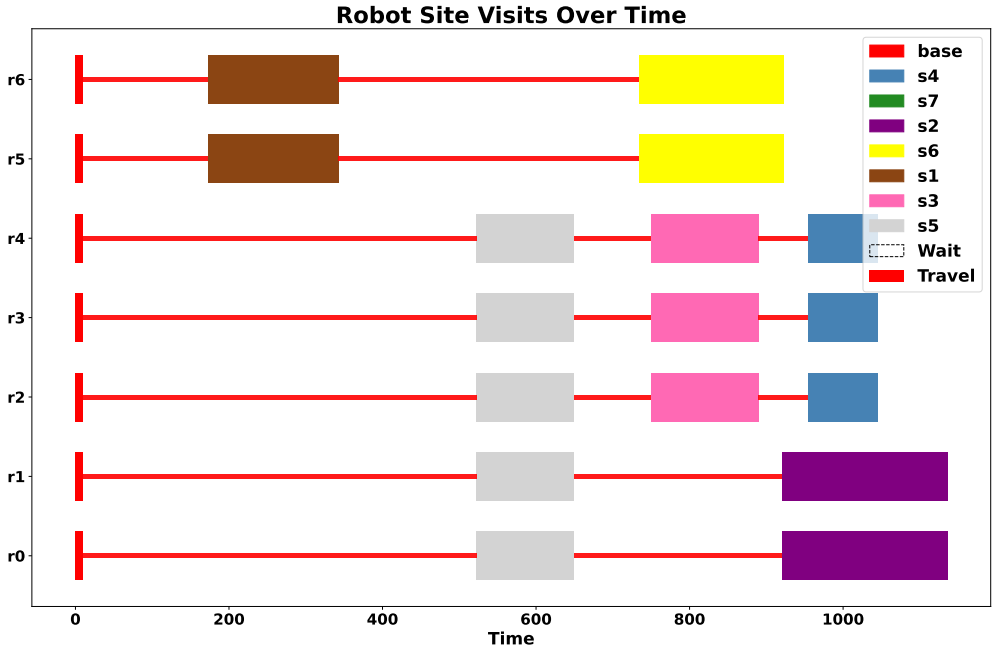


Figure 6.3: Robot site visits over time for the homogeneous case.

For the homogeneous scenario (D1), the OPTIC direct solution yields a makespan of **1178.0 s** in **14.9 s** of planning time, whereas AMA-Plan achieves a makespan of **1135.5 s** in only **0.8 s**. These results confirm that AMA-Plan introduces negligible overhead while producing comparable (slightly better in this case) makespan in trivial homogeneous missions.

Heterogeneous scenario (D2). Presented in Figure 6.2 (domain available in appendix A.2), Figures 6.7–6.10 illustrate the heterogeneous case. Robots are differentiated by role (e.g., relays vs. samplers), requiring allocation constraints to be respected. The Gantt chart (Fig. 6.7) shows how these roles translate into distinct site allocations, while the topological graph (Fig. 6.8) and path clustering (Fig. 6.9) show how AMA partitions robots into feasible coalitions. The STN representation (Fig. 6.10) captures the team-level precedence relations.

For the heterogeneous scenario (D2), the OPTIC direct solution yields a makespan of **3265.8 s** in **20.3 s** of planning time, whereas AMA-Plan achieves a makespan of **3074.2 s** in only **0.5 s**. These

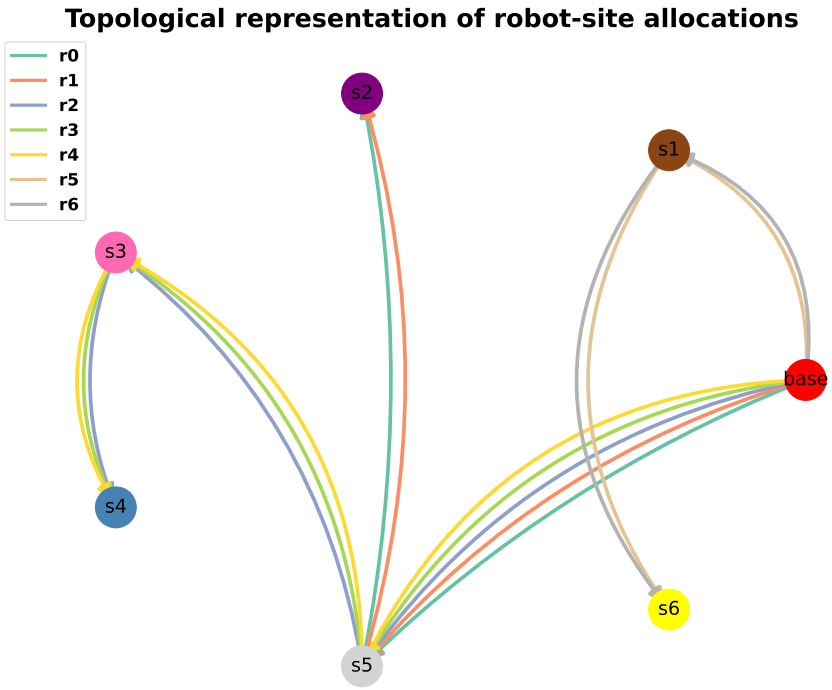


Figure 6.4: Topological representation of robot-site allocations (homogeneous case).

results confirm that the decomposition logic scales effectively under heterogeneous allocation constraints, with stable overhead and faster planning.

6.1.4 Discussion

This case study demonstrates that AMA performs reliably in both homogeneous and heterogeneous settings (results reported on Table 6.2), validating its generality and showing that it does not introduce unnecessary overhead in simple missions. More importantly, it highlights that the same architecture can seamlessly handle the progression from trivial homogeneous domains to constrained heterogeneous ones — a prerequisite before tackling the full complexity of trans-media scenarios in Case 3.

More detailed comparative results (aggregated makespan and planning time curves across team sizes) are reported later in Figure 6.12.

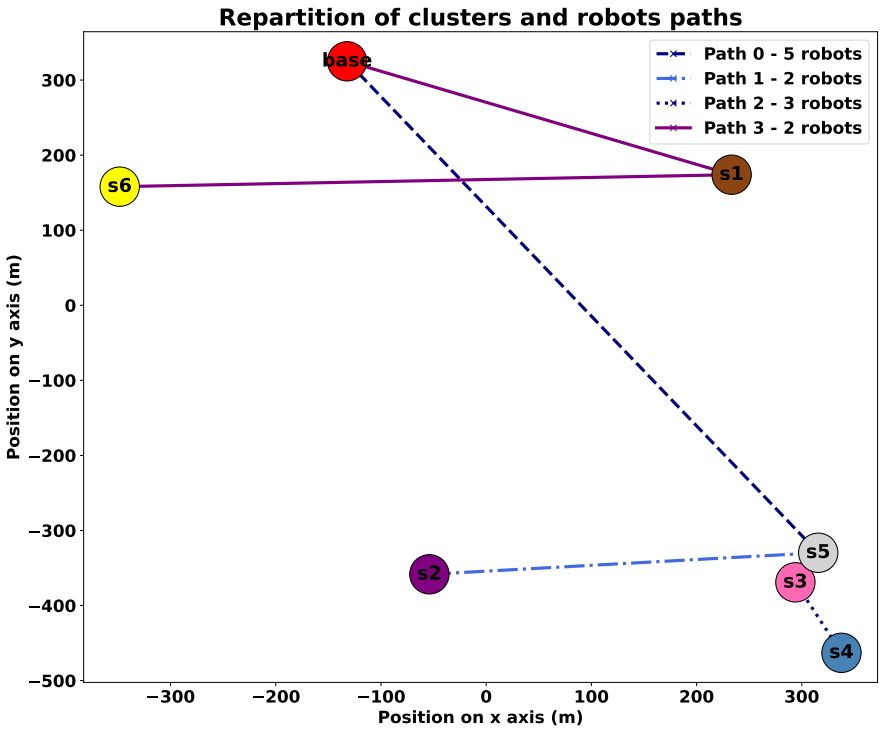


Figure 6.5: Repartition of clusters and robot paths (homogeneous case).

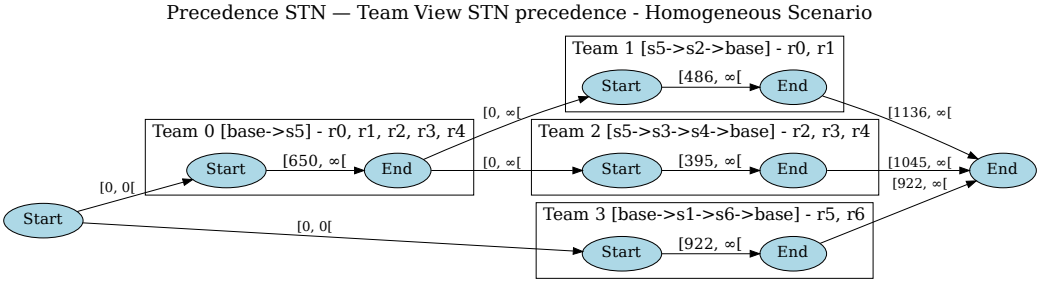


Figure 6.6: Precedence STN of teams (homogeneous case). Each team is abstracted as a single Start–End interval containing the resolution of its assigned sites and the travel time in between. Precedence arcs ensure synchronization across teams. In this example, four homogeneous teams are formed, each covering a disjoint set of sites.

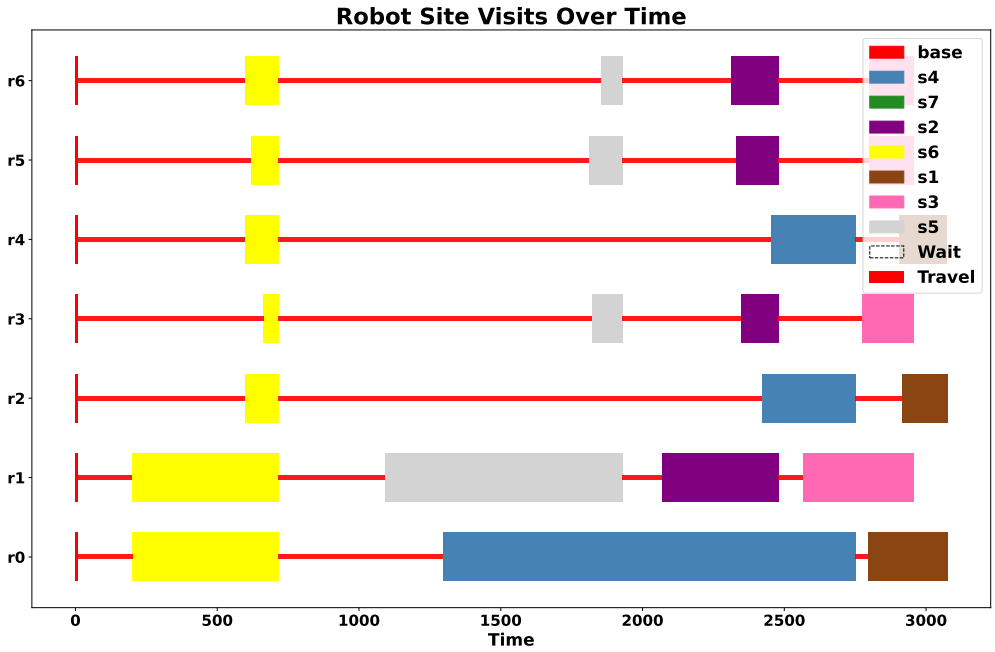


Figure 6.7: Robot site visits over time for the heterogeneous case.

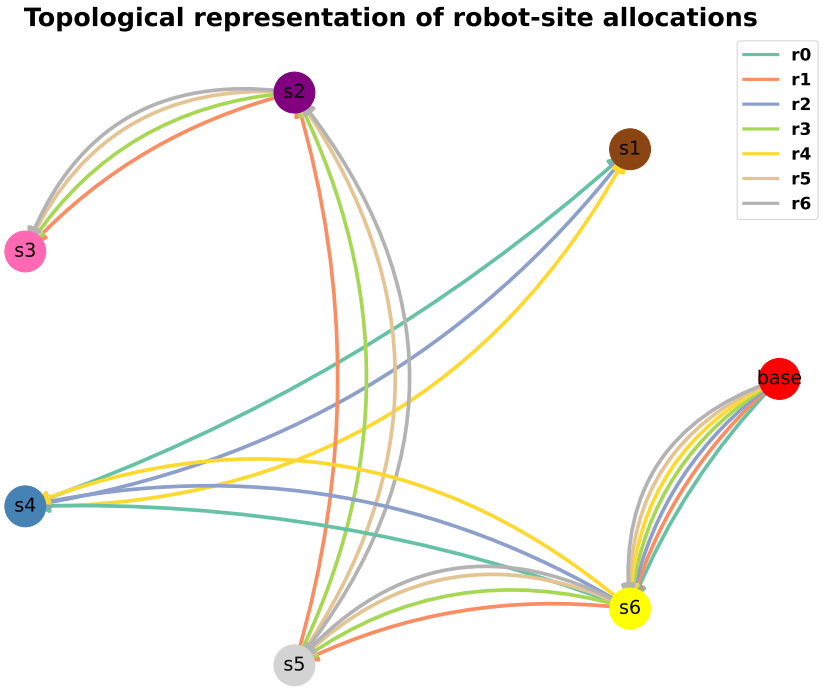


Figure 6.8: Topological representation of robot-site allocations (heterogeneous case).

6.2 Case Study 2: Heterogeneous Robots vs. Trans-Media Robots

6.2.1 Motivation and Scenario

The second case study investigates the fundamental difference between **heterogeneous teams** and **trans-media teams** in the con-

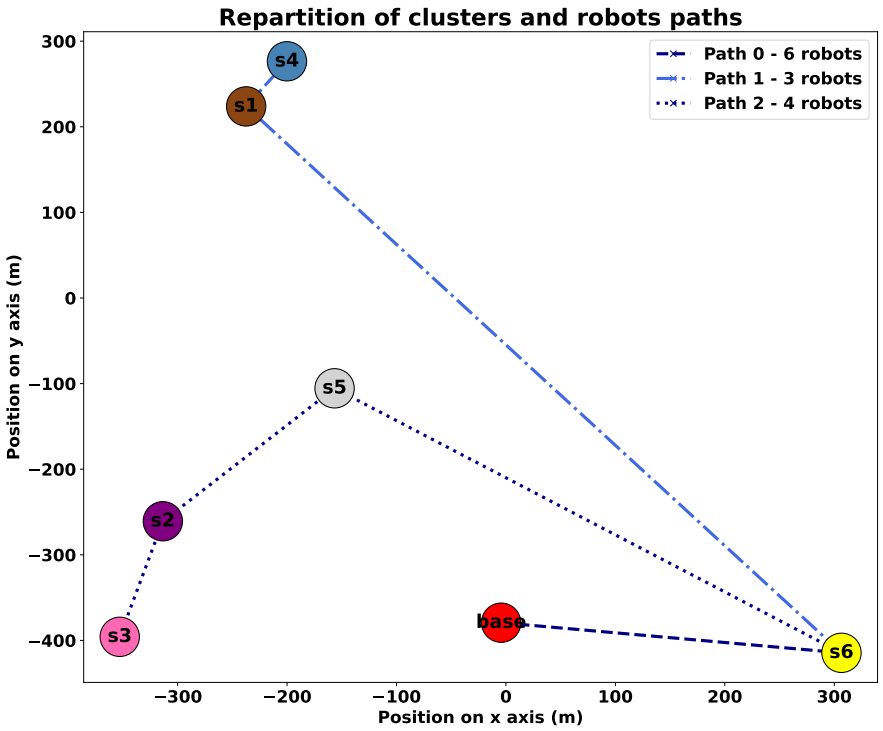


Figure 6.9: Repartition of clusters and robot paths (heterogeneous case).

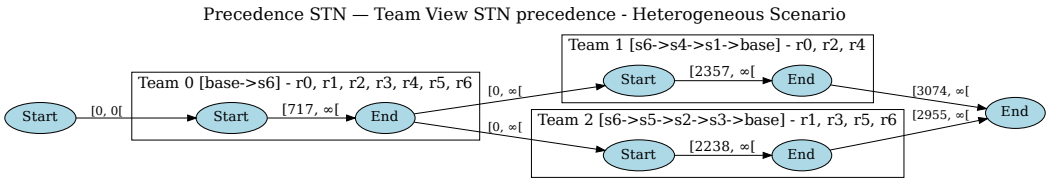


Figure 6.10: Precedence STN of teams (heterogeneous case).As in the homogeneous case, each team is abstracted as a Start–End interval, but here Team_0 completes first before being divided into Team_1 and Team_2, highlighting tighter role distribution and dependencies.

text of PDDL-based planning. In the heterogeneous setting (Domain 3, *Heterog*), robots are restricted to their native medium (air, surface, underwater). In the trans-media setting (Domain 4, *Trans-Med*, domain available in appendix A.3), robots can change medium through dedicated switching operators.

The introduction of switching capabilities increases mission flexibility and reduces execution cost, but it also enlarges the search space considerably. This case study therefore quantifies the trade-off between *planning complexity* and *execution efficiency* when solving the

Table 6.1: **Quantitative summary of initial study cases.** Comparison between OPTIC and AMA-Plan for the homogeneous (D1) and heterogeneous (D2) scenarios. The table reports the average makespan, total planning time, and relative speed-up achieved by AMA-Plan.

Scenario	Planner	Makespan (s)	Planning time (s)
D1 – Homogeneous	OPTIC	1178.0	14.9
	AMA-Plan	1135.5	0.8
D2 – Heterogeneous	OPTIC	3265.8	20.3
	AMA-Plan	3074.2	0.5

Table 6.2: Summary table of Case 1: D1 - D2 demonstration results

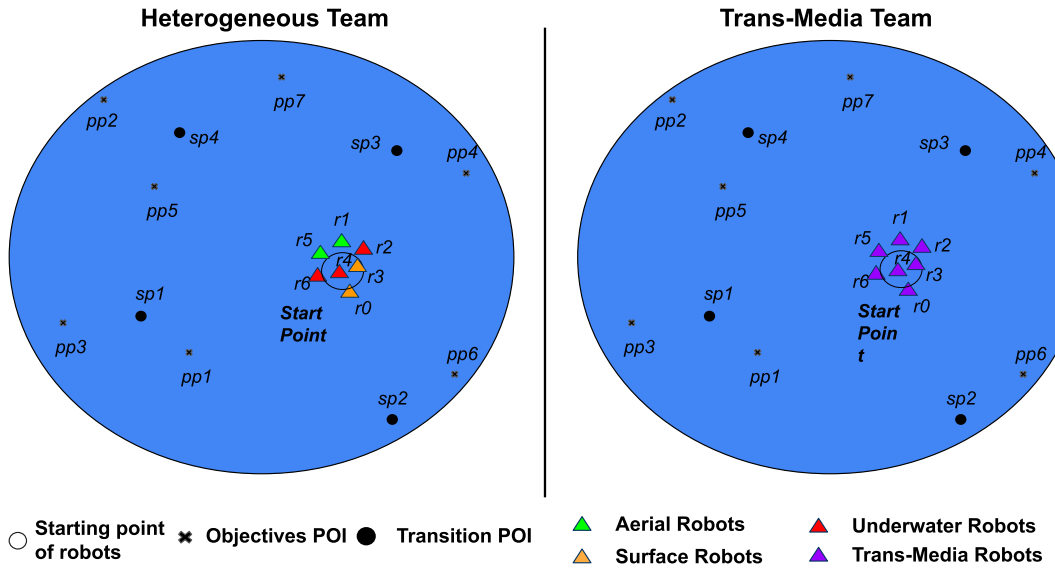


Figure 6.11: Illustration of the Step 2 scenario. On the left, a heterogeneous team with separate aerial (green), surface (surface), and underwater robots (rouge). On the right, a trans-media team where each robot can switch across media. The evaluation compares the resolution of these two domains with OPTIC for varying parameters.

same single-site mission.

