FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Enhancing Centralized Multi-Robot Path Planning and Coordination

Afonso Tomás de Magalhães Mateus

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Pedro Luís Cerqueira Gomes da Costa

July 30, 2026

Abstract

This dissertation addresses centralized *Multi-Robot Path Planning* (MRPP) for fleets of *Autonomous Mobile Robots* (AMRs) operating in shared environments. In this context, multiple robots must reach their assigned goals while avoiding collisions and satisfying temporal coordination constraints. The work focuses on the application of MRPP to a centralized multi-robot fleet coordination framework, where planning decisions must remain computationally efficient and compatible with practical execution constraints.

The work builds on an existing centralized coordination framework that relies on a sequential *Time-Enhanced A** (TEA*) planner, where robot paths are planned one at a time according to a priority ordering. Although this approach is suitable for practical deployment, its performance is highly sensitive to the selected ordering, and exhaustive evaluation of all possible permutations is infeasible for large fleets.

To address this limitation, this dissertation develops prioritization heuristics that generate alternative robot orderings using distance, topology, spatial distribution, predicted conflicts, and replanning feedback. These orderings are evaluated in an isolated planner testing framework and integrated into a parallel planning approach, where multiple TEA* instances are executed in parallel under a fixed planning time budget.

The experimental results show that heuristic-generated robot sequences improved the reliability of TEA* in finding valid solutions when compared with random priority selection. In the 180-robot large-scale scenarios, the heuristic-guided configurations increased the valid-run rate from $0/100$ to $100/100$ in one scenario and from $2/100$ to $100/100$ in another. The parallel approach further increased the ability to identify effective orderings within the available planning time. Among the heuristic-guided configurations, increasing parallelism reduced the mean time to the first valid solution from $7.32\,\mathrm{s}$ with a single worker to $2.83\,\mathrm{s}$ with five workers in the most restrictive large-scale scenario. Finally, the dissertation also introduces batching mechanisms to improve integrated execution consistency, extending the centralized coordination framework through prioritization heuristics, parallel TEA* planning, and more consistent processing of planning requests and robot state updates.

Keywords: Multi-Robot Path Planning, Autonomous Mobile Robots, Centralized Fleet Coordination, Prioritization Heuristics, Time-Enhanced A*, Parallel Planning, Robot Coordination.

Resumo

Esta dissertação aborda o planeamento de caminhos multi-robô, *Multi-Robot Path Planning* (MRPP), para frotas de robôs móveis autónomos, do inglês *Autonomous Mobile Robots* (AMRs), que operam em ambientes partilhados. Neste contexto, vários robôs devem alcançar os seus objetivos, evitando colisões e respeitando restrições temporais de coordenação. O trabalho foca-se na aplicação de MRPP a uma arquitetura centralizada de coordenação de frotas multi-robô, onde as decisões de planeamento devem ser computacionalmente eficientes e compatíveis com restrições de execução prática.

O trabalho tem por base uma arquitetura centralizada de coordenação que utiliza um planeador sequencial *Time-Enhanced A** (TEA*), no qual os robôs são planeados um de cada vez segundo uma ordem de prioridades. Embora esta abordagem seja adequada a cenários de aplicação prática, o seu desempenho é altamente sensível à ordem escolhida, sendo inviável avaliar exaustivamente todas as possíveis permutações para frotas de maior dimensão.

Para mitigar esta limitação, esta dissertação desenvolve heurísticas de priorização que geram ordens alternativas dos robôs com base em distância, topologia, distribuição espacial, conflitos previstos e informação de replaneamento. Estas ordens são avaliadas num ambiente de teste isolado do planeador e integradas numa abordagem de planeamento paralelo, onde várias instâncias do TEA* são executadas em paralelo sob um tempo máximo de planeamento.

Os resultados experimentais mostram que as sequências geradas por heurísticas melhoraram a fiabilidade na obtenção de soluções válidas pelo TEA* quando comparadas com a seleção aleatória de prioridades. Nos cenários de grande escala com 180 robôs, as configurações guiadas por heurísticas aumentaram a taxa de execuções válidas de $0/100$ para $100/100$ num cenário e de $2/100$ para $100/100$ noutro. A abordagem paralela aumentou ainda a capacidade de identificar ordens úteis dentro do tempo disponível. Entre as configurações guiadas por heurísticas, o aumento do paralelismo reduziu o tempo médio até à primeira solução válida de 7.32 s com uma única thread para 2.83 s com cinco threads no cenário de grande escala mais restritivo. Por fim, a dissertação contribui também com mecanismos de *batching* para melhorar a consistência da execução integrada, estendendo a arquitetura centralizada de coordenação através de heurísticas de priorização, planeamento TEA* paralelo e processamento mais consistente de pedidos de planeamento e atualização do estado dos robôs.

Palavras-chave: Planeamento de Caminhos Multi-Robô, Robôs Móveis Autónomos, Coordenação Centralizada de Frotas, Heurísticas de Priorização, Time-Enhanced A*, Planeamento Paralelo, Coordenação de Robôs.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) [1] provide a global framework to achieve a better and more sustainable future for all. The framework comprises 17 goals addressing global challenges such as poverty, inequality, climate change, environmental degradation, peace, and justice.

This dissertation relates to the SDGs through its contribution to methods aimed at making centralized multi-robot fleet coordination more robust and efficient. The work can support more reliable and scalable robotic coordination in industrial and logistics environments through prioritization heuristics and parallel planning. The following table identifies the most relevant SDGs, the associated targets, the contribution of the dissertation, and the corresponding performance indicators.

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.

SDG 12 Ensure sustainable consumption and production patterns.

SDG	Target	Contribution	Indicators and Metrics
8	8.2	The proposed methods can contribute to technological upgrading and operational productivity by improving the robustness and effectiveness of centralized multi-robot fleet coordination in industrial and logistics scenarios.	Planning success rate; time to first valid solution; solution cost; planning performance in large-scale scenarios.
9	9.4	The proposed planning approach may support more efficient operation of robotic infrastructure by improving the ability to obtain valid coordination plans within a limited planning time.	Time to first valid solution; number of valid runs; solution cost; scalability across large-scale scenarios.
	9.5	The dissertation contributes to research and innovation in industrial robotic systems through the design, implementation, and evaluation of prioritization heuristics and a parallel planning framework.	Comparison with random priority selection; performance across controlled and large-scale scenarios; effect of worker thread configurations.
12	12.2	By reducing redundant planner executions and improving the use of the available planning budget, the proposed methods may indirectly reduce computational effort in the supporting computing infrastructure.	Time to complete the heuristic pool; solution cost; planning performance in large-scale scenarios.

Acknowledgements

I would like to express my sincere gratitude to Professor Pedro Costa, my supervisor, for his guidance, availability, and constant support throughout this dissertation. His advice helped me follow the right direction at important moments, and his calm and practical approach made the process clearer and less overwhelming.

I am also deeply grateful to Diogo Matos, my advisor at INESC TEC, for his continuous presence and support. His guidance, technical insight, and readiness to discuss ideas and decisions were essential throughout the development of this dissertation.

To Marco Moreira, my colleague in the mobile robotics team, I would like to express a special thank you for his friendship, good humor, and constant willingness to help throughout this journey. His presence made this process much more enjoyable.

I would also like to thank Héber Sobreira for always being available, supportive, and ready to offer good advice. His good humor and reassuring presence made many moments of this journey lighter.

To all members of the mobile robotics group, thank you for the companionship, collaboration, and mutual support during these months. Being surrounded by people who were always willing to help and share ideas made this experience much more meaningful.

I also want to thank all members of the 5DPO robotics team, with whom I shared a journey that I will always remember. More than teammates, many became friends for life. In particular, I would like to thank Daniel Silva and Pedro Lopes for all the work, challenges, laughter, and moments we shared together. I am also grateful to Professor António Paulo Moreira and Professor Paulo Costa for the support they always showed, both in curricular matters and in the extracurricular path of the team.

To Leonor, my love, I cannot thank you enough. Your patience and constant support carried me through the most difficult moments. Thank you for believing in me when I doubted myself, for listening, and for always being by my side. Without you, this path would have been much harder.

To my father José, my mother Rosa and my sister Diana, thank you for being my foundation. Your support and trust have always given me the strength to move forward. Everything I achieve is also a reflection of what you have given me, and this dissertation would not have been possible without you.

Finally, to my friends, thank you for being present and for helping me disconnect when I needed it most. Your support and friendship were essential to keep balance throughout this period.

Afonso Mateus

"For the things we have to learn before we can do them, we learn by doing them."

Aristotle

x

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AMR	*Autonomous Mobile Robot*
APF	*Artificial Potential Field*
CBS	*Conflict-Based Search*
CBSH	*Heuristic-Guided Conflict-Based Search*
CBSH-RTC	*Real-Time Conflict-Based Search with Heuristics*
CCBS	*Continuous-Time Conflict-Based Search*
CRIIS	*Centre for Robotics in Industry and Intelligent Systems*
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*
DoF	*Degrees of Freedom*
ECBS	*Enhanced Conflict-Based Search*
ECDF	*Empirical Cumulative Distribution Function*
iiLab	*Industry and Innovation Laboratory*
INESC TEC	*Institute for Systems and Computer Engineering, Technology and Science*
MAE	*Mean Absolute Error*
MAPF	*Multi-Agent Path Finding*
MRPP	*Multi-Robot Path Planning*
NP-hard	*Nondeterministic Polynomial-time Hard*
OOP	*Object-Oriented Programming*
ROS	*Robot Operating System*
SDGs	*Sustainable Development Goals*
SIPP	*Safe Interval Path Planning*
TEA*	*Time-Enhanced A**
YAML	*YAML Ain't Markup Language*

Chapter 1

Introduction

1.1 Context

The increasing adoption of Autonomous Mobile Robots (AMRs) in industrial, logistics, and service environments has intensified the need for effective Multi-Robot Path Planning (MRPP) in shared spaces. In such scenarios, robots must navigate efficiently while avoiding collisions and mutual interference, often under strict temporal and operational constraints. As fleet sizes grow, coordination becomes a key challenge that directly impacts system performance, scalability, and reliability.

A common formal abstraction for studying this problem is Multi-Agent Path Finding (MAPF), which focuses on computing collision-free trajectories for multiple agents moving from initial to goal states in a shared environment [2]. Although MAPF is usually formulated in terms of generic agents, its space-time representation of motion and conflicts provides a useful foundation for studying MRPP in centralized and decentralized coordination frameworks.

MAPF is computationally challenging, with optimal variants being Nondeterministic Polynomial-time hard (NP-hard) in the general case [3]. Consequently, a wide spectrum of solution approaches has been proposed, ranging from optimal but computationally demanding methods [4], [5] to heuristic and approximate techniques that favor scalability and real-time performance [6]. Within this landscape, MAPF serves as a foundational framework for reasoning about coordination and scalability in multi-agent systems, while this dissertation focuses on its application to centralized MRPP.

1.2 Motivation and Problem

This dissertation is developed in the context of research on centralized multi-robot fleet coordination conducted at the Institute for Systems and Computer Engineering, Technology and Science (INESC TEC), namely at the Centre for Robotics in Industry and Intelligent Systems (CRIIS) and the Industry and Innovation Laboratory (iiLab). Within this context, a centralized fleet coordination framework has been proposed to address multi-robot navigation under realistic

operational constraints, relying on a shared environment representation and a sequential *Time-Enhanced A** (TEA*) planning component [7], [8].

In this framework, robots are planned sequentially according to a predefined priority order, which avoids conflicts by construction within the planner's space-time reservation model, but makes the resulting solution highly sensitive to robot ordering. Different priority permutations can lead to substantially different outcomes, particularly in constrained environments with shared resources. As the number of possible orderings grows factorially with fleet size, exhaustive exploration becomes computationally infeasible.

This limitation motivates the development of prioritization heuristics capable of generating informative robot orderings without enumerating all possible permutations. By exploiting different properties of the planning instance, these heuristics aim to produce orderings that are more relevant than purely random priority selections.

However, no single heuristic is expected to perform best across all scenarios, since the usefulness of a priority rule depends on the spatial configuration of the robots, the structure of the environment, and the interactions between their paths. This motivates the investigation of a parallel planning strategy in which multiple heuristic-generated orderings are evaluated within the same planning cycle. By exploiting multithreading to execute several TEA* instances in parallel, the impact of ordering sensitivity can be reduced while preserving compatibility with the existing TEA* coordination framework.

1.3 Objectives

Building on these motivations, the primary objective of this dissertation is to study and improve the robustness and effectiveness of centralized MRPP within the existing TEA* planning framework used by the CRIIS fleet coordination system. This objective involves analyzing how robot sequencing affects sequential TEA* planning and developing prioritization heuristics capable of generating informative robot sequences.

Beyond the design of individual heuristics, this dissertation aims to evaluate how multiple priority orderings can be explored within the same planning cycle through parallel execution. The proposed work therefore seeks to reduce the dependence on a single robot sequence while preserving compatibility with the existing coordination architecture.

Finally, the dissertation also aims to analyze temporal consistency issues observed during integrated execution of the complete coordination stack, with particular attention to redundant replanning and inconsistent processing of robot state updates.

1.4 Contributions

This dissertation builds upon the centralized fleet coordination system described by Matos et al. [7] and proposes an extension of the sequential TEA* planner aimed at reducing its sensitivity to robot priority ordering. One contribution is the development of a set of prioritization heuristics that

generate alternative robot orderings from different properties of the planning instance, providing structured alternatives to random priority selection.

A second contribution is the integration of these heuristics into a multithreaded planning approach, where multiple TEA* instances explore different priority orderings within the same planning time. The goal is to assess whether more robust coordination solutions with lower plan cost can be obtained without modifying the core architecture of the existing system.

By evaluating a broader and more informative set of priority orderings, the proposed approach aims to improve performance in the constrained multi-robot scenarios considered in this dissertation, where the ordering strongly affects solution feasibility.

An additional outcome of this dissertation is the preparation and submission of a scientific paper to the 9th Iberian Robotics Conference, ROBOT2026. The submitted work reports part of the heuristic prioritization study developed in this dissertation, contributing to the dissemination of the proposed approach within the robotics research community.

1.5 Document Structure

This dissertation is organized as follows. Chapter 2 presents the fundamental concepts required to support the work, including MRPP, the MAPF abstraction, graph search, machine learning concepts for clustering and runtime analysis, and parallel computing. Chapter 3 reviews the state of the art in multi-robot coordination and related MAPF approaches.

Chapter 4 describes the CRIIS fleet coordination system used as the starting point for this work, including its centralized planning pipeline and the role of TEA* as the underlying path planning method.

Chapter 5 presents the proposed prioritization heuristics, which generate alternative robot orderings for TEA* from different sources of information, including distance, roadmap structure, local robot distribution, predicted conflicts, and replanning feedback. Their individual behavior is evaluated in Chapter 6 using an isolated Path Planner testing framework, allowing the effect of each ordering strategy to be analyzed independently from the remaining coordination stack.

Chapter 7 introduces the proposed parallel planning framework, including the worker thread implementation, heuristic dispatching mechanism, timeout handling, and redundant heuristic selection strategy. The performance of this framework is then assessed in Chapter 8, with emphasis on the effect of the number of worker threads and on the contribution of the heuristic dispatching strategy.

Finally, Chapter 9 analyzes temporal consistency issues observed during execution of the complete CRIIS coordination stack, focusing on redundant replanning operations and on the batching mechanisms introduced for planning requests and robot state updates. Chapter 10 summarizes the main conclusions and outlines possible directions for future work.

Chapter 2

Background and Fundamental Aspects

This chapter presents the fundamental concepts and background knowledge required to contextualize the research developed in this dissertation. Section 2.1 introduces the MRPP problem considered in this work and explains how it can be formalized through the MAPF abstraction. Section 2.2 discusses the environment representations used to instantiate this type of planning problem. Section 2.3 explores classical graph search algorithms that serve as building blocks for many MRPP methods. Section 2.4 introduces machine learning concepts used to support the heuristic analysis. Finally, Section 2.5 presents foundational notions of parallel and concurrent computing.

2.1 MRPP Problem and MAPF Abstraction

The problem addressed in this dissertation is centralized MRPP, where multiple robots must move from their initial states to their goal states while sharing the same environment. Each robot must reach its goal without colliding with other robots, and the resulting set of trajectories should satisfy a global performance criterion.

A widely used abstraction for formalizing this type of coordination problem is MAPF. In MAPF, the moving entities are usually referred to as agents, but in the context of this dissertation they correspond to robots in a fleet coordination system. The formulation adopted here follows the conceptual definition of MAPF presented by Stern et al. [2], while being interpreted in the MRPP setting considered throughout this work.

From a formal perspective, let there be a set of m robots $R = \{r_1, r_2, \ldots, r_m\}$. Each robot $r_i \in R$ is associated with an initial state s_i and a goal state g_i.

A solution for robot r_i is defined as a trajectory P_i, which specifies the evolution of the robot state over time. In discrete formulations, a trajectory can be represented as a sequence of states

$$P_i = \langle p_i^0, p_i^1, \ldots, p_i^T \rangle \tag{2.1}$$

where $p_i^0 = s_i$ and $p_i^T = g_i$. The explicit consideration of time is essential, since interactions and conflicts between robots arise from the simultaneous execution of their trajectories in a shared space-time domain [2].

A valid solution must ensure that no two robots are in conflict during their motion, where conflicts arise when robots attempt to occupy incompatible states in space and time. The most common type of conflict is the position conflict, which occurs when two robots occupy the same state at the same time step, i.e., $p_i^t = p_j^t$ for $i \neq j$. Other types of conflicts may arise depending on the motion model and representation adopted, but all correspond to violations of mutual feasibility constraints among robots [2].

The objective is to compute a set of conflict-free trajectories $\{P_1, P_2, \ldots, P_m\}$ that minimize a global cost function C. This cost function is defined over the joint set of robot trajectories and encodes the chosen optimization criterion. Common objectives include minimizing the time at which the last robot reaches its goal, known as the makespan, or minimizing the sum of individual traversal costs. Formally:

Find $\{P_1, P_2, \ldots, P_m\}$ such that all trajectories are conflict-free and $C(\{P_i\})$ is minimized.

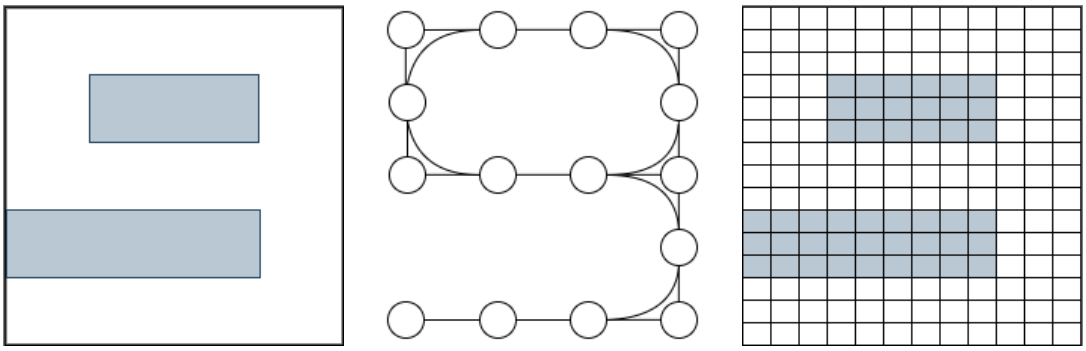
This formulation inherits the combinatorial nature of MAPF. In the general case, optimal MAPF is NP-hard, as formally established in earlier work [3] and refined in subsequent complexity analyses [9]. This inherent complexity motivates the development of graph-based planning algorithms that balance solution quality with computational efficiency in practical MRPP systems [10].

2.2 Environment Representation

Although MRPP can be described independently of a specific environment representation, practical planning methods require a concrete state space in which robot motion and conflicts can be evaluated. In multi-robot planning literature, different representations have been adopted, primarily differing in how space and time are modeled and in the resulting level of abstraction.

Common approaches include discrete representations, such as grids and graphs, where robots occupy a finite set of states and move between neighboring states in discrete time steps, as well as continuous formulations, where robot motion is described by continuous trajectories and conflicts arise from geometric interactions. Figure 2.1 illustrates how the same workspace can be modeled under different discrete representations.

While continuous formulations offer higher modeling fidelity, discrete representations are widely used in classical MAPF and MRPP settings due to their simplicity and structured abstraction. In this dissertation, the focus is placed on grid and graph representations, with particular emphasis on graph representations because the considered fleet coordination framework performs planning over a shared roadmap.



(a) Workspace layout (b) Graph representation (c) Grid representation

Figure 2.1: Illustration of different discrete environment representations for the same workspace.

2.2.1 Grid Representation

In grid formulations, the environment is discretized into a regular two- or three-dimensional array of uniformly sized cells, each classified as free or occupied. Robots move between neighboring cells according to a fixed connectivity structure, yielding a simple and intuitive abstraction that has been widely adopted in early MAPF studies [2].

A key advantage of grid representations is their simplicity and direct correspondence to occupancy grids used in robotic perception. However, grid resolution introduces an inherent trade-off. Coarse grids may fail to capture narrow passages or thin obstacles, while fine grids incur substantial memory and computational costs [11], [12]. In addition, the discrete nature of grid motion complicates the integration of continuous-time dynamics, often requiring post-processing to obtain executable trajectories [12].

For these reasons, graph representations are particularly relevant in this dissertation, since the considered fleet coordination framework performs planning over a shared roadmap [7]. By allowing arbitrary topologies and connectivity patterns, graph representations provide a convenient abstraction for modeling structured environments and for supporting centralized coordination.

2.2.2 Graph Representation

In graph formulations the environment is modeled as a discrete graph that captures admissible robot states and their connectivity. This follows the modeling approach described by Stern et al. [2], while being interpreted here in the context of multi-robot fleet coordination.

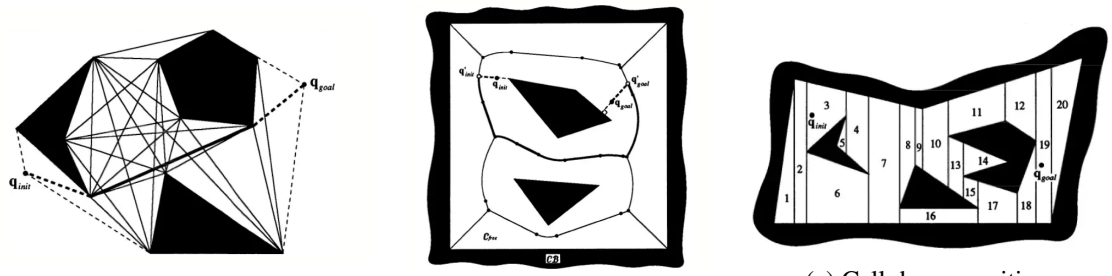
Formally, the environment is represented as a graph $G = (V, E)$, where each vertex $v \in V$ corresponds to a valid state that a robot may occupy, and each edge $(u, v) \in E$ encodes a feasible transition between states. The graph may be directed or undirected, depending on the characteristics of the environment and on the motion constraints imposed by the planning system.

In this representation, the state p_i^t of robot r_i at time step t corresponds to a vertex of the graph. At the next time step, the robot may either remain at the same vertex or move to an adjacent vertex connected by an edge, corresponding respectively to waiting and movement actions [2].

2.2.2.1 Graph Construction Approaches

Several classical approaches have been proposed to construct graph representations of planning environments. These include roadmap approaches such as visibility graphs, which connect mutually visible obstacle vertices [13], Voronoi diagram representations that emphasize obstacle clearance [14], and cell decomposition techniques that subdivide the free space into regions whose adjacency defines a connectivity graph [15], [16]. Representative examples of these constructions are illustrated in Figure 2.2.

In this dissertation, the objective is not to compare graph construction methods, but to perform multi-robot coordination over an existing graph representation. The underlying roadmap is therefore assumed to be available, and the planning problem consists of computing coordinated robot trajectories on top of this discrete abstraction.



(a) Visibility graph (b) Voronoi diagram (c) Cell decomposition

Figure 2.2: Classical approaches for constructing graph representations [15].

2.3 Search and Planning Foundations

Classical graph search techniques provide an important foundation for MRPP and MAPF methods, since they are commonly used to compute individual robot paths and to support coordination strategies. In the context of this dissertation, these methods are particularly relevant because the considered fleet coordination framework performs planning over a graph representation of the environment.

Accordingly, this section reviews graph search and planning concepts that underlie the TEA* planner used in this work and many related multi-robot coordination approaches. The focus is first placed on shortest path search in graphs and then on heuristic search, with particular emphasis on Dijkstra's algorithm and A*.

2.3.1 Search Problems on Graphs

A search problem on a graph consists of finding a sequence of transitions that leads from a given initial state to a goal state while satisfying feasibility constraints and optimizing a specified cost criterion [17].

Formally, a search problem is defined by a state space, an initial state, a set of goal states, a transition model that specifies the allowable moves between states, and a cost function that assigns a non-negative cost to each transition or path [17]. When the state space is represented explicitly as a graph, vertices correspond to states and edges encode feasible transitions, possibly associated with traversal costs.

A solution to a graph search problem is a path in the graph that connects the initial vertex to a goal vertex. Among all feasible solutions, particular interest is often placed on optimal solutions, defined as those that minimize the cumulative cost along the path. The shortest path problem in weighted graphs is a canonical instance of this formulation and has been extensively studied in both theoretical and applied contexts [18].

Search algorithms typically operate by incrementally exploring the graph starting from the initial state, expanding states according to a specific strategy and maintaining a frontier of candidate paths. Different search strategies define different orders in which states are expanded, which directly affects properties such as completeness, optimality, and computational efficiency [19].

2.3.1.1 Dijkstra's Algorithm

Dijkstra's algorithm is a classical graph search algorithm for solving the shortest path problem in weighted graphs with non-negative edge costs. Originally introduced by Dijkstra [20], it provides a systematic procedure for computing the minimum-cost path from a given initial vertex to all other reachable vertices in the graph.

The algorithm operates by incrementally expanding vertices in order of increasing accumulated path cost from the initial state. At each step, the vertex with the lowest known cost is selected for expansion, and its outgoing edges are relaxed to update the cost of reaching neighboring vertices. This process continues until the goal vertex is reached or all reachable vertices have been explored [18]. A key property of Dijkstra's algorithm is that, under the assumption of non-negative edge costs, once a vertex is expanded, the cost associated with it is guaranteed to be optimal. As a result, the algorithm is both complete and optimal for this class of graphs [18].

From a search perspective, Dijkstra's algorithm can be viewed as a best-first search strategy in which the expansion order is determined by the accumulated path cost from the initial vertex. This accumulated cost, commonly denoted by $g(n)$ for a vertex n, corresponds to the sum of the costs of the edges along the path from the initial vertex to n

$$g(n) = \sum_{(u,v)\in P_n} c(u,v), \tag{2.2}$$

where P_n denotes the path to n and $c(u,v)$ is the cost of the edge from vertex u to vertex v.

Regarding computational complexity, the cost of Dijkstra's algorithm depends on the data structure used to manage the frontier. When implemented with a binary heap priority queue and an

adjacency-list graph representation, the algorithm has worst-case time complexity

$$O\big((|V|+|E|)\log|V|\big), \tag{2.3}$$

where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph [18]. This bound results from inserting and updating vertices in the priority queue while relaxing the graph edges.

In MRPP settings, Dijkstra's algorithm is often used as a baseline within more complex coordination frameworks, serving as a reference point for evaluating the benefits of more advanced search strategies.