6.2.2 Evaluation Focus

Two sweeps are considered:

- **Increasing the number of objectives** ($\mathcal{K} = 4$–23) while keeping the team size fixed at $\mathcal{R} = 4$;

- **Increasing the number of robots** ($\mathcal{R} = 4$–14) while keeping the number of objectives fixed at $\mathcal{K} = 14$.

For each configuration, we record:

1. the planning time measured by OPTIC;
2. the mission makespan of the produced plan;
3. coverage, i.e., the percentage of batches solved within a 300 s limit. Timeout cases are explicitly annotated in the plots.

6.2.3 Results

Figure 6.12 reports the aggregated results across ten randomized batches. The top row corresponds to planning time, while the bottom row shows mission makespan.

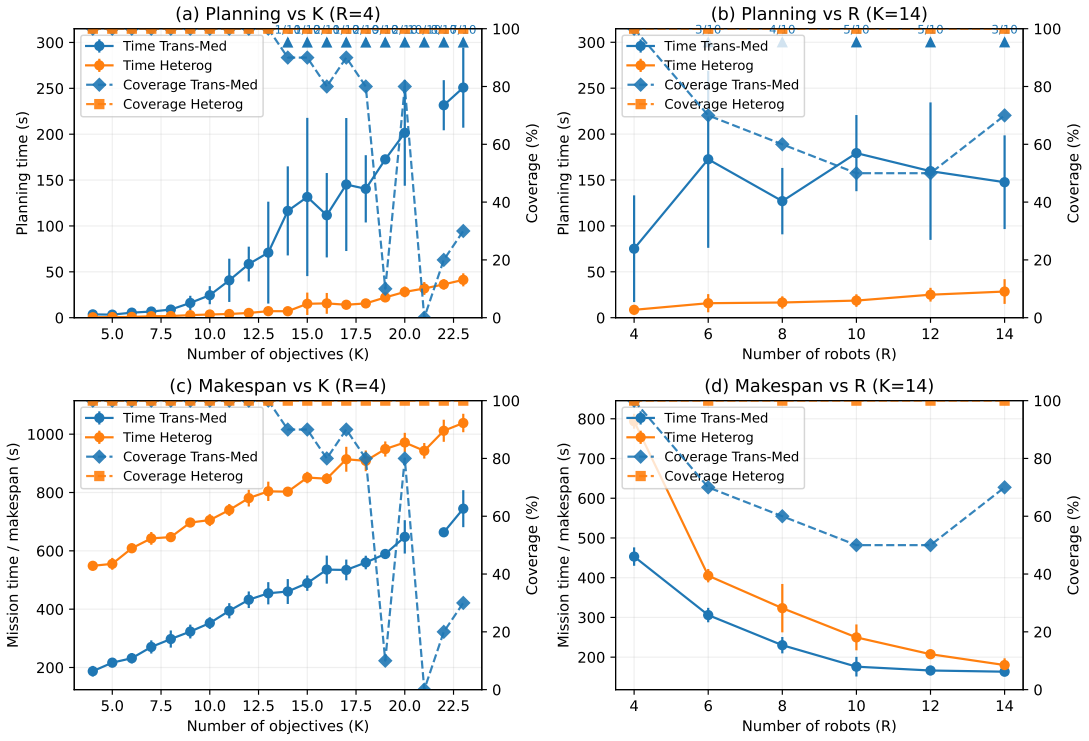


Figure 6.12: Comparison of heterogeneous and trans-media domains. (a,c) Increasing number of objectives $\mathcal{K}$ with fixed $\mathcal{R} = 4$. (b,d) Increasing number of robots $\mathcal{R}$ with fixed $\mathcal{K} = 14$. Curves report mean ± std over ten randomized batches. Coverage (right axis) indicates percentage of solved instances, with caret markers showing the number of timeouts out of ten runs.

Planning time vs. objectives ($\mathcal{R} = 4$).

Heterogeneous's planning time remains fast ($< 20\,\text{s}$) and fully reliable up to $\mathcal{K} = 23$. Trans-Media team's planning time shows rapidly growing complexity: beyond $\mathcal{K} \approx 20$ planning frequently exceeds 200 s with partial coverage. This reflects the branching introduced by switch preconditions, alternative entry/egress choices, and extra causal links.

Planning time vs. robots ($\mathcal{K} = 14$).

With more robots, heterogeneous planning time decreases (more parallelizable allocations) while keeping full coverage. In trans-media, coverage remains partial (roughly 40–70% in our runs); mean planning time drops slightly with R but stays an order of magnitude higher than the heterogeneous team due to persistent switching choices.

Mission makespan.

Execution flips the picture: in both sweeps, the trans-media team achieves substantially shorter makespan—often about half of the heterogeneous at large $\mathcal{K}$—by exploiting shorter cross-medium routes and better synchronization. The makespan advantage persists as $\mathcal{R}$ increases.

6.2.4 Discussion

Case 2 reveals a clear trade-off: **planning tractability** (heterogeneous) versus **execution efficiency** (trans-media). Heterogeneous teams are easier to plan but deliver longer missions due to rigid role separation. Trans-media teams plan more slowly and with reduced coverage, yet execute faster thanks to flexible routing and on-demand switching. These results motivate AMA-Plan's decomposition demonstrated in Chapter 4 and the use-case (Case 3): mitigating combinatorial explosion while preserving the operational benefits of switching.

6.3 Case Study 3: Trans-media Robots Multi-Site Pollution Assessment

Trans-Media Robots in pollution assessment mission: The most challenging scenario that we have been working on, where trans-media robots must navigate an environment that includes ground, air and

water. This case study evaluates how well AMA-Plan and AMA-Exec handle complex domain definition, higher dimension problems, mission execution monitoring and plan-level recovery mechanisms.(surface)

6.3.1 Motivation and Scenario

The pollution assessment mission stresses both layers of the architecture. Containing Case 1 (multi-site scenario, larger scale) and case 2 (trans-media switch action available) complexity components. Case 3 is issued as the idea scenario to challenge AMA's capability to handle complex multi-robot mission planning and execution. It couples team formation, adaptability of robots and robot-site allocation (AMA-Plan) with temporal monitoring and recovery (AMA-Exec), matching realistic field deployments where multi-modal vehicles operate under execution uncertainties.

6.3.1.1 Action Control Structures with Behavior Trees

As detailed in Subsection 3.2.2, AMA-Exec relies on PlanSys2, plan-level PDDL $\rightarrow$ BT transformation to supervise the plans. Furthermore, PlanSys2 allows for definition of action's level BT structure to supervise individual robot actions. While this is still not fully developed, action-level BTs were designed to follow a specific structure to add a action-level recovery capability to the system.

BT node legend. We follow the standard BehaviorTree.CPP library (BT.CPP) node types, ensuring compatibility with existing execution engines. The following roles are used uniformly across all actions: *Sequence* [$\rightarrow$] progresses until one child fails; *Fallback* [!] tries its children until one succeeds; *Parallel(ALL)* executes `Start` and `Guard` nodes concurrently and only succeeds if both do; *Condition* [$\checkmark$] acts as an online guard for feasibility or safety; *Action* [$\triangleright$] triggers a robot skill; and, when required, a custom *Operator* node [*Op*] acts as an asynchronous blocking action until a human decision is provided. Figure 6.13 illustrates these node set legends.

Generic three-arm structure. At the action level, we rely on a fixed three-arm *Fallback* pattern, designed to balance immediate execution with guarded retries and alternatives:

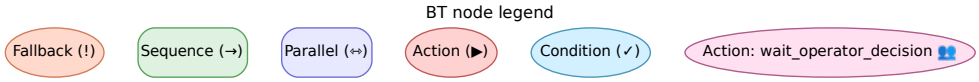


Figure 6.13: **BT node legend used in this chapter.** Only some of the nodes available in BT.CPP (plus a custom Operator node) are considered: Fallback, Sequence, Parallel, Action, Condition.

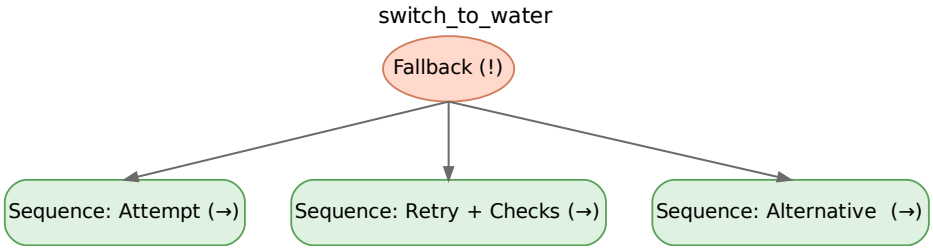


Figure 6.14: **Three-arm BT structure for actions.** A root *Fallback* supervises three branches: immediate attempt, retry with checks, and an alternative branch. This pattern is reused across all trans-media actions.

1. **Attempt (left)** — perform the action immediately within a *Parallel(ALL)* of `Start` and `Guard`, ensuring that the actuator is launched and the goal condition is monitored online;

2. **Retry + checks (middle)** — re-attempt the action only after verifying explicit preconditions such as entry feasibility, energy, or environment conditions;

3. **Alternative (right)** — perform auxiliary steps that may lead to the same goal (e.g., navigate to an alternate entry point, use a relay path, or—in operator-enabled variants—branch to human decision).

This pattern provides a uniform template for trans-media robot skills. Takeoff and landing actions typically omit the right arm, while more complex switching, sampling or data transmission actions include it.

Example: switching to water. Figure 6.15, in landscape, illustrates the BT structure adopted for the `switch_to_water` action. The left arm executes the switch directly, monitoring readiness at the surface; the middle arm retries after checking action's preconditions like entry point, energy, and environment state; the right arm encodes alternatives. In its simplest form, this branch navigates to a different entry point. In an operator-enabled extension, it is replaced by an operator decision sequence: after waiting for human input, a *Fallback* executes one of three guarded sub-branches (*retry elsewhere*, *observe again*, or *fail*).

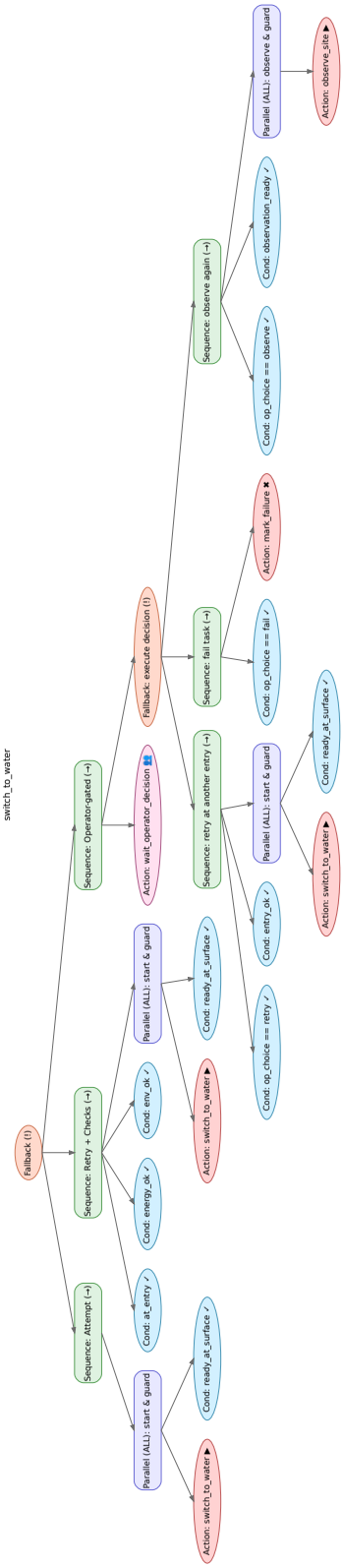


Figure 6.15: **Behavior Tree for `switch_to_water`.** The arms seen before are kept, with an operator-enabled decision variant, the alternative action is chosen by a guarded human decision.

6.3.1.2 Integration of Planning and Execution

In this mission, AMA-Plan generates teams of robots and encodes dependencies in the $\text{STN}_{\text{teams}}$. These structures are then instantiated into executable BTs plan for each team. Temporal constraints propagate through the STN_{act}, ensuring coordination across modes (e.g., a surface relay finishing before an underwater robot completes its sampling). At runtime, AMA-Exec monitors these constraints: if delays or failures occur, the team-level STN_{act} recovery mechanisms attempt local temporal compensation first, before escalating to team failure, with local replanning, $\text{STN}_{\text{teams}}$ reallocation or full replanning. This integration illustrates how the architecture bridges symbolic planning with reactive execution in a multi-modal environment. As these examples where demonstrated in the illustrative scenarios at both AMA-Plan level in Section 4.1.2 and AMA-Exec level in Section 5.3, the next subsection present a quantitative study of both of these aspects for a varying scenario and mission parameters.

6.3.2 Quantitative Study. Characterisation of Planning Time in AMA-Plan and Recovery Mechanisms in AMA-Exec

We are now focusing on the variation of the allocation of time for the various components of the re-allocation and resolution in AMA-Plan.

6.3.2.1 Limits and test cases

Following with the quantitative study, we first assess the runtime cost of AMA-Plan across all tested configurations ($|\mathcal{S}| \in \{4, \ldots, 20\}$, $|\mathcal{R}| \in \{4, \ldots, 20\}$) with 30 batches of randomized inputs for a total of 2,430 planning problem instances. Each instance corresponds to a complete mission planning run, including symbolic decomposition and $\text{STN}_{\text{teams}}$ resolution. Higher-$|\mathcal{S}|$ scenarios contain multiple points of interest (PoIs), yielding between 25 and more than 50 sampling objectives (pollution points pp in P_{spl}). Grounded PDDL action counts routinely exceed 100, as an example taking back our 7 sites and 7 robots illustrative mission (Figure 1.2) achieves sampling of $|P_{spl}| = 18$ and compiles into 143 grounded actions distributed across team-specific subproblems presented in Figure 4.7b for a total mission time of 2,236–2,246 s depending on temporal drift during execution.

We chose 300 s as a timeout threshold as our study focuses on planning at execution level, and planning time is an important vari-

able to avoid idle time in that context. Each PDDL subproblem was also capped at 60 s wall-clock to reflect the same constraints.

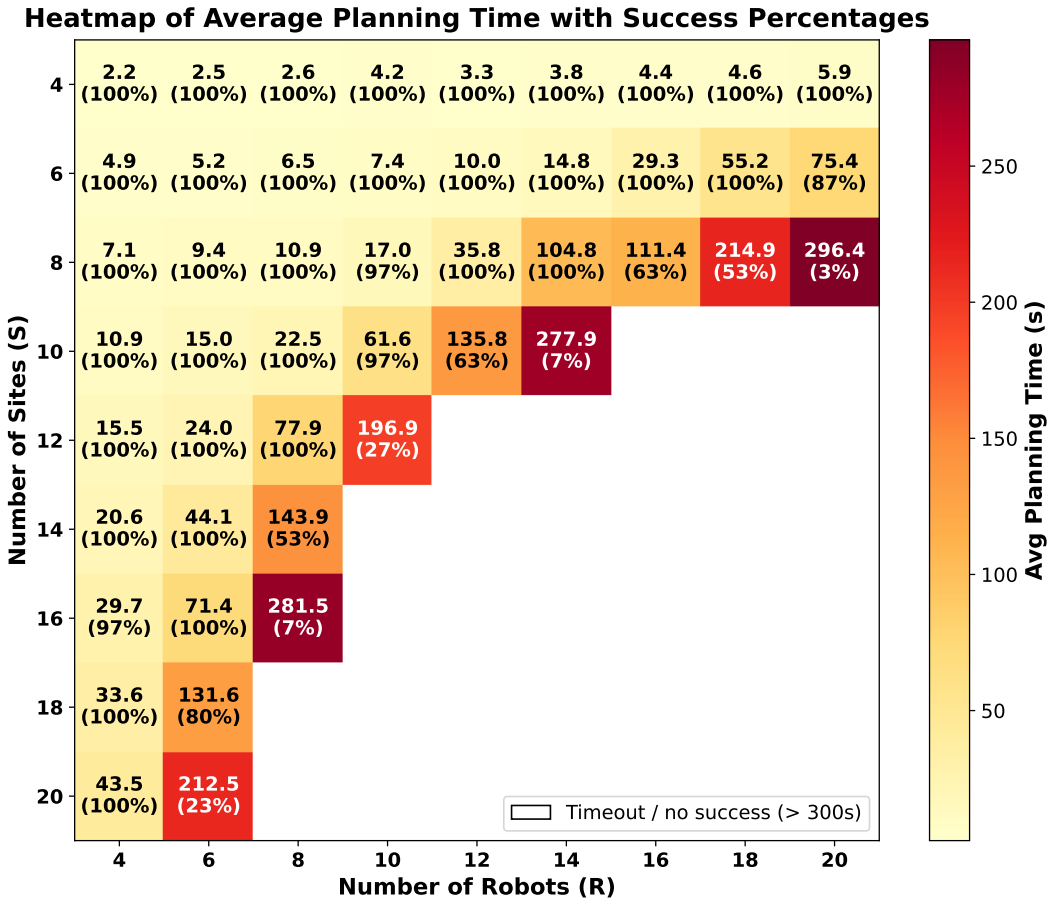


Figure 6.16: Heatmap of average planning time (seconds) with success rate (parentheses). Each cell aggregates 30 batches. White borders denote at least one run exceeding the 300 s timeout.

6.3.2.2 Characterisation of Planning Time

Figure 6.17 decomposes planning cost at fixed $|\mathcal{R}|$. The stacked bars show that STN solving is the dominant contributor at lower scales, while clustering and robot assignment remain comparatively minor. As $|\mathcal{S}|$ grows, per-scenario construction (sites and clusters) increases sharply and eventually overtakes STN solving, explaining the steep rise in planning times. The 95% confidence intervals (top) confirm that these averages are stable across batches. Overall, scalability degrades gracefully up to $|\mathcal{S}| \approx 12$, after which the combined growth of scenario construction and STN solving causes some runs to approach

or exceed the 300 s timeout. We therefore treat the 300 s planning cutoff as our practical real-time deployment threshold: configurations exceeding this limit are considered operationally infeasible due to the idle time they would induce during execution.

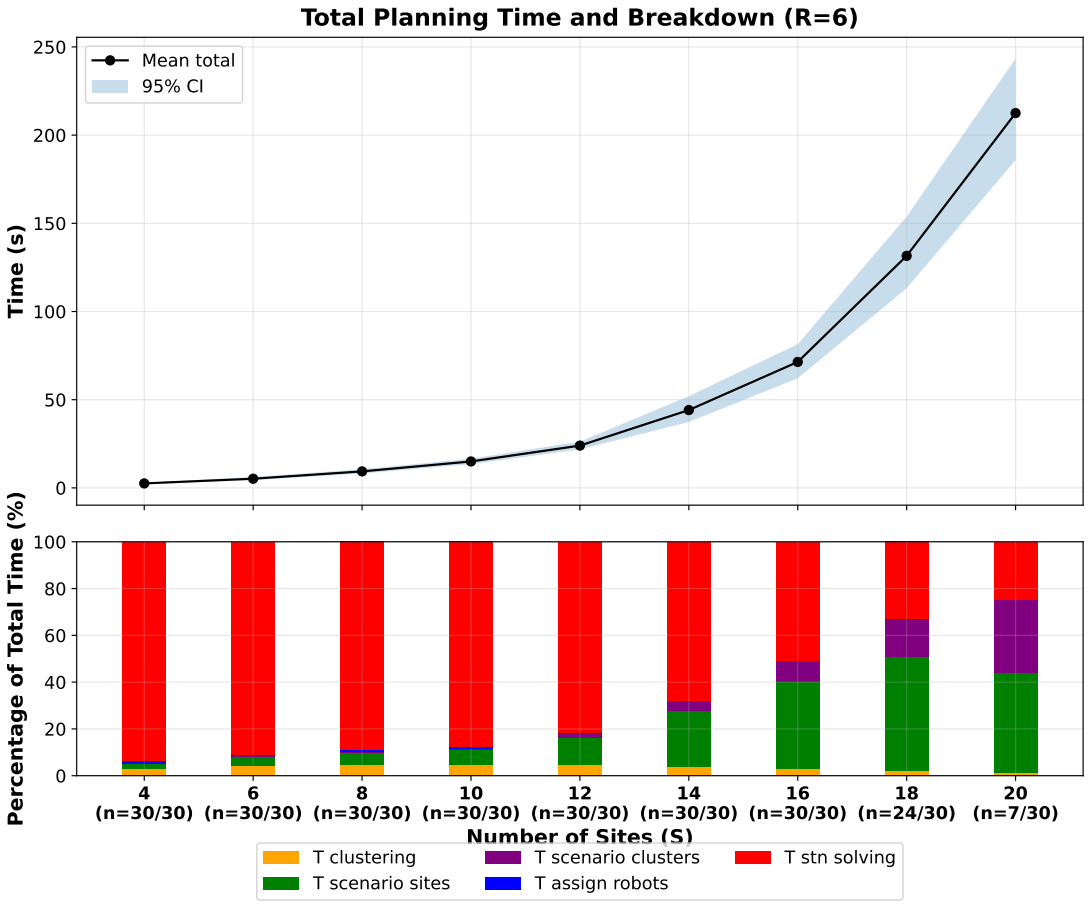


Figure 6.17: Total planning time and component breakdown for $|\mathcal{R}| = 6$. Top: mean total planning time with 95% bootstrap confidence intervals. Bottom: percentage of time per component (clustering, site-level construction, cluster-level construction, robot assignment, STN creation and solving).