2.3.1.2 A* Search Algorithm

A* is a heuristic-based graph search algorithm that extends Dijkstra's algorithm by incorporating problem-specific information to guide the search toward the goal more efficiently. Originally introduced by Hart, Nilsson, and Raphael [19], A* is designed to compute optimal paths while reducing the number of explored states when compared to uninformed search methods.

The algorithm evaluates each vertex using an objective function

$$f(n) = g(n) + h(n), \tag{2.4}$$

where $g(n)$ denotes the accumulated cost from the initial vertex to vertex n, as defined in Equation (2.2), and $h(n)$ is a heuristic estimate of the remaining cost from n to a goal vertex.

This algorithm maintains a frontier of candidate vertices ordered by increasing values of $f(n)$ and iteratively expands the vertex with the lowest estimated total cost. When the heuristic function is admissible, meaning that it never overestimates the true remaining cost to the goal, A* is guaranteed to be both complete and optimal [19], [21].

Figure 2.3 illustrates the resulting difference in exploration behavior between Dijkstra's algorithm and A*. The light blue marker denotes the initial state, the green marker denotes the goal state, and darker blue cells correspond to expanded nodes. The environment is modeled as a grid to facilitate visualization, and the Manhattan distance heuristic is used to guide the A* search. As can be observed, in this particular example, the heuristic effectively directs the search toward the goal, resulting in a focused expansion pattern, whereas Dijkstra's algorithm exhibits a nearly radial expansion that explores the state space more uniformly. However, this figure should be interpreted as an illustrative case rather than as a general performance guarantee. The advantage of A* depends on the informativeness of the heuristic function and on the structure of the search problem. With a weak or uninformative heuristic, its exploration behavior may become much closer to that of Dijkstra's algorithm.

From a search perspective, Dijkstra's algorithm can be seen as a special case of A* in which the heuristic function is identically zero. Under this interpretation, A* generalizes Dijkstra by allowing heuristic guidance while preserving optimality guarantees under appropriate conditions [18].

The computational complexity of A* also depends on the implementation of the frontier and on the effectiveness of the heuristic function. In the worst case, when the heuristic provides no useful guidance, A* may explore the same search space as Dijkstra's algorithm. With a binary heap priority queue and an adjacency-list graph representation, its worst-case time complexity is therefore the same as in Equation (2.3).

In practice, an informative admissible heuristic can reduce the number of expanded vertices, but the worst-case bound remains the same.

(a) Dijkstra (b) A*

Figure 2.3: Comparison of node expansion patterns produced by Dijkstra's algorithm and A*.

2.4 Machine Learning Foundations for Heuristic Analysis

The main focus of this dissertation lies in centralized multi-robot path planning and coordination. Within this context, some of the methods used to design and evaluate the proposed heuristics rely on basic machine learning concepts. In particular, clustering methods are relevant for analyzing the spatial distribution of robots, while regression models can be used to study and estimate the computational cost of heuristic functions as a function of relevant problem parameters.

This section introduces the machine learning concepts required to support these components. The goal is not to provide an exhaustive review of machine learning methods, but rather to establish the background necessary to understand the clustering-based heuristics and the runtime modeling procedures used in the experimental evaluation.

2.4.1 Unsupervised and Supervised Learning

Machine learning methods are commonly divided into unsupervised and supervised learning approaches [22]. Unsupervised learning methods operate on data without predefined target labels. Their objective is to identify structure, patterns, or groups within the data. Clustering is one of the most common unsupervised learning tasks, aiming to group samples that are similar according to a given distance or similarity criterion [22].

In contrast, supervised learning methods are trained from examples composed of input variables and known target outputs. The objective is to learn a mapping that can predict the target output for

new, unseen inputs. Regression is a typical supervised learning task, where the target variable is continuous and can be used to model the execution time of heuristic functions.

2.4.2 Clustering Methods

Clustering methods aim to partition a set of data points into groups, called clusters, so that points within the same cluster are more similar to each other than to points in different clusters. The notion of similarity depends on the data representation and analysis objective, and is typically expressed through a proximity measure, such as the Euclidean distance [22].

In multi-robot coordination, clustering can be used to characterize the spatial distribution of robots in the environment. Robots located close to each other may compete for the same spatial resources, create local congestion, or increase the probability of future interactions. For this reason, clustering is useful for designing prioritization heuristics that account not only for individual robot properties, but also for the local structure of the fleet.

In this context, two representative clustering approaches are considered: K-means and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). K-means is a centroid-based method that partitions data into a predefined number of groups [23], [24]. DBSCAN is a density-based method that identifies clusters from local point density and can classify isolated points as noise [25]. These methods were selected because they are simple and widely used, with K-means providing a classical clustering reference and DBSCAN a common density-based alternative.

2.4.2.1 K-means

K-means is a classical centroid-based clustering algorithm originally introduced by MacQueen [23]. The iterative least-squares procedure commonly associated with K-means was later formalized by Lloyd [24], and corresponds to the standard alternating optimization process used to update cluster assignments and centroids.

The algorithm partitions a set of data points into K clusters, where K is defined in advance. Due to its simplicity and computational efficiency, K-means is one of the most widely used clustering methods and is often adopted as a reference approach when discussing spatial grouping problems.

Given a set of observations $X = \{x_1, x_2, \ldots, x_N\}$, the algorithm starts by selecting K initial centroids μ_k, with $k = 1, \ldots, K$. These centroids are typically initialized from existing data points or by using a specific initialization strategy. Each point x_n is then assigned to the cluster whose centroid is closest to it. After all points have been assigned, each centroid is updated as the mean of the points currently assigned to its cluster. This assignment and update process is repeated until the centroids or the cluster assignments no longer change significantly.

Figure 2.4 illustrates this iterative process. The initial data distribution is shown first, followed by the initialization of the centroids and the successive assignment and update steps. As the centroids move toward the center of their assigned points, the cluster assignments become progressively more stable.

Since K-means requires K to be specified beforehand, different procedures can be used to estimate a suitable number of clusters. One common approach is the elbow method [26], which runs K-means for different values of K and analyzes how the within-cluster sum of squares decreases as K increases. The selected value is usually the point where adding more clusters produces only a limited additional reduction in this quantity. Although intuitive, this method can be subjective when the curve does not present a clear elbow.

K-means is simple and computationally efficient, which makes it widely used in practice. However, it has important limitations. First, the number of clusters K must be specified before running the algorithm. Second, all points are forced to belong to one of the clusters, even if they are isolated or do not naturally belong to any group. Third, because clusters are represented by centroids, K-means is better suited to compact and approximately spherical clusters, and may perform poorly when the underlying groups have irregular shapes.

These limitations are relevant in multi-robot path planning scenarios. The number of spatial groups formed by robots is not usually known in advance and may vary between planning requests. Moreover, isolated robots should not necessarily be forced into a cluster, since doing so may artificially influence the priority score of unrelated robots.

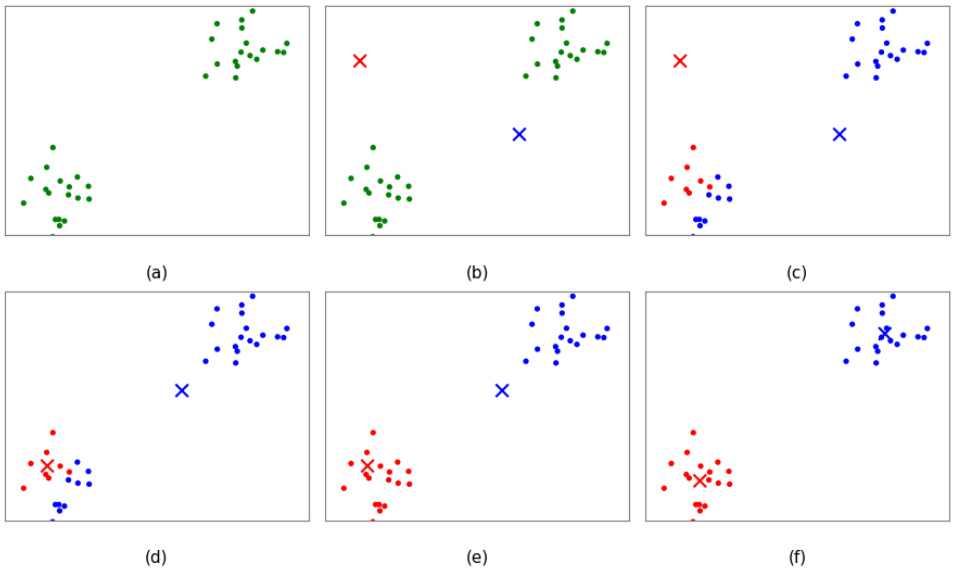


Figure 2.4: Illustration of successive K-means iterations, adapted from [27].

2.4.2.2 DBSCAN

DBSCAN, introduced by Ester et al. [25], is a density-based clustering algorithm designed to identify groups of points from local density patterns. Unlike centroid-based methods such as K-means, DBSCAN does not require the number of clusters to be specified in advance. Instead, clusters are formed by connecting points that belong to sufficiently dense regions of the data space.

The algorithm is based on two parameters: a distance threshold ε and a minimum number of points MinPts. Given a point x_i, its ε-neighborhood is defined as

$$N_\varepsilon(x_i) = \{x_j \in X : \|x_j - x_i\| \leq \varepsilon\}. \tag{2.5}$$

A point is classified as a core point if its ε-neighborhood contains at least MinPts points. A point that does not satisfy this condition, but lies within the ε-neighborhood of a core point, is classified as a border point. Points that are neither core points nor border points are classified as noise.

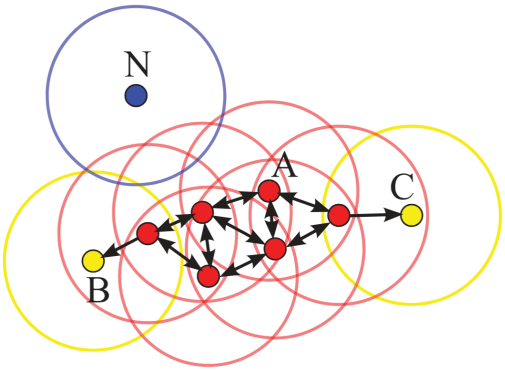


Figure 2.5: Illustration of the DBSCAN cluster model [28].

Figure 2.5 illustrates these concepts. Point *A* represents a core point, since its neighborhood satisfies the density condition. Points *B* and *C* are border points, because they do not define dense regions by themselves but are density-reachable from a core point. Point *N* is not density-reachable from any core point and is therefore classified as noise.

Clusters are obtained by grouping density-connected points. Starting from a core point, DBSCAN expands the cluster by recursively including neighboring points that satisfy the density condition. This process allows the algorithm to identify spatial groups without forcing every point to belong to a cluster.

In multi-robot path planning scenarios, robots may form dense local groups in some regions of the environment, while other robots may remain isolated. Since DBSCAN does not require the number of clusters to be directly defined beforehand and can explicitly classify isolated points as noise, it is well suited for detecting spatial concentrations of robots without introducing artificial cluster assignments.

2.4.3 Linear Regression

Regression methods are supervised learning techniques used to model the relationship between a set of explanatory variables and a continuous target variable [22], [29]. Linear regression is one of the simplest and most interpretable regression models. It assumes that the target variable can be approximated as a linear combination of the input variables.

Given a set of input features $x_1, x_2, \ldots, x_p$, a multiple linear regression model estimates a target variable y as

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \tag{2.6}$$

where $\hat{y}$ is the predicted value, β_0 is the intercept, and β_j are the model coefficients associated with each input feature.

The coefficients are commonly estimated by minimizing the residual sum of squares between the observed values y_i and the predicted values $\hat{y}_i$ [29]:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.7}$$

Due to its simplicity and interpretability, linear regression can be used as an exploratory model for runtime analysis in multi-robot path planning. In this context, the target variable may correspond to the measured execution time of a planning or heuristic function, while the input features may depend on the specific heuristic being analyzed and on the properties of the planning scenario.

Although linear regression assumes a linear relationship between inputs and output, this does not prevent it from being useful for exploratory runtime analysis. In particular, it can help identify observed runtime trends when its input features are chosen in relation to the expected behavior of the algorithm. For this reason, linear regression should be understood as a complementary analysis tool, where theoretical complexity analysis helps suggest relevant features, and regression is used to assess how those features relate to the measured runtime in practice.

2.5 Foundations of Parallel and Concurrent Computing

Coordinating several robots in a shared environment requires the planner to reason about multiple paths, interactions, and timing constraints within the same planning cycle. As the fleet size increases, the computational effort associated with path computation and conflict handling can grow rapidly, making sequential execution a potential limitation.

Concepts from parallel and concurrent computing are therefore relevant for understanding how multiple planning tasks can be executed and synchronized within the same computational system. This section introduces these concepts as a foundation for the parallel planning framework presented later in this dissertation.

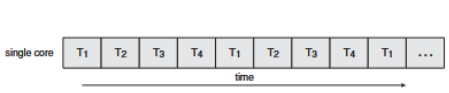
2.5.1 Parallelism and Concurrency

Parallelism and concurrency are closely related but conceptually distinct notions that describe different aspects of task execution within a computational system.

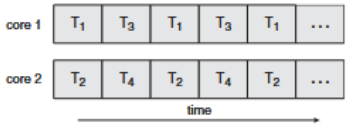
In this context, parallelism refers to the simultaneous execution of multiple computational tasks, typically with the objective of improving performance or reducing overall execution time. It

relies on the availability of multiple processing units, such as multi-core processors, and focuses on exploiting hardware resources to increase computational throughput [30].

Concurrency, in contrast, concerns the logical composition of multiple tasks that make progress over overlapping periods of time, regardless of whether they are executed simultaneously in physical terms. These concepts are visually illustrated in Figure 2.6.



(a) Concurrent execution



(b) Parallel execution

Figure 2.6: Conceptual distinction between concurrency and parallelism [31].

2.5.2 Computational Models and Execution

Computational models provide an abstract description of how tasks are executed and how computational resources are organized during execution. These models are essential for reasoning about the structure, scalability, and coordination of computations involving multiple activities that progress over time.

2.5.2.1 Processes

A process represents an executing instance of a program together with its associated execution context. This context typically includes a private address space, program code, data, and operating system resources required for execution [32].

Processes provide strong isolation between executing programs, as each process operates within its own protected memory space. This isolation improves robustness and fault containment, since errors in one process cannot directly corrupt the state of another. However, inter-process communication generally incurs higher overhead, as it requires explicit mechanisms to exchange data across process boundaries [31].

From an execution perspective, processes constitute units of concurrency that provide strong isolation, at the cost of increased management and communication overhead.

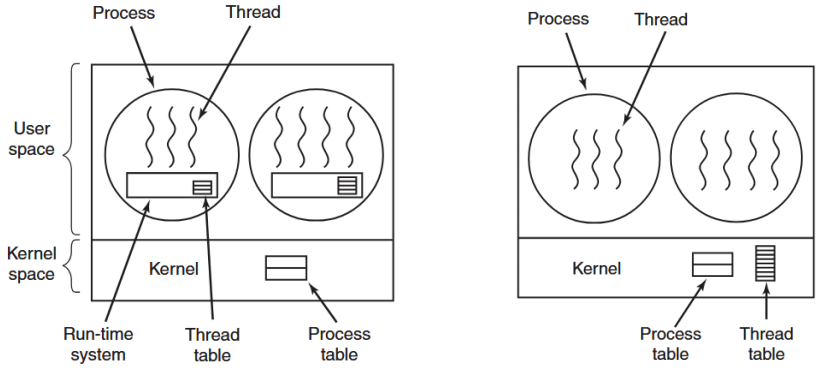
2.5.2.2 Threads

A thread represents a lower-level unit of execution within a process. Multiple threads within the same process share the process address space and resources, while maintaining independent execution contexts such as program counters and stacks [32]. This sharing enables efficient communication and cooperation between concurrent activities, as data can be accessed directly without inter-process communication mechanisms.

Threads can be classified according to how their management is implemented. In user-level threading models, thread creation, scheduling, and synchronization are handled by runtime systems

operating entirely in user space. As depicted in Fig. 2.7(a), the kernel is unaware of individual threads and schedules only processes, while thread tables and scheduling decisions are maintained by the runtime environment. This approach enables fast thread management but limits the ability to exploit multiple processors transparently [31].

In contrast, kernel-level threads, shown in Fig. 2.7(b), are managed directly by the operating system kernel. In this model, the kernel maintains thread tables and schedules threads independently, allowing threads belonging to the same process to be distributed across multiple processing units. While this enables true parallel execution on multi-core systems, it also introduces higher management overhead due to kernel involvement [30].



(a) User-level threading model (b) Kernel-level threading model

Figure 2.7: Conceptual comparison between user-level and kernel-level thread management [32].

In both models, threads within a process typically share memory and other resources, adopting a shared-memory execution model. While this model supports efficient cooperation, it also requires careful coordination to ensure correct access to shared data. The mechanisms used to address these challenges are discussed in the following subsection.

2.5.3 Synchronization and Shared Resource Access

In concurrent execution models, multiple threads or tasks may access shared resources during their execution. While shared access enables cooperation and efficient communication, it also introduces challenges related to correctness, as uncontrolled interactions may lead to inconsistent or unintended system states.

2.5.3.1 Race Conditions

In concurrent execution models, incorrect synchronization may result in race conditions, where the behavior of a computation depends on the timing of concurrent executions. When multiple threads access shared resources without proper coordination, the final system state may vary across different execution schedules, even when the same operations are performed [30].

Race conditions are particularly difficult to detect and reason about, as they often arise only under specific execution orders of concurrent tasks.

2.5.3.2 Critical Sections

To prevent race conditions, concurrent systems identify regions of execution that access shared resources as critical sections. A critical section denotes a portion of code in which a shared resource is accessed or modified and whose execution must be mutually exclusive.

Correct concurrent execution requires that critical sections operating on the same shared resource are not executed simultaneously by multiple threads, thereby ensuring consistency of the shared state [32].

2.5.3.3 Mutexes

Mutual exclusion mechanisms, commonly referred to as mutexes, provide a concrete means to enforce exclusive access to critical sections. A mutex ensures that at most one process or thread may enter a given critical section at any time.

By serializing access to shared resources, mutexes prevent concurrent modifications that may otherwise result in inconsistent system states. This behavior is illustrated in Fig. 2.8, where one process is blocked while another executes a critical section.

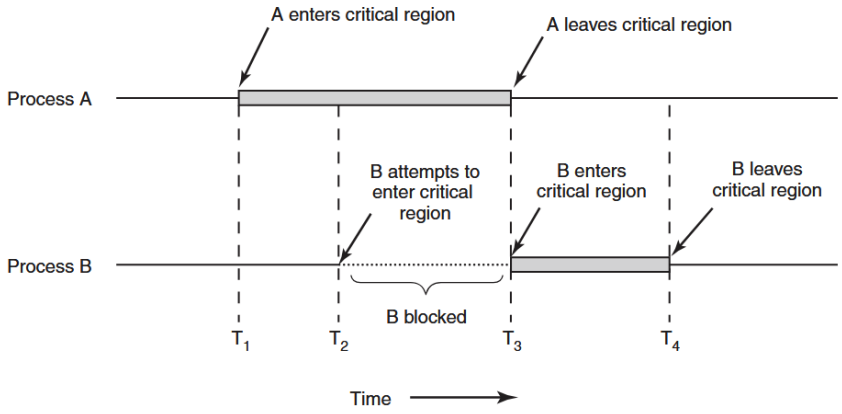


Figure 2.8: Illustration of mutual exclusion in a critical section [32].

2.5.3.4 Semaphores

Semaphores provide a general synchronization mechanism for controlling access to shared resources and enforcing ordering constraints between concurrent tasks. Unlike mutexes, which are primarily intended to enforce exclusive access, semaphores support more flexible coordination patterns [33].

According to Arpaci-Dusseau [30], semaphores are commonly classified into two types:

- **Binary semaphores**, whose value is restricted to zero or one, and which can be used to enforce mutual exclusion in a manner similar to mutexes, though without ownership constraints;
- **Counting semaphores**, which take non-negative integer values and are used to manage access to a finite number of identical resources. The semaphore value represents the number of currently available resource instances and is decremented each time a resource is allocated to a process or thread.

2.5.3.5 Deadlocks

Another fundamental challenge in concurrent systems is deadlock, a state in which a set of threads becomes permanently blocked because each thread is waiting for resources held by others. Deadlocks arise from circular waiting dependencies and result in a loss of system progress, requiring careful design and coordination to avoid [31].

2.6 Summary

This chapter introduced the main background concepts required to support the work developed in this dissertation. First, the centralized MRPP problem considered in this work was formalized through the MAPF abstraction, highlighting the role of time, robot interactions, conflict-free trajectories, and global optimization criteria. The chapter then discussed how such planning problems can be instantiated through discrete environment representations, with particular emphasis on grid and graph abstractions.

Classical graph search methods were also reviewed, since they provide the basis for individual path computation in many multi-robot planning approaches. In particular, Dijkstra's algorithm and A* were introduced as fundamental shortest path methods, together with their main properties and computational complexity.

The chapter further presented basic machine learning concepts relevant to the proposed heuristic analysis. Clustering methods, namely K-means and DBSCAN, were introduced as tools for characterizing the spatial distribution of robots, while linear regression was presented as an exploratory model for runtime analysis. Finally, concepts from parallel and concurrent computing were discussed, including parallelism, concurrency, processes, threads, synchronization mechanisms, race conditions, critical sections, mutexes, semaphores, and deadlocks.

Overall, these topics establish the theoretical and technical foundation for the MRPP methods, prioritization heuristics, runtime analysis, and parallel execution strategies discussed in the following chapters.

Chapter 3

State-of-the-Art

This chapter reviews the State-of-the-Art relevant to centralized MRPP, including MAPF formulations and coordination approaches that provide useful foundations for the methods developed in this work. Section 3.1 surveys the main coordination paradigms and algorithmic approaches, including decentralized, centralized, hybrid, and learning strategies, highlighting their respective strengths and limitations in multi-robot systems.

Section 3.2 examines prioritization heuristics for sequential planning methods, with particular attention to how robot ordering affects solution feasibility, solution quality, and computational performance.

3.1 Algorithmic Approaches to MRPP

The MRPP problem has been addressed through different coordination paradigms, reflecting distinct assumptions about system architecture and the distribution of coordination responsibilities. In multi-robot systems and fleet coordination, these paradigms exhibit characteristic trade-offs in terms of coordination quality, computational complexity, and robustness [7].

This chapter considers the following coordination paradigms:

- **Decentralized coordination**;
- **Centralized coordination**;
- **Hybrid coordination**;
- **Learning-based coordination**.

The following subsections discuss each of these paradigms in turn, introducing their main coordination principles and presenting representative algorithmic approaches to illustrate how these paradigms are realized in practice, with particular emphasis on centralized approaches, which constitute the algorithmic focus of this dissertation.

3.1.1 Decentralized Coordination

Decentralized approaches distribute planning and decision-making among the robots themselves, without relying on a global coordination authority. In this paradigm, robots typically operate based on local information and, when available, limited communication with neighboring robots. Coordination emerges through local interactions, reactive behaviors, or distributed decision mechanisms, as illustrated in Figure 3.1.

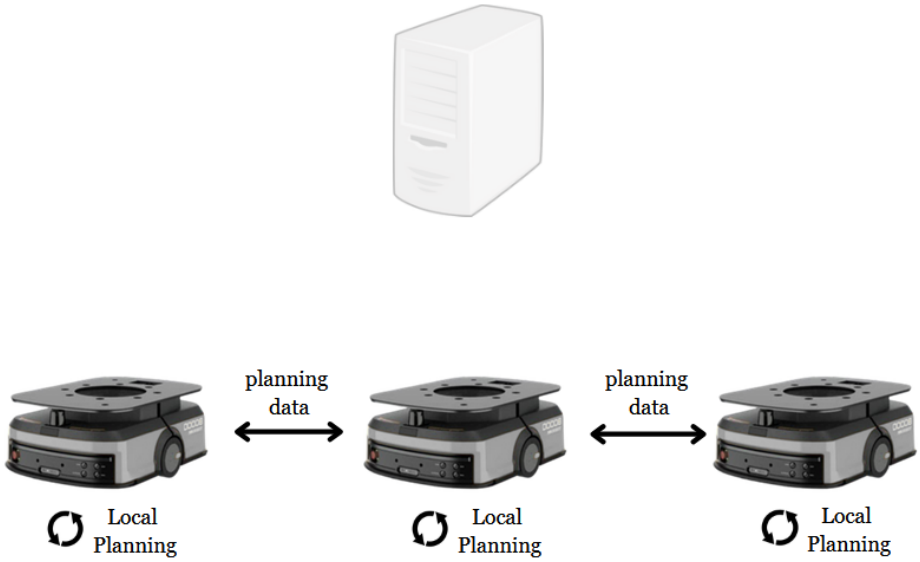


Figure 3.1: Illustration of a decentralized coordination architecture, where robots communicate directly with each other and no central planning entity is responsible for computing globally consistent paths.

Such approaches are particularly attractive in large-scale systems or in scenarios where communication is unreliable, as they avoid centralized computational bottlenecks and single points of failure [34]. Decentralized coordination has been extensively studied in the context of AMRs, where each robot must operate under partial observability and limited global knowledge [35].

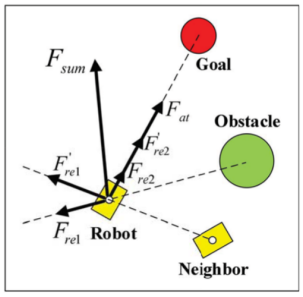
However, the absence of a global planning entity makes it difficult to ensure globally consistent behavior. Decentralized MRPP approaches generally provide limited guarantees with respect to completeness, deadlock avoidance, or solution quality, especially in environments with narrow passages or strong interactions between robots [36]. As a result, while decentralized paradigms offer scalability, they often struggle to meet the predictability and safety requirements of tightly coordinated robotic fleets.

Several algorithmic approaches exemplify decentralized coordination in multi-robot systems. Reactive collision avoidance methods rely on local sensing and interactions between robots to prevent collisions, enabling scalable coordination without global planning, although with limited guarantees [37]. Distributed planning and negotiation-related approaches extend this paradigm through peer-to-peer information exchange and local conflict resolution, improving robustness while avoiding centralized control [36].

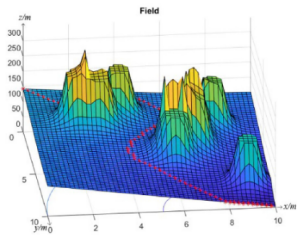
A representative example of decentralized coordination is provided by Artificial Potential Field (APF) methods. Originally proposed for single-robot navigation, APF approaches were designed to guide an agent toward a goal while avoiding obstacles by defining attractive and repulsive potential functions over the workspace [38]. More recent extensions have adapted APF methods to multi-robot settings by incorporating interaction forces between robots, which are treated as dynamic obstacles or sources of repulsive potentials. In such approaches, coordination emerges from local interactions, as each robot computes its motion based on locally available information. An example of this paradigm is the APF-CPP approach proposed by Wang et al. [39], where artificial potential fields are used for online multi-robot coverage path planning in a fully decentralized manner.

The resulting motion emerges from the combination of attractive forces toward goals and repulsive forces arising from obstacles and neighboring robots, allowing each robot to react continuously to its local environment without requiring global coordination, as illustrated in Figure 3.2a.