AMA-Plan demonstrates a broader range of coverage of the various forms of the trans-media problems parameter. It facilitates the solvability of problems of a substantially larger and more complex nature than those addressed by conventional PDDL solvers.

6.3.2.3 Runtime Failure Recovery Analysis

We now turn to the evaluation of AMA-Exec under failure. We categorize the failures by failure types:

- **Failure type 1:** a change in the environment (for example, unusable relay point).
- **Failure type 2:** robot loses capability (e.g., relay, sample);
- **Failure type 3:** robot becomes unavailable (crash);

and failures are injected at different points in the $\text{STN}_{\text{teams}}$:

- **Strategy 1 (Early):** in a team with no predecessor;
- **Strategy 2 (Mid):** in a team with both predecessors and successors;
- **Strategy 3 (Late):** in a terminal team with at least one predecessor.

A total of 9,720 failure cases were tested, covering all $(|\mathcal{S}|, |\mathcal{R}|)$ configurations with 30 randomized batches each, as well as all failure types and strategies. For each failure, we recorded the recovery mechanism selected by AMA-Exec and its estimated cost. As described in Section 5.3, AMA-Exec can recover by local symbolic replanning, by reallocating robots within $\text{STN}_{\text{teams}}$, or by full replanning. Across all runs, missions completed unless planning timed out; we did not observe aborts, since the current quantitative failure tests do not account for high-scale cascading failures or enforce a strict makespan bound that could render a mission unsolvable.

Figure 6.18 confirms that the nature of the failure significantly influences recovery behavior. Robot crashes will have a higher impact on the plans and often require re-allocation or full replanning (only instances where over robots were allocated on the failing team might recover through simple local replanning). On the other hand, robot losing capability and environment changes usually do not lead to severe failure impact, as a simple switch of assigned role or location of action could allow for local replanning.

Finally, as shown in Figure 6.19, most recovery mechanisms are completed significantly faster, and most failures lead to local replanning of the failed team. Our failure handling strategy demonstrates high adaptability and robust recovery output for various failure types while recovery times remain below full mission re-synthesis, validating the modular and adaptive architecture.

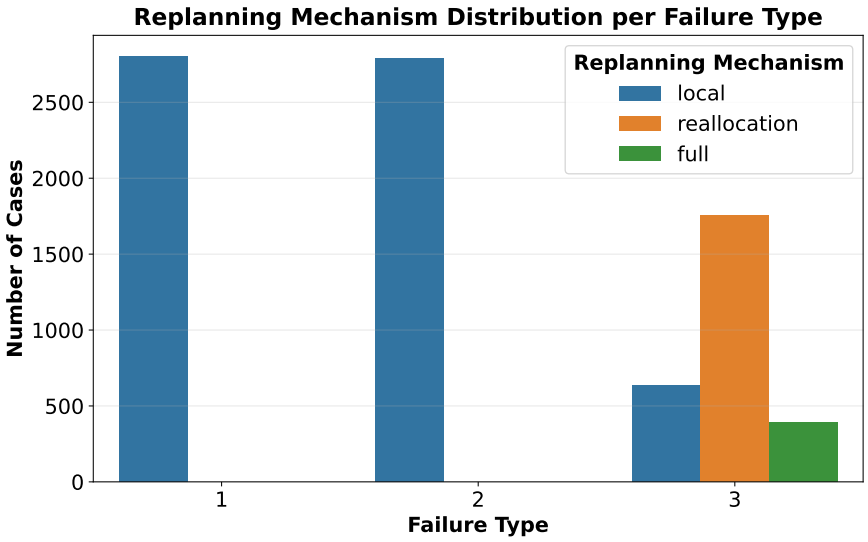


Figure 6.18: Replanning mechanism distribution per failure type. Robot crashes (Failure Type 3) mostly trigger reallocation. Robot's capability loss (Failure Type 2) or environmental changes (Failure Type 1) usually leads to local replanning.

Summary Across all scenarios, the modular decomposition and temporal supervision yield consistent planning times at scale and predominantly local recovery under injected failures. The observed planning and recovery latencies are compatible with ROS 2 real-time operation on multi-robot teams, further supporting the deployability of the architecture beyond simulation.

6.3.2.4 Toward Real-World Deployment

While our evaluation is simulation-based, the simulator mirrors deployed ROS 2 interfaces (actions, topics, services; PlanSys2-compatible), and execution parameters were derived from platform specifications and field observations. AMA-Exec emphasizes executability: the STN-Controller absorbs runtime drift via conservative margins and delay injection, and the DEM localizes replanning to the minimum affected scope. These mechanisms target field realities such as network delays and packet loss (which can desynchronize collaborative actions), sensor noise, partial capability loss (e.g., sampling or relay), and asymmetric costs at air–water transitions. AMA acts as a supervisory layer on top of perception and navigation, maintaining symbolic and temporal coherence even when lower-level behaviors are imperfect.

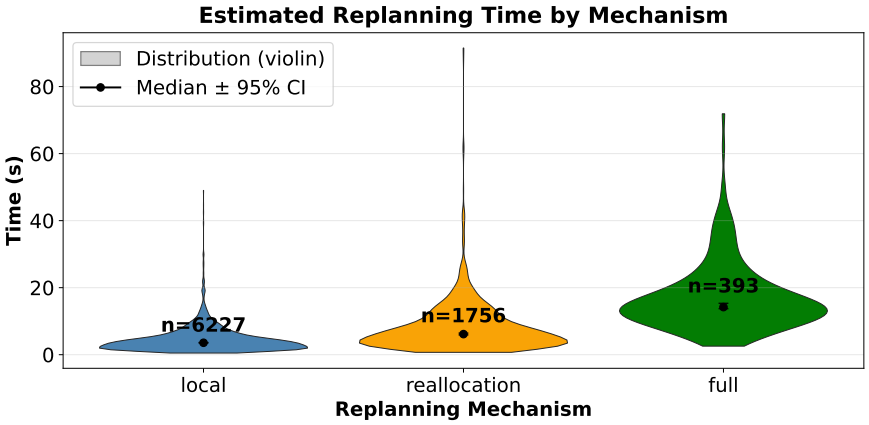


Figure 6.19: Distribution of replanning times across 9,720 injected failures, shown per mechanism as violin plots with median and confidence intervals. Local replanning typically completes within $\leq$5–10 s, reallocations take slightly longer but remain mostly under 20 s, while full replanning exhibits the longest tail and more occasional high-latency cases.

Chapter 7

Conclusion

This thesis was motivated by the emergence of trans-media robotic systems, capable of operating seamlessly across different environments such as air, ground, and water. These hybrid capabilities open new mission opportunities in domains ranging from environmental monitoring to infrastructure inspection and large-scale exploration.

From a planning perspective these challenges expose the limitations of classical MRTA resolution with PDDL-based solvers, which tend to struggle when applied to large-scale missions with tight temporal coupling. From an execution perspective, existing robotic middleware and behavior tree frameworks generally lack the temporal monitoring and recovery mechanisms needed to ensure robust mission execution in such complex domains.

The overarching problem addressed in this thesis is therefore twofold:

- how to design a planning system that scales to large trans-media missions while preserving temporal feasibility and efficiency in terms of planning time
- how to execute such plans in a distributed and resilient manner that can absorb delays, drifts, and failures without resorting systematically to costly full replanning.

Tackling these problems required bridging symbolic mission planning, temporal reasoning, and distributed execution within a single, modular architecture.

In order to address the challenges of mission planning and execution in complex trans-media multi-robot systems, this thesis proposes the Adaptive Modular Architecture (AMA), a generalizable framework designed to integrate scalable planning with resilient distributed execution. The contributions of this work can be summarized along four main axes.

First, we introduced AMA-Plan, a modular planning approach that combines symbolic problem decomposition and restructuring with temporal and coalition reasoning. AMA-Plan automatically divides missions into a series of organized and tractable subproblems. It employs STN to ensure synchronization, thereby reduc-

ing planning times while maintaining temporal feasibility in large-scale multi-robot scenarios. As demonstrated in simulation, AMA-Plan provided significant scalability advantages over classical PDDL solvers, while maintaining tractability and solution quality in missions involving dozens of robots and sites leading to hundreds of goals.

In a second step, we developed AMA-Exec, a distributed execution architecture that extends the capabilities of the PlanSys2 system. AMA-Exec integrates symbolic execution with execution-level temporal supervision through an STN-based controller, a Team Lifecycle Manager, and distributed BT executors. This architecture ensures that plans generated by AMA-Plan can be executed coherently across multiple teams of robots while maintaining resilience to various impediments. A notable feature of AMA-Exec is the introduction of a rapid, constraint-aware recovery mechanism that identifies the impact of failures on the current mission's plan, thereby reducing the level of replanning necessary.

Third, this thesis demonstrated the integration of AMA-Plan and AMA-Exec into a coherent planning–acting framework. AMA offers a method for linking mission-level decomposition with execution-level supervision by maintaining both symbolic and temporal consistency across the planning and execution layers. This integration facilitates the effective execution of complex missions by multi-robot teams, while also ensuring the continuity of operations in the face of unanticipated disturbances.

Finally, we conducted a comprehensive empirical evaluation of the proposed architecture across diverse simulated environments and mission scenarios. The results highlight AMA-Plan's scalability advantage over monolithic solvers, and AMA-Exec's capacity to absorb the majority of execution failures through partial recovery strategies. Together, these findings emphasize the value of modularity and temporal reasoning as foundational principles for robust multi-robot planning and execution in trans-media domains.

Although mission efficiency is important, this thesis has primarily focused on robust execution under uncertainty. Since our missions involve non-deterministic action durations and medium-dependent risks, our objective was to design a system capable of absorbing delays and maintaining continuity of operation rather than enforcing strict timing constraints.

Lessons This work highlights three central lessons. First, modularity is key to scalability and robustness, both in planning (through decomposition) and in execution (through distributed supervision).

Second, temporal reasoning is indispensable for synchronizing teams and managing delays or drifts. Finally, effective failure handling often requires only local adjustments, with higher level replanning serving as a fallback. Together, these lessons emphasize the value of a clear separation of concerns between planning and execution, while ensuring their consistency through shared temporal reasoning.

Limitations This thesis represents a step toward comprehensive mission planning and execution in complex multi-robot systems. Nevertheless, it is important to acknowledge some of the limitations.

A first limitation concerns the assumptions on robot knowledge and feedback. The architecture relies on accurate models of robot capabilities and consistent simulation feedback. While this assumption simplifies integration and evaluation, its applicability is restricted to scenarios where information is incomplete, noisy, or partially observable. The extension of AMA to address the uncertainty and partiality inherent in information and knowledge remains an unresolved challenge. Additionally, as the system is capable of comprehending the knowledge and actions of simulated robots at any given moment, decisions are made exclusively by the monolithic DEM. Although this is pivotal to the centralized management of subproblems in distributed teams, extending AMA to enable team-level decision-making at a lower level could result in a more equitable distribution of decision-making resources.

A second limitation lies in the scope of validation. The evaluation presented in this thesis was primarily conducted in simulation, albeit on large-scale and diverse scenarios. Simulation facilitates systematic testing and extensive experimental campaigns; however, it is incapable of entirely replicating the complexity and unpredictability characteristic of real-world deployments for which the system was designed. In the context of real-world tests, the true non-deterministic nature of robots actions necessitates the incorporation of more robust monitoring and adaptation components within its control system. Demonstrating the ability to do so in field experiments with actual hybrid robotic platforms will be a necessary step toward operational maturity.

In addition to simulation, the architecture has now reached a level of maturity that should enables its direct deployment on real robotic platforms. The current ROS 2-native implementation of AMA-Exec, including its distributed Behavior Tree executors through PlanSys2 (already tested on real-robots) and temporal supervision modules, has been integrated and tested in ROS 2 environments. These components have demonstrated the ability to handle real execution feed-

back and timing uncertainties without modification to the symbolic or temporal layers, confirming the readiness of the system for field experiments. Accordingly, future work will focus on validating AMA in more realistic multi-robot missions to quantitatively assess execution latency, robustness, and adaptation under real-world conditions.

Thirdly, the current framework provides only limited consideration of human supervision. In the course of designing the pollution scenario and in our preliminary approaches, we endeavored to incorporate a human-in-the-loop decision for the data sampling completion request and the determination of the site's importance and the sequence of visits. While the importance of human operators in complex mission contexts has been acknowledged, their integration was not addressed beyond initial design perspectives. A more comprehensive human-in-the-loop approach, enabling operators to monitor, guide, and intervene in planning and execution, remains an area of active research.

Beyond technical contributions, this thesis advances the understanding and modeling of automated planning and acting for trans-media robots. Analyzing the mechanical challenges they face, such as medium transitions, limited communication, and heterogeneous capability evolution, reveals structural discontinuities that cannot be reduced to simple capability constraints. Based on this analysis, we propose a structured representation of trans-media missions in terms of subproblems and synchronization requirements. This modeling effort provides a foundation for scalable planning and execution and retroactively informs the design requirements of trans-media robots. For example, our study shows that cooperation mechanisms (including integrating dual communication modalities (e.g., acoustic for underwater and radio for aerial/surface) and coordinating task sharing) are central elements of autonomy for these platforms, not optional add-ons.

Finally, although AMA integrates temporal reasoning and monitoring through STNs, several challenges remain in the temporal-symbolic interface. More advanced mechanisms, such as the STN-BT builder recently added to PlanSys2, have not yet been exploited. Furthermore, the transformation of symbolic PDDL plans into BTs introduces non-negligible overhead: while BT synthesis preserves causal structure and facilitates monitoring, it can become slow for long or temporally rich plans with many synchronization points. These limitations impede responsiveness in scenarios necessitating frequent replanning or partial plan updates. The exploration of planning-during-acting strategies, wherein AMA-Plan functions as an online component of execution rather than a snapshot planner, in

conjunction with more efficient plan-to-BT pipelines or hybrid execution structures, has the potential to markedly enhance robustness and responsiveness. Similarly, the AMA does not currently incorporate learning methods that could enable adaptation to recurring drifts or dynamic optimization of execution strategies.

Future research directions These limitations outline the boundaries of the present work while also pointing to promising directions for future research.

A first direction is the generalization of constraint extraction and restructuring in AMA-Plan. At present, constraints and weights reflecting domain knowledge are manually specified and then leveraged by the planner. Automating this process by analyzing action preconditions and effects would allow AMA-Plan to derive domain-specific restructuring strategies autonomously, leading to more scalable and adaptable planning in new environments.

A second perspective involves human-in-the-loop mission supervision. In some realistic scenarios, domain experts (e.g., environmental scientists) are expected to validate data and influence the prioritization of sites to visit. Embedding this kind of decision making into AMA could be achieved by allowing human operators to adjust heuristic weights for restructuring or task priorities dynamically, thereby shaping both planning and execution. Such integration would improve mission relevance while preserving autonomy at the execution level.

A third direction concerns the development of a universal aquatic–aerial–terrestrial simulation environment for hybrid missions. While AMA has been validated in diverse simulation settings, current tools (especially in ROS 2) lack the ability to realistically model trans-media missions where robots transition between mediums. Building such an environment would enable more realistic evaluation of AMA-Plan and AMA-Exec.

Beyond these concrete directions, the future development of AMA also involves:

- Integrating advanced temporal-symbolic mechanisms such as the STN-BT builder,
- Reducing the overhead of PDDL-to-BT transformation,
- Automating site designation in broader domains, as AMA-Plan uses coalition and goal regrouping to distribute allocation, the automatic construction of sites regrouping mission goals would allow the system to be used on simpler application domains.

Together, these directions aim to consolidate AMA as a general-purpose architecture for resilient, large-scale multi-robot planning and execution.

As a final summary, this thesis has demonstrated the benefits of integrating modularity, temporal reasoning, and distributed supervision into a cohesive architecture to address the challenges posed by complex multi-robot missions. Although the conceptualization of AMA occurred within the paradigm of trans-media robotics, the foundational principles established therein have demonstrated a degree of applicability across a more extensive array of robotic systems, typified by heterogeneity and substantial scale. The tight coupling between AMA-Plan and AMA-Exec has also been reinforced through a direct symbolic–temporal interface, enabling plans produced by the planner to be executed by the same distributed architecture with minimal translation overhead. This end-to-end coherence, recently demonstrated in our ROS 2 implementation, confirms the operational feasibility of the planning–execution pipeline envisioned in this thesis. By integrating symbolic planning with resilient execution, AMA establishes the foundation for unified frameworks that enable future autonomous systems to coordinate effectively, and adapt robustly in demanding real-world environments.

Bibliography

Allen, James F (1981). "An Interval-Based Representation of Temporal Knowledge." In: *International Joint Conference on Artificial Intelligence*. URL: http://www.ijcai.org/Past%20Proceedings/IJCAI-81-VOL%201/PDF/045.pdf (Cited on page 107).

Alzu'bi, Hamzeh, Iyad Mansour, and Osamah. Rawashdeh (Aug. 2018). "Loon Copter: Implementation of a hybrid unmanned aquatic–aerial quadcopter with active buoyancy control". en. In: *Journal of Field Robotics* 35.5, pp. 764–778. DOI: 10.1002/rob.21777 (Cited on pages 2, 18, 19).

Baines, Robert, Sree Kalyan Patiballa, Joran Booth, Luis Ramirez, Thomas Sipple, Andonny Garcia, Frank Fish, and Rebecca Kramer-Bottiglio (Oct. 2022). "Multi-environment robotic transitions through adaptive morphogenesis". en. In: *Nature* 610.7931, pp. 283–289. DOI: 10.1038/s41586-022-05188-w (Cited on pages 14, 15).

Belbachir, Assia, Félix Ingrand, and Simon Lacroix (Apr. 2012). "A cooperative architecture for target localization using multiple AUVs". In: *Intelligent Service Robotics* 5.2, pp. 119–132. DOI: 10.1007/s11370-012-0107-1 (Cited on page 61).

Benton, J., Amanda Coles, and Andrew Coles (May 2012). "Temporal Planning with Preferences and Time-Dependent Continuous Costs". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 22, pp. 2–10. DOI: 10.1609/icaps.v22i1.13509 (Cited on pages 37, 44, 52, 92, 128).

Bi, Yuanbo, Yufei Jin, Chenxin Lyu, Zheng Zeng, and Lian Lian (July 2022). "Nezha-Mini: Design and Locomotion of a Miniature Low-Cost Hybrid Aerial Underwater Vehicle". In: *IEEE Robotics and Automation Letters* 7.3, pp. 6669–6676. DOI: 10.1109/LRA.2022.3176438 (Cited on page 18).

Bit-Monnot, Arthur (2023). "Experimenting with Lifted Plan-Space Planning as Scheduling: Aries in the 2023 IPC". In: *2023 International Planning Competition at the 33rd International Conference on Automated Planning and Scheduling* (Cited on pages 92, 128).

Bit-Monnot, Arthur, Malik Ghallab, Félix Ingrand, and David E Smith (2020). "FAPE: a Constraint-based Planner for Generative and Hierarchical Temporal Planning". In: *arXiv.org* (Cited on page 36).

Brenner, Michael and Bernhard Nebel (2006). "A Multiagent Planning Language". In: *European Conference on Artificial Intelligence*, pp. 418–422 (Cited on page 35).

— (June 2009). "Continual planning and acting in dynamic multiagent environments". English. In: *Autonomous Agents and Multi-Agent Systems* 19.3, pp. 297–331. DOI: 10.1007/s10458-009-9081-1 (Cited on page 60).

Bylander, Tom (1994). "The computational complexity of propositional STRIPS planning". In: *Artificial Intelligence* 69.1-2, pp. 165–204 (Cited on page 35).

Carreno, Yaniel, Èric Pairet, Yvan Petillot, and Ronald P. A. Petrick (May 2020). "Task Allocation Strategy for Heterogeneous Robot Teams in Offshore Missions". In: *In-*

ternational Conference on Autonomous Agents and MultiAgent Systems. Richland, SC, pp. 222–230. ISBN: 9781450375184 (Cited on pages 4, 45, 48, 52).

Cashmore, Michael, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras (2015). "ROSPlan: Planning in the Robot Operating System". In: *International Conference on Automated Planning and Scheduling*. DOI: 10.1609/icaps.v25i1.13699 (Cited on pages 49, 53).

Chen, Xinye, Ping Zhang, Guanglong Du, and Fang Li (Aug. 2019). "A distributed method for dynamic multi-robot task allocation problems with critical time constraints". en. In: *Robotics and Autonomous Systems* 118, pp. 31–46. DOI: 10.1016/j.robot.2019.04.012 (Cited on page 20).

Chen, Yufeng et al. (Oct. 2017). "A biologically inspired, flapping-wing, hybrid aerial-aquatic microrobot". en. In: *Science Robotics* 2.11. DOI: 10.1126/scirobotics.aao5619 (Cited on pages 18, 20, 21).

Coles, Amanda, Andrew Coles, Maria Fox, and Derek Long (2010). "Forward-Chaining Partial-Order Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (Cited on pages 37, 52, 92, 128).

Colledanchise, Michele and Petter Ögren (2018). *Behavior Trees in Robotics and AI*. CRC Press. DOI: 10.1201/9780429489105 (Cited on page 50).

Crespi, Alessandro and Auke Jan Ijspeert (2009). "Salamandra Robotica: A Biologically Inspired Amphibious Robot that Swims and Walks". en. In: *Artificial Life Models in Hardware*. Ed. by Andrew Adamatzky and Maciej Komosinski. London: Springer London, pp. 35–64. DOI: 10.1007/978-1-84882-530-7_3 (Cited on pages 14, 15).

Daler, Ludovic, Stefano Mintchev, Cesare Stefanini, and Dario Floreano (Jan. 2015). "A bioinspired multi-modal flying and walking robot". In: *Bioinspiration & Biomimetics* 10.1, p. 016005. DOI: 10.1088/1748-3190/10/1/016005 (Cited on pages 14, 15).

De La Rochefoucauld, Virgile, Simon Lacroix, Photchara Ratsamee, and Haruo Takemura (2024). "Solving Multi-Robot Task Allocation and Planning in Trans-media Scenarios". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. DOI: 10.1109/IROS58592.2024.10801394 (Cited on page 8).

Dechter, Rina, Itay Meiri, and Judea Pearl (1991). "Temporal constraint networks". In: *Artificial Intelligence* 49.1-3, pp. 61–95. DOI: 10.1016/0004-3702(91)90006-6 (Cited on pages 52, 71).

Despouys, Olivier and Félix Ingrand (1999). "Propice-Plan: Toward a Unified Framework for Planning and Execution". In: *European Workshop on Planning*. DOI: 10.1007/10720246_22 (Cited on page 49).

Di Rocco, Maurizio, Federico Pecora, and Alessandro Saffiotti (2013). "When robots are late: Configuration planning for multiple robots with dynamic goals". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. DOI: 10.1109/IROS.2013.6697214 (Cited on page 61).

Dias, M.B., R. Zlot, N. Kalra, and A. Stentz (2006). "Market-Based Multirobot Coordination: A Survey and Analysis". In: *Proceedings of the IEEE* 94.7, pp. 1257–1270. DOI: 10.1109/JPROC.2006.876939 (Cited on page 43).

Do, Minh B. and Subbarao Kambhampati (2003). "LP-RPG: Leveraging Planning Graphs with Linear Programming". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (Cited on page 37).

Drews, Paulo L. J., Armando Alves Neto, and Mario F. M. Campos (Sept. 2014). "Hybrid Unmanned Aerial Underwater Vehicle: Modeling and simulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4637–4642. DOI: 10.1109/IROS.2014.6943220 (Cited on page 19).

Erol, Kutluhan, James Hendler, and Dana S Nau (1994). "HTN Planning: Complexity and Expressivity". In: *AAAI*, pp. 1123–1128 (Cited on page 38).

Estrada, Matthew A., Stefano Mintchev, David L. Christensen, Mark R. Cutkosky, and Dario Floreano (Oct. 2018). "Forceful manipulation with micro air vehicles". en. In: *Science Robotics* 3.23, eaau6903. DOI: 10.1126/scirobotics.aau6903 (Cited on pages 14, 15).

Fikes, Richard E. and Nils J. Nilsson (1971). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3–4, pp. 189–208 (Cited on page 33).

Firby, R James (1987). "An investigation into reactive planning in complex domains". In: *AAAI Conference*. Seattle, WA. URL: http://www.aaai.org/Papers/AAAI/1987/AAAI87-036.pdf (Cited on page 49).

Fox, Maria and Derek Long (Dec. 2003). "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20, pp. 61–124. DOI: 10.1613/jair.1129 (Cited on page 34).

— (2006). "Modelling mixed discrete-continuous domains for planning". In: *Journal of Artificial Intelligence Research* 27, pp. 235–297 (Cited on pages 34, 35).

Fu, Zhangjie, Yiming Chen, Yongjie Ding, and Daojing He (2019). "Pollution Source Localization Based on Multi-UAV Cooperative Communication". In: *IEEE Access* 7, pp. 29304–29312. DOI: 10.1109/ACCESS.2019.2900475 (Cited on page 4).

Gazen, B. C. and C. L. Knoblock (1999). "Combining the Expressiveness of UCPOP with the Efficiency of Graphplan". In: *IJCAI-99 Workshop on Scheduling and Planning* (Cited on page 37).

Gerevini, Alfonso, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos (2009). "PDDL3: An extension to PDDL 2.1 for expressing preferences and soft constraints". In: *Technical Report IC-2009-01, Dept. of Information Engineering, University of Brescia* (Cited on page 34).

Gerevini, Alfonso and Ivan Serina (2002). "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs." In: *Aips*. Vol. 2, pp. 281–290. URL: https://dl.acm.org/doi/10.5555/3036884.3036887 (Cited on pages 92, 128).

Gerkey, Brian P. and Maja J. Matarić (2003). "Multi-robot task allocation: analyzing the complexity and optimality of key architectures". In: *IEEE International Conference on Robotics and Automation*. Taipei, Taiwan. DOI: 10.1109/ROBOT.2003.1242189 (Cited on page 3).

Gerkey, Brian P. and Maja J. Matarić (Sept. 2004). "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". en. In: *The International Journal of Robotics Research* 23.9, pp. 939–954. DOI: 10.1177/0278364904045564 (Cited on pages 3, 40, 65).

Ghallab, Malik, Craig Knoblock, et al. (Aug. 1998). *PDDL - The Planning Domain Definition Language*. Tech. rep. AIPS. URL: https://engineering.purdue.edu/~relation/surf/papers/pddl.pdf (Cited on page 51).

Ghallab, Malik and H Laruelle (1994). "Representation and Control in IxTeT, a Temporal Planner". In: *International Conference on AI Planning Systems*, pp. 61–67 (Cited on pages 36, 49).

Ghallab, Malik, Dana S Nau, and Paolo Traverso (2016). *Automated Planning and Acting*. Cambridge University Press. DOI: 10.1017/CBO9781139583923. URL: https://projects.laas.fr/planning/ (Cited on pages 51, 53, 61).

Ghzouli, Razan, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski, and Swaib Dragule (2023). "Behavior trees and state machines in robotics applications". In: *IEEE Transactions on Software Engineering* 49.9, pp. 4243–4267. DOI: 10.1109/TSE.2023.3269081 (Cited on page 61).

Hassanalian, M. and A. Abdelkefi (2017). "Classifications, applications, and design challenges of drones: A review". In: *Progress in Aerospace Sciences* 91, pp. 99–131. DOI: https://doi.org/10.1016/j.paerosci.2017.04.003 (Cited on page 17).

Helmert, Malte (2006). "The Fast Downward Planning System". In: *International Conference on Automated Planning and Scheduling*, pp. 151–154 (Cited on page 36).

Hoffmann, Jorg and Bernhard Nebel (2001). "The FF planning system: Fast plan generation through heuristic search". In: *Journal of Artificial Intelligence Research*. Vol. 14, pp. 253–302 (Cited on page 36).

Höller, Daniel, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford (2020). "HDDL: An Extension To PDDL for Expressing Hierarchical Planning Problems". In: *International Joint Conference on Artificial Intelligence*, pp. 9883–9891. DOI: 10.24963/ijcai.2020/137 (Cited on pages 35, 38).

Howey, Richard, Derek Long, and Maria Fox (2004). "VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL". In: *International Conference on Tools with Artificial Intelligence*, pp. 294–301. DOI: 10.1109/ICTAI.2004.120 (Cited on page 113).

Ijspeert, Auke Jan, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen (Mar. 2007). "From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model". en. In: *Science* 315.5817, pp. 1416–1420. DOI: 10.1126/science.1138353 (Cited on pages 2, 14).

Ingrand, Félix (2023). *ProSkill*. Version 1.0b0. URL: https://laas.hal.science/hal-04738223 (Cited on pages 50, 51, 53).

— (Oct. 2024). *BT2Fiacre (Behavior Tree 2 Fiacre)*. Version 1.0b0. URL: https://laas.hal.science/hal-04720141 (Cited on page 50).

Ingrand, Félix, Raja Chatilla, and Rachid Alami (2001). "An Architecture for Dependable Autonomous Robots". In: *IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments* (Cited on page 52).

Ingrand, Félix, Michael Georgeff, and Anan Rao (1992). "An architecture for real-time reasoning and system control". In: *IEEE Expert* 7.6, pp. 34–44. DOI: 10.1109/64.180407 (Cited on page 49).

Jeon, Sang Yeob, In Seok Yang, Byung Hoon Chi, Han Ul Yoo, and Hyouk Ryeol Choi (2022). "Behavior Tree-Based Task Planning for Multiple Mobile Robots Using a Data Distribution Service". In: *Electronics* 11.4, p. 601. DOI: 10.3390/electronics11040601 (Cited on pages 50, 61).

Jin, Yufei, Yuanbo Bi, Chenxin Lyu, Yulin Bai, Zheng Zeng, and Lian Lian (Mar. 2024). "Nezha-IV: A hybrid aerial underwater vehicle in real ocean environments". en. In: *Journal of Field Robotics* 41.2, pp. 420–442. DOI: 10.1002/rob.22274 (Cited on page 18).

Jin, Yufei, Zheng Zeng, and Lian Lian (Jan. 2025). "Nezha-SeaDart: A tail-sitting fixed-wing vertical takeoff and landing hybrid aerial underwater vehicle". en. In: *Journal of Field Robotics* 42.1, pp. 137–152. DOI: 10.1002/rob.22399 (Cited on page 19).

Joyeux, Sylvain, Rachid Alami, Simon Lacroix, and R. Philippsen (2009). "A plan manager for multi-robot systems". In: *The International Journal of Robotics Research* 28.2, p. 220. DOI: 10.1177/0278364908098370 (Cited on page 60).

Kaelbling, Leslie Pack and Tomas Lozano-Perez (2011). "Hierarchical task and motion planning in the now". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1470–1477 (Cited on pages 46, 48).

Kautz, Henry A. and Bart Selman (1992). "Planning as Satisfiability". In: *European Conference on Artificial Intelligence (ECAI)*, pp. 359–363 (Cited on page 37).

Khamis, Alaa, Ahmed Hussein, and Ahmed Elmogy (2015). "Multi-robot Task Allocation: A Review of the State-of-the-Art". en. In: *Cooperative Robots and Sensor Networks 2015*. Ed. by Anis Koubâa and J.Ramiro Martı´nez-de Dios. Cham: Springer International Publishing, pp. 31–51. DOI: 10.1007/978-3-319-18299-5_2 (Cited on pages 42, 44, 45).

Kim, Kyunam, Patrick Spieler, Elena-Sorina Lupu, Alireza Ramezani, and Soon-Jo Chung (Oct. 2021). "A bipedal walking robot that can fly, slackline, and skateboard". en. In: *Science Robotics* 6.59, eabf8136. DOI: 10.1126/scirobotics.abf8136 (Cited on page 2).

Kim, Sungho, Yeongjoon Jo, and Dong-Ok Kwak (2016). "Task planning for multi-robot systems using PDDL with optimization objectives". In: *IEEE/SICE International Symposium on System Integration (SII)*, pp. 388–393 (Cited on pages 46, 48).

Korsah, G. Ayorkor, Anthony Stentz, and M. Bernardine Dias (Oct. 2013). "A comprehensive taxonomy for multi-robot task allocation". en. In: *The International Journal of Robotics Research* 32.12, pp. 1495–1512. DOI: 10.1177/0278364913496484 (Cited on page 42).

Kumar, Manoj and Pratap Padhy (Dec. 2015). "Environmental Perspectives of Pond Ecosystems: Global Issues, Services and Indian Scenarios". In: *Current World En-*

vironment 10.3, pp. 848–867. DOI: 10.12944/CWE.10.3.16 (Cited on pages 1, 5).

Lee, Jaeho, Marcus J Huber, Edmund H Durfee, and Patrick G Kenny (1994). "UM-PRS: An Implementation Of The Procedural Reasoning System For Multirobot Applications". In: *AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*. NASA, pp. 842–842. URL: http://www.researchgate.net/publication/2656892_UM-PRS_An_Implementation_Of_The_Procedural_Reasoning_System_For_Multirobot_Applications/file/60b7d5230773e39700.pdf (Cited on page 49).

Lemai-Chenevier, Solange (2004). "IxTeT-eXeC: planning, plan repair and execution control with time and resource management". PhD thesis. Institut National Polytechnique Toulouse. URL: https://theses.hal.science/tel-00009496/ (Cited on page 52).

Lesire, Charles, David Doose, and Christophe Grand (Oct. 2020). "Formalization of Robot Skills with Descriptive and Operational Models". In: *International Conference on Intelligent Robots and Systems*, pp. 1–6. DOI: 10.1109/IROS45743.2020.9340698 (Cited on pages 50, 51, 53).

Lesire, Charles and Franck Pommereau (Sept. 2018). "ASPiC: an Acting system based on Skill Petri net Composition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1–7. DOI: 10.1109/IROS.2018.8594328 (Cited on page 51).

Li, Lei et al. (May 2022). "Aerial-aquatic robots capable of crossing the air-water boundary and hitchhiking on surfaces". en. In: *Science Robotics* 7.66, eabm6695. DOI: 10.1126/scirobotics.abm6695 (Cited on pages 18, 19).

Liang, Jianhong, Xingbang Yang, Tianmiao Wang, Guocai Yao, and Wendi Zhao (Sept. 2013). "Design and Experiment of a Bionic Gannet for Plunge-Diving". en. In: *Journal of Bionic Engineering* 10.3, pp. 282–291. DOI: 10.1016/S1672-6529(13)60224-3 (Cited on pages 20, 21).

Liu, Xuchen, Minghao Dou, Dongyue Huang, Biao Wang, Jinqiang Cui, Qinyuan Ren, Lihua Dou, Zhi Gao, Jie Chen, and Ben M. Chen (Feb. 2023). *TJ-FlyingFish: Design and Implementation of an Aerial-Aquatic Quadrotor with Tiltable Propulsion Units*. DOI: 10.48550/arXiv.2301.12344 (Cited on page 20).

Macenski, Steve, Ruffin White, and Joshua Wallace (2024). *Nav2 Behavior Trees*. URL: https://docs.nav2.org/behavior_trees/index.html (Cited on page 56).

Martı´n, Francisco, Jonatan Ginés Clavero, Vicente Matellán Olivera, and Francisco J Rodrı´guez (2021). "PlanSys2: A planning system framework for ROS2". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. DOI: 10.1109/IROS51168.2021.9636544 (Cited on pages 49, 53).

Martı´n, Francisco, Matteo Morelli, Huascar Espinoza, Francisco J. Rodrı´guez-Lera, and Vicente Matellán Olivera (2021). "Optimized Execution of PDDL Plans Using Behavior Trees". In: *International Conference on Autonomous Agents and Multi-*

Agent Systems. Virtual Event, United Kingdom, pp. 1596–1598. DOI: 10.5555/3463952.3464171 (Cited on page 54).

Martynov, Mikhail, Zhanibek Darush, Aleksey Fedoseev, and Dzmitry Tsetserukou (Mar. 2024). *MorphoGear: An UAV with Multi-Limb Morphogenetic Gear for Rough-Terrain Locomotion*. arXiv:2403.08340. DOI: 10.48550/arXiv.2403.08340 (Cited on page 2).

Mayer, Marta Cialdea, Andrea Orlandini, and Alessandro Umbrico (Dec. 2015). "Planning and execution with flexible timelines: a formal account". In: *Acta Informatica*, pp. 1–32. DOI: 10.1007/s00236-015-0252-z (Cited on page 52).

McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). *PDDL – The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003. Yale University: Yale Center for Computational Vision and Control (Cited on page 34).

Micheli, Andrea et al. (2025). "Unified Planning: Modeling, manipulating and solving AI planning problems in Python". In: *SoftwareX* 29, p. 102012. DOI: 10.1016/j.softx.2024.102012 (Cited on pages 92, 128).

Muscettola, Nicola, P Pandurang Nayak, B Pell, and Brian C Williams (1998). "Remote Agent: to boldly go where no AI system has gone before". English. In: *Artificial Intelligence* 103, pp. 5–47. URL: http://citeseer.ist.psu.edu/134737.html (Cited on page 49).

Musliner, David J and Edmund H Durfee (1993). "CIRCA: a cooperative intelligent real-time control architecture". English. In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.6, pp. 1561–1574. DOI: 10.1109/21.257754 (Cited on pages 49, 52).

Nau, Dana S, Thomas Au, Omar Ilghami, Ugur Kuter, James W Murdock, Daniel Wu, and Fusun Yaman (2003). "SHOP2: An HTN planning system". In: *Journal of Artificial Intelligence Research* 20, pp. 379–404 (Cited on page 38).

Patra, Sunandita, Malik Ghallab, Dana S Nau, and Paolo Traverso (Sept. 2019). "APE: An Acting and Planning Engine". In: *Advances in Cognitive Systems*, pp. 1–20. URL: https://univ-tlse2.hal.science/hal-01959098v1 (Cited on pages 49, 53).

Patra, Sunandita, James Mason, Malik Ghallab, Dana Nau, and Paolo Traverso (2021). "Deliberative acting, planning and learning with hierarchical operational models". In: *Artificial Intelligence* 299. DOI: 10.1016/j.artint.2021.103523 (Cited on pages 49, 53).