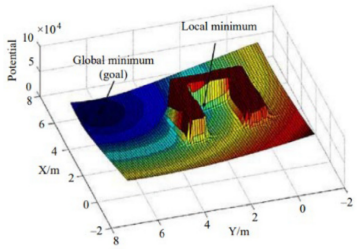
Although Figures 3.2b and 3.2c do not explicitly represent a multi-robot setting, they are useful for illustrating the fundamental behavior of potential fields and the emergence of local minima. Figure 3.2b depicts an idealized potential field, while Figure 3.2c shows how the interaction between attractive and repulsive potentials can give rise to undesired local minima. These local minima constitute a well-known limitation of APF methods, as robots may become trapped despite the existence of a feasible path to the goal [40].



(a) Attractive and repulsive forces acting on a robot in a multi-robot setting [39].



(b) Potential field [40].



(c) Potential field exhibiting local minima induced by obstacles [40].

Figure 3.2: Illustration of APF concepts.

The APF approach discussed above illustrates several characteristic limitations of decentralized coordination methods. In particular, the reliance on local and reactive decision-making can lead to undesired behaviors such as local minima and deadlocks, reflecting the difficulty of achieving globally consistent solutions. As a result, decentralized approaches typically provide limited guarantees regarding global completeness or solution quality.

3.1.2 Centralized Coordination

Centralized coordination approaches rely on a single planning entity with access to the global state of the system, including the environment representation and the start and goal locations of all robots. This entity is responsible for computing coordinated, conflict-free trajectories that ensure globally consistent behavior across the fleet. A typical centralized coordination architecture is depicted in Figure 3.3.

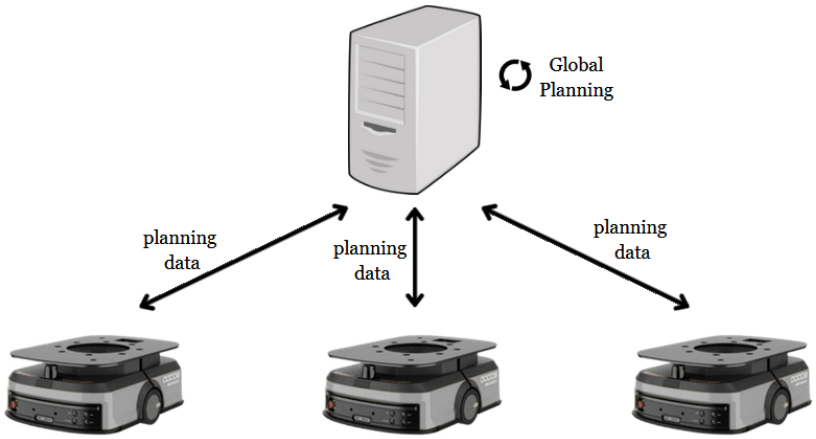


Figure 3.3: Illustration of a centralized coordination architecture, in which a central planning entity computes coordinated, conflict-free trajectories for all robots based on global information.

Centralized coordination has long been adopted in industrial and structured environments, where reliable communication and centralized supervision are available [41]. This paradigm enables strong guarantees regarding safety, predictability, and coordination quality, and aligns well with fleet management systems that incorporate execution monitoring and replanning capabilities.

From a computational perspective, centralized multi-robot planning faces inherent scalability challenges as the number of robots increases. As discussed in the background, MAPF formulations are NP-hard in the general case, and practical MRPP systems inherit this combinatorial difficulty when multiple robots must be coordinated in shared spaces. Planning therefore becomes increasingly expensive in densely populated or highly constrained environments. Consequently, practical centralized planners must carefully balance solution quality with computational efficiency. Nevertheless, access to global information allows centralized methods to reason explicitly about robot interactions and to resolve conflicts in a systematic and predictable manner, which remains a key advantage in scenarios requiring strong coordination guarantees.

Several algorithmic approaches have been proposed for centralized coordination, differing in how conflicts are represented, detected, and resolved. This subsection focuses on two representative methods, TEA* [8] and Conflict-Based Search (CBS) [4]. TEA* is directly relevant to the robotic fleet coordination framework considered in this dissertation, while CBS is introduced as a reference MAPF algorithm for conflict-driven centralized planning.

3.1.2.1 TEA*

TEA* is a centralized multi-robot coordination approach that builds directly upon the classical A* search algorithm described in the background by extending the search space with an explicit temporal dimension. Instead of reasoning solely over spatial states, TEA* models robot motion in a space-time representation, where each state is augmented with a discrete time index. This enables the planner to reason explicitly about future robot positions and to avoid conflicts during execution [8].

In this formulation, time progresses in discrete steps that correspond to the traversal of graph edges. Each feasible motion between two adjacent spatial nodes induces a transition to the next temporal layer, and the search is propagated forward up to a predefined temporal horizon $k_{\max}$. This layered space-time graph, illustrated in Figure 3.4, allows TEA* to encode both spatial feasibility and temporal ordering directly within the search process. By reserving nodes and edges across successive time layers during planning, the algorithm ensures that robots do not occupy the same resource at the same time, preventing future conflicts before execution rather than detecting and resolving them after paths have been computed.

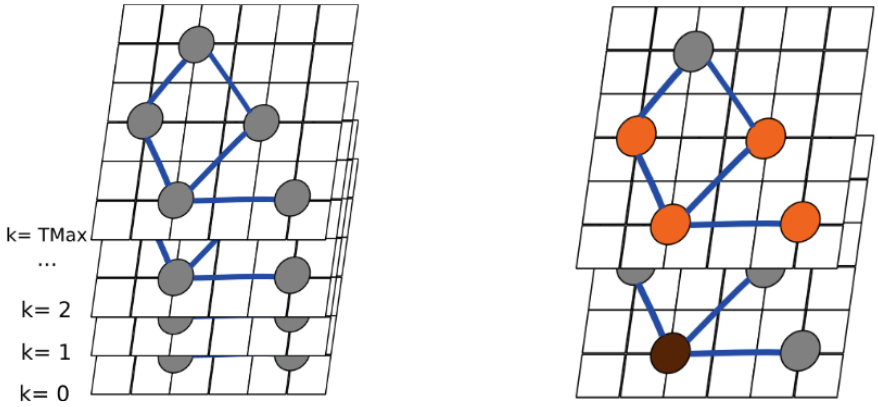


Figure 3.4: Illustration of the TEA* space-time representation [42].

In TEA*, paths are planned sequentially for each robot over a shared environment representation, which is commonly modeled as a graph. Once a path is computed for a given robot, the corresponding space-time trajectory is reserved and effectively treated as a dynamic obstacle for subsequently planned robots. During planning, nodes and edges occupied at specific time steps are marked as unavailable, preventing later robots from generating paths that would conflict with already planned motions. This temporal reservation mechanism allows TEA* to generate collision-free solutions without requiring a separate conflict detection and resolution stage [42].

A key characteristic of TEA* is its suitability for online execution and replanning. By maintaining a time-indexed representation of resource usage, the planner can update or recompute paths in response to execution delays, dynamic disturbances, or unexpected robot behavior. This capability makes TEA* particularly attractive for industrial and warehouse environments, where strict coordination requirements coexist with execution uncertainty and frequent replanning needs [8].

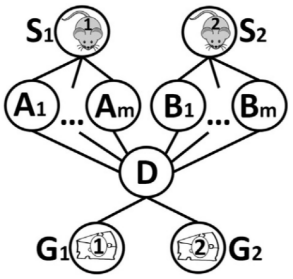
Several extensions of the original TEA* formulation have been proposed to address limitations arising from simplifying assumptions in the base model. In particular, Moura et al. [43] introduce a temporal optimization that accounts for robot kinematics, such as orientation changes and rotation costs, reducing the mismatch between the discrete planning model and real-world execution for differential-drive robots.

In summary, the sequential nature of TEA* implies that the resulting solution may depend on the order in which robots are planned, potentially affecting solution quality or feasibility in densely constrained environments. This limitation can be mitigated through planning heuristics and robot ordering strategies. While this design choice sacrifices optimality guarantees under standard MAPF cost formulations, it offers a favorable trade-off in terms of computational efficiency and practical deployability.

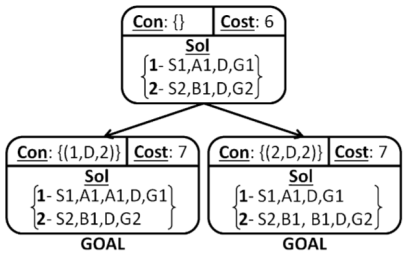
3.1.2.2 CBS

CBS is a centralized algorithm originally proposed for MAPF. It resolves coordination through a hierarchical planning framework composed of a high-level conflict resolution process and a low-level path planner [4]. The key idea underlying CBS is to decouple individual path planning from conflict management, allowing agents to plan largely independently while explicitly resolving interactions when conflicts arise.

Figure 3.5a illustrates a simple MAPF instance in which two agents navigate a shared environment with alternative routes between start and goal locations. Although individual shortest paths may exist for each agent, conflicts can arise when agents attempt to occupy the same location or traverse the same edge at the same time, motivating the need for explicit coordination.



(a) Example MAPF instance illustrating potential conflicts between agents navigating a shared environment.



(b) Conflict Tree expansion in CBS, where conflicts are resolved by branching over alternative constraints.

Figure 3.5: Overview of Conflict-Based Search. Adapted from [4].

The high-level search maintains a Conflict Tree (CT), illustrated in Figure 3.5b, where each node corresponds to a set of constraints and associated individual paths. When a conflict is detected, the corresponding CT node is expanded by generating alternative constraint sets, each prohibiting one of the conflicting agents from performing the conflicting action. While this mechanism guarantees completeness and optimality under standard MAPF cost formulations, the number of conflicts and

CT nodes may grow exponentially in the worst case, motivating several extensions to improve scalability.

At the low level, each agent computes a shortest path subject to a set of constraints. Recent extensions of CBS adopt Safe Interval Path Planning (SIPP) as the low-level planner in order to handle continuous time. This combination, commonly referred to as Continuous-Time CBS (CCBS), integrates SIPP into the CBS framework to enable optimal planning without discretizing time [44], [45]. SIPP operates by representing, for each spatial location, intervals of time during which the location is safe to occupy, allowing waiting actions and motion durations to be handled efficiently under temporal constraints. This integration makes CBS particularly well suited for domains where temporal feasibility plays a central role [45].

Several variants of CBS have been proposed to address scalability limitations:

- **ECBS:** Bounded-suboptimal CBS introduces focal search to trade optimality for efficiency, allowing solutions within a predefined suboptimality bound [46].
- **CBSH:** Heuristic-guided CBS augments the high-level search with admissible heuristics that estimate the cost of unresolved conflicts, reducing the size of the Conflict Tree while preserving optimality [47].
- **CBSH2-RTC:** CBS with symmetry reasoning mitigates the combinatorial explosion of collision resolutions by identifying and pruning pairwise symmetric path combinations, improving scalability and search efficiency [48].

Overall, CBS and its variants exemplify a conflict-driven centralized MAPF paradigm, in which interactions between agents are explicitly detected and resolved through constraint generation at the high level. Although CBS is not the algorithm used in the CRIIS fleet coordination system, it is a useful reference for contrasting conflict resolution approaches with the conflict avoidance strategy adopted by TEA*.

3.1.2.3 Contrasting Centralized Coordination Strategies

Although TEA* and CBS are both centralized planning approaches, they originate from different planning contexts and follow distinct coordination principles. CBS is a reference MAPF algorithm based on conflict resolution, while TEA* is a robotic planning approach that relies on conflict avoidance through space-time reservations.

This distinction leads to different trade-offs. CBS provides strong guarantees under standard MAPF formulations, but may become computationally expensive as the number of conflicts increases. TEA*, in contrast, sacrifices optimality guarantees but offers a practical and computationally efficient structure for online fleet coordination. In this dissertation, CBS is considered as a reference conflict resolution approach, while the subsequent work focuses on TEA* and on strategies for improving its performance within the CRIIS fleet coordination framework.

3.1.3 Hybrid Coordination

Hybrid coordination approaches seek to combine elements of decentralized and centralized coordination in order to leverage the advantages of both paradigms. Typically, these approaches adopt a hierarchical structure, in which higher level coordination decisions are made centrally, while lower level planning or execution decisions are delegated to individual robots, as illustrated in Figure 3.6.

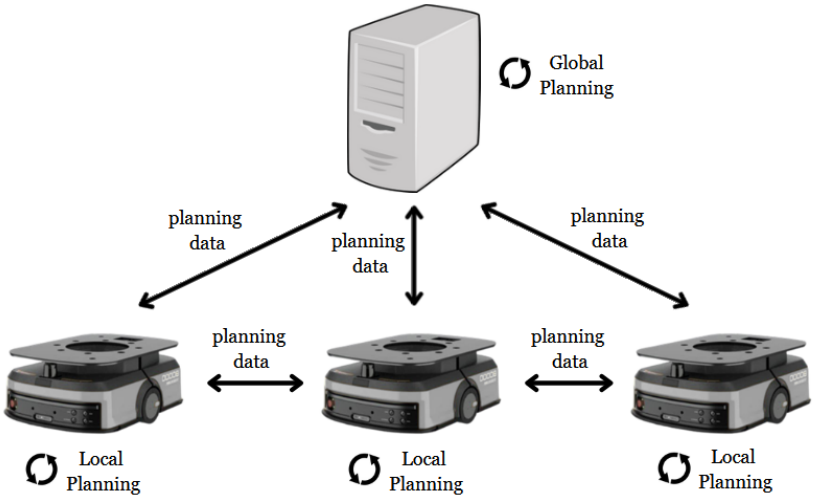


Figure 3.6: Illustration of a hybrid coordination architecture combining centralized decision making at higher levels with decentralized planning or execution at the robot level.

The motivation for hybrid coordination lies primarily in scalability. By abstracting parts of the problem or limiting centralized planning to higher level decisions, hybrid approaches aim to reduce the computational burden associated with fully centralized coordination, while still preserving some degree of global consistency [49].

Despite their potential benefits, hybrid and hierarchical approaches introduce additional complexity and may compromise theoretical guarantees. In particular, the separation between planning layers can lead to suboptimal solutions or coordination failures in complex or highly constrained environments. As such, hybrid paradigms are often tailored to specific application domains and system architectures.

Several representative approaches illustrate this paradigm. Hierarchical MAPF methods employ centralized planning at a coarse level, such as region or waypoint assignment, while delegating fine-grained path planning to individual agents [50]. Other hybrid strategies combine centralized global guidance with decentralized local execution, allowing robots to react autonomously to local disturbances while respecting global coordination constraints [49]. Region coordination approaches further reduce complexity by enforcing centralized constraints at the level of areas or traffic corridors, while allowing robots to plan locally within these regions [50], [51].

3.1.4 Learning-Based Coordination

More recently, learning-based approaches have been explored as an alternative paradigm for multi-agent coordination and path planning. These methods typically rely on machine learning techniques, such as reinforcement learning or imitation learning, to learn coordination policies from data rather than explicitly computing plans through search or optimization.

Learning-based coordination is motivated by the potential to generalize across environments and to adapt to complex, dynamic scenarios. In multi-agent settings, learning-based methods have been applied to collision avoidance, cooperative navigation, and decentralized decision-making [52], [53].

Representative examples include learned decentralized collision-avoidance and navigation policies, where each agent reacts to nearby agents through interaction with the environment rather than relying on explicit global planning [54]. Such methods emphasize scalability and adaptability, but typically lack formal guarantees.

Another prominent line of work combines classical MAPF planners with learning-based components. In these hybrid approaches, learning is used to guide heuristic selection, conflict resolution, or priority assignment within search-based planners, improving efficiency while preserving the underlying planning structure [52].

Despite promising results, learning-based approaches face significant challenges in the context of MAPF. These include the need for large amounts of training data and the difficulty of providing strong guarantees regarding safety, completeness, or optimality. As a result, learning-based coordination is often considered complementary to classical MAPF methods rather than a direct replacement, particularly in safety-critical fleet management applications.

3.2 Prioritization Heuristics

Prioritized planning constitutes an important class of MRPP approaches, particularly in centralized settings where robots are planned sequentially according to a predefined priority ordering. In this paradigm, higher-priority robots reserve their paths first, while lower-priority robots must plan around the space-time constraints introduced by previously planned robots. Although this decomposition makes the planning problem more tractable, it also makes the resulting solution highly dependent on the selected ordering. Different priority permutations can lead to different levels of feasibility, solution quality and computational effort [55], [56].

The importance of priority assignment has therefore motivated several heuristic, randomized, adaptive, and learning-based strategies for selecting or modifying robot orderings. As highlighted by recent surveys on prioritized multi-robot path planning, no single priority rule is consistently superior across all environments and task configurations [57]. Instead, performance depends strongly on factors such as robot distribution, path length, local density, environmental bottlenecks, and the degree of interaction between robots. This motivates the use of diverse prioritization criteria rather than relying on a single fixed ordering strategy.

3.2.1 Randomized Strategies

A simple way to reduce the sensitivity of prioritized planning to robot ordering is to evaluate multiple priority permutations. Bennewitz, Burgard and Thrun [58] explore this idea by modifying priority schemes through a hill-climbing process, allowing the planner to search for better orderings without enumerating all possible permutations.

Randomized orderings are also commonly used in MAPF as baselines or restart mechanisms [55], [56]. Although they can help escape poor deterministic orderings, they do not provide an explicit criterion for why a given ordering should be effective.

3.2.2 Distance and Path Length Heuristics

Distance and path length criteria are among the first prioritization rules introduced in the literature on prioritized planning. This family of heuristics assigns priority according to geometric distance or estimated path length. These measures are attractive because they are computationally lightweight and can be computed before the space-time planning stage.

Van den Berg and Overmars [59] use a roadmap-distance priority criterion in which robots farther from their goals are given higher priority. The intuition is that robots requiring longer movements are more likely to traverse a larger portion of the shared environment and may therefore benefit from being planned before the roadmap becomes constrained by reservations from other robots.

However, distance and path length rules only approximate coordination difficulty, since they do not explicitly capture the interaction structure of the problem, making them less reliable in bottlenecked or highly coupled environments.

3.2.3 Environment Heuristics

Several prioritization strategies exploit structural properties of the environment, since paths through constrained or highly shared regions may become harder to accommodate after other reservations have been introduced.

A representative example is the *Path Prospects* heuristic proposed by Wu, Bhattacharya and Prorok [60], which prioritizes robots according to the number of meaningful route alternatives available between their current position and goal. In their formulation, the number of path prospects of robot r_n at time t is estimated from the number of effective obstacles in the relevant forward region of the robot:

$$P_n^{(t)} = 2^{\kappa_n^{(t)}}, \tag{3.1}$$

where $\kappa_n^{(t)}$ denotes the number of effective obstacles considered relevant for robot r_n at time t. The base two reflects the intuition that each effective obstacle can introduce a binary route choice, causing the number of possible alternatives to grow exponentially.

The relevant obstacles are identified after first determining the part of the graph that may still belong to a reasonable route to the goal. This is done by expanding from the current robot position and retaining forward vertices v satisfying $g(v) + d_{\text{true}}(v, g_n) \leq T$, where $g(v)$ is the cost from the current position to v, $d_{\text{true}}(v, g_n)$ is the shortest graph distance from v to the goal g_n, and T defines the maximum cost of paths considered relevant. Effective obstacles that fall within this forward region are then counted to estimate $\kappa_n^{(t)}$. Robots with fewer path prospects are considered more constrained and are therefore planned earlier, since delaying them may make their paths harder to accommodate after reservations from other robots have been introduced.

Although this heuristic provides an informative estimate of route availability, it requires expanding the graph from the current robot position, identifying the forward region of vertices that might still belong to a reasonable route, and checking which effective obstacles fall within that region.

A lighter strategy that evaluates the role of shared infrastructure is considered by ter Mors [61]. In this approach, robots that depend on the busiest parts of the infrastructure are prioritized earlier, under the assumption that highly used resources are more likely to become difficult to access after other plans have been introduced. This criterion is simpler to apply once information about infrastructure demand is available, but it depends on prior estimates or statistics about which parts of the infrastructure are expected to be heavily used.

3.2.4 Local Density Heuristics

Another class of prioritization methods uses the local surroundings of each robot as an estimate of planning difficulty. The underlying assumption is that robots located in more cluttered or locally constrained regions can have fewer immediate movement options and might therefore benefit from being planned earlier.

An early example of this idea is the dynamic priority system proposed by Clark, Bretl and Rock [62], where priority is given to the robot whose local workspace is more crowded. This corresponds to a simple surroundings heuristic, since priority is influenced by the degree of local crowding around the robot.

Wu, Bhattacharya and Prorok [60] later evaluate this idea as *Naive Surroundings* and introduce a coupled variant that accounts for the relationship between robot mobility and the environment. While *Naive Surroundings* counts original obstacles within a fixed range, *Coupled Surroundings* counts effective obstacles. In this case, obstacles that are geometrically distinct may be treated as a single obstacle for a robot if its dimensions or motion constraints prevent it from moving between them.

These heuristics show that local spatial context can provide a lightweight priority signal, but their effectiveness depends on whether the chosen neighborhood reflects actual motion constraints.

3.2.5 Conflict and Interaction Prioritization

Methods based on conflict information use interactions between agents competing for the same vertices, edges, or temporal resources as a source of priority information. Although CBS is not

itself a prioritization heuristic, it demonstrates the importance of conflict information in MAPF by detecting conflicts between paths and resolving them through alternative constraint sets [4]. This makes conflicts a useful signal for identifying coupled agents and difficult coordination decisions.

An indirect form of interaction awareness is considered by ter Mors [61], where one heuristic evaluates the quality of a context-aware route using the ratio

$$q_i = \frac{c_i}{\ell_i}, \tag{3.2}$$

where c_i is the cost of the context-aware plan of agent i, and ℓ_i is the length of its shortest path. Higher values of q_i indicate that the route is more strongly affected by existing plans or shared-resource usage, making this criterion an indirect measure of interaction pressure [61].

Overall, while conflicts can provide useful indicators of agent interaction, the literature does not appear to provide many explicit prioritization heuristics that use predicted space-time conflicts as a lightweight preplanning score for constructing an initial priority ordering.

3.2.6 Rescheduling and Adaptive Priority Adjustment

Priority orderings do not necessarily remain fixed throughout the planning process. In rescheduling methods, feedback from previous planning attempts, such as failures, is used to adjust the ordering during replanning.

Regele and Levi [63] propose a heuristic priority-adjustment mechanism in which robots initially have low priority values, which are increased when planning becomes difficult or unsuccessful. Andreychuk and Yakovlev [64] later simplify this idea through deterministic rescheduling, where a robot that fails to find a route is assigned maximum priority in the next planning attempt. This follows the assumption that failure indicates strong dependence on the current ordering.

Other adaptive strategies modify only part of the priority order. Local priority adjustment focuses on robots involved around a deadlock region, while priority tuning increases the priority of robots whose routes are least optimal and repeats this process until no significant improvement is obtained [57]. Overall, these methods show that replanning feedback can be used to refine priority decisions instead of treating each ordering as independent.

3.2.7 Learning Priority Orderings

Learning priority approaches provide an alternative to manually defined prioritization rules. Instead of selecting a fixed heuristic a priori, Zhang et al. [56] propose learning priority orderings for prioritized MAPF from training data. Their method first generates candidate solutions by running prioritized planning with different orderings, including heuristic and random priority assignments, and then uses the best-performing orderings as supervision for learning.

The authors consider two learning formulations. The first learns a total priority ordering, where all agents are ranked before planning. The second learns partial priority relations between agents, allowing the method to focus on the ordering decisions that are more relevant to solution

quality. The learned models use features related to start-goal distances, path alternatives, and potential interactions between agents, combining several signals that are often treated separately in handcrafted heuristics.

Although this allows priority rules to be inferred from data, learning-based methods require training instances, feature design, and generalization across maps and problem distributions. Therefore, handcrafted heuristics remain relevant when interpretability, low computational cost, and direct control over the prioritization criterion are important.

3.3 Summary

This chapter reviewed the main coordination paradigms and algorithmic approaches relevant to MRPP. The discussion covered decentralized, centralized, hybrid, and learning approaches, with particular emphasis on centralized planning methods. While decentralized, hybrid, and learning approaches offer advantages in terms of scalability, adaptability, or architectural flexibility, their weaker guarantees regarding safety, predictability, and solution quality can limit their applicability in tightly coordinated fleet management scenarios. Centralized approaches, in contrast, provide a more structured way to reason about robot interactions, although they must balance coordination quality with computational efficiency.

The chapter also reviewed prioritization strategies for sequential planning methods, where the order in which robots are planned can strongly affect feasibility, solution quality, and computational effort. The surveyed approaches include randomized strategies, distance and path length criteria, environment and local density heuristics, conflict and interaction measures, adaptive rescheduling mechanisms, and learning methods for priority assignment.

The literature shows that priority ordering is an important factor in sequential multi-robot planning, but also that no single ordering criterion is consistently effective across all scenarios. This motivates the design of diverse prioritization heuristics and multiple ordering strategies, which are developed and evaluated in this work.

Chapter 4

CRIIS Fleet Coordination System

This chapter describes the existing CRIIS fleet coordination system, which serves as the reference architecture for the work developed in this dissertation. The system is a centralized coordination framework built around TEA*, supported by supervision and execution monitoring mechanisms for online operation in real-world scenarios [7].

Section 4.1 presents the graph and trajectory representation used by the system. Section 4.2 describes the main components of the Central Coordination Module.

4.1 Trajectory Representation

The CRIIS fleet coordination system represents the operating environment as a graph, as illustrated in Figure 4.1. Vertices correspond to decision states, while edges represent executable actions, including robot motion and operations such as opening a door, calling an elevator, or docking with an object [7]. This representation allows planning to consider both robot displacement and actions that influence the temporal coordination of the fleet.

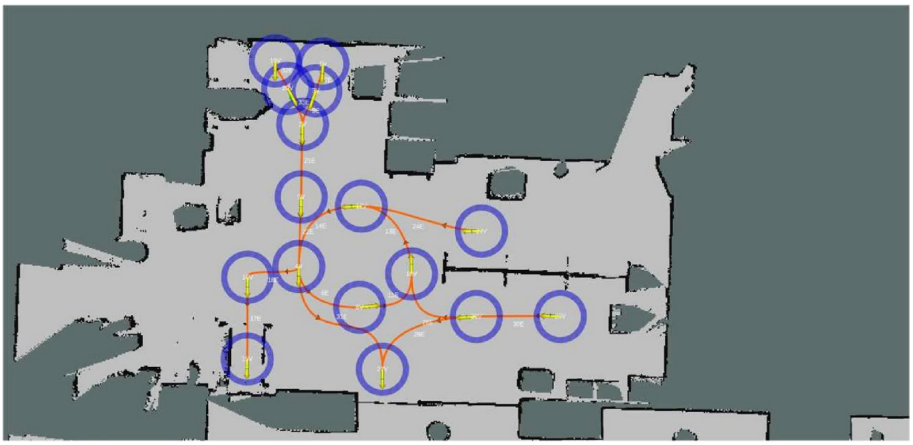


Figure 4.1: Graph representation of the operating environment [7].

The graph abstracts the physical workspace into a set of vertices and directed edges, allowing the planner to compute coordinated routes over a compact symbolic representation while the robot controllers execute the corresponding physical trajectories.