Pedroso, Adir A., Andressa C. Da Silva, Pedro M. Pinheiro, and Paulo L. J. Drews (Oct. 2022). "Prototyping and Construction of a Hybrid Unmanned Aerial Underwater Vehicles". In: *Latin American Robotics Symposium (LARS), Brazilian Symposium on Robotics (SBR), and Workshop on Robotics in Education (WRE)*. São Bernardo do Campo, Brazil: IEEE, pp. 61–66. DOI: 10.1109/LARS/SBR/WRE56824.2022.9995873 (Cited on pages 2, 18–20).

Pelletier, Baptiste, Charles Lesire, and Karen Godary-Dejean (2025). "A formal framework for the specification and verification of robotic skills composition for au-

tonomous behaviors". In: *Robotics and Autonomous Systems*, p. 105041. DOI: 10.1016/j.robot.2025.105041 (Cited on pages 50, 51, 53).

Penberthy, J. Scott and Daniel S. Weld (Oct. 25, 1992). "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *International Conference on Principles of Knowledge Representation and Reasoning*. KR'92. Foundational POCL planner - one of the first sound and complete partial-order planners. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 103–114 (Cited on page 36).

Richter, Silvia and Malte Westphal (2010). "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research*. Vol. 39, pp. 127–177 (Cited on page 36).

Rintanen, Jussi (2004). "Complexity of concurrent temporal planning". In: *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 280–288 (Cited on page 35).

Rockenbauer, Friedrich M. et al. (July 2021). "Dipper: A Dynamically Transitioning Aerial-Aquatic Unmanned Vehicle". en. In: *Robotics: Science and Systems*, p. 48. DOI: 10.15607/RSS.2021.XVII.048 (Cited on pages 20, 21).

Röger, Gabriele, Patrick Eyerich, and Robert Mattmüller (2008). "TFD: A numeric temporal extension to Fast Downward". In: *International Planning Competition (IPC)*. Syndey, Australia, p. 114 (Cited on pages 52, 92, 128).

Shi, Liwei, Shuxiang Guo, Shilian Mao, Chunfeng Yue, Maoxun Li, and Kinji Asaka (Oct. 2013). "Development of an Amphibious Turtle-Inspired Spherical Mother Robot". In: *Journal of Bionic Engineering* 10.4, pp. 446–455. DOI: 10.1016/S1672-6529(13)60248-6 (Cited on page 2).

Shin, Won Dong, Hoang-Vu Phan, Monica A. Daley, Auke J. Ijspeert, and Dario Floreano (2024). "Fast ground-to-air transition with avian-inspired multifunctional legs". In: DOI: 10.48550/ARXIV.2412.02389 (Cited on page 2).

Shkurti, Florian et al. (Oct. 2012). "Multi-domain monitoring of marine environments using a heterogeneous robot team". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1747–1753. DOI: 10.1109/IROS.2012.6385685 (Cited on pages 1, 4).

Stedl, John and Brian C Williams (2005). "Managing communication limitations in partially controllable multi-agent plans". In: *ICAPS Workshop on Multiagent Planning and Scheduling*, pp. 8–14. URL: https://groups.csail.mit.edu/mers/old-site/papers/HRA_ICAPS_Workshop.pdf (Cited on pages 52, 60).

Tan, Yu Herng and Ben M. Chen (Oct. 2019). "Design of a Morphable Multirotor Aerial-Aquatic Vehicle". In: *OCEANS 2019 MTS/IEEE SEATTLE*. Seattle, WA, USA: IEEE, pp. 1–8. DOI: 10.23919/OCEANS40490.2019.8962867 (Cited on pages 18, 19).

— (July 2021). "Survey on the Development of Aerial–Aquatic Hybrid Vehicles". en. In: *Unmanned Systems* 09.03, pp. 263–282. DOI: 10.1142/S2301385021410028 (Cited on pages 2, 16).

Turi, Jérémy and Arthur Bit-Monnot (2022). "Extending a Refinement Acting Engine for Fleet Management: Concurrency and Resources". In: *International Conference*

on Tools with Artificial Intelligence (ICTAI). DOI: 10.1109/ICTAI56018.2022.00216 (Cited on pages 49, 53).

Umbrico, Alessandro, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini (2017). "PLATINUm: A New Framework for Planning and Acting". In: *AI*IA*. Springer International Publishing. DOI: 10.1007/978-3-319-70169-1_37 (Cited on pages 50, 52, 61).

Umbrico, Alessandro, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini (Nov. 2019). "Evaluating Robustness of an Acting Framework over Temporally Uncertain Domains". In: *AI*IA Advances in Artificial Intelligence*. Springer International Publishing, pp. 1–14. DOI: 10.1007/978-3-030-35166-3_18 (Cited on pages 52, 53, 60).

Valentini, Alessandro, Andrea Micheli, and Alessandro Cimatti (2020). "Temporal planning with intermediate conditions and effects". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06, pp. 9975–9982 (Cited on page 92).

Vasilijevic, Antonio, Dula Nad, Filip Mandic, Nikola Miskovic, and Zoran Vukic (June 2017). "Coordinated Navigation of Surface and Underwater Marine Robotic Vehicles for Ocean Sampling and Environmental Monitoring". In: *IEEE/ASME Transactions on Mechatronics* 22.3, pp. 1174–1184. DOI: 10.1109/TMECH.2017.2684423 (Cited on page 1).

Weisler, Warren, William Stewart, Mark B. Anderson, Kara J. Peters, Ashok Gopalarathnam, and Matthew Bryant (Oct. 2018). "Testing and Characterization of a Fixed Wing Cross-Domain Unmanned Vehicle Operating in Aerial and Underwater Environments". In: *IEEE Journal of Oceanic Engineering* 43.4, pp. 969–982. DOI: 10.1109/JOE.2017.2742798 (Cited on pages 18, 19).

Yang, Xingbang, Tianmiao Wang, Jianhong Liang, Guocai Yao, and Miao Liu (Apr. 2015). "Survey on the novel hybrid aquatic–aerial amphibious aircraft: Aquatic unmanned aerial vehicle (AquaUAV)". en. In: *Progress in Aerospace Sciences* 74, pp. 131–151. DOI: 10.1016/j.paerosci.2014.12.005 (Cited on pages 2, 16).

Yao, Guocai, Yanze Li, Hanyi Zhang, Yaotong Jiang, Tianmiao Wang, Fuchun Sun, and Xingbang Yang (May 2023). "Review of hybrid aquatic-aerial vehicle (HAAV): Classifications, current status, applications, challenges and technology perspectives". en. In: *Progress in Aerospace Sciences* 139, p. 100902. DOI: 10.1016/j.paerosci.2023.100902 (Cited on pages 2, 4, 14, 16, 22, 23).

Younes, H. L.S. and R. G. Simmons (Dec. 1, 2003). "VHPOP: Versatile Heuristic Partial Order Planner". In: *Journal of Artificial Intelligence Research* 20. Improved POCL planner with heuristics and temporal reasoning capabilities, pp. 405–430. DOI: 10.1613/jair.1136. URL: https://jair.org/index.php/jair/article/view/10363 (Cited on page 36).

Zapf, Josh, Marco Roveri, Francisco Martin, and Juan Carlos Manzanares (2024). "Constructing Behavior Trees from Temporal Plans for Robotic Applications". In: *arXiv preprint*. DOI: 10.48550/arXiv.2406.17379 (Cited on pages 55, 60, 120).

Zheng, Xiaoming and Sven Koenig (2009). "Negotiation with reaction functions for solving complex task allocation problems". In: *2009 IEEE/RSJ International Con-*

ference on Intelligent Robots and Systems, pp. 4811–4816. DOI: 10.1109/IROS.2009.5354326 (Cited on page 44).

Appendix A

PDDL domains, problems and BT output

A.1 PDDL domains for cases 1–3

Listing A.1: Homogeneous PDDL domain

```
(define (domain HomogeneousDomain)
(:requirements :strips :typing :fluents :negative-preconditions :disjunctive-
    preconditions :durative-actions :universal-preconditions timed-initial-literals)

;; Types ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:types

robot
pointofinterest
site

);; end Types ;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Predicates ;;;;;;;;;;;;;;;;;;;;;;;;;;
(:predicates

;; site predicates

(observed ?s - site)
(beaconed ?s - site)

;; robot predicates

(at ?r - robot ?poi - pointofinterest)
(at_site ?r - robot ?s - site)
(airconf ?r - robot)
(groundconf ?r - robot)
(available ?r - robot)
(canobs ?r - robot)
(hasbeacons ?r - robot)
(camera_ready ?r - robot)

;; Poi predicates

(transition_poi ?poi - pointofinterest)
(survey_poi ?poi - pointofinterest)
(air ?poi - pointofinterest)
(checked ?poi - pointofinterest)
(ground ?poi - pointofinterest)
(partofsite ?poi - pointofinterest ?s - site)

```

```
41
42 );; end Predicates ;;;;;;;;;;;;;;;;;;;
43
44 ;; Functions ;;;;;;;;;;;;;;;;;;;;;;;;
45 (:functions
46 (distance ?poi1 ?poi2 - pointofinterest)
47
48 (site_size ?s - site)
49 (speedair ?r - robot)
50 (landingduration_airground ?r - robot)
51 (takeoffduration_groundair ?r - robot)
52
53 );; end Functions ;;;;;;;;;;;;;;;;;;;
54
55 ;; Actions ;;;;;;;;;;;;;;;;;;;;;;;;;;
56 (:durative-action navigation_air
57   :parameters (?r - robot ?poi1 ?poi2 - pointofinterest ?s - site)
58   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
59   :condition (and
60     (at start(available ?r))
61     (at start (partofsite ?poi1 ?s))
62     (at start (partofsite ?poi2 ?s))
63     (at start (at ?r ?poi1))
64     (at start (air ?poi1))
65     (at start (air ?poi2))
66     (at start (airconf ?r))
67   )
68   :effect (and
69       (at start (not (available ?r)))
70       (at start (not (at ?r ?poi1)))
71       (at end (at ?r ?poi2))
72       (at end (available ?r))
73     )
74 )
75
76 (:durative-action takeoff
77   :parameters (?r - robot ?poi - pointofinterest)
78   :duration ( = ?duration (takeoffduration_groundair ?r))
79   :condition (and
80     (at start (available ?r))
81     (at start (at ?r ?poi))
82     (at start (transition_poi ?poi))
83     (at start (groundconf ?r))
84     (at start (air ?poi))
85     (at start (ground ?poi))
86   )
87   :effect (and
88       (at start (not (available ?r)))
89       (at start (not (groundconf ?r)))
90       (at end (airconf ?r))
91     (at end (at ?r ?poi))
92       (at end (available ?r))
93     (at end (available ?r))
94     )
95 )
96
97 (:durative-action landing
98   :parameters (?r - robot ?poi - pointofinterest)
99   :duration ( = ?duration (landingduration_airground ?r))
100  :condition (and
101    (at start(available ?r))
102    (over all (at ?r ?poi))
```

```
    (over all (transition_poi ?poi))
    (at start (airconf ?r))
    (over all (air ?poi))
    (over all (ground ?poi))
  )
  :effect (and
       (at start (not (available ?r)))
       (at start (not (airconf ?r)))
       (at end (groundconf ?r))
       (at end (available ?r))
    )
)

(:durative-action observe
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration (/ (site_size ?s) (speedair ?r)))
  :condition (and
    (at start (at ?r ?poi))
    (at start (at_site ?r ?s))
    (at start (survey_poi ?poi))
    (at start (partofsite ?poi ?s))
    (at start (air ?poi))
    (at start (available ?r))
    (at start (airconf ?r))
  )
  :effect (and
    (at start (not(available ?r)))
    (at end (available ?r))
    (at end (observed ?s))
  )
)

(:durative-action observe_2r
  :parameters (?r1 ?r2 - robot ?poi - pointofinterest ?s - site)
  :duration (= ?duration (/ (/ (site_size ?s) (speedair ?r1)) 2))
  :condition (and
    (at start (at ?r1 ?poi))
    (at start (at_site ?r1 ?s))
    (at start (at ?r2 ?poi))
    (at start (at_site ?r2 ?s))
    (at start (survey_poi ?poi))
    (at start (partofsite ?poi ?s))
    (at start (air ?poi))
    (at start (available ?r1))
    (at start (available ?r2))
    (at start (airconf ?r1))
    (at start (airconf ?r2))
  )
  :effect (and
    (at start (not(available ?r1)))
    (at start (not(available ?r2)))
    (at end (available ?r1))
    (at end (available ?r2))
    (at end (observed ?s))
  )
)


(:durative-action check_transition_point
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration 30)
  :condition (and
```

```
    (over all (at ?r ?poi))
    (at start (available ?r))
    (over all (canobs ?r))
    (over all (airconf ?r))
    (over all (air ?poi))
    (over all (transition_poi ?poi))
    (over all (partofsite ?poi ?s))
    (over all (observed ?s))
  )
  :effect (and
    (at start (not(available ?r)))
    (at end (available ?r))
    (at end (checked ?poi))
  )
)

(:durative-action Pose_beacon
  :parameters (?r1 ?r2 - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration 30)
  :condition (and
    (over all (at ?r1 ?poi))
    (at start (available ?r1))
    (over all (at ?r2 ?poi))
    (at start (available ?r2))
    (at start (hasbeacons ?r1))
    (over all (airconf ?r1))
    (over all (airconf ?r2))
    (over all (air ?poi))
    (over all (transition_poi ?poi))
    (over all (partofsite ?poi ?s))
    (at start (observed ?s))
    (at start (checked ?poi))
    (over all (camera_ready ?r2))
  )
  :effect (and
    (at start (not(available ?r1)))
    (at start (not(available ?r2)))
    (at end (available ?r1))
    (at end (available ?r2))
    (at end (beaconed ?s))
  )
)

(:durative-action change_site
  :parameters (?r - robot ?s1 ?s2 - site ?poi1 ?poi2 - pointofinterest)
  :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
  :condition (and
    (at start (at_site ?r ?s1))
    (at start (at ?r ?poi1))
    (over all (air ?poi1))
    (over all (air ?poi2))
    (over all (partofsite ?poi1 ?s1))
    (over all (partofsite ?poi2 ?s2))
    (at start (available ?r))
    (over all (airconf ?r))
  )
  :effect (and
    (at start (not(available ?r)))
    (at start (not(at_site ?r ?s1)))
    (at start (not(at ?r ?poi1)))
    (at end (available ?r))
    (at end (at_site ?r ?s2))
```

```
    (at end (at ?r ?poi2))
  )
)


);; end Domain ;;;;;;;;;;;;;;;;;;;;;;;;;
```

Listing A.2: Heterogeneous PDDL domain

```
(define (domain MMdomainextended)
(:requirements :strips :typing :fluents :negative-preconditions :disjunctive-
    preconditions :durative-actions :universal-preconditions timed-initial-literals)

;; Types ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:types

robot
pointofinterest
site

);; end Types ;;;;;;;;;;;;;;;;;;;;;;;;;

;; Predicates ;;;;;;;;;;;;;;;;;;;;;;;;;
(:predicates

;; site predicates

(observed ?s - site)

;; robot predicates

(at ?r - robot ?poi - pointofinterest)
(at_site ?r - robot ?s - site)
(waterconf ?r - robot)
(airconf ?r - robot)
(groundconf ?r - robot)
(available ?r - robot)
(canrelay ?r - robot)
(cansample ?r - robot)
(canobs ?r - robot)

;; Poi predicates

(sample_poi ?poi - pointofinterest)
(transition_poi ?poi - pointofinterest)
(survey_poi ?poi - pointofinterest)
(water ?poi - pointofinterest)
(air ?poi - pointofinterest)
(ground ?poi - pointofinterest)
(checked ?poi - pointofinterest)
(connected ?s - site)
(partofsite ?poi - pointofinterest ?s - site)
(sampled ?poi - pointofinterest)
(isrelay ?poi - pointofinterest)

);; end Predicates ;;;;;;;;;;;;;;;;;;;;

;; Functions ;;;;;;;;;;;;;;;;;;;;;;;;;
(:functions
(distance ?poi1 ?poi2 - pointofinterest)

(site_size ?s - site)
```

```
53
54 (speedair ?r - robot)
55 (speedwater ?r - robot)
56
57 );; end Functions ;;;;;;;;;;;;;;;;;;;
58
59 ;; Actions ;;;;;;;;;;;;;;;;;;;;;;;;;;;
60 (:durative-action navigation_air
61   :parameters (?r - robot ?poi1 ?poi2 - pointofinterest ?s - site)
62   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
63   :condition (and
64     (at start(available ?r))
65     (at start (partofsite ?poi1 ?s))
66     (at start (partofsite ?poi2 ?s))
67     (at start (at ?r ?poi1))
68     (at start (air ?poi1))
69     (at start (air ?poi2))
70     (at start (airconf ?r))
71   )
72   :effect (and
73       (at start (not (available ?r)))
74       (at start (not (at ?r ?poi1)))
75       (at end (at ?r ?poi2))
76       (at end (available ?r))
77     )
78 )
79
80 (:durative-action navigation_water
81   :parameters (?r - robot ?poi1 ?poi2 - pointofinterest ?s - site)
82   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedwater ?r)))
83   :condition (and
84     (at start(available ?r))
85     (at start (partofsite ?poi1 ?s))
86     (at start (partofsite ?poi2 ?s))
87     (at start (at ?r ?poi1))
88     (at start (water ?poi1))
89     (at start (water ?poi2))
90     (at start (waterconf ?r))
91   )
92   :effect (and
93       (at start (not (available ?r)))
94       (at start (not (at ?r ?poi1)))
95       (at end (at ?r ?poi2))
96       (at end (available ?r))
97     )
98 )
99
100 (:durative-action observe
101   :parameters (?r - robot ?poi - pointofinterest ?s - site)
102   :duration ( = ?duration (/ (site_size ?s) (speedair ?r)))
103   :condition (and
104     (at start (at ?r ?poi))
105     (at start (at_site ?r ?s))
106     (at start (survey_poi ?poi))
107     (at start (partofsite ?poi ?s))
108     (at start (air ?poi))
109     (at start (available ?r))
110     (at start (airconf ?r))
111   )
112   :effect (and
113     (at start (not(available ?r)))
114     (at end (available ?r))
```

```
    (at end (observed ?s))
  )
)

(:durative-action observe_2r
  :parameters (?r1 ?r2 - robot ?poi - pointofinterest ?s - site)
  :duration (= ?duration (/ (/ (site_size ?s) (speedair ?r1)) 2))
  :condition (and
    (at start (at ?r1 ?poi))
    (at start (at_site ?r1 ?s))
    (at start (at ?r2 ?poi))
    (at start (at_site ?r2 ?s))
    (at start (survey_poi ?poi))
    (at start (partofsite ?poi ?s))
    (at start (air ?poi))
    (at start (available ?r1))
    (at start (available ?r2))
    (at start (airconf ?r1))
    (at start (airconf ?r2))
  )
  :effect (and
    (at start (not(available ?r1)))
    (at start (not(available ?r2)))
    (at end (available ?r1))
    (at end (available ?r2))
    (at end (observed ?s))
  )
)

(:durative-action translate_data
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration 45)
  :condition (and
    (over all (at ?r ?poi))
    (over all (transition_poi ?poi))
    (over all (partofsite ?poi ?s))
    (over all (observed ?s))
    (over all (checked ?poi))
    (over all (isrelay ?poi))
    (at start (available ?r))
    (over all (waterconf ?r))
    (over all (canrelay ?r))
  )
  :effect (and
    (at start (not (available ?r)))
    (at start (connected ?s))
    (at end (not (connected ?s)))
    (at end (available ?r))
  )
)

(:durative-action sample
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration 30)
  :condition (and
    (over all (at ?r ?poi))
    (at start (available ?r))
    (over all (cansample ?r))
    (over all (waterconf ?r))
    (over all (water ?poi))
    (over all (sample_poi ?poi))
    (over all (partofsite ?poi ?s))
```

```
177     (over all (connected ?s))
178   )
179   :effect (and
180     (at start (not(available ?r)))
181     (at end (available ?r))
182     (at end (sampled ?poi))
183   )
184 )
185
186 (:durative-action check_transition_point
187   :parameters (?r - robot ?poi - pointofinterest ?s - site)
188   :duration ( = ?duration 30)
189   :condition (and
190     (over all (at ?r ?poi))
191     (at start (available ?r))
192     (over all (canobs ?r))
193     (over all (airconf ?r))
194     (over all (air ?poi))
195     (over all (transition_poi ?poi))
196     (over all (partofsite ?poi ?s))
197     (over all (observed ?s))
198   )
199   :effect (and
200     (at start (not(available ?r)))
201     (at end (available ?r))
202     (at end (checked ?poi))
203   )
204 )
205
206 (:durative-action change_site_air
207   :parameters (?r - robot ?s1 ?s2 - site ?poi1 ?poi2 - pointofinterest)
208   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
209   :condition (and
210     (at start (at_site ?r ?s1))
211     (at start (at ?r ?poi1))
212     (over all (air ?poi1))
213     (over all (air ?poi2))
214     (over all (partofsite ?poi1 ?s1))
215     (over all (partofsite ?poi2 ?s2))
216     (at start (available ?r))
217     (over all (airconf ?r))
218   )
219   :effect (and
220     (at start (not(available ?r)))
221     (at start (not(at_site ?r ?s1)))
222     (at start (not(at ?r ?poi1)))
223     (at end (available ?r))
224     (at end (at_site ?r ?s2))
225     (at end (at ?r ?poi2))
226   )
227 )
228
229 (:durative-action change_site_water
230   :parameters (?r - robot ?s1 ?s2 - site ?poi1 ?poi2 - pointofinterest)
231   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedwater ?r)))
232   :condition (and
233     (at start (at_site ?r ?s1))
234     (at start (at ?r ?poi1))
235     (over all (water ?poi1))
236     (over all (water ?poi2))
237     (over all (partofsite ?poi1 ?s1))
238     (over all (partofsite ?poi2 ?s2))
```

```
    (at start (available ?r))
    (over all (waterconf ?r))
  )
  :effect (and
    (at start (not(available ?r)))
    (at start (not(at_site ?r ?s1)))
    (at start (not(at ?r ?poi1)))
    (at end (available ?r))
    (at end (at_site ?r ?s2))
    (at end (at ?r ?poi2))
  )
)


);; end Domain ;;;;;;;;;;;;;;;;;;;;;;;;
```