Each vertex is associated with a unique identifier, a reference frame, a planar position, and an orientation. The orientation value is used to define the direction of outgoing and incoming travel edges, supporting the construction of smooth trajectories. Each edge is associated with an origin vertex, a destination vertex, a unique identifier, a curve type, and parameters describing the corresponding motion or action. For travel edges, the physical trajectory is encoded as a parametric Bézier curve. The system also stores forward and backward execution velocities for these edges, allowing the same geometric connection to have different temporal costs depending on the direction of traversal.

Figure 4.2 illustrates how the curve parameters and vertex orientations define the shape of a travel edge between two graph vertices. This parametric representation allows the system to generate intermediate points along each edge, producing an executable trajectory for the robot controller.

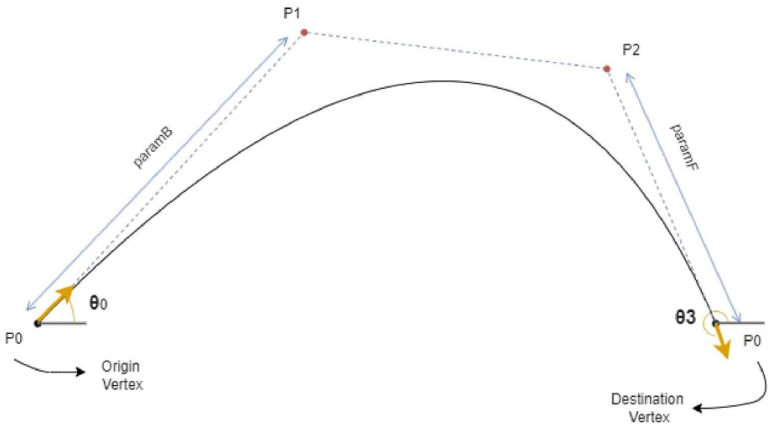


Figure 4.2: Parametric representation of a travel edge, defined from the origin and destination vertices, their orientations, and the curve parameters used to shape the trajectory [7].

The cost of each edge is expressed in temporal terms. For travel edges, it is computed from the length of the corresponding parametric curve and the nominal velocity in the direction of traversal. For actions such as docking or elevator interaction, the cost is estimated from real execution data. This allows TEA* to reason over execution time rather than only geometric distance.

A robot plan is therefore represented as an ordered sequence of graph edges. During execution, each robot follows this sequence locally and reports its symbolic state to the central coordination module, typically through the current edge and the progress along it. This allows the central system to coordinate the fleet at graph level while continuous trajectory tracking remains handled by the robot-side control layer.

4.2 System Architecture

The CRIIS fleet coordination system follows a centralized and modular architecture implemented on top of the Robot Operating System (ROS), which provides the communication middleware between the central coordination components and the robot-side control modules [7]. As illustrated in Figure 4.3, the system is organized around two main parts: the Central Coordination Module and the Agent Control Module.

The Central Coordination Module is responsible for global planning, supervision, and coordination, maintaining a global view of the environment and the fleet state. In contrast, the Agent Control Module, instantiated on each robot, executes the received edge sequences, interfaces with the local navigation stack, and provides execution feedback to the central coordination layer. This separation allows each robot to retain autonomy in localization and low-level control, while the fleet remains coordinated through centralized planning.

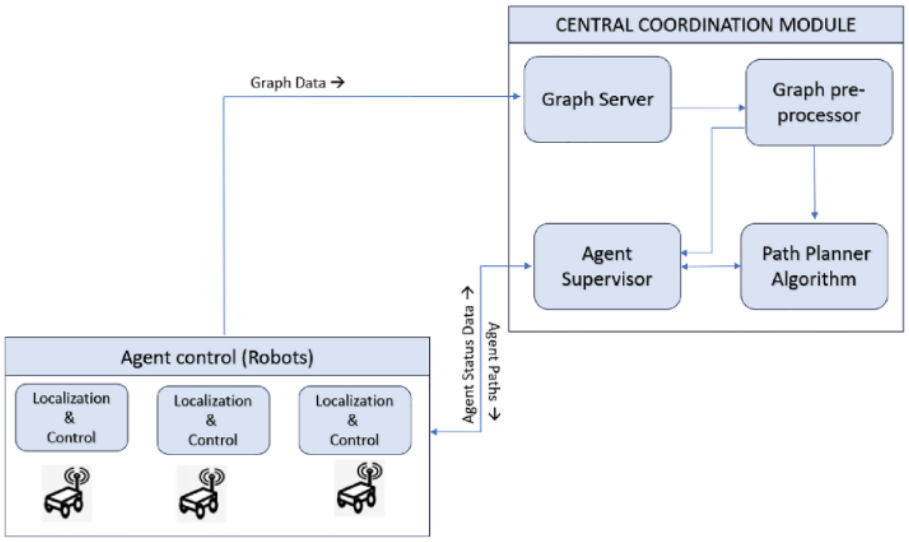


Figure 4.3: High-level architecture of the CRIIS fleet coordination system [7].

The following subsections describe the main components involved in this coordination workflow. The Graph Server, Graph Pre-Processor, Path Planner, and Supervisor form the central coordination side of the system. The Navigation Handler, although not explicitly represented as an independent block in Figure 4.3, is also described because it provides the interface between the centralized coordination logic and the robot-side navigation stack.

Figure 4.4 summarizes the interaction between these components during the coordination workflow, including the exchange of planning requests, graph information, computed paths, and execution feedback.

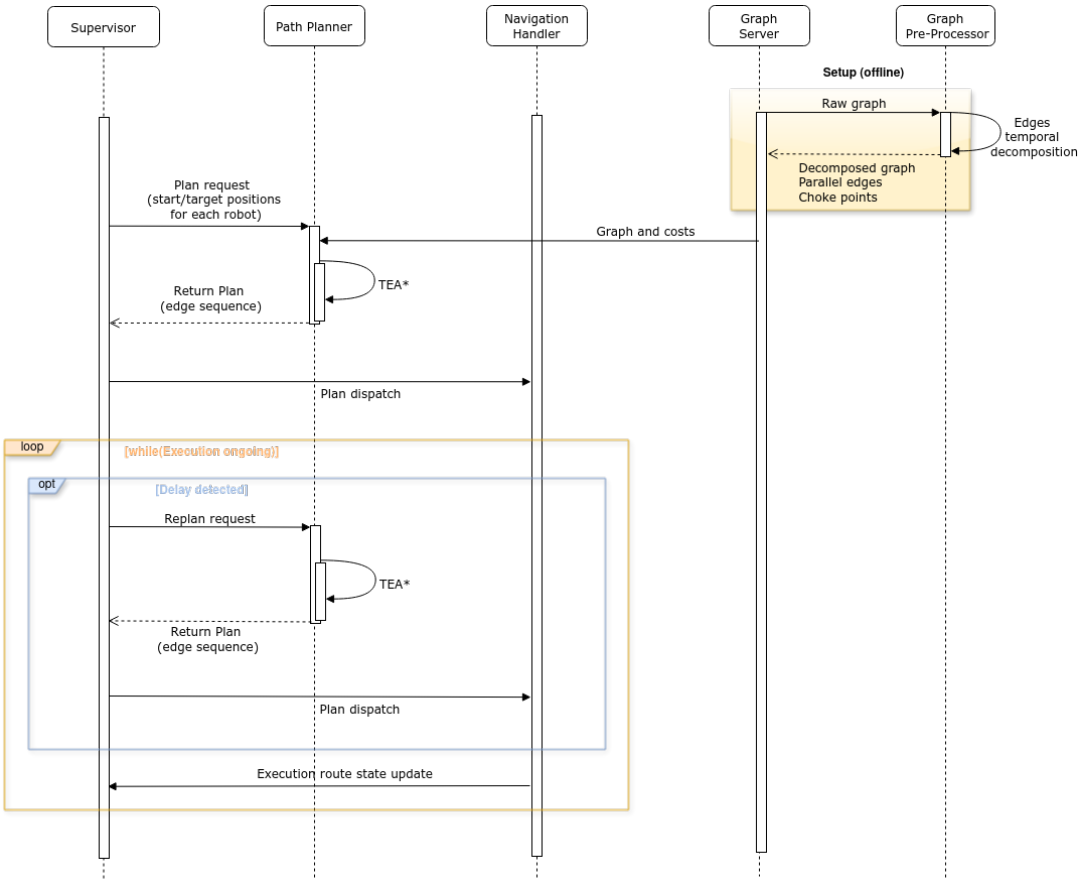


Figure 4.4: Sequence diagram of the CRIIS fleet coordination workflow.

4.2.1 Graph Server

The Graph Server is responsible for storing and providing access to the global navigation graph shared by all agents in the fleet. This graph encodes the topological structure of the environment, including vertices, directed edges, and their associated traversal costs. In the CRIIS fleet coordination system, these costs are expressed in temporal terms, enabling time-aware planning and synchronization among agents [7].

Additionally, the Graph Server exposes a query interface that allows other components, namely the Path Planner, to retrieve graph topology, edge attributes, and metadata required for coordination. By decoupling graph management from planning logic, the system promotes modularity and facilitates future extensions, such as dynamic cost updates or graph augmentation strategies.

4.2.2 Graph Pre-Processor

The Graph Pre-Processor operates during an offline configuration phase and augments the raw navigation graph with information required for multi-robot coordination, transforming a purely geometric representation into a coordination-aware graph [7]. Its responsibilities include identifying footprint-related constraints, such as parallel edges that cannot be safely occupied simultaneously,

and detecting critical shared resources, such as narrow corridors or doorways that are prone to congestion and deadlocks.

Additionally, since TEA* advances time in discrete steps associated with edge traversals, the pre-processor performs edge discretization and normalization according to a global temporal resolution. This ensures consistent traversal times across the graph, improving temporal alignment between agents and enabling more accurate space-time reasoning during planning, thereby reducing synchronization errors and unnecessary replanning [65].

4.2.3 Path Planner

The Path Planner is the core decision-making component responsible for computing coordinated, collision-free trajectories for the entire fleet. In the CRIIS system, this functionality is implemented using the TEA* algorithm [7], which follows a centralized and sequential prioritized planning approach.

Planning requests include the current symbolic positions and goals of all agents. Upon receiving a request, the Path Planner retrieves the navigation graph and associated traversal costs from the Graph Server and computes time-consistent sequences of edges for each agent. By reasoning over a shared space-time representation, the planner can explicitly account for interactions between agents, reserving graph resources over time and preventing conflicts through temporal exclusion mechanisms.

The centralized nature of the Path Planner can produce globally coherent solutions, particularly in environments with shared resources or high robot density. This approach allows conflicts to be resolved at planning time rather than during execution, which can reduce the likelihood of deadlocks, emergency stops, or excessive local replanning [7].

A previous extension of this planner had already explored a multi-threaded TEA* implementation to evaluate multiple robot priority orderings in parallel [66]. This dissertation builds on that direction in Chapters 7 and 8, where the parallel execution logic is reorganized, combined with the proposed prioritization heuristics, and evaluated on large-scale planning scenarios.

4.2.4 Navigation Handler

The Navigation Handler serves as the integration layer between the centralized coordination logic and the robots' low-level navigation systems. Although it is not represented as an independent component in the high-level architecture shown in Figure 4.3, it plays an important role in the execution workflow by mediating navigation requests, plan dispatch, and execution feedback.

In particular, the Navigation Handler exposes an API through which a robot can be requested to navigate to a given goal vertex. When this service is called, the Navigation Handler forwards the request to the Supervisor, which in turn invokes the Path Planner to compute a feasible path within the shared graph representation. Once the resulting plan is received, the Navigation Handler translates the symbolic sequence of graph edges into navigation commands compatible with the robot navigation stack.

During execution, the Navigation Handler collects feedback such as the current symbolic state, edge completion status, and execution delays, and reports this information to the Supervisor. By isolating execution-specific concerns from planning logic, it enhances system modularity and allows the coordination framework to remain agnostic to the underlying robot platforms and navigation implementations [7].

4.2.5 Supervisor

The Supervisor is responsible for orchestrating the overall coordination loop by closing the feedback cycle between planning and execution. It aggregates execution feedback provided by the Navigation Handler, monitors the progress of each agent, and maintains a global view of the system state [7].

Based on this information, the Supervisor detects abnormal or unexpected conditions, such as execution delays, deviations between planned and executed trajectories, or emerging conflicts caused by dynamic disturbances. When such situations are identified, the Supervisor triggers replanning requests to the Path Planner.

This supervision mechanism is a key contributor to system robustness under real-world uncertainty, including variable execution times, communication latency, and dynamic obstacles. By enabling adaptive replanning while preserving centralized coordination, the Supervisor helps the fleet maintain safe and efficient operation even when assumptions made at planning time are violated [7].

4.3 Summary

This chapter described the existing CRIIS fleet coordination system used as the reference architecture for this dissertation. First, the graph and trajectory representation adopted by the system was presented, highlighting how vertices define decision states and edges encode executable actions, including both travel motions and operations not directly related to navigation. The use of temporal edge costs allows TEA* to reason over execution time, while robot plans are represented as ordered sequences of graph edges.

The chapter then introduced the modular architecture of the system, implemented in ROS, and distinguished between the Central Coordination Module and the Agent Control Module. The main components involved in the coordination workflow were then described, including the Graph Server, Graph Pre-Processor, Path Planner, Navigation Handler, and Supervisor. Although the Navigation Handler is not represented as an independent block in the high-level architecture, it was included because it provides the interface between centralized coordination and robot-side navigation. This architecture provides the foundation upon which the prioritization heuristics, parallel planning framework, and batching mechanisms developed in the following chapters are introduced.

Chapter 5

Prioritization Heuristics Design and Development

Due to the sequential nature of TEA*, each planned path constrains the following ones, making the selected robot ordering directly affect solution cost and feasibility. This chapter presents the prioritization heuristics developed to generate alternative robot orderings for TEA*.

Section 5.1 introduces the motivation and design objectives, while Section 5.2 presents the common score-based formulation. The remaining sections describe the proposed heuristic families, namely roadmap distance, topology, surroundings, conflict-based, and replanning heuristics, including their scoring criteria, ordering variants, algorithms, and computational costs.

5.1 Motivation and Design Objectives

The motivation for developing prioritization heuristics arises from the fact that the same planning instance can produce substantially different results depending only on the order in which robots are planned. In constrained environments, assigning priority to one robot may reduce its own planning difficulty while increasing the constraints imposed on the remaining robots.

Since the best ordering depends on robot distribution, goals, roadmap structure, and path interactions, no single prioritization rule is expected to perform well in all scenarios. For this reason, the objective of this work is not to identify one dominant heuristic, but to design a diverse set of complementary ordering strategies. These heuristics should produce priority orderings that are informative enough to improve planning feasibility or solution quality, while remaining computationally lightweight when compared with the execution time of TEA*. Given that prioritization is performed before planner execution, excessive heuristic computation time may reduce or even eliminate the practical gains obtained from improved orderings.

5.2 Heuristic Design Methodology

The heuristics developed in this work follow a common score-based formulation. Given a set of robots

$$R = \{r_1, r_2, \ldots, r_n\}, \tag{5.1}$$

each heuristic assigns a priority score, $score_i$, to every robot, r_i, according to the information available in the current planning instance. For a given heuristic, the priority score assigned to robot r_i is denoted as:

$$score_i = h(d_i), \tag{5.2}$$

where d_i denotes the input data used by the heuristic to evaluate robot r_i.

Depending on the heuristic being applied, d_i may include information such as the robot's current position, goal vertex, estimated path, predicted conflicts, roadmap-related properties, or data from previous coordination cycles.

After computing the scores, the priority ordering is obtained using a common sorting procedure. The sorting direction is defined by the parameter *mode*. In the *First* mode, robots with higher scores are planned earlier, while in the *Last* mode, robots with lower scores are planned earlier. This operation is denoted as

$$\pi = \textsc{SortByScore}(R, score, mode), \tag{5.3}$$

where π represents the resulting priority permutation of the robots.

The developed heuristics are organized into five main groups according to the type of information they exploit:

- **Roadmap distance heuristics**, based on distances or estimated path lengths in the roadmap graph;
- **Topological heuristics**, based on structural properties of the graph and its constrained regions;
- **Surroundings-based heuristics**, based on the spatial distribution of robots around their current positions or goals;
- **Conflict-based heuristics**, based on estimated interactions between tentative robot paths;
- **Replanning heuristics**, based on information from previous plans or coordination cycles.

Together, these groups cover complementary dimensions of the coordination problem, allowing the proposed heuristics to address robot prioritization from different perspectives. Each group is examined in the following sections, where the corresponding heuristics are described in terms of their motivation, scoring procedure, and computational implications.

5.3 Roadmap Distance Heuristics

Roadmap distance heuristics define robot priority orderings using distance measures between each robot and its goal. Following the Roadmap Distance nomenclature reported by Heselden and Das [57], the two heuristics developed in this group correspond to *Roadmap Distance 1*, where the farthest robot is given priority, and *Roadmap Distance 2*, where the robot with the longest path is planned first.

These are the simplest heuristics considered in this work, since they rely only on travel distance as an approximate indicator of planning difficulty. Although computationally lightweight, they do not explicitly account for path conflicts, local robot density, or topological constraints. The two implemented criteria are the direct distance between each robot and its goal and the length of an estimated shortest path in the roadmap graph, giving rise to the *Farthest Robot* and *Longest Path* heuristics, respectively.

5.3.1 Farthest Robot

The *Farthest Robot* heuristic assigns priorities according to the direct distance between each robot and its goal.

For each robot r_i, let $s_i = (x_i, y_i)$ denote its current position and let $g_i = (x_{g_i}, y_{g_i})$ denote its goal position. The score assigned to robot r_i is computed as the Euclidean distance between these two points:

$$score_i = dist(s_i, g_i) = \|g_i - s_i\|_2. \tag{5.4}$$

Figure 5.1 illustrates this computation for a single robot and goal pair. In the example, the score assigned to robot r_1 corresponds to the direct distance between its current position, s_1, and its goal position, g_1.

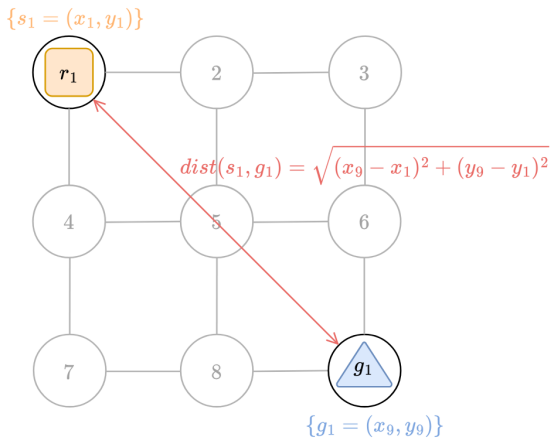


Figure 5.1: Score computation for the Farthest Robot heuristic.

5.3.1.1 Ordering variants

Two ordering variants were considered. In the *Farthest Robot First* variant, robots with higher distance scores are planned earlier, following the intuition that longer movements should be planned before the search space becomes more constrained.

In the *Farthest Robot Last* variant, the ordering is reversed, and robots with lower distance scores are planned earlier. This strategy may be useful in scenarios where the objective is to maximize the number of robots that successfully reach their goals, especially when a complete solution for all robots is difficult or infeasible. By prioritizing robots closer to their goals, the planner may allow a larger subset of robots to complete their paths before the remaining planning problem becomes too constrained.

5.3.1.2 Core Algorithm

The procedure is summarized in Algorithm 1. For each robot, the heuristic computes the Euclidean distance between its current roadmap position and its goal vertex, and then applies the sorting procedure to obtain the final priority ordering.

Algorithm 1 Farthest Robot Heuristic

Require: Set of robots R, current closest-vertex positions s_i on the roadmap, current goal vertices g_i on the roadmap, ordering mode *mode*

Ensure: Priority ordering π

1: **for** each robot $r_i \in R$ **do**
2: $\quad score_i \leftarrow dist(s_i, g_i)$
3: **end for**
4: $\pi \leftarrow \text{SORTBYSCORE}(R, score, mode)$
5: **return** π

In the following complexity analyses, $n = |R|$ denotes the number of robots. Each score is computed in constant time, resulting in $O(n)$ time for all robots. The final score-based sorting step requires $O(n \log n)$ time. Thus, the total computational cost is

$$O(n) + O(n \log n) \tag{5.5}$$

which is dominated by the sorting step.

5.3.2 Longest Path

The *Longest Path* heuristic assigns priorities according to the cost of an estimated shortest path on the roadmap graph. Unlike the *Farthest Robot* heuristic, which uses only the Euclidean distance between the current and goal vertices, this heuristic accounts for the actual connectivity of the roadmap. This is particularly relevant in scenarios where the Euclidean distance does not reflect

the real distance that the robot must travel, for example when obstacles force the robot to follow a longer route through the graph.

For each robot r_i, a single-robot shortest path is computed between its current vertex s_i and its goal vertex g_i using A^*. The resulting path is represented as

$$P_i^* = (v_1, v_2, \ldots, v_k), \tag{5.6}$$

with $v_1 = s_i$ and $v_k = g_i$. The priority score of robot r_i is then defined as the total cost of the edges along this path:

$$score_i = \sum_{\ell=1}^{k-1} w_{v_\ell, v_{\ell+1}}, \tag{5.7}$$

where $w_{v_\ell, v_{\ell+1}}$ denotes the weight of the edge connecting consecutive vertices v_ℓ and $v_{\ell+1}$.

In the example shown in Figure 5.2, the path computed for robot r_1 traverses the sequence of vertices from its initial position to its goal. The corresponding score is obtained by summing the costs of the selected roadmap edges.

$$score_1 = w_{1,2} + w_{2,5} + w_{5,6} + w_{6,9}. \tag{5.8}$$

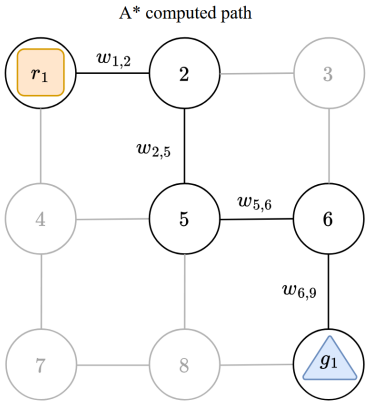


Figure 5.2: Illustration of the Longest Path heuristic.

5.3.2.1 Ordering variants

Following the same logic used for the *Farthest Robot* heuristic, two ordering variants were considered. In the *Longest Path First* variant, robots with higher path-cost scores are planned earlier, while in the *Longest Path Last* variant, robots with lower path-cost scores are planned earlier. Both variants use the same priority score definition and differ only in the sorting direction.

5.3.2.2 Core Algorithm

Algorithm 2 summarizes the *Longest Path* heuristic. For each robot, the algorithm computes a single-agent shortest path on the roadmap, uses the corresponding path cost as the priority score,

and then applies the generic score-based sorting procedure to obtain the final ordering.

Algorithm 2 Longest Path Heuristic

Require: Set of robots R, current positions s_i, goal positions g_i, ordering mode *mode*
Ensure: Priority ordering π
1: **for** each robot $r_i \in R$ **do**
2: $p_i \leftarrow A^*(s_i, g_i)$ ▷ single-robot shortest path on the roadmap
3: $score_i \leftarrow cost(p_i)$ ▷ sum of edge costs along p_i
4: **end for**
5: $\pi \leftarrow \text{SORTBYSCORE}(R, score, mode)$
6: **return** π

The dominant operation in this heuristic is the computation of one A^* path per robot. As discussed in Section 2.3.1.2, in the worst case, A^* may explore the same search space as Dijkstra's algorithm. Therefore, using the complexity given in Equation 2.3, computing the priority scores for all robots requires $O\big(n \cdot (|V| + |E|) \log |V|\big)$ time. Since the final score-based sorting step requires $O(n \log n)$ time, the total computational cost is

$$O\big(n \cdot (|V| + |E|) \log |V|\big) + O(n \log n), \tag{5.9}$$

which is typically dominated by the n shortest-path computations.

5.4 Topological Heuristics

The topological heuristics developed in this work are motivated by the environment heuristics discussed in Section 3.2, which assign priority according to structural properties of the environment. Previous approaches capture this idea either by reasoning about meaningful route alternatives, as in *Path Prospects*, or by considering the expected demand on highly shared infrastructure [60], [61]. However, the former requires additional graph expansion and effective obstacle identification, while the latter depends on prior estimates of infrastructure demand.

In this work, a lighter alternative is adopted by estimating movement flexibility from the roadmap path computed for each robot. This preserves the intuition that robots crossing structurally constrained regions may be more affected by previous reservations, while avoiding the need for prior statistics about infrastructure demand.

To quantify this notion of movement flexibility, the Degree of Freedom (DoF) of roadmap vertices is used. Each roadmap vertex v is associated with a set of directly connected neighboring vertices, denoted by $N(v)$. The DoF of v is defined as

$$\text{DoF}(v) = |N(v)|, \tag{5.10}$$

where $|N(v)|$ represents the number of neighbors of vertex v.

For each robot r_i, the same single-robot shortest path notation introduced in Section 5.3.2 is used. A path P_i^* is computed between the current vertex s_i and the goal vertex g_i using A^*, and the vertices belonging to this path are then evaluated according to their DoF values.

Vertices with lower DoF values typically correspond to more constrained regions of the environment, such as corridors or narrow passages, where robots have fewer movement options. Conversely, vertices with higher DoF values tend to represent more open regions of the roadmap, where robots have greater flexibility to avoid conflicts. Figure 5.3 illustrates this concept on a roadmap path computed with A^*.

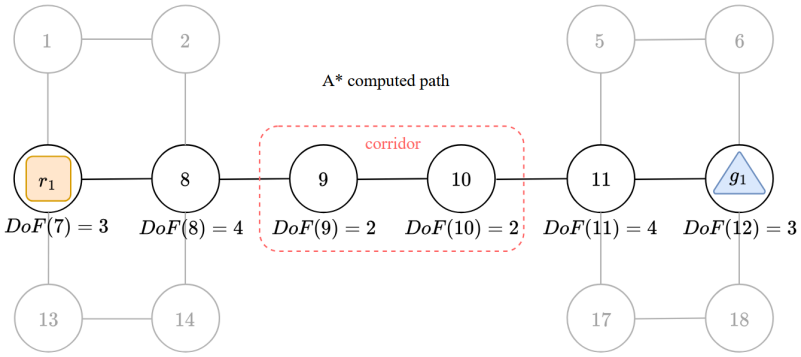


Figure 5.3: Illustration of DoF values along a roadmap path.

Based on this information, two topological criteria are considered. The first summarizes the overall movement flexibility of a path through its average DoF, while the second explicitly counts the number of low-DoF vertices traversed by the path. These criteria give rise to the *Average DoF Path* and *Low DoF Sum* heuristics, respectively.

5.4.1 Average DoF Path

The *Average DoF Path* heuristic assigns priorities according to the average topological flexibility of the roadmap path estimated for each robot. For each robot r_i, a single-robot shortest path P_i^* is first computed between its current vertex s_i and its goal vertex g_i. The DoF values of the vertices traversed by this path are then averaged to obtain the priority score.

Let $V(P_i^*)$ denote the set of distinct roadmap vertices traversed by path P_i^*. The score of robot r_i is defined as

$$score_i = \frac{1}{|V(P_i^*)|} \sum_{v \in V(P_i^*)} \mathrm{DoF}(v). \tag{5.11}$$

Therefore, the score corresponds to the mean roadmap connectivity along the estimated path of the robot. Higher scores indicate paths through more connected roadmap regions, while lower scores indicate more constrained paths with fewer movement alternatives.

This heuristic provides a compact measure of the overall flexibility of a path. However, because it is based on an average value, short but highly constrained portions of the path may be smoothed by more open regions.

Considering the path illustrated in Figure 5.3, the Average DoF Path score for robot r_1 is computed as

$$\text{score}(r_1) = \frac{\text{DoF}(7) + \text{DoF}(8) + \text{DoF}(9) + \text{DoF}(10) + \text{DoF}(11) + \text{DoF}(12)}{6} = 3,$$

where each $\text{DoF}(v)$ denotes the connectivity of vertex v. Therefore, the score corresponds to the mean connectivity of the vertices traversed by the path.

5.4.1.1 Ordering variants

Two ordering variants were considered. In the *High Average DoF Path First* variant, robots with higher average DoF scores are planned earlier, prioritizing paths that traverse more open regions of the roadmap. In the *Low Average DoF Path First* variant, the ordering is reversed, and robots with lower average DoF scores are planned earlier, prioritizing paths that traverse more constrained regions. Both variants use the same score and differ only in the sorting direction.

5.4.1.2 Core Algorithm

Algorithm 3 summarizes the *Average DoF Path* heuristic. For each robot, the algorithm first computes an A^* path between the current and goal vertices. It then identifies the distinct roadmap vertices traversed by that path and evaluates the DoF of each one by counting its distinct neighboring vertices. These values are accumulated and averaged, producing a priority score that represents the mean level of movement flexibility available along the estimated path. The final ordering is obtained by applying the selected sorting mode to these scores.

Algorithm 3 Average DoF Path Heuristic

Require: Set of robots R, roadmap graph G, current positions s_i, goal positions g_i, ordering mode *mode*
Ensure: Priority ordering π

for each robot $r_i \in R$ **do**
  $p_i \leftarrow A^*(s_i, g_i)$ ▷ single-robot shortest path on the roadmap
  $N_i \leftarrow$ distinct vertices along p_i
  $DoFsum_i \leftarrow 0$
  for each vertex $v \in N_i$ **do**
    $N_v \leftarrow \emptyset$ ▷ set of distinct neighbors of v
    for each edge e in AdjacencyList(v) **do**
      $u \leftarrow$ OppositeEndpoint(e, v) ▷ neighbor reached through edge e
      insert u into N_v
    end for
    $\text{DoF}(v) \leftarrow |N_v|$ ▷ number of distinct neighbors of v
    $DoFsum_i \leftarrow DoFsum_i + \text{DoF}(v)$
  end for
  $score_i \leftarrow DoFsum_i / |N_i|$
end for
$\pi \leftarrow$ SortByScore$(R, score, mode)$
return π

The dominant operations in this heuristic are the single-robot path searches and the evaluation of the DoF values along the resulting paths. Using the A^* worst-case complexity given in Equation 2.3, the path computation step is applied once per robot.

After each path is obtained, the heuristic visits its vertices and computes the DoF of each one by checking its adjacency list. If L_i is the number of edges in the path of robot r_i, and $\Delta = \max_{v \in V} \mathrm{DoF}(v)$ is the maximum degree of any roadmap vertex, the DoF aggregation step costs at most $O(L_i \cdot \Delta)$ for that robot. Considering all robots, this cost is upper-bounded by

$$O(n \cdot L_{\max} \cdot \Delta), \tag{5.12}$$

where $L_{\max} = \max_i L_i$. The final score-based sorting step requires $O(n \log n)$ time. Thus, the total computational cost is given by

$$O\big(n \cdot (|V| + |E|) \log |V|\big) + O\big(n \cdot L_{\max} \cdot \Delta\big) + O(n \log n). \tag{5.13}$$

5.4.2 Low DoF Sum

The *Low DoF Sum* heuristic assigns priorities according to the number of constrained vertices traversed by the roadmap path of each robot. Instead of averaging the DoF values along the path, as in the *Average DoF Path* heuristic, this approach explicitly counts how many vertices are classified as topologically restricted.

A vertex is considered constrained when its DoF is lower than or equal to a predefined threshold τ. The priority score of robot r_i is defined as

$$score_i = \sum_{v \in P_i^*} \delta(v), \tag{5.14}$$

with

$$\delta(v) = \begin{cases} 1, & \mathrm{DoF}(v) \leq \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{5.15}$$

Therefore, the score corresponds to the number of low-DoF vertices traversed by the path of the robot.

Compared with the *Average DoF Path* heuristic, the *Low DoF Sum* heuristic provides a more explicit measure of path restriction. While *Average DoF Path* summarizes the overall topological flexibility of a path through a mean value, *Low DoF Sum* directly counts the number of vertices that satisfy a predefined restriction criterion. As a result, it can better emphasize paths that cross several clearly constrained regions, even when the average DoF of the complete path is not particularly low.

This distinction can lead to different priority scores in scenarios where the distribution of constrained vertices along the path is relevant. For example, two paths may have similar average DoF values, but one may traverse several very restricted vertices while the other only passes through

moderately constrained areas. In such cases, the *Low DoF Sum* heuristic tends to penalize the first path more strongly, whereas the average-based score may smooth out this difference.

However, this explicitness also introduces a limitation. The heuristic depends directly on the threshold τ, which defines what is considered a low-DoF vertex. Therefore, its behavior is more sensitive to parameter selection and to the specific connectivity characteristics of the roadmap. In contrast, the *Average DoF Path* heuristic is more general, since it does not require a hard classification of vertices as constrained or unconstrained.

Considering the path illustrated in Figure 5.3, the threshold is set to $\tau = 2$. Therefore, only vertices with $\mathrm{DoF}(v) \leq 2$ are classified as constrained. In this case, vertices 9 and 10 satisfy the restriction condition, while vertices 7, 8, 11, and 12 do not. The Low DoF Sum score for robot r_1 is therefore:

$$\mathrm{score}(r_1) = \delta(7) + \delta(8) + \delta(9) + \delta(10) + \delta(11) + \delta(12) = 0+0+1+1+0+0 = 2.$$

5.4.2.1 Ordering variants

Two ordering variants were considered. In the *Low DoF Sum First* variant, robots whose paths traverse more constrained vertices (a higher score) are planned earlier, prioritizing the most restricted paths. In the *Low DoF Sum Last* variant, the ordering is reversed, and robots with fewer constrained vertices are planned earlier. Both variants use the same score and differ only in the sorting direction.

5.4.2.2 Core Algorithm

Algorithm 4 follows the same structure as Algorithm 3, differing only in the score computation.

Algorithm 4 Low DoF Sum Heuristic

Require: robots R, roadmap graph G, current positions s_i, goal positions g_i, ordering mode *mode*, DoF threshold τ
Ensure: Priority ordering π

for each robot $r_i \in R$ **do**
  $p_i \leftarrow A^*(s_i, g_i)$ ▷ single-agent shortest path on the roadmap
  $N_i \leftarrow$ distinct vertices along p_i
  $score_i \leftarrow 0$
  for each vertex $v \in N_i$ **do**
    $N_v \leftarrow \emptyset$ ▷ set of distinct neighbors of v
    for each edge e in AdjacencyList(v) **do**
      $u \leftarrow$ OppositeEndpoint(e, v) ▷ neighbor reached through edge e
      insert u into N_v
    end for
    $\mathrm{DoF}(v) \leftarrow |N_v|$ ▷ number of distinct neighbors of v
    if $\mathrm{DoF}(v) \leq \tau$ **then**
      $score_i \leftarrow score_i + 1$ ▷ count constrained vertex
    end if
  end for
end for
$\pi \leftarrow$ SortByScore$(R, score, mode)$
return π

This heuristic has the same computational complexity as the *Average DoF Path* heuristic. The only difference lies in the aggregation step, since *Low DoF Sum* counts the vertices that satisfy $\mathrm{DoF}(v) \leq \tau$ instead of averaging the DoF values. Therefore, the total computational cost remains essentially the same as in Equation 5.13.

5.5 Surroundings Heuristics

The surroundings heuristics developed in this work are inspired by local-surroundings priority criteria discussed in Section 3.2. In particular, they follow the same general intuition as *Naive Surroundings*, where priority is influenced by how crowded the region around a robot is. However, instead of counting nearby obstacles, the heuristics proposed here evaluate the spatial distribution of other robots around a selected reference point.

The selected reference point can correspond either to the current position of the robot or to its goal position. When the current position is used, the heuristic estimates local congestion at the beginning of the planning process. When the goal position is used, it estimates goal concentration, capturing cases where several robots may compete for access to the same final area or to nearby roadmap resources.

Thus, these heuristics adapt the surroundings principle to the spatial distribution of robots, measuring local crowding through radius, roadmap, or cluster neighborhoods.

5.5.1 Radius Neighbor Density

The Radius Neighbor Density heuristic assigns priorities according to the local density of robots around a selected reference point. For each robot, the heuristic counts how many other robots are located within a predefined distance threshold. This provides a simple proximity-based estimate of how crowded the surrounding region is.

Let n_i denote the reference point associated with robot r_i, which can correspond either to its current position or to its goal position, depending on the variant being evaluated. Given a distance threshold R_{th}, the score of robot r_i is defined as:

$$score_i = \sum_{\substack{r_j \in R \\ j \neq i}} \rho(i,j), \tag{5.16}$$

where

$$\rho(i,j) = \begin{cases} 1, & d(n_i, n_j) \leq R_{\text{th}}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.17}$$

Here, $d(n_i, n_j)$ represents the Euclidean distance between the reference points associated with robots r_i and r_j. Therefore, the score corresponds to the number of neighboring robots located within radius R_{th} of robot r_i.

Robots with higher scores are located in denser local regions, since more robots are found within their neighborhood. When the current position is used as the reference point, this may

indicate robots that may have more difficulty initiating their movements due to the presence of nearby agents. When the goal position is used instead, higher scores indicate goals located in more concentrated areas, where several robots may compete for access to nearby roadmap resources.

This heuristic is strongly affected by the choice of the radius threshold. If R_{th} is too small, most robots may have few or no neighbors, making the heuristic unable to distinguish between different local configurations. Conversely, if R_{th} is too large, many robots may be counted as neighbors, reducing the locality of the measure. Therefore, the value of R_{th} directly controls the scale at which local density is evaluated. For the configuration illustrated in Figure 5.4, the score of robot r_1 varies with the selected radius, as shown in Table 5.1.

Table 5.1: Effect of the radius threshold on the score of robot r_1.

Radius threshold	$score_1$
R_1	4
R_2	4
R_3	5

This illustrates the direct dependence of the heuristic on the chosen radius threshold.

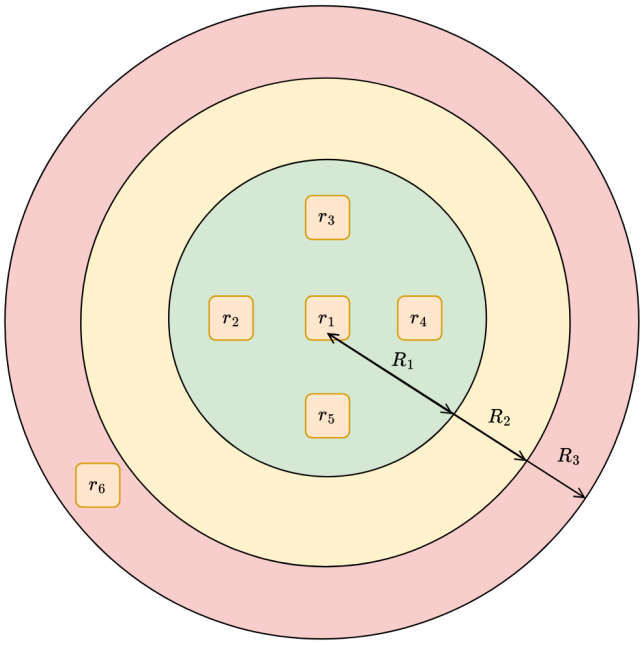


Figure 5.4: Illustration of the Radius Neighbor Density heuristic.

5.5.1.1 Ordering variants

Two ordering variants were considered for the *Radius Neighbor Density* heuristic. The first one, *High Radius Density Goals First*, computes the density score using the goal positions of the robots and prioritizes robots whose goals are located in denser regions. This strategy is motivated by the behavior of TEA* in scenarios where several robots have nearby goals. In such cases, planning first

the robots that must reach the most crowded goal regions may help avoid situations where robots assigned to peripheral goals introduce reservations that later block access to the denser central area.

The second variant, *Low Radius Density Positions First*, computes the density score using the initial positions of the robots and prioritizes robots that start in less crowded regions. This follows the intuition that, when robots are initially clustered, it may be beneficial to first move the robots located closer to the periphery of the cluster. By freeing these outer positions earlier, the planner may create more space for the robots located in more central and constrained starting regions to find feasible paths afterwards.

Therefore, these two variants use the same radius-density score but apply it to different reference points and with opposite sorting directions.

5.5.1.2 Core Algorithm

Algorithm 5 summarizes the *Radius Neighbor Density* heuristic.

Algorithm 5 Radius Neighbor Density Heuristic

Require: Set of robots R, reference function $\mathrm{ref}(\cdot)$, radius threshold R_{th}, ordering mode *mode*
Ensure: Priority ordering π

for each robot $r_i \in R$ **do**
  $n_i \leftarrow \mathrm{ref}(r_i)$ ▷ current position or goal position
  $score_i \leftarrow 0$
  for each robot $r_j \in R$ with $j \neq i$ **do**
    $n_j \leftarrow \mathrm{ref}(r_j)$
    if $d(n_i, n_j) \leq R_{\mathrm{th}}$ **then**
      $score_i \leftarrow score_i + 1$ ▷ count nearby robot
    end if
  end for
end for
$\pi \leftarrow \textsc{SortByScore}(R, score, mode)$
return π

The heuristic computes each density score by comparing the reference point of a robot with the reference points of all remaining robots and counting how many lie within the radius threshold R_{th}. Since every robot is compared with the remaining $n-1$ robots, computing all density scores requires $O(n(n-1)) \approx O(n^2)$ pairwise checks. The final score-based sorting step requires $O(n \log n)$ time. Thus, the total computational cost is given by

$$O(n^2) + O(n \log n). \tag{5.18}$$

It is also important to note that this complexity does not scale with the value of R_{th}. The radius only affects the outcome of each pairwise comparison, not the number of comparisons performed.

5.5.2 *K*-hop Neighbor Density

The *K-hop Neighbor Density* heuristic assigns priorities according to the number of robots located within a graph-based neighborhood of each robot. Unlike the *Radius Neighbor Density* heuristic, which defines proximity using a geometric distance threshold, this heuristic defines neighborhoods directly from the roadmap connectivity.

Given a hop limit k_{hops}, let $N_{k_{\text{hops}}}(n_i)$ denote the set of roadmap vertices that can be reached from n_i in at most k_{hops} graph hops. The score of robot r_i is defined as:

$$score_i = \sum_{\substack{r_j \in R \\ j \neq i}} \kappa(i, j), \tag{5.19}$$

where

$$\kappa(i, j) = \begin{cases} 1, & n_j \in N_{k_{\text{hops}}}(n_i), \\ 0, & \text{otherwise.} \end{cases} \tag{5.20}$$

Therefore, the score corresponds to the number of robots whose reference points are reachable from n_i within at most k_{hops} roadmap edges. Robots with higher scores are located in denser graph-based neighborhoods. For the configuration illustrated in Figure 5.5, increasing the hop depth changes the number of neighboring robots counted for r_1, as shown in Table 5.2.

Table 5.2: Effect of the hop depth on the score of robot r_1.

Hop depth	$score_1$
$k_{\text{hops}} = 1$	4
$k_{\text{hops}} = 2$	4
$k_{\text{hops}} = 3$	5

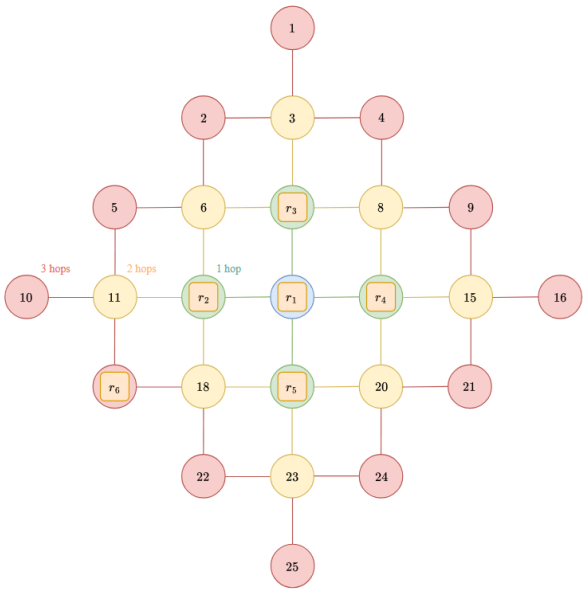


Figure 5.5: Illustration of the *K*-hop Neighbor Density heuristic.

Although conceptually similar to *Radius Neighbor Density*, this heuristic defines neighborhoods through roadmap connectivity rather than Euclidean distance. This distinction is relevant in obstacle-rich environments, where robots may be close in continuous space but far apart on the roadmap due to walls or obstacles that force longer routes.

5.5.2.1 Ordering variants

The ordering variants follow the same logic as in the *Radius Neighbor Density* heuristic. *Low K-hop Density Positions First* and *High K-hop Density Goals First* differ only in the reference point used and in the sorting direction.

5.5.2.2 Core Algorithm

Algorithm 7 summarizes the *K-hop Neighbor Density* heuristic. For each robot, a recursive DFS explores the roadmap from the reference vertex up to k_{hops}, and the score counts the robot reference vertices within that neighborhood.

Algorithm 6 Recursive DFS K-hop Expansion

Require: Roadmap graph $G = (V, E)$, current vertex v, remaining hops h, map $bestK$
Ensure: Updated map $bestK$
if $h \leq 0$ **then**
  return
end if
$h \leftarrow h - 1$
for each neighbor $u \in N(v)$ **do**
  if $u \notin bestK$ or $h > bestK[u]$ **then**
    $bestK[u] \leftarrow h$ ▷ best remaining-hop value found for u
    RECURSIVEKHOP$(G, u, h, bestK)$
  end if
end for

Algorithm 7 K-hop Neighbor Density Heuristic

Require: Set of robots R, roadmap graph $G = (V, E)$, reference function $\text{ref}(\cdot)$, hop limit k_{hops}, ordering mode *mode*
Ensure: Priority ordering π
$robotCountAtVertex \leftarrow \emptyset$
for each robot $r_j \in R$ **do**
  $n_j \leftarrow \text{ref}(r_j)$ ▷ current position or goal position
  $robotCountAtVertex[n_j] \leftarrow robotCountAtVertex[n_j] + 1$ ▷ count robots at each reference vertex
end for
for each robot $r_i \in R$ **do**
  $n_i \leftarrow \text{ref}(r_i)$ ▷ current position or goal position
  $bestK \leftarrow \emptyset$
  $bestK[n_i] \leftarrow k_{\text{hops}}$
  RECURSIVEKHOP$(G, n_i, k_{\text{hops}}, bestK)$
  $N_{k_{\text{hops}}}(n_i) \leftarrow$ keys of $bestK$ ▷ vertices reachable within k_{hops} hops
  $score_i \leftarrow 0$
  for each vertex $v \in N_{k_{\text{hops}}}(n_i)$ **do**
    $score_i \leftarrow score_i + robotCountAtVertex[v]$ ▷ count robots in the k-hop neighborhood
  end for
  $score_i \leftarrow score_i - 1$ ▷ exclude robot r_i from its own density score
end for
$\pi \leftarrow$ SORTBYSCORE$(R, score, mode)$
return π

The computational cost of the *K-hop Neighbor Density* heuristic depends on the structure of the roadmap graph, since the number of vertices reached by a k-hop expansion is determined by the graph connectivity. In this work, the analysis considers a grid-based roadmap, which is a common map representation in path-planning benchmarks and is also used in the experimental evaluation of this document. In this type of roadmap, each vertex has at most four neighbors.

Assuming an infinite 2D grid with four-neighbor connectivity, the number of vertices at exactly j hops from a reference vertex is $4j$. The number of vertices reachable within k_{hops} hops is denoted by N_k and, under this grid assumption, is given by

$$N_k = 1 + \sum_{j=1}^{k_{\text{hops}}} 4j = 1 + 4 \cdot \frac{k_{\text{hops}}(k_{\text{hops}}+1)}{2} = 1 + 2k_{\text{hops}}(k_{\text{hops}}+1). \tag{5.21}$$

However, in the recursive implementation, the number of reachable vertices does not necessarily match the number of vertex expansions performed by the algorithm. This occurs because a vertex may be expanded more than once if it is later reached with a larger remaining hop value. The number of vertex expansions performed during one recursive k-hop procedure is represented by X_k. Since each reachable vertex can be expanded at most k_{hops} times, a conservative upper bound is

$$X_k \leq k_{\text{hops}} \cdot N_k. \tag{5.22}$$

This bound is conservative, since in practice many vertices are expanded fewer times, especially in finite grids and near map boundaries. Nevertheless, it provides a safe upper bound for the recursive implementation. Substituting the expression of N_k gives

$$X_k \leq k_{\text{hops}} \left(1 + 2k_{\text{hops}}(k_{\text{hops}}+1)\right). \tag{5.23}$$

Expanding this expression results in

$$X_k \leq 2k_{\text{hops}}^3 + 2k_{\text{hops}}^2 + k_{\text{hops}}. \tag{5.24}$$

Therefore, the cost of one recursive k-hop expansion is upper-bounded by

$$O(X_k) = O\left(2k_{\text{hops}}^3 + 2k_{\text{hops}}^2 + k_{\text{hops}}\right), \tag{5.25}$$

which can be simplified to

$$O(k_{\text{hops}}^3). \tag{5.26}$$

Since one k-hop expansion is performed for each robot, the total cost of computing the density scores is

$$O(n \cdot k_{\text{hops}}^3). \tag{5.27}$$

Finally, the robots are sorted according to their scores, which requires $O(n \log n)$ time. Therefore,

the total computational cost of the heuristic is

$$O(n \cdot k_{\text{hops}}^3) + O(n \log n). \tag{5.28}$$

As shown by this expression, the computational cost of the *K-hop Neighbor Density* heuristic depends directly on the selected hop limit k_{hops}. Therefore, increasing the depth of the graph exploration increases the cost of the heuristic. This contrasts with the *Radius Neighbor Density* heuristic, whose complexity does not depend on the radius threshold R_{th}, since the radius only affects the result of each pairwise comparison and not the number of comparisons performed.

Consequently, although the *K-hop Neighbor Density* heuristic may provide a density estimate that is more consistent with the roadmap structure, especially in environments with obstacles, it is expected to become less efficient as k_{hops} increases. In such cases, the radius-based heuristic may become computationally more efficient, particularly when larger neighborhoods are considered.

5.5.3 Spatial Cluster-Based

The *Spatial Cluster-Based* heuristic groups robots according to the spatial distribution of their reference points. In the base formulation, these reference points correspond to the current positions of the robots, and the resulting heuristic corresponds to the *High Cluster Peripheral First* variant.

DBSCAN is used to obtain the spatial clusters. As discussed in Subsection 2.4.2.2, this method can identify dense groups without directly specifying the number of clusters beforehand, while also allowing isolated points to be treated as noise. These properties are useful in the present context, since the number of robot groups may vary across planning instances and isolated robots should not necessarily affect the priority score of dense groups.

The suitability of this choice is supported by the benchmark study of Murugesan et al. [67]. Their results show that K-means is more appropriate for compact and convex datasets, whereas DBSCAN is better suited to non-convex data, well-separated dense regions, and datasets where outliers should be explicitly identified. This distinction is relevant for multi-robot planning because robot formations are not always compact around a centroid. Depending on the roadmap and the task configuration, robots may form irregular spatial concentrations along corridors, near bottlenecks, around intersections, or around dense goal regions.

Therefore, DBSCAN is not adopted because it is generally superior to K-means, but because its assumptions better match the intended role of this heuristic. The objective is to detect spatial concentrations of robots that may influence the priority ordering, without requiring a predefined number of groups and without forcing isolated robots into artificial clusters.

After the clusters are computed, the heuristic assigns a priority score to each robot according to the size of its cluster and its relative distance to the cluster centroid. For a robot r_i belonging to cluster C_m, the centroid of the cluster is denoted by c_m, and the current position of the robot is denoted by s_i. The distance between the robot and the centroid is defined as

$$d_i = \|s_i - c_m\|_2. \tag{5.29}$$

The maximum distance between the centroid and any robot in the same cluster is defined as

$$d_{\max,m} = \max_{r_\ell \in C_m} \|s_\ell - c_m\|_2. \tag{5.30}$$

The priority score of robot r_i is then computed as

$$score_i = |C_m| \cdot \frac{d_i}{d_{\max,m}}, \qquad d_{\max,m} \neq 0 \tag{5.31}$$

where $|C_m|$ is the number of robots in cluster C_m.

The score contains two components. The factor $|C_m|$ increases the priority of robots belonging to larger clusters, reflecting the assumption that larger groups are typically more difficult to resolve in a prioritized planner. Within each cluster, the normalized distance term increases the score of robots farther from the centroid. As a result, the heuristic first favors robots in larger clusters and, inside each cluster, prioritizes those located closer to the boundary.

Figure 5.6 illustrates the scoring principle. Cluster C_1 contains more robots than cluster C_2, and therefore receives a higher cluster-size contribution. Within each cluster, the distance to the centroid determines the relative ordering of the robots according to the selected variant.

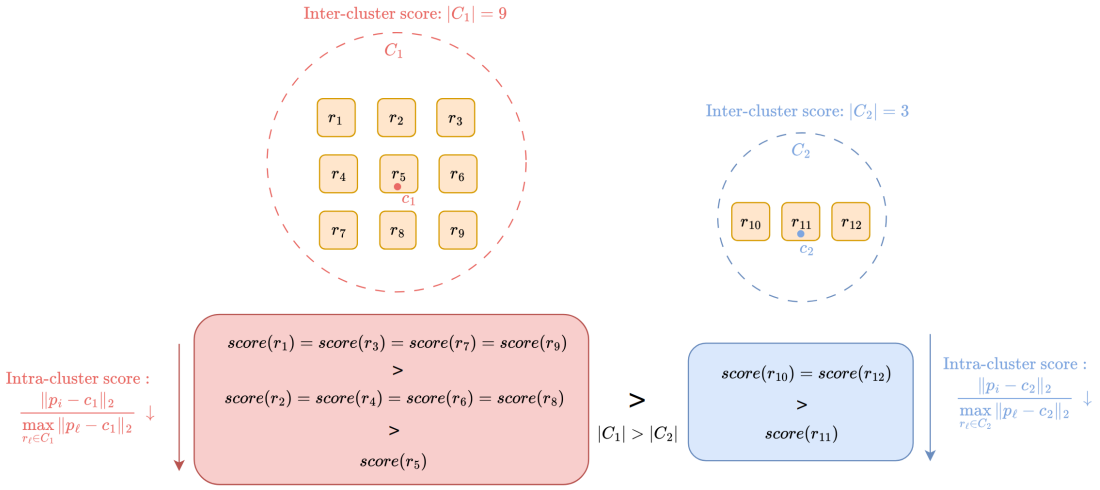


Figure 5.6: High Cluster Peripheral First scoring example.

5.5.3.1 Ordering variants

Two ordering variants were considered for the Spatial Cluster-Based heuristic. The first one is *High Cluster Peripheral First*, which corresponds to the formulation described above. This variant clusters robots according to their current positions and favors robots located farther from the centroid of larger clusters.