Listing A.3: Trans-media PDDL domain

```
(define (domain TMdomainextended)
(:requirements :strips :typing :fluents :negative-preconditions :disjunctive-
    preconditions :durative-actions :universal-preconditions )

;; Types ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:types

robot
pointofinterest
site

);; end Types ;;;;;;;;;;;;;;;;;;;;;;;;

;; Predicates ;;;;;;;;;;;;;;;;;;;;;;;;
(:predicates

;; site predicates

(observed ?s - site)

;; robot predicates

(at ?r - robot ?poi - pointofinterest)
(at_site ?r - robot ?s - site)
(waterconf ?r - robot)
(airconf ?r - robot)
(groundconf ?r - robot)
(available ?r - robot)
(canswitch ?r - robot)
(canrelay ?r - robot)
(cansample ?r - robot)

;; Poi predicates

(sample_poi ?poi - pointofinterest)
(transition_poi ?poi - pointofinterest)
(survey_poi ?poi - pointofinterest)
(water ?poi - pointofinterest)
(air ?poi - pointofinterest)
(ground ?poi - pointofinterest)
(connected ?s - site)
(partofsite ?poi - pointofinterest ?s - site)
(sampled ?poi - pointofinterest)
(isswitchable ?poi - pointofinterest)
```

```
(isrelay ?poi - pointofinterest)

);; end Predicates ;;;;;;;;;;;;;;;;;;;

;; Functions ;;;;;;;;;;;;;;;;;;;;;;;;
(:functions
(distance ?poi1 ?poi2 - pointofinterest)

(site_size ?s - site)

(energy ?r - robot)

(speedair ?r - robot)
(speedwater ?r - robot)

(maintainconsumption_air ?r - robot)
(maintainconsumption_water ?r - robot)

(assesscost ?r - robot )
(observecost ?r - robot)

(navconsumption_water ?r - robot)
(navconsumption_air ?r - robot)

(switchduration_airwater ?r - robot)
(switchcost_airwater ?r - robot)

(landingduration_airground ?r - robot)
(landingcost_airground ?r - robot)

(takeoffduration_groundair ?r - robot)
(takeoffost_groundair ?r - robot)

(switchduration_waterair ?r - robot)
(switchcost_waterair ?r - robot)



);; end Functions ;;;;;;;;;;;;;;;;;;;

;; Actions ;;;;;;;;;;;;;;;;;;;;;;;;;;
(:durative-action navigation_air
 :parameters (?r - robot ?poi1 ?poi2 - pointofinterest ?s - site)
 :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
 :condition (and
   (at start(available ?r))
   (at start (partofsite ?poi1 ?s))
   (at start (partofsite ?poi2 ?s))
   (at start (at ?r ?poi1))
   (at start (air ?poi1))
   (at start (air ?poi2))
   (at start (airconf ?r))
   (at start (>= (energy ?r) (* (distance ?poi1 ?poi2)(navconsumption_air ?r))))
 )
 :effect (and
   (at start (decrease (energy ?r) (* (distance ?poi1 ?poi2) (navconsumption_air ?r)))
       )
      (at start (not (available ?r)))
      (at start (not (at ?r ?poi1)))
      (at end (at ?r ?poi2))
      (at end (available ?r))
   )
```

```
105 )
106
107 (:durative-action navigation_water
108   :parameters (?r - robot ?poi1 ?poi2 - pointofinterest ?s - site)
109   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedwater ?r)))
110   :condition (and
111     (at start(available ?r))
112     (at start (partofsite ?poi1 ?s))
113     (at start (partofsite ?poi2 ?s))
114     (at start (at ?r ?poi1))
115     (at start (water ?poi1))
116     (at start (water ?poi2))
117     (at start (waterconf ?r))
118     (at start (>= (energy ?r) (* (distance ?poi1 ?poi2)(navconsumption_water ?r))))
119   )
120   :effect (and
121     (at start (decrease (energy ?r) (* (distance ?poi1 ?poi2) (navconsumption_water ?r)
          )))
122       (at start (not (available ?r)))
123       (at start (not (at ?r ?poi1)))
124       (at end (at ?r ?poi2))
125       (at end (available ?r))
126     )
127 )
128
129 (:durative-action takeoff
130   :parameters (?r - robot ?poi - pointofinterest)
131   :duration ( = ?duration (takeoffduration_groundair ?r))
132   :condition (and
133     (at start (available ?r))
134     (at start (at ?r ?poi))
135     (at start (transition_poi ?poi))
136     (at start (groundconf ?r))
137     (at start (air ?poi))
138     (at start (ground ?poi))
139     (at start (>= (energy ?r) (takeoffost_groundair ?r)))
140   )
141   :effect (and
142     (at start (decrease (energy ?r) (takeoffost_groundair ?r)))
143       (at start (not (available ?r)))
144       (at start (not (groundconf ?r)))
145       (at end (airconf ?r))
146     (at end (at ?r ?poi))
147       (at end (available ?r))
148     (at end (available ?r))
149     )
150 )
151
152 (:durative-action landing
153   :parameters (?r - robot ?poi - pointofinterest)
154   :duration ( = ?duration (landingduration_airground ?r))
155   :condition (and
156     (at start(available ?r))
157     (over all (at ?r ?poi))
158     (over all (transition_poi ?poi))
159     (at start (airconf ?r))
160     (over all (air ?poi))
161     (over all (ground ?poi))
162     (at start (>= (energy ?r) (landingcost_airground ?r)))
163   )
164   :effect (and
165     (at start (decrease (energy ?r) (landingcost_airground ?r)))
```

```
166       (at start (not (available ?r)))
167       (at start (not (airconf ?r)))
168       (at end (groundconf ?r))
169       (at end (available ?r))
170    )
171 )
172
173
174
175
176 (:durative-action observe
177  :parameters (?r - robot ?poi - pointofinterest ?s - site)
178  :duration ( = ?duration (/ (site_size ?s) (speedair ?r)))
179  :condition (and
180    (at start (at ?r ?poi))
181    (at start (at_site ?r ?s))
182    (at start (survey_poi ?poi))
183    (at start (partofsite ?poi ?s))
184    (at start (air ?poi))
185    (at start (available ?r))
186    (at start (airconf ?r))
187    (at start (>= (energy ?r) (+(* (observecost ?r) 30) (*(maintainconsumption_air ?r)
           60))))
188  )
189  :effect (and
190    (at start (not(available ?r)))
191    (at start (decrease (energy ?r) (+(* (observecost ?r) 30) (*(
           maintainconsumption_air ?r) 60))))
192    (at end (available ?r))
193    (at end (observed ?s))
194  )
195 )
196
197 (:durative-action observe_2r
198  :parameters (?r1 ?r2 - robot ?poi - pointofinterest ?s - site)
199  :duration (= ?duration (/ (/ (site_size ?s) (speedair ?r1)) 2))
200  :condition (and
201    (at start (at ?r1 ?poi))
202    (at start (at_site ?r1 ?s))
203    (at start (at ?r2 ?poi))
204    (at start (at_site ?r2 ?s))
205    (at start (survey_poi ?poi))
206    (at start (partofsite ?poi ?s))
207    (at start (air ?poi))
208    (at start (available ?r1))
209    (at start (available ?r2))
210    (at start (airconf ?r1))
211    (at start (airconf ?r2))
212    (at start (>= (energy ?r1) (+(* (observecost ?r1) 30) (*(maintainconsumption_air ?
           r1) 60))))
213    (at start (>= (energy ?r2) (+(* (observecost ?r2) 30) (*(maintainconsumption_air ?
           r2) 60))))
214  )
215  :effect (and
216    (at start (not(available ?r1)))
217    (at start (not(available ?r2)))
218    (at start (decrease (energy ?r1) (+(* (observecost ?r1) 30) (*(
           maintainconsumption_air ?r1) 60))))
219    (at start (decrease (energy ?r2) (+(* (observecost ?r2) 30) (*(
           maintainconsumption_air ?r2) 60))))
220    (at end (available ?r1))
221    (at end (available ?r2))
```

```
    (at end (observed ?s))
  )
)

(:durative-action switch_waterair
  :parameters (?r - robot ?poi - pointofinterest)
  :duration ( = ?duration (switchduration_waterair ?r))
  :condition (and
    (at start (available ?r))
    (over all (canswitch ?r))
    (at start (waterconf ?r))
    (over all (at ?r ?poi))
    (over all (transition_poi ?poi))
    (over all (isswitchable ?poi))
    (over all (water ?poi))
    (over all (air ?poi))
    (at start (>= (energy ?r) (switchcost_waterair ?r)))
  )
  :effect (and
    (at start (decrease (energy ?r) (switchcost_waterair ?r)))
       (at start (not (available ?r)))
       (at start (not (waterconf ?r)))
       (at end (airconf ?r))
       (at end (available ?r))
    )
)

(:durative-action switch_airwater
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration (switchduration_airwater ?r))
  :condition (and
    (at start(available ?r))
    (over all (canswitch ?r))
    (at start (airconf ?r))
    (over all (at ?r ?poi))
    (over all (observed ?s))
    (over all (partofsite ?poi ?s))
    (over all (transition_poi ?poi))
    (over all (isswitchable ?poi))
    (over all (air ?poi))
    (over all (water ?poi))
    (at start (>= (energy ?r) (switchcost_airwater ?r)))
  )
  :effect (and
    (at start (decrease (energy ?r) (switchcost_airwater ?r)))
       (at start (not (available ?r)))
       (at start (not (airconf ?r)))
       (at end (waterconf ?r))
       (at end (available ?r))
    )
)


(:durative-action relay_data
  :parameters (?r - robot ?poi - pointofinterest ?s - site)
  :duration ( = ?duration 45)
  :condition (and
    (over all (at ?r ?poi))
    (over all (transition_poi ?poi))
    (over all (partofsite ?poi ?s))
    (over all (observed ?s))
    (over all (isrelay ?poi))
```

```
284     (at start (available ?r))
285     (over all (waterconf ?r))
286     (over all (canrelay ?r))
287     (at start (>= (energy ?r) (* (maintainconsumption_water ?r) 45) ))
288   )
289   :effect (and
290     (at start (not (available ?r)))
291     (at start (connected ?s))
292     (at start (decrease (energy ?r) (* (maintainconsumption_water ?r) 45)))
293     (at end (not (connected ?s)))
294     (at end (available ?r))
295   )
296 )
297
298 (:durative-action sample
299   :parameters (?r - robot ?poi - pointofinterest ?s - site)
300   :duration ( = ?duration 30)
301   :condition (and
302     (over all (at ?r ?poi))
303     (at start (available ?r))
304     (over all (cansample ?r))
305     (over all (waterconf ?r))
306     (over all (water ?poi))
307     (over all (sample_poi ?poi))
308     (over all (partofsite ?poi ?s))
309     (over all (connected ?s))
310     (at start (>= (energy ?r) (+(* (assesscost ?r) 60) (*(maintainconsumption_water ?r)
            60)) ))
311   )
312   :effect (and
313     (at start (not(available ?r)))
314     (at start (decrease (energy ?r) (+(* (assesscost ?r) 60) (*(
            maintainconsumption_water ?r) 60))))
315     (at end (available ?r))
316     (at end (sampled ?poi))
317   )
318 )
319
320 (:durative-action change_site
321   :parameters (?r - robot ?s1 ?s2 - site ?poi1 ?poi2 - pointofinterest)
322   :duration ( = ?duration (/ (distance ?poi1 ?poi2) (speedair ?r)))
323   :condition (and
324     (at start (at_site ?r ?s1))
325     (at start (at ?r ?poi1))
326     (over all (air ?poi1))
327     (over all (air ?poi2))
328     (over all (partofsite ?poi1 ?s1))
329     (over all (partofsite ?poi2 ?s2))
330     (at start (available ?r))
331     (over all (airconf ?r))
332     (at start (>= (energy ?r) (* (distance ?poi1 ?poi2)(navconsumption_air ?r))))
333   )
334   :effect (and
335     (at start (not(available ?r)))
336     (at start (not(at_site ?r ?s1)))
337     (at start (not(at ?r ?poi1)))
338     (at start (decrease (energy ?r) (* (distance ?poi1 ?poi2)(navconsumption_air ?r))))
339     (at end (available ?r))
340     (at end (at_site ?r ?s2))
341     (at end (at ?r ?poi2))
342   )
343 )
```

```
344
345
346 );; end Domain ;;;;;;;;;;;;;;;;;;;;;;;;
```

A.2 Problem and PDDL plans of team_0 example

This section shows how the plan for team_0 in Fig.4.7b was obtained by successively solving three subproblems corresponding to base→site3, site3→site5, and site5→base:

1. The team starts at the base and lifts off to solve site3.
2. From their last state at site3, they continue to solve site5.
3. Finally, they land back at the base.

PDDL subproblems

Listing A.4: PDDL problem: team_0 base to site3

```
1  (define (problem subp_base_to_site3)
2  ( :domain mmdomainextended )
3  ( :objects
4  robot0 robot1 - robot
5  cpbase cpsite3 pp5 pp6 pp7 sp6 sp7 sp8 sp9 - pointofinterest
6  base site3 - site
7  )
8  ( :init
9  ( at robot0 cpbase )
10 ( at_site robot0 base )
11 ( groundconf robot0 )
12 ( available robot0 )
13 ( canswitch robot0 )
14 ( canrelay robot0 )
15 ( cansample robot0 )
16 ( at robot1 cpbase )
17 ( at_site robot1 base )
18 ( groundconf robot1 )
19 ( available robot1 )
20 ( canswitch robot1 )
21 ( canrelay robot1 )
22 ( cansample robot1 )
23 ( transition_poi cpbase )
24 ( isswitchable cpbase )
25 ( isrelay cpbase )
26 ( ground cpbase )
27 ( air cpbase )
28 ( partofsite cpbase base )
29 ( survey_poi cpsite3 )
30 ( air cpsite3 )
31 ( partofsite cpsite3 site3 )
32 ( sample_poi pp5 )
33 ( water pp5 )
34 ( partofsite pp5 site3 )
```

```
( sample_poi pp6 )
( water pp6 )
( partofsite pp6 site3 )
( sample_poi pp7 )
( water pp7 )
( partofsite pp7 site3 )
( transition_poi sp6 )
( isswitchable sp6 )
( isrelay sp6 )
( water sp6 )
( air sp6 )
( partofsite sp6 site3 )
( transition_poi sp7 )
( isswitchable sp7 )
( isrelay sp7 )
( water sp7 )
( air sp7 )
( partofsite sp7 site3 )
( transition_poi sp8 )
( isswitchable sp8 )
( isrelay sp8 )
( water sp8 )
( air sp8 )
( partofsite sp8 site3 )
( transition_poi sp9 )
( isswitchable sp9 )
( isrelay sp9 )
( water sp9 )
( air sp9 )
( partofsite sp9 site3 )
( = ( speedair robot0 ) 1.5000000000 )
( = ( speedwater robot0 ) 0.5000000000 )
( = ( energy robot0 ) 10000.0000000000 )
( = ( recharge_rate robot0 ) 10.0000000000 )
( = ( assesscost robot0 ) 0.0200000000 )
( = ( observecost robot0 ) 0.0100000000 )
( = ( maintainconsumption_water robot0 ) 0.0200000000 )
( = ( maintainconsumption_air robot0 ) 0.0800000000 )
( = ( navconsumption_water robot0 ) 0.5000000000 )
( = ( navconsumption_air robot0 ) 0.8000000000 )
( = ( switchduration_airwater robot0 ) 5.0000000000 )
( = ( switchcost_airwater robot0 ) 20.0000000000 )
( = ( takeoffduration_groundair robot0 ) 4.0000000000 )
( = ( takeoffost_groundair robot0 ) 10.0000000000 )
( = ( landingduration_airground robot0 ) 3.0000000000 )
( = ( landingcost_airground robot0 ) 5.0000000000 )
( = ( switchduration_waterair robot0 ) 8.0000000000 )
( = ( switchcost_waterair robot0 ) 30.0000000000 )
( = ( speedair robot1 ) 1.5000000000 )
( = ( speedwater robot1 ) 0.5000000000 )
( = ( energy robot1 ) 10000.0000000000 )
( = ( recharge_rate robot1 ) 10.0000000000 )
( = ( assesscost robot1 ) 0.0200000000 )
( = ( observecost robot1 ) 0.0100000000 )
( = ( maintainconsumption_water robot1 ) 0.0200000000 )
( = ( maintainconsumption_air robot1 ) 0.0800000000 )
( = ( navconsumption_water robot1 ) 0.5000000000 )
( = ( navconsumption_air robot1 ) 0.8000000000 )
( = ( switchduration_airwater robot1 ) 5.0000000000 )
( = ( switchcost_airwater robot1 ) 20.0000000000 )
( = ( takeoffduration_groundair robot1 ) 4.0000000000 )
( = ( takeoffost_groundair robot1 ) 10.0000000000 )
```

```
( = ( landingduration_airground robot1 ) 3.0000000000 )
( = ( landingcost_airground robot1 ) 5.0000000000 )
( = ( switchduration_waterair robot1 ) 8.0000000000 )
( = ( switchcost_waterair robot1 ) 30.0000000000 )
( = ( site_size base ) 10.0000000000 )
( = ( distance cpbase cpsite3 ) 439.4300000000 )
( = ( distance cpbase sp6 ) 425.4100000000 )
( = ( distance cpbase sp7 ) 425.8000000000 )
( = ( distance cpbase sp8 ) 438.6100000000 )
( = ( distance cpbase sp9 ) 442.5200000000 )
( = ( site_size site3 ) 30.0000000000 )
( = ( distance cpsite3 cpbase ) 439.4300000000 )
( = ( distance cpsite3 sp6 ) 14.2200000000 )
( = ( distance cpsite3 sp7 ) 13.7200000000 )
( = ( distance cpsite3 sp8 ) 9.8800000000 )
( = ( distance cpsite3 sp9 ) 7.4500000000 )
( = ( distance pp5 pp6 ) 23.7500000000 )
( = ( distance pp5 pp7 ) 23.2000000000 )
( = ( distance pp5 sp6 ) 8.1900000000 )
( = ( distance pp5 sp7 ) 11.1700000000 )
( = ( distance pp5 sp8 ) 22.0700000000 )
( = ( distance pp5 sp9 ) 15.0900000000 )
( = ( distance pp6 pp5 ) 23.7500000000 )
( = ( distance pp6 pp7 ) 8.9500000000 )
( = ( distance pp6 sp6 ) 25.4500000000 )
( = ( distance pp6 sp7 ) 25.3400000000 )
( = ( distance pp6 sp8 ) 17.6100000000 )
( = ( distance pp6 sp9 ) 9.6700000000 )
( = ( distance pp7 pp5 ) 23.2000000000 )
( = ( distance pp7 pp6 ) 8.9500000000 )
( = ( distance pp7 sp6 ) 22.1100000000 )
( = ( distance pp7 sp7 ) 20.8600000000 )
( = ( distance pp7 sp8 ) 9.4000000000 )
( = ( distance pp7 sp9 ) 12.7200000000 )
( = ( distance sp6 cpbase ) 425.4100000000 )
( = ( distance sp6 cpsite3 ) 14.2200000000 )
( = ( distance sp6 pp5 ) 8.1900000000 )
( = ( distance sp6 pp6 ) 25.4500000000 )
( = ( distance sp6 pp7 ) 22.1100000000 )
( = ( distance sp6 sp7 ) 3.3800000000 )
( = ( distance sp6 sp8 ) 17.7200000000 )
( = ( distance sp6 sp9 ) 17.6700000000 )
( = ( distance sp7 cpbase ) 425.8000000000 )
( = ( distance sp7 cpsite3 ) 13.7200000000 )
( = ( distance sp7 pp5 ) 11.1700000000 )
( = ( distance sp7 pp6 ) 25.3400000000 )
( = ( distance sp7 pp7 ) 20.8600000000 )
( = ( distance sp7 sp6 ) 3.3800000000 )
( = ( distance sp7 sp8 ) 15.3300000000 )
( = ( distance sp7 sp9 ) 18.4600000000 )
( = ( distance sp8 cpbase ) 438.6100000000 )
( = ( distance sp8 cpsite3 ) 9.8800000000 )
( = ( distance sp8 pp5 ) 22.0700000000 )
( = ( distance sp8 pp6 ) 17.6100000000 )
( = ( distance sp8 pp7 ) 9.4000000000 )
( = ( distance sp8 sp6 ) 17.7200000000 )
( = ( distance sp8 sp7 ) 15.3300000000 )
( = ( distance sp8 sp9 ) 16.9800000000 )
( = ( distance sp9 cpbase ) 442.5200000000 )
( = ( distance sp9 cpsite3 ) 7.4500000000 )
( = ( distance sp9 pp5 ) 15.0900000000 )
( = ( distance sp9 pp6 ) 9.6700000000 )
```

```
( = ( distance sp9 pp7 ) 12.7200000000 )
( = ( distance sp9 sp6 ) 17.6700000000 )
( = ( distance sp9 sp7 ) 18.4600000000 )
( = ( distance sp9 sp8 ) 16.9800000000 )
)
(:goal
( and
(sampled pp5)(sampled pp6)(sampled pp7))
)
)
```