The second variant is *High Cluster Central Goal First*. This variant applies the same clustering principle to the goal positions, but reverses the intra-cluster preference by favoring robots whose goals are closer to the centroid of their goal cluster.

For this variant, g_i represents the goal position of robot r_i, and $d_i = \|g_i - c_m\|_2$ is its distance to the centroid of the corresponding goal cluster. The priority score is defined as

$$score_i = \frac{|C_m|}{d_i}, \qquad d_i \neq 0 \tag{5.32}$$

where $|C_m|$ is the size of the goal cluster. When $d_i = 0$, the robot is assigned the highest priority within the cluster, since its goal coincides with the cluster centroid. Unlike the peripheral variant, where the score increases with the distance to the centroid, the central-goal variant assigns higher scores to robots closer to the centroid. Thus, both variants use cluster size to favor larger groups, but differ in the selected reference point and in the intra-cluster ordering criterion.

5.5.3.2 Core Algorithm

Algorithm 8 summarizes the *Spatial Cluster-Based* heuristic. The algorithm extracts the selected reference point of each robot and applies DBSCAN to identify spatial groups of nearby robots. Since DBSCAN may classify isolated points as noise, robots outside any cluster keep their initial score, set to zero in the implementation. Thus, only clustered robots are affected by the clustering component, with scores computed from the cluster size and the distance to the corresponding centroid.

Algorithm 8 Spatial Cluster-Based Heuristic

Require: Set of robots R, reference function $\text{ref}(\cdot)$, score function $\text{score}(\cdot)$, ordering mode *mode*
Ensure: Priority ordering π

$P \leftarrow \emptyset$ ▷ reference points used for clustering
for each robot $r_i \in R$ **do**
  $q_i \leftarrow \text{ref}(r_i)$ ▷ current position or goal position
  add $(q_i, \text{id}(r_i))$ to P
  $score_i \leftarrow 0$
end for
$C \leftarrow \text{DBSCAN}(P)$ ▷ spatial clusters of robot reference points
for each cluster $C_m \in C$ **do**
  $c_m \leftarrow$ centroid of C_m
  $d_{\max,m} \leftarrow 0$
  for each point $q_i \in C_m$ **do**
    $d_i \leftarrow \|q_i - c_m\|_2$
    $d_{\max,m} \leftarrow \max(d_{\max,m}, d_i)$
  end for
  for each point $q_i \in C_m$ associated with robot r_i **do**
    $score_i \leftarrow \text{score}(|C_m|, d_i, d_{\max,m})$
  end for
end for
$\pi \leftarrow \text{SortByScore}(R, score, mode)$
return π

The algorithm starts by building the set of reference points, which requires processing each robot once and therefore has $O(n)$ time complexity.

The computational cost of DBSCAN depends on the implementation used to perform neighborhood queries. In the implementation considered in this work, the ε-neighborhood of each point is obtained by scanning all other points. Since there are at most n points, each neighborhood query costs $O(n)$, and the complete clustering step has complexity $O(n^2)$.

After clustering, the heuristic computes the centroid of each cluster, the distance between each cluster member and the centroid, and the maximum distance inside the cluster. Since each robot belongs to at most one cluster, the sum of the cluster sizes is at most n. Therefore, the total score-computation cost over all clusters is $O(n)$. Thus, the total computational cost is given by

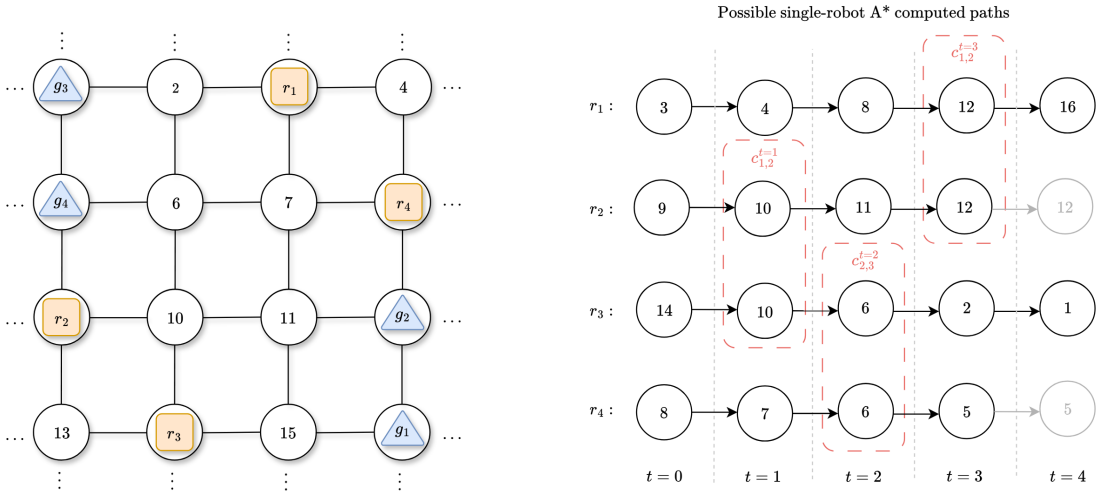
$$O(n^2) + O(n) + O(n \log n). \tag{5.33}$$

5.6 Conflict-Based Heuristics

The *Conflict-Based* heuristics developed in this work are motivated by the conflict and interaction criteria discussed in Section 3.2. As described in that section, previous work has considered interaction mostly through indirect indicators, such as the effect of existing plans on route cost [61]. In contrast, the heuristics proposed here use predicted space-time conflicts explicitly as priority information.

This information is obtained by first computing an individual path for each robot while ignoring the remaining robots. The resulting paths are then compared over time to identify predicted interactions between pairs of robots.

Figure 5.7 illustrates this process by combining the roadmap scenario with one possible set of independently computed single-robot A^* paths. Figure 5.7a shows the initial robot and goal configuration, while Figure 5.7b shows the predicted space-time conflicts obtained from comparing the individual paths over time.



(a) Conflict scenario.

(b) Single-robot A^* paths and predicted conflicts.

Figure 5.7: Illustrative example for the conflict-based heuristics.

For each robot r_i, the associated conflict occurrences are represented by the set O_i. Each conflict occurrence contributes to the priority score through a coefficient α. The priority score is then defined as

$$score_i = \sum_{c \in O_i} \alpha^t_{R_c}, \tag{5.34}$$

where R_c denotes the set of robots involved in conflict occurrence c, and t denotes the time step at which the conflict occurs.

Different conflict-based heuristics can be obtained by changing the definition of α. The underlying intuition is that robots involved in more relevant predicted conflicts are more likely to be difficult to coordinate in a prioritized planner such as TEA*. By assigning priorities according to conflict information, these heuristics attempt to plan first or last the robots whose individual paths are expected to interfere more strongly with the rest of the system.

5.6.1 Space-Time Conflicts

The Space-Time Conflicts heuristic corresponds to the simplest conflict-based variant. In this case, all predicted conflict occurrences are treated with the same importance. Therefore, each conflict occurrence has unit weight

$$\alpha^t_{R_c} = 1. \tag{5.35}$$

Substituting this definition into Equation 5.34, the priority score of robot r_i becomes

$$\text{score}(r_i) = |O_i|. \tag{5.36}$$

Thus, the score is simply the number of predicted space-time conflicts involving robot r_i. Robots involved in more conflicts receive higher priority scores, since their individual paths are expected to interfere more frequently with the paths of the remaining robots. For the example shown in Figure 5.7b, the predicted conflict occurrences are

$$c^{t=1}_{2,3}, \qquad c^{t=2}_{3,4}, \qquad c^{t=3}_{1,2}.$$

Since each conflict has unit weight, the resulting scores are

$$score_1 = 1, \qquad score_2 = 2, \qquad score_3 = 2, \qquad score_4 = 1.$$

Therefore, robots r_2 and r_3 obtain the highest conflict scores in this example, while robots r_1 and r_4 obtain lower scores. However, this score should be interpreted only as a static estimate of conflict involvement. The heuristic does not account for the fact that resolving an earlier conflict may change the subsequent paths and consequently remove conflicts that were originally predicted.

5.6.1.1 Ordering variants

Two ordering variants were considered for the *Space-Time Conflicts* heuristic. In the *More Conflicts First* variant, robots are sorted by decreasing score, so robots involved in more predicted conflicts are planned earlier. For the example in Figure 5.7b, this produces the priority grouping $\pi = \langle \{r_2, r_3\}, \{r_1, r_4\} \rangle$.

In the *Less Conflicts First* variant, the ordering is reversed, and robots with fewer predicted conflicts are planned earlier. The resulting priority grouping is $\pi = \langle \{r_1, r_4\}, \{r_2, r_3\} \rangle$.

5.6.1.2 Core Algorithm

The conflict extraction step is common to the conflict-based heuristics. Algorithm 9 computes an independent single-robot path for each robot and compares the resulting paths over time to obtain the set of predicted conflict occurrences O_i associated with each robot.

Algorithm 9 Predicted Conflict Extraction

Require: Set of robots R, current positions s_i, goal positions g_i
Ensure: Conflict occurrence sets O_i for all robots $r_i \in R$

for each robot $r_i \in R$ **do**
  $p_i \leftarrow A^*(s_i, g_i)$ ▷ single-robot shortest path on the roadmap
  $O_i \leftarrow \emptyset$ ▷ conflict occurrences involving robot r_i
end for
$L_{\max} \leftarrow \max_i |p_i|$
for $t = 0, 1, \ldots, L_{\max} - 1$ **do**
  for each pair of robots (r_i, r_j) with $i < j$ **do**
    if $t < |p_i|$ **and** $t < |p_j|$ **then**
      $e_i(t) \leftarrow$ edge traversed by robot r_i at timestep t
      $e_j(t) \leftarrow$ edge traversed by robot r_j at timestep t
      $V_{\text{conf}} \leftarrow$ shared endpoint vertices of $e_i(t)$ and $e_j(t)$
      if $V_{\text{conf}} \neq \emptyset$ **then**
        add conflict occurrence $(t, V_{\text{conf}}, r_j)$ to O_i
        add conflict occurrence $(t, V_{\text{conf}}, r_i)$ to O_j
      end if
    end if
  end for
end for
return $\{O_i\}_{r_i \in R}$

For each robot, the heuristic first computes a single-robot shortest path on the roadmap using A^*, and the resulting independent paths are then used to estimate the number of space-time conflicts involving each robot.

Using the graph-search complexity given in Equation 2.3, the path generation step is applied once per robot.

After the individual paths are obtained, the heuristic compares them over time. The maximum path length among all robots is denoted by $L_{\max} = \max_i |p_i|$. For each timestep, all unordered pairs of robots are checked for conflicts. Since there are $O(n^2)$ robot pairs and each pairwise conflict check requires constant time, the conflict detection step costs

$$O(L_{\max} \cdot n^2). \tag{5.37}$$

The score of each robot is obtained by counting the conflict occurrences stored in O_i. This operation is bounded by the number of detected conflicts and, in the worst case, does not exceed the pairwise conflict detection cost. The final score-based sorting step requires $O(n \log n)$ time. Thus, the total computational cost is given by

$$O\big(n \cdot (|V| + |E|) \log |V|\big) + O\big(L_{\max} \cdot n^2\big) + O(n \log n). \tag{5.38}$$

5.6.2 Time-DoF Weighted Conflicts

The *Time-DoF Weighted Conflicts* heuristic extends the previous conflict-based formulation by assigning a different weight to each predicted conflict occurrence. The objective is to distinguish conflicts not only by how often they occur, but also by when and where they occur along the individually planned paths.

For each conflict occurrence $c \in O_i$, the timestep at which the conflict occurs is denoted by t_c, the set of robots involved is denoted by R_c, and the set of vertices associated with the conflict is denoted by V_c. The coefficient assigned to occurrence c is defined as

$$\alpha_{R_c}^{t_c} = \frac{1}{\min\limits_{v \in V_c} \mathrm{DoF}(v)} \cdot \frac{1}{1 + t_c}, \tag{5.39}$$

where V_c contains the vertices associated with the conflict occurrence. The term $\min_{v \in V_c} \mathrm{DoF}(v)$ makes the coefficient depend on the most constrained vertex involved in the conflict. For vertex conflicts, this corresponds to the shared vertex. For edge interactions, V_c includes the vertices associated with the interacting edges, and the minimum DoF is used to represent the most constrained part of that interaction.

The temporal term gives higher weight to conflicts that occur earlier in the individual paths, based on the assumption that the first steps are more likely to be executed as originally computed. As time progresses, reservations introduced by other robots may force path changes, making later predicted conflicts less reliable indicators of actual future interference. This partially addresses a limitation of the previous heuristic, where all conflicts were counted equally, even though resolving an earlier conflict may alter subsequent paths and remove conflicts that were initially detected.

The DoF term gives higher weight to conflicts occurring in more constrained roadmap regions. When a conflict involves vertices with low DoF, the affected robots have fewer alternative movements available around that location. Such conflicts are therefore considered more critical than conflicts occurring in highly connected regions, where robots may have more possible escape routes or alternative paths.

Substituting the weight of Equation 5.39 into the general conflict-based score defined in Equation 5.34, the score of robot r_i becomes

$$score_i = \sum_{c \in O_i} \left(\frac{1}{\min\limits_{v \in V_c} \mathrm{DoF}(v)} \cdot \frac{1}{1+t_c} \right). \tag{5.40}$$

For the example shown in Figure 5.7b, assume that all vertices involved in the predicted conflicts have $\mathrm{DoF}(v) = 4$. Under this assumption, the coefficients are determined only by the timestep of each conflict:

$$\alpha_{2,3}^{t=1} = \frac{1}{4} \cdot \frac{1}{1+1} = \frac{1}{8}, \qquad \alpha_{3,4}^{t=2} = \frac{1}{4} \cdot \frac{1}{1+2} = \frac{1}{12}, \qquad \alpha_{1,2}^{t=3} = \frac{1}{4} \cdot \frac{1}{1+3} = \frac{1}{16}.$$

The resulting weighted scores are

$$score_1 = \frac{1}{16}, \qquad score_2 = \frac{1}{8} + \frac{1}{16} = \frac{3}{16}, \qquad score_3 = \frac{1}{8} + \frac{1}{12} = \frac{5}{24}, \qquad score_4 = \frac{1}{12}.$$

5.6.2.1 Ordering variants

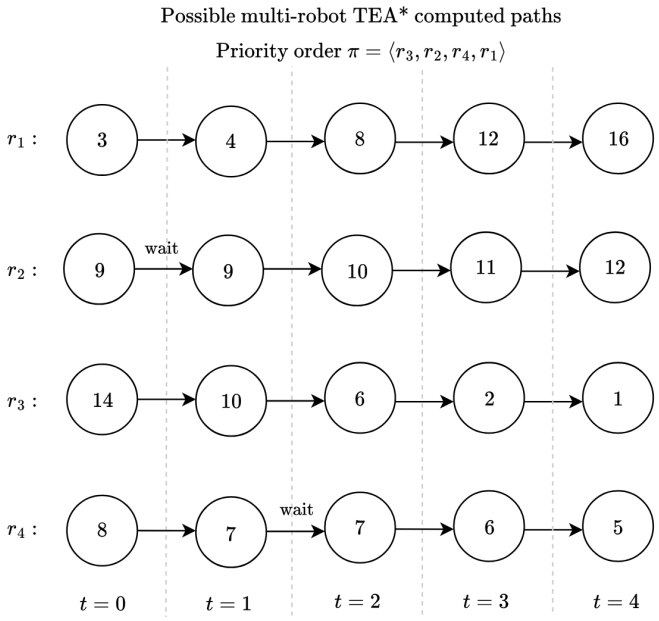
Two ordering variants were considered. *More Time-DoF Weighted Conflicts First* plans robots with higher weighted conflict scores earlier, giving priority to conflicts that are predicted to occur earlier and in more constrained roadmap regions. For the example shown in Figure 5.7b, this produces the priority ordering $\pi = \langle r_3, r_2, r_4, r_1 \rangle$.

Figure 5.8 shows a possible conflict-free set of paths generated by TEA* using this ordering, first as vertex sequences over time and then as a projection onto the roadmap structure. The sequence view makes the waiting actions explicit, while the projected view shows how the same paths evolve through space and time.

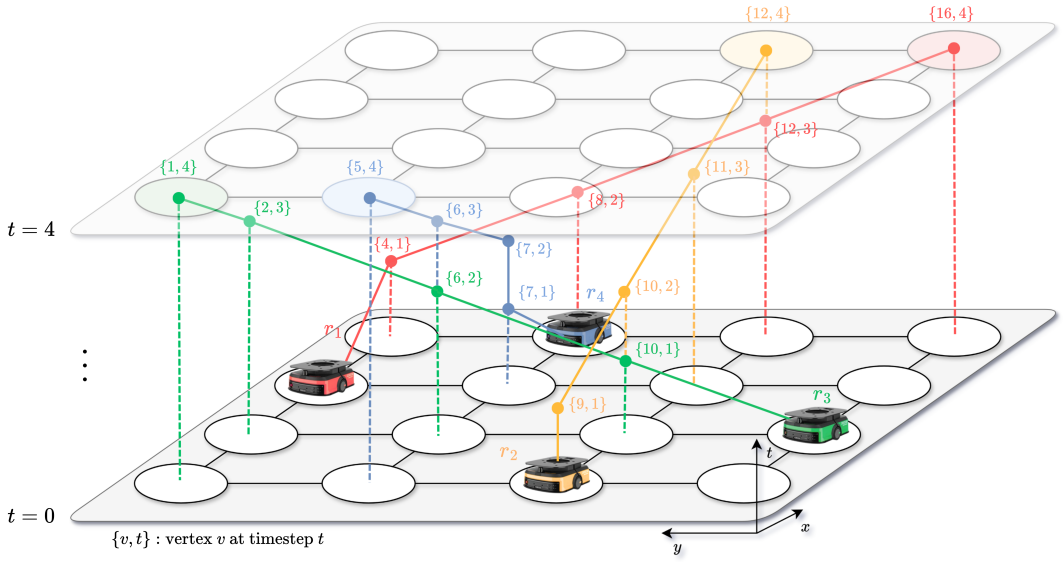
In this example, the priority ordering allows TEA* to introduce waiting actions that avoid the predicted conflicts shown in the independently computed paths. Robot r_2 waits at timestep $t = 1$, while robot r_4 waits at timestep $t = 2$. These waits change the temporal alignment of the paths and resolve the earlier interactions involving robots r_2, r_3, and r_4.

As a consequence, the later predicted conflict between r_1 and r_2 at timestep $t = 3$, visible in Figure 5.7b, no longer occurs in the final TEA* plan, since the waiting action introduced for r_2 changes its subsequent position sequence. This illustrates why later predicted conflicts should be weighted less than earlier ones, as they may disappear once previous interactions have been resolved.

This behavior motivates the temporal component of the *Time-DoF Weighted Conflicts* heuristic. By assigning lower weight to later conflicts, the heuristic gives more influence to interactions that are more likely to remain relevant during the final prioritized planning process. In the *Less Time-DoF Weighted Conflicts First* variant, the ordering is reversed, with lower weighted conflict scores planned earlier.



(a) Vertex-time sequences.



(b) Space-time projection.

Figure 5.8: Possible conflict-free TEA* plan obtained with the *More Time-DoF Weighted Conflicts First* ordering.

5.6.2.2 Core Algorithm

The *Time-DoF Weighted Conflicts* heuristic uses the same predicted conflict sets O_i computed by Algorithm 9, but replaces the unit conflict weight with a coefficient that depends on the conflict timestep and on the DoF of the associated roadmap vertices. The value $\mathrm{DoF}(v)$ is computed as defined in Section 5.4, by counting the distinct roadmap neighbors of vertex v.

Algorithm 10 Time-DoF Weighted Conflicts Scoring

Require: Set of robots R, conflict occurrence sets O_i, ordering mode *mode*
Ensure: Priority ordering π

for each robot $r_i \in R$ **do**
  $score_i \leftarrow 0$
  for each occurrence $(t, V_{\text{conf}}, r_j) \in O_i$ **do**
    $DoF_{\min} \leftarrow \min_{v \in V_{\text{conf}}} \mathrm{DoF}(v)$
    $\alpha \leftarrow \dfrac{1}{DoF_{\min}} \cdot \dfrac{1}{1+t}$
    $score_i \leftarrow score_i + \alpha$
  end for
end for
$\pi \leftarrow \textsc{SortByScore}(R, score, mode)$
return π

The *Time-DoF Weighted Conflicts* heuristic follows the same general structure as the *Space-Time Conflicts* heuristic, but each detected conflict is additionally weighted using time and DoF information. Since the DoF computation adds a factor bounded by the maximum roadmap degree Δ, the total computational cost becomes

$$O\big(n \cdot (|V| + |E|) \log |V|\big) + O\big(L_{\max} \cdot n^2 \cdot (1+\Delta)\big) + O(n \log n). \tag{5.41}$$

5.7 Replanning Heuristics

The replanning heuristic developed in this work is motivated by the rescheduling strategies discussed in Section 3.2, where feedback from previous planning attempts is used to adjust the priority ordering.

In this work, this idea is adapted to the behavior of TEA*. Rather than relying only on whether a robot failed to find a path, the proposed heuristic uses search statistics collected during a previous planning attempt. In particular, it considers how strongly each robot was affected by reservation conflicts during the TEA* search, using this information to compute a revised ordering for a subsequent planning attempt.

The resulting heuristic, named *Exploration Conflicts Pressure*, is therefore tailored to the information naturally produced by a prioritized planner. It uses replanning feedback to identify

robots whose search was more constrained by previous reservations, allowing the next priority ordering to reflect the difficulties observed in the previous attempt.

5.7.1 Exploration Conflicts Pressure

The *Exploration Conflicts Pressure* heuristic is designed for replanning scenarios, especially when a previous TEA* attempt fails. Unlike the initial-ordering heuristics, this heuristic does not estimate difficulty from distances, topology, local density, or predicted conflicts between independently planned paths. Instead, it uses runtime feedback collected during the actual prioritized planning process to revise the robot ordering.

The main idea is to identify robots whose search was strongly constrained by reservations introduced by previously planned robots. During TEA*, a candidate expansion may be rejected if the corresponding vertex or edge is already reserved in space-time by a higher-priority robot. If this happens frequently during the search of robot r_i, it suggests that the robot was negatively affected by its position in the previous ordering. Rather than simply reversing the previous permutation, the heuristic performs a targeted reordering, moving earlier only the robots that show evidence of being constrained by existing reservations.

For robot r_i, N_i^{rej} denotes the number of reservation-based rejections observed during the search, and E_i denotes the total number of expanded nodes. Instead of using N_i^{rej} directly, the heuristic normalizes this value by the total exploration effort E_i. This avoids assigning high scores only because a robot required a longer or more extensive search, which naturally creates more opportunities for reservation conflicts to occur. The priority score assigned to robot r_i is therefore defined as

$$score_i = \begin{cases} \dfrac{N_i^{\text{rej}}}{E_i}, & \text{if } E_i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.42}$$

This score measures the fraction of the search effort affected by reservation conflicts. Higher values indicate that a larger proportion of the attempted exploration was blocked by existing space-time reservations, rather than simply indicating that the robot performed a larger search. Therefore, robots with higher scores are interpreted as being more constrained by the previous ordering and may benefit from receiving higher priority in a subsequent planning attempt.

Figure 5.9 illustrates the type of runtime information used by this heuristic. In the represented TEA* expansion step, robot r_2 is being planned after other robots have already introduced space-time reservations. Three possible successor expansions are considered. Two of them are rejected because the corresponding roadmap resources are already reserved by previously planned robots, while only one expansion remains feasible. Therefore, this step contributes two reservation-based rejections to the value of N_2^{rej}.

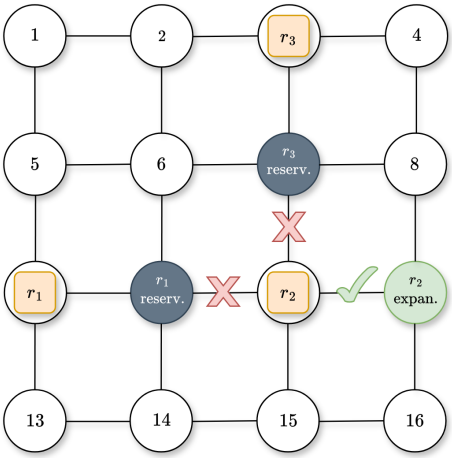


Figure 5.9: TEA* expansion step illustrating two reservation-based rejections.

This example shows why the ratio in Equation 5.42 can be useful during replanning. If similar rejections occur frequently throughout the search of a robot, this indicates that the robot was strongly affected by reservations imposed by higher-priority robots. The heuristic then uses this information to adjust the ordering in a subsequent planning attempt.

5.7.1.1 Core Algorithm

Algorithm 11 summarizes the *Exploration Conflicts Pressure* heuristic, which is applied only in a retry context. The heuristic uses statistics from a previous TEA* attempt, namely the number of expanded space-time nodes E_i and the number of reservation-based rejections N_i^{rej} observed while planning each robot r_i.

No additional path computation or pairwise conflict enumeration is performed, since the heuristic relies only on statistics already collected during the previous TEA* execution.

Algorithm 11 Exploration Conflicts Pressure Heuristic

Require: Planning queue Q, search statistics S from the previous planning attempt, ordering mode *mode*
Ensure: Revised priority ordering π

for each robot $r_i \in Q$ **do**
  if $r_i \notin S$ **or** $S[r_i].E_i = 0$ **then**
    $score_i \leftarrow 0$
  else
    $N_i^{\text{rej}} \leftarrow S[r_i].N_i^{\text{rej}}$ ▷ reservation-based rejections
    $E_i \leftarrow S[r_i].E_i$ ▷ expanded nodes
    $score_i \leftarrow \dfrac{N_i^{\text{rej}}}{E_i}$
  end if
end for
$\pi \leftarrow \text{SORTBYSCORE}(Q, score, mode)$
return π

The number of robots in the planning queue is denoted by $n = |Q|$. For each robot, the heuristic retrieves the corresponding search statistics and computes the ratio between reservation-based rejections and expanded nodes. Assuming that the statistics are accessed in constant time, the score computation step requires $O(n)$ time.

The final score-based sorting step requires $O(n \log n)$ time. Thus, the total computational cost is given by

$$O(n) + O(n \log n). \tag{5.43}$$

The cost of this heuristic is small compared with the TEA* planning attempt that produced the statistics. Its practical overhead is therefore limited, since it only reorders the robots using information already collected during the previous search.

5.8 Summary

This chapter presented the prioritization heuristics developed to generate alternative robot orderings for the TEA* planner. Since TEA* plans robots sequentially, the priority order directly affects the reservations introduced in the space-time search space and, consequently, the feasibility and quality of the resulting solution.

The proposed heuristics were organized into five groups, each capturing a different source of planning difficulty. Roadmap distance heuristics use travel-distance information, topological heuristics evaluate movement flexibility along estimated paths, surroundings heuristics consider the spatial distribution of robots, conflict-based heuristics use predicted space-time interactions, and replanning heuristics exploit feedback collected from previous TEA* attempts. These groups are not intended to define a single dominant priority rule, but rather to provide complementary criteria for constructing diverse robot orderings.

Table 5.3 summarizes the conceptual role of each heuristic, highlighting the priority criterion used by each family. Table 5.4 then provides the corresponding score functions and computational complexities. In both cases, the reported score defines the value assigned to each robot, while the final priority ordering also depends on the selected sorting mode, as defined in Section 5.2.

The computational analysis shows that the heuristics differ substantially in the amount of pre-planning effort they require. Direct distance and feedback-based criteria introduce limited overhead, while heuristics that depend on graph searches, neighborhood expansion, clustering, or predicted conflict extraction are more expensive. This distinction is important because heuristic computation is performed before TEA* execution, meaning that a more informative ordering is only useful if its computational cost remains justified by the improvement obtained during planning.

Overall, the chapter defines a diverse set of priority-ordering mechanisms and establishes their formal scoring rules and computational implications. The following chapter evaluates these heuristics experimentally, focusing on how their different sources of information affect planning feasibility, solution quality, and runtime across distinct coordination scenarios.

Table 5.3: Conceptual summary of the developed priority heuristics.

Family	**Heuristic**	**Priority criterion**
Roadmap distance	Farthest Robot	Direct distance between the current position and the goal.
	Longest Path	Cost of the individual shortest path on the roadmap.
Topology	Average DoF Path	Average roadmap connectivity along the estimated path.
	Low DoF Sum	Number of constrained vertices traversed by the estimated path.
Surroundings	Radius Neighbor Density	Number of robots within a fixed radius of the reference point.
	K-hop Neighbor Density	Number of robots within a bounded roadmap neighborhood.
	Spatial Cluster-Based	Cluster size weighted by relative position inside the cluster.
Conflicts	Space-Time Conflicts	Number of predicted space-time conflicts involving the robot.
	Time-DoF Weighted Conflicts	Predicted conflicts weighted by time and connectivity.
Replanning	Exploration Conflicts Pressure	Ratio of reservation-based rejections to expanded nodes.

Table 5.4: Score functions and computational complexity of the developed priority heuristics.

Heuristic	**Score function**	**Computational complexity**
Farthest Robot	$score_i = \|g_i - s_i\|_2$	$O(n) + O(n \log n)$
Longest Path	$score_i = \sum_{\ell=1}^{k-1} w_{v_\ell, v_{\ell+1}}$	$O\big(n(\|V\| + \|E\|) \log \|V\|\big) + O(n \log n)$
Average DoF Path	$score_i = \frac{1}{\|N_i\|} \sum_{v \in N_i} \mathrm{DoF}(v)$	$O\big(n(\|V\| + \|E\|) \log \|V\|\big) + O(nL_{\max}\Delta) + O(n \log n)$
Low DoF Sum	$score_i = \sum_{v \in P_i^*} \delta(v)$	$O\big(n(\|V\| + \|E\|) \log \|V\|\big) + O(nL_{\max}\Delta) + O(n \log n)$
Radius Neighbor Density	$score_i = \sum_{j \neq i} \rho(i, j)$	$O(n^2) + O(n \log n)$
K-hop Neighbor Density	$score_i = \sum_{j \neq i} \kappa(i, j)$	$O(nk_{\text{hops}}^3) + O(n \log n)$
Spatial Cluster-Based	$score_i = \|C_m\| \frac{d_i}{d_{\max,m}}$	$O(n^2) + O(n) + O(n \log n)$
Space-Time Conflicts	$score_i = \|O_i\|$	$O\big(n(\|V\| + \|E\|) \log \|V\|\big) + O(L_{\max} n^2) + O(n \log n)$
Time-DoF Weighted Conflicts	$score_i = \sum_{c \in O_i} \frac{1}{\min_{v \in V_c} \mathrm{DoF}(v)} \frac{1}{1 + t_c}$	$O\big(n(\|V\| + \|E\|) \log \|V\|\big) + O\big(L_{\max} n^2 (1 + \Delta)\big) + O(n \log n)$
Exploration Conflicts Pressure	$score_i = \frac{N_i^{\text{rej}}}{E_i}$	$O(n) + O(n \log n)$

$$\delta(v) = \begin{cases} 1, & \mathrm{DoF}(v) \leq \tau, \\ 0, & \text{otherwise}, \end{cases} \qquad \rho(i,j) = \begin{cases} 1, & d(n_i, n_j) \leq R_{\text{th}}, \\ 0, & \text{otherwise}, \end{cases} \qquad \kappa(i,j) = \begin{cases} 1, & n_j \in N_{k_{\text{hops}}}(n_i), \\ 0, & \text{otherwise}. \end{cases}$$

$$G = (V, E), \qquad n = |R|, \qquad L_{\max} = \max_i |P_i^*|, \qquad \Delta = \max_{v \in V} \mathrm{DoF}(v).$$

For the *Exploration Conflicts Pressure* heuristic, the score is set to zero when $E_i = 0$.

Chapter 6

Experimental Evaluation of Prioritization Heuristics

This chapter evaluates the prioritization heuristics introduced in Chapter 5 in an isolated testing environment. The objective is not yet to assess the complete planning framework, but rather to analyze how different priority orderings affect the behavior of the sequential TEA* planner.

Section 6.1 describes the testing framework prepared for this evaluation, including the workspace generation process, the conversion between grid-based and graph-based representations, and the execution architecture used to run planning instances. Section 6.2 then presents the experimental evaluation of the proposed heuristics, first in controlled scenarios designed to highlight specific planning conditions, and then in larger factory operation scenarios with 180 robots. The observations obtained in this chapter provide the experimental basis for the heuristic selection and usage mechanisms explored in the following chapter.

6.1 Isolated Planner Testing Framework

Before evaluating the developed prioritization heuristics, an isolated testing framework was prepared for the Path Planner. Since the heuristics developed in this work affect exclusively the planning layer, running the complete coordination stack was not necessary for the experimental evaluation.

Using an isolated version of the Path Planner provides two main advantages. First, it improves experimental control by limiting the number of external components involved in each test. This makes the observed behavior easier to associate with the planning algorithm and with the selected priority ordering. Second, this setup reduces the computational resources required to execute large-scale experiments, since robot navigation nodes and other execution-level components are not required to evaluate the centralized planning process itself.

6.1.1 Workspace Generation and Map Representation

All test environments used in this evaluation were represented as grid-based roadmaps with four-neighbor connectivity. This choice follows the type of representation adopted in several path-planning and MAPF benchmarks, where problems are commonly defined over 2D grid maps with discrete start and goal locations. In particular, the developed interface was inspired by the Moving AI benchmark format, which is widely used to support reproducible comparisons between path-planning algorithms [2], [68].

In this representation, each map is described by a text file in which each character corresponds to one grid cell. Following the same principle, the testing interface uses a *.txt* file, represented as *grid.txt* in Figure 6.1, where free cells are represented by "." and obstacles by "T". This format provides a simple and intuitive way of defining test environments, while keeping the scenarios close to commonly used benchmark inputs.

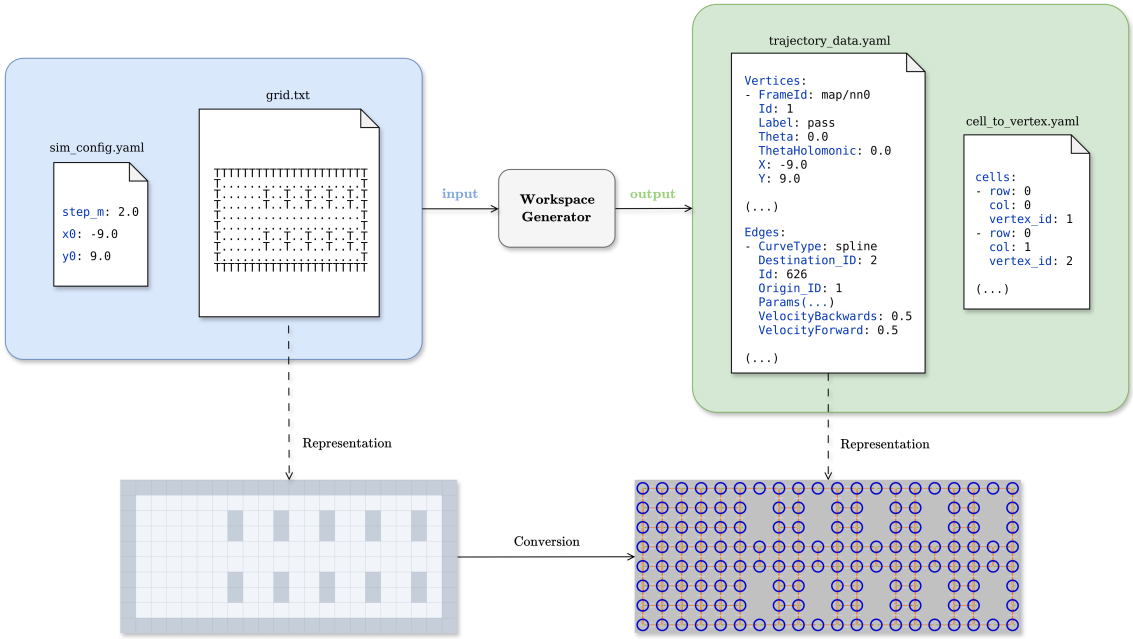


Figure 6.1: Workspace generation process used to convert grid-based maps into the graph representation required by the existing planning system.

Although the grid-based representation is convenient for defining maps and robot start and goal configurations, the existing CRIIS planning stack operates over a graph representation stored in a YAML (YAML Ain't Markup Language) file, represented as *trajectory_data.yaml* in Figure 6.1. For this reason, a conversion step was required to use the grid-based interface while preserving the original Path Planner and Graph Server inputs, minimizing the changes required to the existing source code.

To perform this conversion, a Python script named *Workspace Generator* was developed, as illustrated in Figure 6.1. The script receives the *grid.txt* file and a configuration file, *sim_config.yaml*, which defines general parameters such as the distance between adjacent vertices and the coordinates

of the first roadmap vertex. From these inputs, the script generates the *trajectory_data.yaml* file required by the existing system, together with the auxiliary *cell_to_vertex.yaml* mapping used to relate grid coordinates to graph vertex identifiers.

During this conversion, the grid-based representation shown on the left of Figure 6.1 is mapped into the graph-based representation shown on the right, with each free cell becoming a graph vertex. Obstacle cells are ignored, and edges are created between adjacent free cells according to four-neighbor connectivity. The generated *cell_to_vertex.yaml* file stores the correspondence between grid cells and graph vertex identifiers, allowing later stages of the testing pipeline to translate information between the grid-based and graph-based representations.

6.1.2 Test Execution Architecture

After generating the graph representation of the workspace, the isolated testing stack can be executed using only the components required for path computation. As shown in Figure 6.2, these components are the *Graph Server*, the *Path Planner*, and the newly developed *Planner Tester* node.

The *Graph Server*, already described in Chapter 4, reads the *trajectory_data.yaml* file and provides the graph representation required by the planning module. This graph is used by the *Path Planner* as the environment representation for TEA*, while the planning requests are generated separately by the *Planner Tester*. This testing node defines the robot initial positions and goals and sends the corresponding request to the Path Planner, which then computes the requested plan.

Each planning instance is defined in a file, *planning_instance.yaml*, containing the initial and goal grid positions of all robots. This representation makes scenarios easier to define manually, since robot positions can be specified in grid coordinates instead of directly using graph vertex identifiers.

Before sending the request, the *Planner Tester* uses the *cell_to_vertex.yaml* file to convert each initial and goal grid position into the corresponding graph vertex identifier. However, the Path Planner request interface describes the current robot position using not only a vertex, but also the current graph edge and the percentage of completion along that edge, providing the more continuous position representation used by the original system.

Therefore, the *Planner Tester* also uses the graph information provided by the *Graph Server* to build requests compatible with the existing Path Planner interface, while still allowing scenarios to be defined through simple grid coordinates.

Once the request is received, the Path Planner computes the corresponding TEA* plan using the selected priority ordering or prioritization heuristic under evaluation. The resulting plan is returned to the *Planner Tester*, which records the relevant outputs for later analysis. This setup enables the prioritization heuristics to be evaluated while preserving the existing planner interfaces and requiring only minimal modifications to the source code of the original system components.

After receiving the computed plan, the *Planner Tester* also exports the resulting robot paths to a *.txt* file using a grid-based path format. This file was then used with a visualization interface adapted from the CBSH2-RTC solver repository [69].

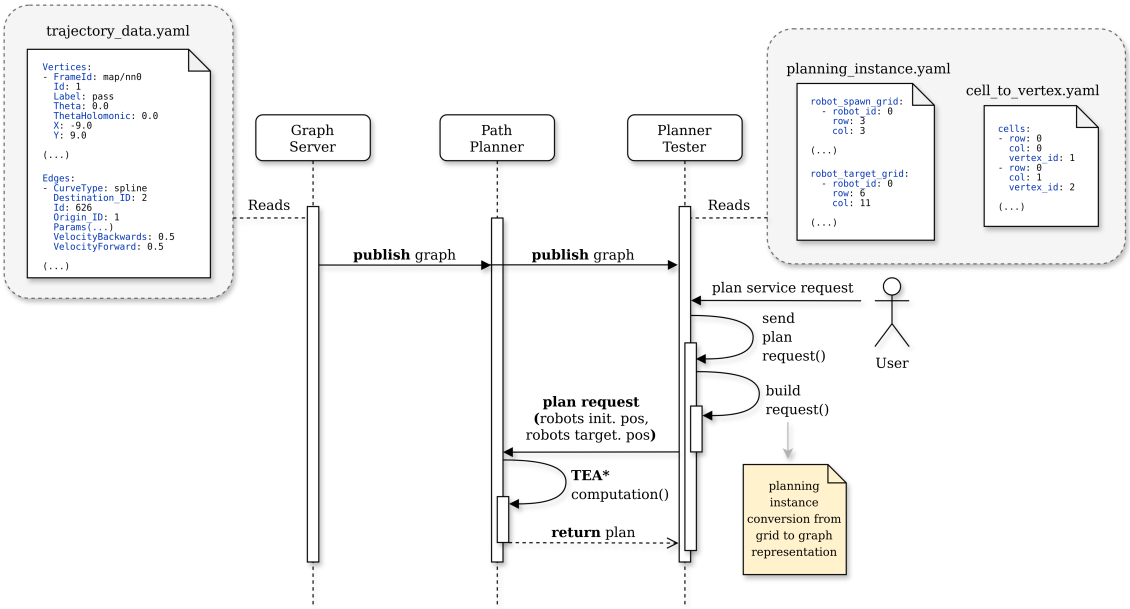


Figure 6.2: Execution flow of the isolated planner testing framework.

6.2 Heuristic Evaluation Experiments

This section presents the experimental evaluation performed to compare the prioritization heuristics developed in Chapter 5. The evaluation was divided into two stages. The first stage uses controlled scenarios designed to highlight planning conditions where different heuristic groups may lead to more favorable results. The second stage considers larger factory operation scenarios, with 180 robots, intended to assess the behavior of the heuristics under conditions closer to normal warehouse operation.

6.2.1 Experimental Setup

All experiments were executed on the same machine, using the hardware and software configuration summarized in Table 6.1. This information is reported because the measured planning and permutation computation times depend on the execution environment.

Table 6.1: Hardware and software configuration used in the experimental evaluation.

Component	**Specification**
Operating system	Ubuntu 20.04.6 LTS
CPU	AMD Ryzen 5 7535HS with Radeon Graphics
CPU cores	6 cores
RAM	14 GiB
ROS distribution	ROS Noetic
Programming language	C++

6.2.2 Heuristic Parameters

The heuristic parameters used throughout the evaluation are summarized in Table 6.2. These values were kept fixed across all scenarios to ensure that the observed differences were caused by the priority rules themselves rather than by parameter tuning adjusted for individual scenarios.

Table 6.2: Heuristic parameters used in the experimental evaluation.

Heuristic group	**Parameter**	**Value**	**Rationale**
Spatial clustering	ε	2.5	Considers the 2 m spacing between adjacent roadmap vertices with a small margin.
Spatial clustering	minClusterSize	2	Avoids isolated positions being treated as clusters, while still detecting the smallest relevant local groups.
Topological	DoF_{th}	2	Identifies corridor-like vertices, since $DoF \leq 2$ typically indicates limited movement alternatives.
Radius-based density	R_{th}	5	Covers approximately two roadmap steps around each robot, while keeping the neighborhood local.
K-hop density	K	2	Limits the topological neighborhood analysis to two graph hops around each robot.

The spatial clustering threshold was kept relatively small because clusters can expand transitively through neighboring robots. Thus, ε only needs to connect robots around adjacent roadmap vertices, considering the 2 m vertex spacing with a small margin. In contrast, the radius-based and k-hop parameters define fixed local neighborhoods around each robot, and were therefore set to cover approximately two roadmap steps without approximating a global search.

6.2.3 Evaluation Metrics

In the diagrams used to represent the test scenarios, orange squares indicate the robot initial positions, while blue triangles indicate their goal positions. The result tables report *Robot completion*, *t permutation*, *Excess cost*, *Cost improvement*, and *Permutation time of total*. These correspond, respectively, to the number of successfully planned robots, the priority permutation computation time, the additional solution cost over independent single-robot shortest paths, the relative improvement over the successful random baseline, and the percentage of total planning time spent computing the permutation.

When a heuristic failed to compute valid paths for all robots, its excess cost was represented as ∞, since the plan was incomplete. For complete solutions, the reported cost is the excess trajectory cost relative to the sum of the optimal individual robot costs: $C = \sum_{i=1}^{n}(c_i - c_i^*)$, where c_i is the coordinated trajectory cost of robot r_i and c_i^* is its optimal individual cost when planned without other robots. This subtraction removes the constant individual planning offset and makes the effective coordination overhead easier to observe.

A random baseline was also included to represent the behavior of the original planning approach, where the robot planning order is not determined by a specific prioritization heuristic. For each scenario, this baseline was evaluated through 100 repetitions with randomly generated priority permutations. The reported results include the success rate and the average excess cost of the successful executions, allowing the proposed heuristics to be compared against the variability introduced by random priority assignments. Given the mean excess cost C_{rand} of the successful random runs and the excess cost C_h obtained by a heuristic ordering, the reported cost improvement is computed as $\text{Gain} = \frac{C_{\text{rand}} - C_h}{C_{\text{rand}}} \times 100$, expressed as a percentage, so that positive values indicate a lower coordination cost than the random baseline. Incomplete heuristic runs are assigned infinite cost and are therefore not associated with a percentage gain.

The *Exploration Conflicts Pressure* heuristic was evaluated only in scenarios where at least one heuristic failed to compute valid paths for all robots, reflecting its corrective role as a method that uses information from unsuccessful TEA* attempts to generate a new priority ordering that more intelligently reorders robots affected by previous planning failures. In these cases, the reservation-based rejections and node expansions produced by the incomplete heuristic executions were accumulated per robot and used in Eq. 5.42 to compute the replanning score.

6.2.4 Heuristic Behavior in Controlled Scenarios

The following controlled scenarios were designed to highlight different planning conditions in which the selected priority ordering may have a significant impact on TEA* performance. In each case, all heuristics were tested under the same initial and goal configurations, allowing their behavior to be compared under equivalent conditions.

6.2.4.1 Initial Position Density Scenario

The first controlled scenario, illustrated in Figure 6.3, was designed to evaluate how the heuristics behave when the difficulty of the planning instance is strongly influenced by the initial spatial distribution of the robots. In this scenario, several robots start in a compact group, while the remaining robots have goals distributed across the central and right regions.

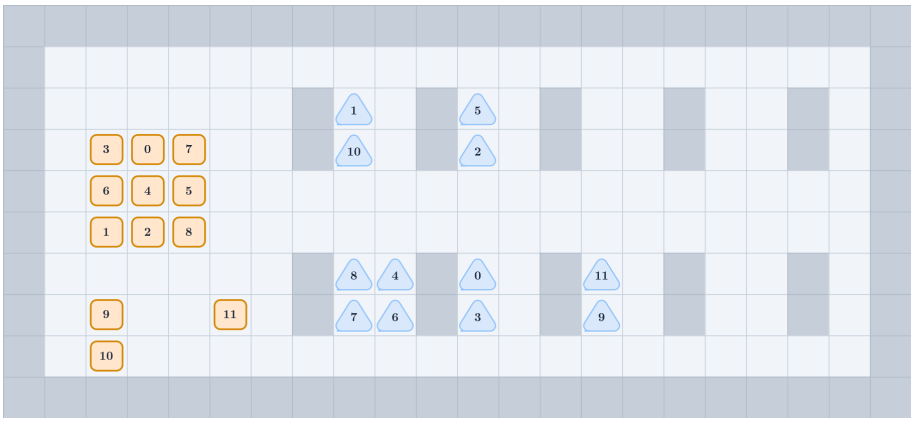


Figure 6.3: Initial Position Density Scenario with a compact initial robot distribution.

This configuration creates a local blockage, making the feasibility of the solution highly dependent on whether the robots blocking the exits of the cluster are planned before the robots trapped inside it. In particular, the feasibility of this scenario is strongly affected by the position of robot 4 in the priority ordering. Due to the sequential nature of TEA*, a valid solution is only obtained when robot 4 is planned after at least one of the robots blocking its available exits, namely robots 0, 2, 5, or 6. If all these blocking robots are planned after robot 4, the surrounding space remains reserved in a way that prevents robot 4 from finding a valid path.

This behavior is also reflected in the random baseline results shown in Table 6.3. Considering only the five relevant robots, robot 4 and the four robots blocking its exits, a random ordering fails when robot 4 appears before all four blocking robots. Since each of these five robots has the same probability of being the first one in this subset, the expected success probability is

$$P(\text{success}) = 1 - \frac{1}{5} = \frac{4}{5} = 80\%. \tag{6.1}$$

The experimental result of 81 successful runs out of 100 random repetitions is therefore consistent with this expected behavior.

Table 6.3: Random baseline performance over 100 repetitions for the Initial Position Density Scenario.

Baseline	**Repetitions**	**Success rate**	**Mean excess cost of successful runs (s)**
Random ordering	100	81/100	274.55

Table 6.4 compares the results obtained with the proposed heuristics. Several heuristics were able to compute valid paths for all robots, but the results also show that the choice of priority ordering has a clear impact on both feasibility and solution cost. Some heuristics, such as *High Average DoF Path First*, *Low DoF Sum Last*, *Farthest Robot Last*, *Longest Path Last*, and the goal-density heuristics, failed to plan all robots in this scenario.

The results show that the *Surroundings (pos)* heuristics were the most consistent group in this scenario. This is coherent with the structure of the instance, since these heuristics explicitly consider the initial spatial distribution of the robots, which is the main factor affecting feasibility in this case.

Nevertheless, the lowest excess cost was obtained by the *Less Time-DoF Weighted Conflicts First* heuristic. In this specific scenario, robots located near the exits of the initial cluster tend to generate fewer weighted conflicts in the early stages of planning, which allows this conflict-based heuristic to produce a particularly efficient ordering. However, this result does not necessarily imply that conflict-based heuristics are always preferable for this type of configuration. If the robots near the cluster boundary had goals on the opposite side of the group, their individually preferred paths would cross the central region and generate additional interactions.

Therefore, this scenario illustrates two important aspects of the evaluation. First, the initial spatial distribution of the robots can be decisive for the feasibility of the TEA* solution. Second,

although conflict-based heuristics may achieve the lowest cost in specific instances, heuristics based on initial-position surroundings provide a more direct and robust criterion for scenarios where the main difficulty is to release locally blocked robots from a dense initial configuration.

Table 6.4: Comparison of heuristics for the Initial Position Density Scenario.

Group	Heuristic	Robot completion	t perm. (ms)	Excess cost (s)	Cost improv. (%)	Perm. time of total (%)
Conflicts	Less Conflicts First	12/12	2.79	229.51	+16.4	27.6
Conflicts	**Less Time-DoF Weighted Conflicts First**	**12/12**	**3.26**	**213.65**	**+22.2**	**32.3**
Conflicts	More Conflicts First	12/12	3.02	301.06	-9.7	29.9
Conflicts	More Time-DoF Weighted Conflicts First	12/12	2.43	324.93	-18.3	24.1
Topological	High Average DoF Path First	11/12	3.53	∞	—	26.8
Topological	Low Average DoF Path First	12/12	3.20	258.69	+5.8	24.2
Topological	Low DoF Sum First	12/12	2.16	292.27	-6.5	16.4
Topological	Low DoF Sum Last	11/12	3.34	∞	—	25.3
Roadmap dist.	Farthest Robot First	12/12	0.04	252.14	+8.2	0.3
Roadmap dist.	Farthest Robot Last	11/12	0.15	∞	—	1.3
Roadmap dist.	Longest Path First	12/12	1.28	277.06	-0.9	10.6
Roadmap dist.	Longest Path Last	11/12	1.15	∞	—	9.6
Surroundings (goal)	High Cluster Central Goal First	11/12	0.04	∞	—	0.4
Surroundings (goal)	High K-hop Density Goals First	11/12	0.26	∞	—	2.3
Surroundings (goal)	High Radius Density Goals First	11/12	0.14	∞	—	1.2
Surroundings (pos)	High Cluster Peripheral First	12/12	0.06	321.06	-16.9	0.7
Surroundings (pos)	Low K-hop Density Positions First	12/12	0.37	248.01	+9.7	4.4
Surroundings (pos)	Low Radius Density Positions First	12/12	0.16	248.01	+9.7	1.9
Replanning	Exploration Conflicts Pressure	12/12	0.03	290.11	-5.7	0.1

6.2.4.2 High Conflict Density Scenario

The next controlled scenario, shown in Figure 6.4, was designed to analyze heuristic behavior when planning difficulty is mainly driven by conflicts between robot paths.

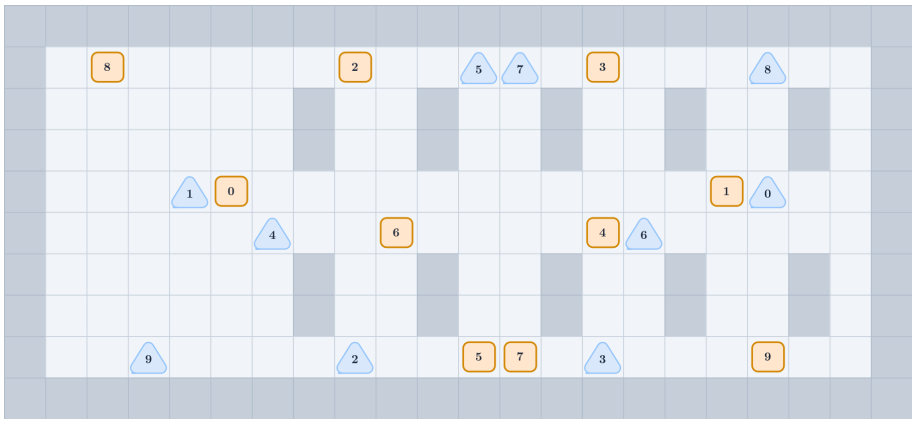


Figure 6.4: High Conflict Density Scenario, with several expected interactions between robot paths.

In this configuration, the initial and goal positions create several expected interactions across shared roadmap regions, making it useful to evaluate how different conflict priority rules affect the final TEA* solution cost.

The random baseline results are presented in Table 6.5. In this scenario, all 100 random priority permutations produced complete solutions, indicating that the instance is not especially difficult in terms of feasibility. However, the mean cost of the successful random executions remains higher than the best heuristic result, showing that the selected priority ordering still has a clear impact on solution quality.

Table 6.5: Random baseline performance over 100 repetitions for the High Conflict Density Scenario.

Baseline	**Repetitions**	**Success rate**	**Mean excess cost of successful runs (s)**
Random ordering	100	100/100	274.42

The results in Table 6.6 show that all heuristics produced complete solutions, although the excess cost varied considerably across priority orderings.