Listing A.5: PDDL problem: team_0 site3 to site5

```
(define (problem subp_site3_to_site5)
( :domain mmdomainextended )
( :objects
robot0 robot1 - robot
cpsite3 pp5 pp6 pp7 sp6 sp7 sp8 sp9 cpsite5 pp9 pp10 pp11 pp12 sp11 sp12 sp13 sp14
    sp15 - pointofinterest
site3 site5 - site
)
( :init
( at robot0 sp9 )
( at_site robot0 site3 )
( waterconf robot0 )
( available robot0 )
( canswitch robot0 )
( canrelay robot0 )
( cansample robot0 )
( at robot1 pp5 )
( at_site robot1 site3 )
( waterconf robot1 )
( available robot1 )
( canswitch robot1 )
( canrelay robot1 )
( cansample robot1 )
( survey_poi cpsite3 )
( air cpsite3 )
( partofsite cpsite3 site3 )
( sample_poi pp5 )
( water pp5 )
( partofsite pp5 site3 )
( sample_poi pp6 )
( water pp6 )
( partofsite pp6 site3 )
( sample_poi pp7 )
( water pp7 )
( partofsite pp7 site3 )
( transition_poi sp6 )
( isswitchable sp6 )
( isrelay sp6 )
( water sp6 )
( air sp6 )
( partofsite sp6 site3 )
( transition_poi sp7 )
( isswitchable sp7 )
( isrelay sp7 )
( water sp7 )
( air sp7 )
( partofsite sp7 site3 )
( transition_poi sp8 )
( isswitchable sp8 )
```

```
( isrelay sp8 )
( water sp8 )
( air sp8 )
( partofsite sp8 site3 )
( transition_poi sp9 )
( isswitchable sp9 )
( isrelay sp9 )
( water sp9 )
( air sp9 )
( partofsite sp9 site3 )
( survey_poi cpsite5 )
( air cpsite5 )
( partofsite cpsite5 site5 )
( sample_poi pp9 )
( water pp9 )
( partofsite pp9 site5 )
( sample_poi pp10 )
( water pp10 )
( partofsite pp10 site5 )
( sample_poi pp11 )
( water pp11 )
( partofsite pp11 site5 )
( sample_poi pp12 )
( water pp12 )
( partofsite pp12 site5 )
( transition_poi sp11 )
( isswitchable sp11 )
( isrelay sp11 )
( water sp11 )
( air sp11 )
( partofsite sp11 site5 )
( transition_poi sp12 )
( isswitchable sp12 )
( isrelay sp12 )
( water sp12 )
( air sp12 )
( partofsite sp12 site5 )
( transition_poi sp13 )
( isswitchable sp13 )
( isrelay sp13 )
( water sp13 )
( air sp13 )
( partofsite sp13 site5 )
( transition_poi sp14 )
( isswitchable sp14 )
( isrelay sp14 )
( water sp14 )
( air sp14 )
( partofsite sp14 site5 )
( transition_poi sp15 )
( isswitchable sp15 )
( isrelay sp15 )
( water sp15 )
( air sp15 )
( partofsite sp15 site5 )
( = ( speedair robot0 ) 1.5000000000 )
( = ( speedwater robot0 ) 0.5000000000 )
( = ( energy robot0 ) 10000.0000000000 )
( = ( recharge_rate robot0 ) 10.0000000000 )
( = ( assesscost robot0 ) 0.0200000000 )
( = ( observecost robot0 ) 0.0100000000 )
( = ( maintainconsumption_water robot0 ) 0.0200000000 )
```

```
( = ( maintainconsumption_air robot0 ) 0.0800000000 )
( = ( navconsumption_water robot0 ) 0.5000000000 )
( = ( navconsumption_air robot0 ) 0.8000000000 )
( = ( switchduration_airwater robot0 ) 5.0000000000 )
( = ( switchcost_airwater robot0 ) 20.0000000000 )
( = ( takeoffduration_groundair robot0 ) 4.0000000000 )
( = ( takeoffost_groundair robot0 ) 10.0000000000 )
( = ( landingduration_airground robot0 ) 3.0000000000 )
( = ( landingcost_airground robot0 ) 5.0000000000 )
( = ( switchduration_waterair robot0 ) 8.0000000000 )
( = ( switchcost_waterair robot0 ) 30.0000000000 )
( = ( speedair robot1 ) 1.5000000000 )
( = ( speedwater robot1 ) 0.5000000000 )
( = ( energy robot1 ) 10000.0000000000 )
( = ( recharge_rate robot1 ) 10.0000000000 )
( = ( assesscost robot1 ) 0.0200000000 )
( = ( observecost robot1 ) 0.0100000000 )
( = ( maintainconsumption_water robot1 ) 0.0200000000 )
( = ( maintainconsumption_air robot1 ) 0.0800000000 )
( = ( navconsumption_water robot1 ) 0.5000000000 )
( = ( navconsumption_air robot1 ) 0.8000000000 )
( = ( switchduration_airwater robot1 ) 5.0000000000 )
( = ( switchcost_airwater robot1 ) 20.0000000000 )
( = ( takeoffduration_groundair robot1 ) 4.0000000000 )
( = ( takeoffost_groundair robot1 ) 10.0000000000 )
( = ( landingduration_airground robot1 ) 3.0000000000 )
( = ( landingcost_airground robot1 ) 5.0000000000 )
( = ( switchduration_waterair robot1 ) 8.0000000000 )
( = ( switchcost_waterair robot1 ) 30.0000000000 )
( = ( site_size site3 ) 30.0000000000 )
( = ( distance cpsite3 sp6 ) 14.2200000000 )
( = ( distance cpsite3 sp7 ) 13.7200000000 )
( = ( distance cpsite3 sp8 ) 9.8800000000 )
( = ( distance cpsite3 sp9 ) 7.4500000000 )
( = ( distance cpsite3 cpsite5 ) 617.3400000000 )
( = ( distance cpsite3 sp11 ) 593.6300000000 )
( = ( distance cpsite3 sp12 ) 616.1500000000 )
( = ( distance cpsite3 sp13 ) 628.3000000000 )
( = ( distance cpsite3 sp14 ) 633.9000000000 )
( = ( distance cpsite3 sp15 ) 596.3900000000 )
( = ( distance pp5 pp6 ) 23.7500000000 )
( = ( distance pp5 pp7 ) 23.2000000000 )
( = ( distance pp5 sp6 ) 8.1900000000 )
( = ( distance pp5 sp7 ) 11.1700000000 )
( = ( distance pp5 sp8 ) 22.0700000000 )
( = ( distance pp5 sp9 ) 15.0900000000 )
( = ( distance pp6 pp5 ) 23.7500000000 )
( = ( distance pp6 pp7 ) 8.9500000000 )
( = ( distance pp6 sp6 ) 25.4500000000 )
( = ( distance pp6 sp7 ) 25.3400000000 )
( = ( distance pp6 sp8 ) 17.6100000000 )
( = ( distance pp6 sp9 ) 9.6700000000 )
( = ( distance pp7 pp5 ) 23.2000000000 )
( = ( distance pp7 pp6 ) 8.9500000000 )
( = ( distance pp7 sp6 ) 22.1100000000 )
( = ( distance pp7 sp7 ) 20.8600000000 )
( = ( distance pp7 sp8 ) 9.4000000000 )
( = ( distance pp7 sp9 ) 12.7200000000 )
( = ( distance sp6 cpsite3 ) 14.2200000000 )
( = ( distance sp6 pp5 ) 8.1900000000 )
( = ( distance sp6 pp6 ) 25.4500000000 )
( = ( distance sp6 pp7 ) 22.1100000000 )
```

```
( = ( distance sp6 sp7 ) 3.3800000000 )
( = ( distance sp6 sp8 ) 17.7200000000 )
( = ( distance sp6 sp9 ) 17.6700000000 )
( = ( distance sp6 cpsite5 ) 631.5200000000 )
( = ( distance sp6 sp11 ) 607.8100000000 )
( = ( distance sp6 sp12 ) 630.3400000000 )
( = ( distance sp6 sp13 ) 642.4800000000 )
( = ( distance sp6 sp14 ) 648.0900000000 )
( = ( distance sp6 sp15 ) 610.5700000000 )
( = ( distance sp7 cpsite3 ) 13.7200000000 )
( = ( distance sp7 pp5 ) 11.1700000000 )
( = ( distance sp7 pp6 ) 25.3400000000 )
( = ( distance sp7 pp7 ) 20.8600000000 )
( = ( distance sp7 sp6 ) 3.3800000000 )
( = ( distance sp7 sp8 ) 15.3300000000 )
( = ( distance sp7 sp9 ) 18.4600000000 )
( = ( distance sp7 cpsite5 ) 630.7100000000 )
( = ( distance sp7 sp11 ) 607.0000000000 )
( = ( distance sp7 sp12 ) 629.4300000000 )
( = ( distance sp7 sp13 ) 641.6900000000 )
( = ( distance sp7 sp14 ) 647.2300000000 )
( = ( distance sp7 sp15 ) 609.7700000000 )
( = ( distance sp8 cpsite3 ) 9.8800000000 )
( = ( distance sp8 pp5 ) 22.0700000000 )
( = ( distance sp8 pp6 ) 17.6100000000 )
( = ( distance sp8 pp7 ) 9.4000000000 )
( = ( distance sp8 sp6 ) 17.7200000000 )
( = ( distance sp8 sp7 ) 15.3300000000 )
( = ( distance sp8 sp9 ) 16.9800000000 )
( = ( distance sp8 cpsite5 ) 617.0800000000 )
( = ( distance sp8 sp11 ) 593.3700000000 )
( = ( distance sp8 sp12 ) 615.6000000000 )
( = ( distance sp8 sp13 ) 628.1100000000 )
( = ( distance sp8 sp14 ) 633.4900000000 )
( = ( distance sp8 sp15 ) 596.2000000000 )
( = ( distance sp9 cpsite3 ) 7.4500000000 )
( = ( distance sp9 pp5 ) 15.0900000000 )
( = ( distance sp9 pp6 ) 9.6700000000 )
( = ( distance sp9 pp7 ) 12.7200000000 )
( = ( distance sp9 sp6 ) 17.6700000000 )
( = ( distance sp9 sp7 ) 18.4600000000 )
( = ( distance sp9 sp8 ) 16.9800000000 )
( = ( distance sp9 cpsite5 ) 615.2200000000 )
( = ( distance sp9 sp11 ) 591.5200000000 )
( = ( distance sp9 sp12 ) 614.2400000000 )
( = ( distance sp9 sp13 ) 626.1300000000 )
( = ( distance sp9 sp14 ) 631.9000000000 )
( = ( distance sp9 sp15 ) 594.2300000000 )
( = ( site_size site5 ) 40.0000000000 )
( = ( distance cpsite5 cpsite3 ) 617.3400000000 )
( = ( distance cpsite5 sp6 ) 631.5200000000 )
( = ( distance cpsite5 sp7 ) 630.7100000000 )
( = ( distance cpsite5 sp8 ) 617.0800000000 )
( = ( distance cpsite5 sp9 ) 615.2200000000 )
( = ( distance cpsite5 sp11 ) 23.7300000000 )
( = ( distance cpsite5 sp12 ) 18.3700000000 )
( = ( distance cpsite5 sp13 ) 11.8500000000 )
( = ( distance cpsite5 sp14 ) 19.2700000000 )
( = ( distance cpsite5 sp15 ) 21.3400000000 )
( = ( distance pp9 pp10 ) 11.3000000000 )
( = ( distance pp9 pp11 ) 28.2100000000 )
( = ( distance pp9 pp12 ) 13.7300000000 )
```

```
( = ( distance pp9 sp11 ) 31.2200000000 )
( = ( distance pp9 sp12 ) 4.9100000000 )
( = ( distance pp9 sp13 ) 25.0800000000 )
( = ( distance pp9 sp14 ) 17.7900000000 )
( = ( distance pp9 sp15 ) 31.8000000000 )
( = ( distance pp10 pp9 ) 11.3000000000 )
( = ( distance pp10 pp11 ) 17.0200000000 )
( = ( distance pp10 pp12 ) 20.3900000000 )
( = ( distance pp10 sp11 ) 32.8900000000 )
( = ( distance pp10 sp12 ) 13.6200000000 )
( = ( distance pp10 sp13 ) 14.4400000000 )
( = ( distance pp10 sp14 ) 9.6700000000 )
( = ( distance pp10 sp15 ) 31.7200000000 )
( = ( distance pp11 pp9 ) 28.2100000000 )
( = ( distance pp11 pp10 ) 17.0200000000 )
( = ( distance pp11 pp12 ) 36.2100000000 )
( = ( distance pp11 sp11 ) 43.6800000000 )
( = ( distance pp11 sp12 ) 30.3000000000 )
( = ( distance pp11 sp13 ) 9.6900000000 )
( = ( distance pp11 sp14 ) 13.9600000000 )
( = ( distance pp11 sp15 ) 40.8300000000 )
( = ( distance pp12 pp9 ) 13.7300000000 )
( = ( distance pp12 pp10 ) 20.3900000000 )
( = ( distance pp12 pp11 ) 36.2100000000 )
( = ( distance pp12 sp11 ) 19.1500000000 )
( = ( distance pp12 sp12 ) 11.6800000000 )
( = ( distance pp12 sp13 ) 30.0600000000 )
( = ( distance pp12 sp14 ) 29.0300000000 )
( = ( distance pp12 sp15 ) 21.1100000000 )
( = ( distance sp11 cpsite3 ) 593.6300000000 )
( = ( distance sp11 sp6 ) 607.8100000000 )
( = ( distance sp11 sp7 ) 607.0000000000 )
( = ( distance sp11 sp8 ) 593.3700000000 )
( = ( distance sp11 sp9 ) 591.5200000000 )
( = ( distance sp11 cpsite5 ) 23.7300000000 )
( = ( distance sp11 pp9 ) 31.2200000000 )
( = ( distance sp11 pp10 ) 32.8900000000 )
( = ( distance sp11 pp11 ) 43.6800000000 )
( = ( distance sp11 pp12 ) 19.1500000000 )
( = ( distance sp11 sp12 ) 28.6300000000 )
( = ( distance sp11 sp13 ) 34.9700000000 )
( = ( distance sp11 sp14 ) 41.3400000000 )
( = ( distance sp11 sp15 ) 4.9500000000 )
( = ( distance sp12 cpsite3 ) 616.1500000000 )
( = ( distance sp12 sp6 ) 630.3400000000 )
( = ( distance sp12 sp7 ) 629.4300000000 )
( = ( distance sp12 sp8 ) 615.6000000000 )
( = ( distance sp12 sp9 ) 614.2400000000 )
( = ( distance sp12 cpsite5 ) 18.3700000000 )
( = ( distance sp12 pp9 ) 4.9100000000 )
( = ( distance sp12 pp10 ) 13.6200000000 )
( = ( distance sp12 pp11 ) 30.3000000000 )
( = ( distance sp12 pp12 ) 11.6800000000 )
( = ( distance sp12 sp11 ) 28.6300000000 )
( = ( distance sp12 sp13 ) 25.8800000000 )
( = ( distance sp12 sp14 ) 19.8000000000 )
( = ( distance sp12 sp15 ) 29.5100000000 )
( = ( distance sp13 cpsite3 ) 628.3000000000 )
( = ( distance sp13 sp6 ) 642.4800000000 )
( = ( distance sp13 sp7 ) 641.6900000000 )
( = ( distance sp13 sp8 ) 628.1100000000 )
( = ( distance sp13 sp9 ) 626.1300000000 )
```

```
( = ( distance sp13 cpsite5 ) 11.8500000000 )
( = ( distance sp13 pp9 ) 25.0800000000 )
( = ( distance sp13 pp10 ) 14.4400000000 )
( = ( distance sp13 pp11 ) 9.6900000000 )
( = ( distance sp13 pp12 ) 30.0600000000 )
( = ( distance sp13 sp11 ) 34.9700000000 )
( = ( distance sp13 sp12 ) 25.8800000000 )
( = ( distance sp13 sp14 ) 15.3800000000 )
( = ( distance sp13 sp15 ) 31.9100000000 )
( = ( distance sp14 cpsite3 ) 633.9000000000 )
( = ( distance sp14 sp6 ) 648.0900000000 )
( = ( distance sp14 sp7 ) 647.2300000000 )
( = ( distance sp14 sp8 ) 633.4900000000 )
( = ( distance sp14 sp9 ) 631.9000000000 )
( = ( distance sp14 cpsite5 ) 19.2700000000 )
( = ( distance sp14 pp9 ) 17.7900000000 )
( = ( distance sp14 pp10 ) 9.6700000000 )
( = ( distance sp14 pp11 ) 13.9600000000 )
( = ( distance sp14 pp12 ) 29.0300000000 )
( = ( distance sp14 sp11 ) 41.3400000000 )
( = ( distance sp14 sp12 ) 19.8000000000 )
( = ( distance sp14 sp13 ) 15.3800000000 )
( = ( distance sp14 sp15 ) 39.9100000000 )
( = ( distance sp15 cpsite3 ) 596.3900000000 )
( = ( distance sp15 sp6 ) 610.5700000000 )
( = ( distance sp15 sp7 ) 609.7700000000 )
( = ( distance sp15 sp8 ) 596.2000000000 )
( = ( distance sp15 sp9 ) 594.2300000000 )
( = ( distance sp15 cpsite5 ) 21.3400000000 )
( = ( distance sp15 pp9 ) 31.8000000000 )
( = ( distance sp15 pp10 ) 31.7200000000 )
( = ( distance sp15 pp11 ) 40.8300000000 )
( = ( distance sp15 pp12 ) 21.1100000000 )
( = ( distance sp15 sp11 ) 4.9500000000 )
( = ( distance sp15 sp12 ) 29.5100000000 )
( = ( distance sp15 sp13 ) 31.9100000000 )
( = ( distance sp15 sp14 ) 39.9100000000 )
)
(:goal
( and
(sampled pp9)(sampled pp10)(sampled pp11)(sampled pp12))
)
)
```

Listing A.6: PDDL problem: team_0 site5 to base

```
(define (problem subp_site5_to_base)
( :domain mmdomainextended )
( :objects
robot0 robot1 - robot
cpbase cpsite5 pp9 pp10 pp11 pp12 sp11 sp12 sp13 sp14 sp15 - pointofinterest
base site5 - site
)
( :init
( at robot0 sp12 )
( at_site robot0 site5 )
( waterconf robot0 )
( available robot0 )
( canswitch robot0 )
( canrelay robot0 )
( cansample robot0 )
( at robot1 pp12 )
```

```
( at_site robot1 site5 )
( waterconf robot1 )
( available robot1 )
( canswitch robot1 )
( canrelay robot1 )
( cansample robot1 )
( transition_poi cpbase )
( isswitchable cpbase )
( isrelay cpbase )
( ground cpbase )
( air cpbase )
( partofsite cpbase base )
( survey_poi cpsite5 )
( air cpsite5 )
( partofsite cpsite5 site5 )
( sample_poi pp9 )
( water pp9 )
( partofsite pp9 site5 )
( sample_poi pp10 )
( water pp10 )
( partofsite pp10 site5 )
( sample_poi pp11 )
( water pp11 )
( partofsite pp11 site5 )
( sample_poi pp12 )
( water pp12 )
( partofsite pp12 site5 )
( transition_poi sp11 )
( isswitchable sp11 )
( isrelay sp11 )
( water sp11 )
( air sp11 )
( partofsite sp11 site5 )
( transition_poi sp12 )
( isswitchable sp12 )
( isrelay sp12 )
( water sp12 )
( air sp12 )
( partofsite sp12 site5 )
( transition_poi sp13 )
( isswitchable sp13 )
( isrelay sp13 )
( water sp13 )
( air sp13 )
( partofsite sp13 site5 )
( transition_poi sp14 )
( isswitchable sp14 )
( isrelay sp14 )
( water sp14 )
( air sp14 )
( partofsite sp14 site5 )
( transition_poi sp15 )
( isswitchable sp15 )
( isrelay sp15 )
( water sp15 )
( air sp15 )
( partofsite sp15 site5 )
( = ( speedair robot0 ) 1.5000000000 )
( = ( speedwater robot0 ) 0.5000000000 )
( = ( energy robot0 ) 10000.0000000000 )
( = ( recharge_rate robot0 ) 10.0000000000 )
( = ( assesscost robot0 ) 0.0200000000 )
```

```
( = ( observecost robot0 ) 0.0100000000 )
( = ( maintainconsumption_water robot0 ) 0.0200000000 )
( = ( maintainconsumption_air robot0 ) 0.0800000000 )
( = ( navconsumption_water robot0 ) 0.5000000000 )
( = ( navconsumption_air robot0 ) 0.8000000000 )
( = ( switchduration_airwater robot0 ) 5.0000000000 )
( = ( switchcost_airwater robot0 ) 20.0000000000 )
( = ( takeoffduration_groundair robot0 ) 4.0000000000 )
( = ( takeoffost_groundair robot0 ) 10.0000000000 )
( = ( landingduration_airground robot0 ) 3.0000000000 )
( = ( landingcost_airground robot0 ) 5.0000000000 )
( = ( switchduration_waterair robot0 ) 8.0000000000 )
( = ( switchcost_waterair robot0 ) 30.0000000000 )
( = ( speedair robot1 ) 1.5000000000 )
( = ( speedwater robot1 ) 0.5000000000 )
( = ( energy robot1 ) 10000.0000000000 )
( = ( recharge_rate robot1 ) 10.0000000000 )
( = ( assesscost robot1 ) 0.0200000000 )
( = ( observecost robot1 ) 0.0100000000 )
( = ( maintainconsumption_water robot1 ) 0.0200000000 )
( = ( maintainconsumption_air robot1 ) 0.0800000000 )
( = ( navconsumption_water robot1 ) 0.5000000000 )
( = ( navconsumption_air robot1 ) 0.8000000000 )
( = ( switchduration_airwater robot1 ) 5.0000000000 )
( = ( switchcost_airwater robot1 ) 20.0000000000 )
( = ( takeoffduration_groundair robot1 ) 4.0000000000 )
( = ( takeoffost_groundair robot1 ) 10.0000000000 )
( = ( landingduration_airground robot1 ) 3.0000000000 )
( = ( landingcost_airground robot1 ) 5.0000000000 )
( = ( switchduration_waterair robot1 ) 8.0000000000 )
( = ( switchcost_waterair robot1 ) 30.0000000000 )
( = ( site_size base ) 10.0000000000 )
( = ( distance cpbase cpsite5 ) 1054.6500000000 )
( = ( distance cpbase sp11 ) 1030.9600000000 )
( = ( distance cpbase sp12 ) 1052.3700000000 )
( = ( distance cpbase sp13 ) 1065.8200000000 )
( = ( distance cpbase sp14 ) 1070.6400000000 )
( = ( distance cpbase sp15 ) 1033.9400000000 )
( = ( site_size site5 ) 40.0000000000 )
( = ( distance cpsite5 cpbase ) 1054.6500000000 )
( = ( distance cpsite5 sp11 ) 23.7300000000 )
( = ( distance cpsite5 sp12 ) 18.3700000000 )
( = ( distance cpsite5 sp13 ) 11.8500000000 )
( = ( distance cpsite5 sp14 ) 19.2700000000 )
( = ( distance cpsite5 sp15 ) 21.3400000000 )
( = ( distance pp9 pp10 ) 11.3000000000 )
( = ( distance pp9 pp11 ) 28.2100000000 )
( = ( distance pp9 pp12 ) 13.7300000000 )
( = ( distance pp9 sp11 ) 31.2200000000 )
( = ( distance pp9 sp12 ) 4.9100000000 )
( = ( distance pp9 sp13 ) 25.0800000000 )
( = ( distance pp9 sp14 ) 17.7900000000 )
( = ( distance pp9 sp15 ) 31.8000000000 )
( = ( distance pp10 pp9 ) 11.3000000000 )
( = ( distance pp10 pp11 ) 17.0200000000 )
( = ( distance pp10 pp12 ) 20.3900000000 )
( = ( distance pp10 sp11 ) 32.8900000000 )
( = ( distance pp10 sp12 ) 13.6200000000 )
( = ( distance pp10 sp13 ) 14.4400000000 )
( = ( distance pp10 sp14 ) 9.6700000000 )
( = ( distance pp10 sp15 ) 31.7200000000 )
( = ( distance pp11 pp9 ) 28.2100000000 )
```

```
( = ( distance pp11 pp10 ) 17.0200000000 )
( = ( distance pp11 pp12 ) 36.2100000000 )
( = ( distance pp11 sp11 ) 43.6800000000 )
( = ( distance pp11 sp12 ) 30.3000000000 )
( = ( distance pp11 sp13 ) 9.6900000000 )
( = ( distance pp11 sp14 ) 13.9600000000 )
( = ( distance pp11 sp15 ) 40.8300000000 )
( = ( distance pp12 pp9 ) 13.7300000000 )
( = ( distance pp12 pp10 ) 20.3900000000 )
( = ( distance pp12 pp11 ) 36.2100000000 )
( = ( distance pp12 sp11 ) 19.1500000000 )
( = ( distance pp12 sp12 ) 11.6800000000 )
( = ( distance pp12 sp13 ) 30.0600000000 )
( = ( distance pp12 sp14 ) 29.0300000000 )
( = ( distance pp12 sp15 ) 21.1100000000 )
( = ( distance sp11 cpbase ) 1030.9600000000 )
( = ( distance sp11 cpsite5 ) 23.7300000000 )
( = ( distance sp11 pp9 ) 31.2200000000 )
( = ( distance sp11 pp10 ) 32.8900000000 )
( = ( distance sp11 pp11 ) 43.6800000000 )
( = ( distance sp11 pp12 ) 19.1500000000 )
( = ( distance sp11 sp12 ) 28.6300000000 )
( = ( distance sp11 sp13 ) 34.9700000000 )
( = ( distance sp11 sp14 ) 41.3400000000 )
( = ( distance sp11 sp15 ) 4.9500000000 )
( = ( distance sp12 cpbase ) 1052.3700000000 )
( = ( distance sp12 cpsite5 ) 18.3700000000 )
( = ( distance sp12 pp9 ) 4.9100000000 )
( = ( distance sp12 pp10 ) 13.6200000000 )
( = ( distance sp12 pp11 ) 30.3000000000 )
( = ( distance sp12 pp12 ) 11.6800000000 )
( = ( distance sp12 sp11 ) 28.6300000000 )
( = ( distance sp12 sp13 ) 25.8800000000 )
( = ( distance sp12 sp14 ) 19.8000000000 )
( = ( distance sp12 sp15 ) 29.5100000000 )
( = ( distance sp13 cpbase ) 1065.8200000000 )
( = ( distance sp13 cpsite5 ) 11.8500000000 )
( = ( distance sp13 pp9 ) 25.0800000000 )
( = ( distance sp13 pp10 ) 14.4400000000 )
( = ( distance sp13 pp11 ) 9.6900000000 )
( = ( distance sp13 pp12 ) 30.0600000000 )
( = ( distance sp13 sp11 ) 34.9700000000 )
( = ( distance sp13 sp12 ) 25.8800000000 )
( = ( distance sp13 sp14 ) 15.3800000000 )
( = ( distance sp13 sp15 ) 31.9100000000 )
( = ( distance sp14 cpbase ) 1070.6400000000 )
( = ( distance sp14 cpsite5 ) 19.2700000000 )
( = ( distance sp14 pp9 ) 17.7900000000 )
( = ( distance sp14 pp10 ) 9.6700000000 )
( = ( distance sp14 pp11 ) 13.9600000000 )
( = ( distance sp14 pp12 ) 29.0300000000 )
( = ( distance sp14 sp11 ) 41.3400000000 )
( = ( distance sp14 sp12 ) 19.8000000000 )
( = ( distance sp14 sp13 ) 15.3800000000 )
( = ( distance sp14 sp15 ) 39.9100000000 )
( = ( distance sp15 cpbase ) 1033.9400000000 )
( = ( distance sp15 cpsite5 ) 21.3400000000 )
( = ( distance sp15 pp9 ) 31.8000000000 )
( = ( distance sp15 pp10 ) 31.7200000000 )
( = ( distance sp15 pp11 ) 40.8300000000 )
( = ( distance sp15 pp12 ) 21.1100000000 )
( = ( distance sp15 sp11 ) 4.9500000000 )
```

```
( = ( distance sp15 sp12 ) 29.5100000000 )
( = ( distance sp15 sp13 ) 31.9100000000 )
( = ( distance sp15 sp14 ) 39.9100000000 )
)
(:goal
( and
( groundconf robot0)
( groundconf robot1)
)
)
)
```

OPTIC output plans

Listing A.7: OPTIC plan for team_0 base to site3 problem

```
0.000: (takeoff robot1 cpbase) [4.000]
0.000: (takeoff robot0 cpbase) [4.000]
4.001: (change_site robot0 base site3 cpbase sp9) [295.013]
4.001: (change_site robot1 base site3 cpbase sp6) [283.606]
287.608: (navigation_air robot1 sp6 cpsite3 site3) [9.480]
299.015: (navigation_air robot0 sp9 cpsite3 site3) [4.966]
303.982: (observe_2r robot1 robot0 cpsite3 site3) [10.000]
313.983: (navigation_air robot0 cpsite3 sp9 site3) [4.966]
313.983: (navigation_air robot1 cpsite3 sp9 site3) [4.966]
318.950: (switch_airwater robot0 sp9 site3) [5.000]
318.950: (switch_airwater robot1 sp9 site3) [5.000]
323.951: (navigation_water robot1 sp9 pp7 site3) [25.440]
334.392: (relay_data robot0 sp9 site3) [45.000]
349.392: (sample robot1 pp7 site3) [30.000]
379.393: (navigation_water robot1 pp7 pp6 site3) [17.900]
382.294: (relay_data robot0 sp9 site3) [45.000]
397.294: (sample robot1 pp6 site3) [30.000]
427.295: (navigation_water robot1 pp6 pp5 site3) [47.500]
459.796: (relay_data robot0 sp9 site3) [45.000]
474.796: (sample robot1 pp5 site3) [30.000]
```

Listing A.8: OPTIC plan for team_0 site3 to site5 problem

```
0.000: (switch_waterair robot0 sp9) [8.000]
0.000: (navigation_water robot1 pp5 sp9 site3) [30.180]
8.001: (change_site robot0 site3 site5 sp9 cpsite5) [410.146]
30.181: (switch_waterair robot1 sp9) [8.000]
38.182: (change_site robot1 site3 site5 sp9 sp11) [394.346]
418.148: (observe robot0 cpsite5 site5) [26.666]
432.529: (navigation_air robot1 sp11 sp13 site5) [23.313]
444.815: (navigation_air robot0 cpsite5 sp14 site5) [12.846]
455.843: (switch_airwater robot1 sp13 site5) [5.000]
457.662: (navigation_air robot0 sp14 sp12 site5) [13.200]
460.844: (navigation_water robot1 sp13 pp11 site5) [19.380]
470.863: (switch_airwater robot0 sp12 site5) [5.000]
475.864: (relay_data robot0 sp12 site5) [45.000]
480.225: (sample robot1 pp11 site5) [30.000]
510.226: (navigation_water robot1 pp11 pp10 site5) [34.040]
529.267: (relay_data robot0 sp12 site5) [45.000]
544.267: (sample robot1 pp10 site5) [30.000]
574.268: (navigation_water robot1 pp10 pp9 site5) [22.600]
581.869: (relay_data robot0 sp12 site5) [45.000]
596.869: (sample robot1 pp9 site5) [30.000]
```

```
626.870: (navigation_water robot1 pp9 pp12 site5) [27.460]
639.331: (relay_data robot0 sp12 site5) [45.000]
654.331: (sample robot1 pp12 site5) [30.000]
```

Listing A.9: OPTIC plan for team_0 site5 to base problem

```
0.000: (switch_waterair robot0 sp12) [8.000]
0.000: (navigation_water robot1 pp12 sp12 site5) [23.360]
8.001: (change_site robot0 site5 base sp12 cpbase) [701.580]
23.361: (switch_waterair robot1 sp12) [8.000]
31.362: (change_site robot1 site5 base sp12 cpbase) [701.580]
709.582: (landing robot0 cpbase) [3.000]
732.943: (landing robot1 cpbase) [3.000]
```

A.3 PDDL-to-BT example for team_0

Following the synchronization and split of team_0 into two subteams described in Subsubsection 4.2.3.3, Fig. A.1 shows the Behavior Tree (BT) produced by the PlanSys2 BT Executor from the temporal plan of Listing A.7.

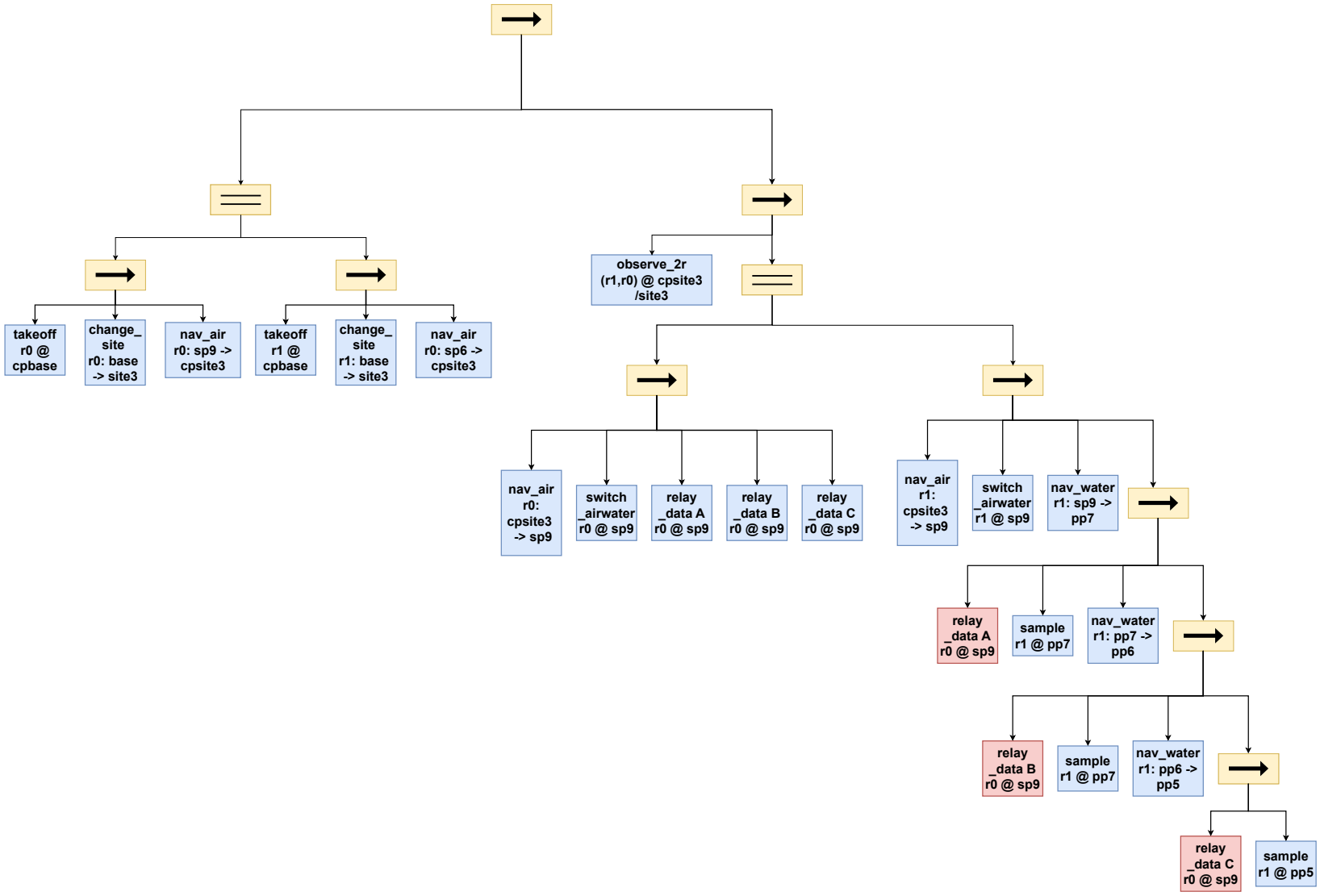


Figure A.1: Behavior Tree generated by the PlanSys2 BT Executor for team_0.