Table 6.6: Comparison of heuristics for the High Conflict Density Scenario.

Group	**Heuristic**	**Robot completion**	**t perm. (ms)**	**Excess cost (s)**	**Cost improv. (%)**	**Perm. time of total (%)**
Conflicts	Less Conflicts First	10/10	1.59	311.35	-13.5	18.5
Conflicts	Less Time-DoF Weighted Conflicts First	10/10	1.00	315.61	-15.0	11.8
Conflicts	More Conflicts First	10/10	1.72	274.83	-0.1	20.1
Conflicts	**More Time-DoF Weighted Conflicts First**	**10/10**	**1.22**	**197.25**	**+28.1**	**14.3**
Topological	High Average DoF Path First	10/10	4.79	198.70	+27.6	41.4
Topological	Low Average DoF Path First	10/10	4.37	254.30	+7.3	37.7
Topological	Low DoF Sum First	10/10	2.08	288.01	-5.0	18.0
Topological	Low DoF Sum Last	10/10	0.93	245.51	+10.5	10.8
Roadmap dist.	Farthest Robot First	10/10	0.04	349.32	-27.3	0.4
Roadmap dist.	Farthest Robot Last	10/10	0.11	208.59	+24.0	1.0
Roadmap dist.	Longest Path First	10/10	1.12	363.03	-32.3	10.0
Roadmap dist.	Longest Path Last	10/10	1.09	208.59	+24.0	9.7
Surroundings (goal)	High Cluster Central Goal First	10/10	0.04	241.12	+12.1	0.6
Surroundings (goal)	High K-hop Density Goals First	10/10	0.23	241.12	+12.1	3.0
Surroundings (goal)	High Radius Density Goals First	10/10	0.13	241.12	+12.1	1.7
Surroundings (pos)	High Cluster Peripheral First	10/10	0.04	241.12	+12.1	0.4
Surroundings (pos)	Low K-hop Density Positions First	10/10	0.30	260.59	+5.0	3.1
Surroundings (pos)	Low Radius Density Positions First	10/10	0.16	260.59	+5.0	1.7

As expected, conflict-based heuristics were particularly relevant in this scenario. Although all tested priority orderings produced complete solutions, *More Time-DoF Weighted Conflicts First* achieved the lowest cost. This matches the instance structure, where several robots interact along their paths, making it beneficial to plan the most restrictive movements earlier.

However, prioritizing the robots involved in more conflicts is not always the best strategy in conflict-intensive scenarios. To illustrate this, a second conflict-oriented configuration was tested, as shown in Figure 6.5.

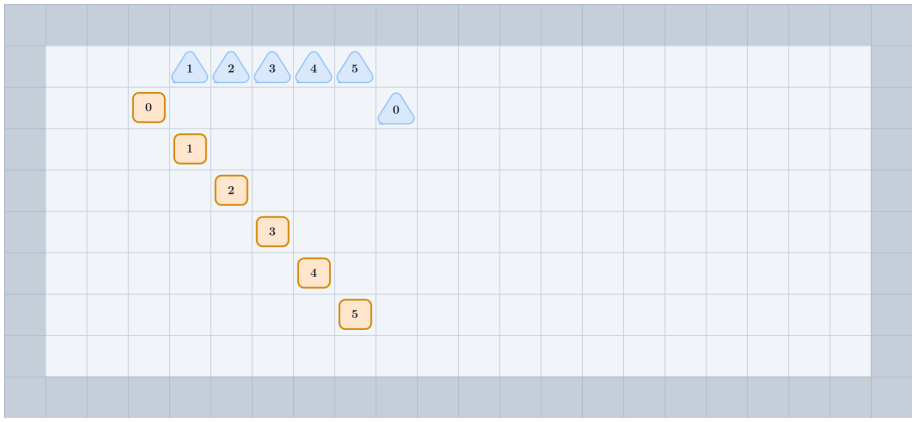


Figure 6.5: Scenario where most expected conflicts are concentrated around one robot.

In this case, most expected conflicts are concentrated around a single robot. If this robot is planned first, its path forces several other robots to wait, increasing the total solution cost. Conversely, if the remaining robots are planned first, the additional waiting cost is mostly concentrated on that single robot.

Table 6.7: Random baseline performance over 100 repetitions for the Dominant Conflict Robot Scenario.

Baseline	**Repetitions**	**Success rate**	**Mean excess cost of successful runs (s)**
Random ordering	100	100/100	47.60

The results in Table 6.8 show the opposite behavior. Here, the best conflict-based results were obtained by *Less Conflicts First* and *Less Time-DoF Weighted Conflicts First*, both with an excess cost of 12.39 s. In contrast, prioritizing the robot involved in more conflicts led to a substantially higher cost, with both *More Conflicts First* and *More Time-DoF Weighted Conflicts First* obtaining 66.10 s.

These two scenarios show that the best conflict-based strategy depends on the distribution of conflicts and on the level of environmental constraint. When several robots are mutually involved in restrictive interactions, especially in regions with few topological alternatives, prioritizing the most conflict-prone robots can reduce the global coordination cost by resolving the most constrained movements earlier.

However, when conflicts are mainly concentrated around a single robot, planning that robot first may force several simpler movements to wait. In that case, planning the less conflicting robots first can be more efficient, because the additional waiting cost is mostly absorbed by the dominant conflict robot instead of being imposed on the rest of the fleet.

Table 6.8: Comparison of heuristics for the Dominant Conflict Robot Scenario.

Group	Heuristic	Robot completion	t perm. (ms)	Excess cost (s)	Cost improv. (%)	Perm. time of total (%)
Conflicts	**Less Conflicts First**	**6/6**	**0.23**	**12.39**	**+74.0**	**5.5**
Conflicts	**Less Time-DoF Weighted Conflicts First**	**6/6**	**0.25**	**12.39**	**+74.0**	**5.9**
Conflicts	More Conflicts First	6/6	0.28	66.10	-38.9	6.6
Conflicts	More Time-DoF Weighted Conflicts First	6/6	0.75	66.10	-38.9	17.6
Topological	High Average DoF Path First	6/6	0.32	66.10	-38.9	7.6
Topological	**Low Average DoF Path First**	**6/6**	**0.33**	**12.39**	**+74.0**	**7.7**
Topological	Low DoF Sum First	6/6	0.26	49.44	-3.9	6.1
Topological	Low DoF Sum Last	6/6	0.26	49.44	-3.9	6.0
Roadmap dist.	Farthest Robot First	6/6	0.03	65.71	-38.0	0.7
Roadmap dist.	**Farthest Robot Last**	**6/6**	**0.01**	**12.39**	**+74.0**	**0.3**
Roadmap dist.	Longest Path First	6/6	0.14	65.71	-38.0	3.4
Roadmap dist.	Longest Path Last	6/6	0.31	28.39	+40.4	7.3
Surroundings (goal)	**High Cluster Central Goal First**	**6/6**	**0.02**	**12.39**	**+74.0**	**0.4**
Surroundings (goal)	**High K-hop Density Goals First**	**6/6**	**0.05**	**12.39**	**+74.0**	**1.1**
Surroundings (goal)	**High Radius Density Goals First**	**6/6**	**0.06**	**12.39**	**+74.0**	**1.3**
Surroundings (pos)	High Cluster Peripheral First	6/6	0.02	65.71	-38.0	0.4
Surroundings (pos)	Low K-hop Density Positions First	6/6	0.06	49.44	-3.9	1.3
Surroundings (pos)	Low Radius Density Positions First	6/6	0.03	49.44	-3.9	0.6

6.2.4.3 Topological Constraint Scenario

The next controlled scenario, illustrated in Figure 6.6, was designed to evaluate the behavior of the heuristics in a configuration where the planning difficulty is mainly caused by a narrow corridor. In this case, the critical interaction occurs between robots 0 and 1. Robot 1 starts inside the constrained region and must leave it, while robot 0 has a goal located in that same area. Since the corridor provides limited room for alternative movements, planning robot 0 before robot 1 may block the latter's only viable exit and prevent a complete solution from being found.

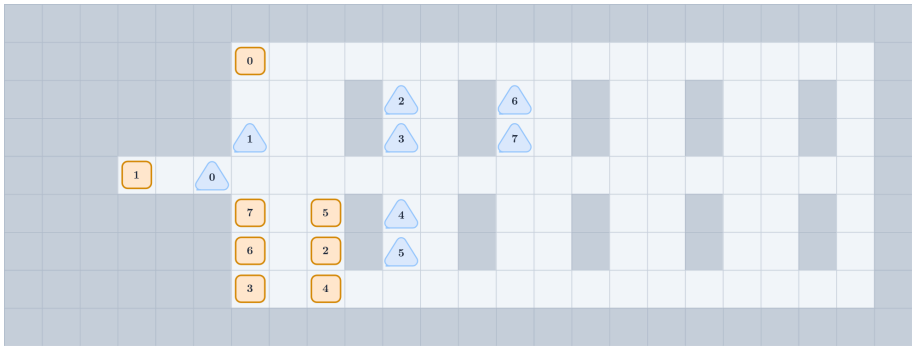


Figure 6.6: Topological Constraint Scenario, where a narrow corridor makes the relative ordering between two robots critical for solution feasibility.

This dependency is reflected in the random baseline results presented in Table 6.9. Considering only the two critical robots, a random priority ordering succeeds when robot 1 is planned before

robot 0. Since both relative orders are equally likely, the expected success probability is

$$P(\text{success}) = \frac{1}{2} = 50\%. \tag{6.2}$$

The observed success rate, with 49 valid solutions out of 100 random permutations, matches this expected dependency between the two critical robots.

Table 6.9: Random baseline performance over 100 repetitions for the Topological Constraint Scenario.

Baseline	**Repetitions**	**Success rate**	**Mean excess cost of successful runs (s)**
Random ordering	100	49/100	122.75

As shown in Table 6.10, most heuristics failed to produce complete solutions because they did not generate the critical ordering required between robots 1 and 0. In this scenario, complete solutions were obtained only by the topological heuristics *Low Average DoF Path First* and *Low DoF Sum First*, with total costs of 107.28 s and 136.46 s, respectively, and by the corrective *Exploration Conflicts Pressure* heuristic, with an excess cost of 134.63 s.

Table 6.10: Comparison of heuristics for the Topological Constraint Scenario.

Group	**Heuristic**	**Robot completion**	**t perm. (ms)**	**Excess cost (s)**	**Cost improv. (%)**	**Perm. time of total (%)**
Conflicts	Less Conflicts First	7/8	1.95	∞	—	25.4
Conflicts	Less Time-DoF Weighted Conflicts First	7/8	2.43	∞	—	31.7
Conflicts	More Conflicts First	7/8	1.88	∞	—	24.5
Conflicts	More Time-DoF Weighted Conflicts First	7/8	2.65	∞	—	34.6
Topological	High Average DoF Path First	7/8	3.58	∞	—	41.6
Topological	**Low Average DoF Path First**	**8/8**	**3.76**	**107.28**	**+12.6**	**43.7**
Topological	Low DoF Sum First	8/8	3.66	136.46	-11.2	42.6
Topological	Low DoF Sum Last	7/8	2.85	∞	—	33.1
Roadmap dist.	Farthest Robot First	7/8	0.04	∞	—	0.6
Roadmap dist.	Farthest Robot Last	7/8	0.07	∞	—	1.1
Roadmap dist.	Longest Path First	7/8	0.73	∞	—	12.7
Roadmap dist.	Longest Path Last	7/8	0.71	∞	—	12.5
Surroundings (goal)	High Cluster Central Goal First	7/8	0.03	∞	—	0.6
Surroundings (goal)	High K-hop Density Goals First	7/8	0.16	∞	—	3.5
Surroundings (goal)	High Radius Density Goals First	7/8	0.09	∞	—	1.9
Surroundings (pos)	High Cluster Peripheral First	7/8	0.04	∞	—	1.0
Surroundings (pos)	Low K-hop Density Positions First	7/8	0.13	∞	—	3.1
Surroundings (pos)	Low Radius Density Positions First	7/8	0.07	∞	—	1.7
Replanning	Exploration Conflicts Pressure	8/8	0.01	134.63	-9.7	0.1

The successful topological heuristics are consistent with the structure of the planning instance. The main difficulty is created by the narrow corridor, where robot 1 has limited movement alternatives and can easily be blocked if other robots reserve the critical region first. Therefore, heuristics

that prioritize more constrained robots are able to capture the relevant topological restriction and produce complete solutions.

6.2.4.4 Goal Density Scenario

The final controlled scenario, illustrated in Figure 6.7, presents a structure similar to the Initial Position Density Scenario, but with the main constraint shifted from the initial robot configuration to the goal distribution. In this case, the robots do not start in a congested region. Instead, several goals are concentrated in the same area, creating a dense arrival region that may become blocked depending on the planning order.

For this reason, this scenario is expected to favor heuristics from the *Surroundings (goal)* group as these heuristics analyze the spatial distribution of the goal positions and tend to prioritize robots whose goals are located in more critical regions of the goal cluster.

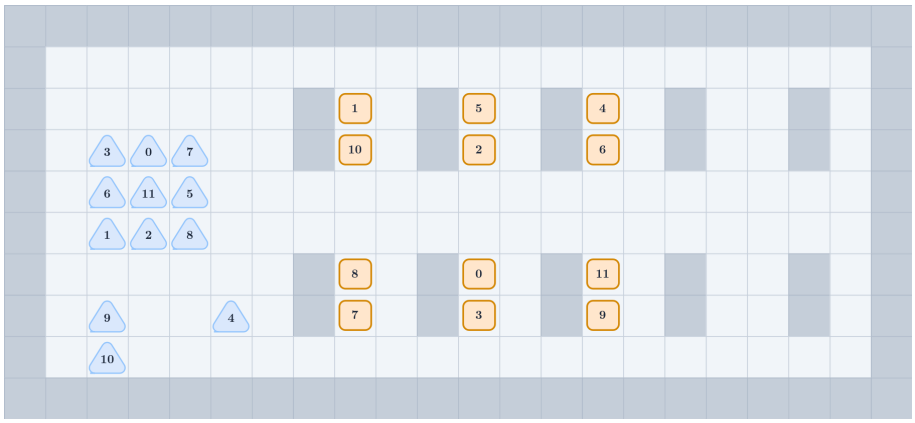


Figure 6.7: Goal Density Scenario, where several goal positions are concentrated in the same region.

The random baseline results are presented in Table 6.11. As in the Initial Position Density Scenario, the success rate can be explained by the relative position of a small group of critical robots in the priority ordering. In this case, the relevant condition involves robot 11 and robots 0, 2, 5, and 6. A valid solution tends to be obtained when robot 11 is planned before at least one of these four robots, preventing the goal region from becoming fully blocked before robot 11 reaches it.

Considering only these five robots, a random permutation fails when robot 11 appears after all robots 0, 2, 5, and 6. Since each of the five robots has the same probability of being the last one within this subset, the expected failure probability is $1/5$. Therefore, the expected success probability is

$$P(\text{success}) = 1 - \frac{1}{5} = \frac{4}{5} = 80\%. \tag{6.3}$$

The baseline result, with 76 successful executions out of 100 random permutations, is close to this expected value.

Table 6.11: Random baseline performance over 100 repetitions for the Goal Density Scenario.

Baseline	**Repetitions**	**Success rate**	**Mean excess cost of successful runs (s)**
Random ordering	100	76/100	283.85

As shown in Table 6.12, all *Surroundings (goal)* heuristics computed complete solutions, which is consistent with the structure of the scenario. Since the main difficulty is associated with the concentration of goal positions, priority rules that explicitly account for goal density are directly aligned with the planning condition being tested.

Table 6.12: Comparison of heuristics for the Goal Density Scenario.

Group	**Heuristic**	**Robot completion**	**t perm. (ms)**	**Excess cost (s)**	**Cost improv. (%)**	**Perm. time of total (%)**
Conflicts	Less Conflicts First	12/12	2.71	299.88	-5.6	13.0
Conflicts	Less Time-DoF Weighted Conflicts First	12/12	2.26	297.32	-4.7	10.9
Conflicts	More Conflicts First	11/12	3.81	∞	—	18.3
Conflicts	More Time-DoF Weighted Conflicts First	11/12	3.77	∞	—	18.2
Topological	High Average DoF Path First	12/12	3.60	246.17	+13.3	22.7
Topological	Low Average DoF Path First	12/12	3.15	303.88	-7.1	19.9
Topological	Low DoF Sum First	11/12	2.71	∞	—	17.1
Topological	Low DoF Sum Last	11/12	3.01	∞	—	19.0
Roadmap dist.	Farthest Robot First	12/12	0.06	319.61	-12.6	0.4
Roadmap dist.	**Farthest Robot Last**	**12/12**	**0.17**	**206.43**	**+27.3**	**1.2**
Roadmap dist.	Longest Path First	12/12	1.53	359.68	-26.7	10.4
Roadmap dist.	Longest Path Last	11/12	1.47	∞	—	10.0
Surroundings (goal)	High Cluster Central Goal First	12/12	0.04	284.80	-0.3	0.5
Surroundings (goal)	High K-hop Density Goals First	12/12	0.16	284.80	-0.3	1.9
Surroundings (goal)	High Radius Density Goals First	12/12	0.15	284.80	-0.3	1.8
Surroundings (pos)	High Cluster Peripheral First	11/12	0.06	∞	—	0.4
Surroundings (pos)	Low K-hop Density Positions First	11/12	0.33	∞	—	2.1
Surroundings (pos)	Low Radius Density Positions First	11/12	0.21	∞	—	1.3
Replanning	Exploration Conflicts Pressure	11/12	0.02	∞	—	0.0

Although *Farthest Robot Last* obtained the lowest numerical cost, the most consistent behavior in this scenario was observed in the *Surroundings (goal)* group. All goal-density heuristics planned the complete set of robots and produced the same excess cost, showing that they captured the relevant structure of the instance. In contrast, the *Surroundings (pos)* heuristics failed because the initial robot distribution is not the main source of difficulty in this case.

This result complements the Initial Position Density Scenario. In both cases, the planning difficulty is caused by a dense spatial configuration, but the relevant density appears in different parts of the problem. When the critical region is formed by the initial positions, the *Surroundings (pos)* heuristics are more appropriate. When the critical region is formed by the goal positions, as

in this scenario, the *Surroundings (goal)* heuristics provide the more directly aligned prioritization rule.

6.2.5 Large-Scale Factory Operation Scenarios

After the controlled scenarios, a second set of experiments was performed using a larger factory environment. The objective was to evaluate how the prioritization heuristics behave in planning instances closer to realistic large-scale warehouse operation, and to compare their performance with that of the original planner, where robot priorities are not defined using heuristics.

The considered environment, shown in Figure 6.8, represents a large factory layout with multiple warehouse regions distributed along the roadmap, designed to increase the number of robots and the density of possible path interactions while preserving a structured layout similar to the type of environment expected in industrial logistics applications.

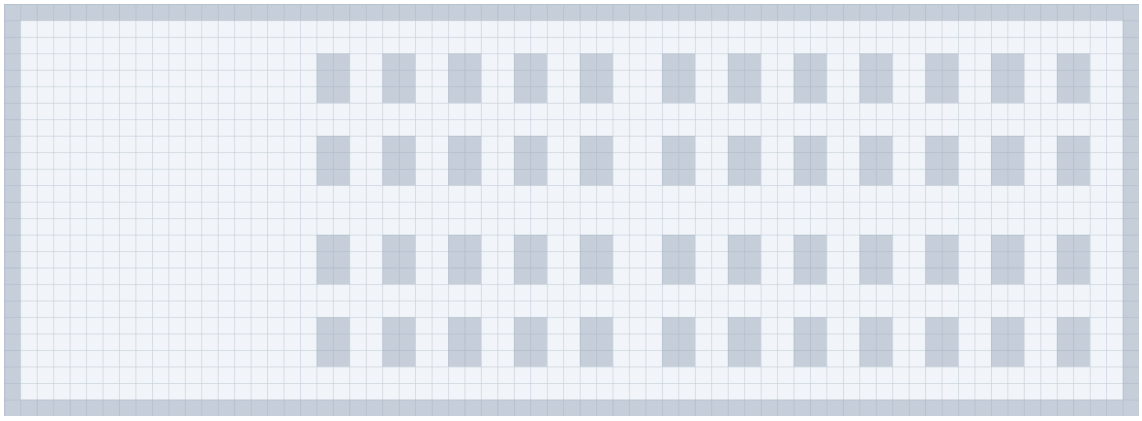


Figure 6.8: Large-scale factory environment used in the warehouse operation scenarios.

Three operation scenarios were defined in this environment, each involving 180 robots and representing a different stage of a large-scale warehouse operation. These scenarios include the dispatch of robots from clustered docking areas, movements between warehouse regions, and the return of robots to their original clusters. An additional video illustrating the execution of the three large-scale scenarios is provided in Appendix A.1, which complements the scenario descriptions and the experimental results reported in this subsection.

6.2.5.1 Scenario 1: Docking Areas to Warehouse Regions

The first large-scale scenario is shown in Figure 6.9. In this case, the robots start from compact clusters located on the left side of the factory layout, representing docking or parking areas, and are assigned goals distributed across the warehouse regions. This configuration creates a demanding planning instance, since many robots start close to each other and must enter a structured roadmap composed of narrow passages between warehouse blocks.

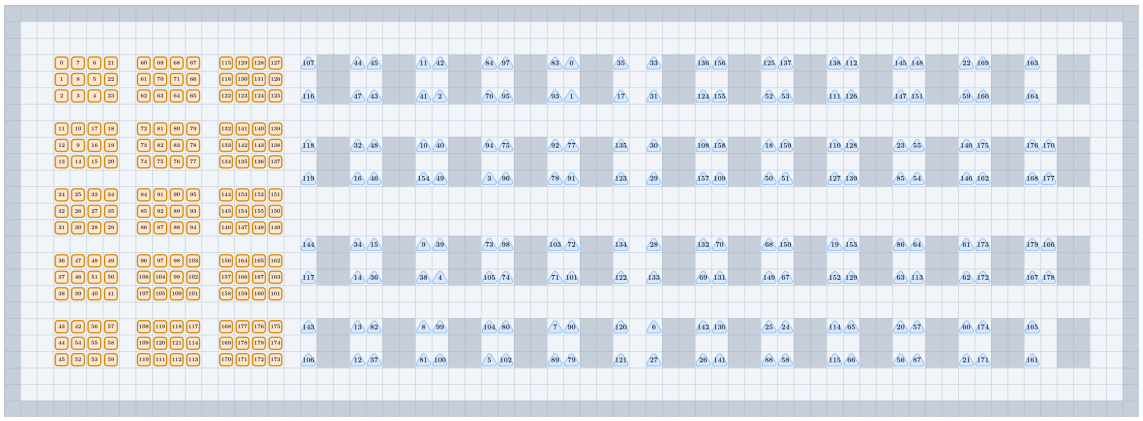


Figure 6.9: Large-Scale Scenario 1, where robots depart from docking areas toward warehouse regions.

Table 6.13 compares the baseline TEA* behavior with the best heuristic result obtained for this scenario. The baseline, represented by 100 random priority permutations, failed to produce a complete solution in every execution. Although it planned an average of 172.9 robots, none of the generated permutations was sufficient to plan the full set of 180 robots.

Table 6.13: Comparison between baseline TEA* and heuristic TEA* for Large-Scale Scenario 1.

Method	**Robots**	**Mean robot completion**	**Valid solutions**	**Excess cost (s)**	**Perm. time of total (%)**	**Best heuristic**
Baseline TEA*	180	172.9/180	0/100	∞	–	–
Heuristic TEA*	**180**	**180/180**	**1/1**	**19944.65**	**0.1**	**Low K-hop Density Positions First**

The heuristic-based version was able to compute a complete plan, with the best result obtained by *Low K-hop Density Positions First*. This is consistent with the structure of the scenario, where the main difficulty is caused by dense initial robot clusters. By prioritizing robots according to their initial spatial distribution, the planner is able to release the docking areas more effectively and avoid the incomplete plans observed with random ordering.

This scenario therefore shows a clear feasibility improvement. While the baseline TEA* approach failed in all 100 random attempts, the heuristic-based version produced a valid plan for all 180 robots.

6.2.5.2 Scenario 2: Movements Between Warehouse Regions

The second large-scale scenario is shown in Figure 6.10. In this case, the robots start from warehouse regions and are assigned new goals in other warehouse areas. Unlike the previous scenario, the robots are already distributed across the factory layout, making the main difficulty less related to the release of dense initial clusters and more associated with the interactions between paths crossing shared corridors and transition regions.

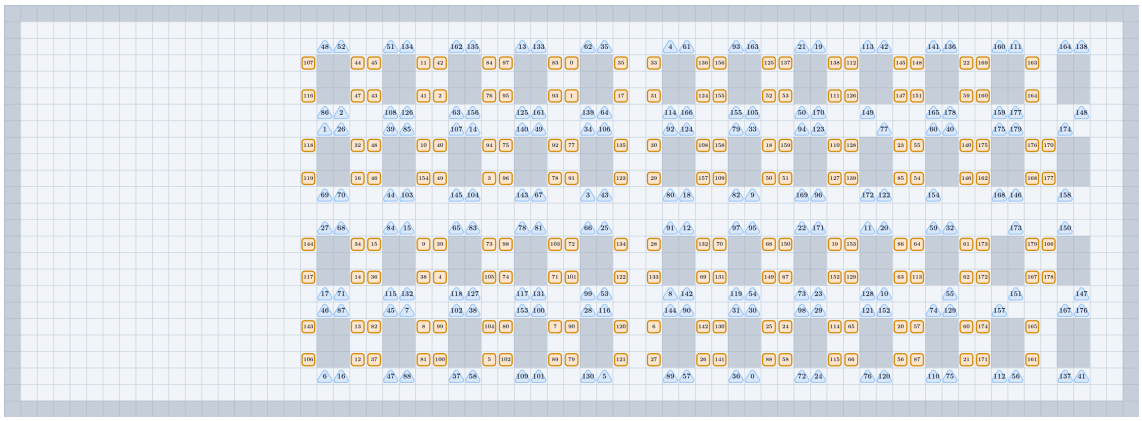


Figure 6.10: Large-Scale Scenario 2, where robots move between different warehouse regions.

Table 6.14 compares the baseline TEA* behavior with the best heuristic result obtained for this scenario. The baseline, represented by 100 random priority permutations, produced complete solutions in only 54 executions. Although the mean robot completion was 179.3/180, the remaining failed executions still resulted in unusable plans, highlighting the limitations of random priority orderings in large-scale warehouse movements.

Table 6.14: Comparison between baseline TEA* and heuristic TEA* for Large-Scale Scenario 2.

Method	Robots	Mean robot completion	Valid solutions	Excess cost (s)	Cost improv. (%)	Perm. time of total (%)	Best heuristic
Baseline TEA*	180	179.3/180	54/100	19456.72	–	–	–
Heuristic TEA*	**180**	**180/180**	**1/1**	**17354.28**	**+10.8**	**1.3**	**Less Conflicts First**

In this scenario, the heuristic-based version improved both reliability and solution quality. While the baseline failed in 46% of the random permutations, the best heuristic ordering produced a complete plan for all 180 robots. More importantly, *Less Conflicts First* reduced the total cost from the baseline mean of 19456.72 s to 17354.28 s, corresponding to a 10.8% improvement over the successful baseline executions.

This result indicates that, for movements between warehouse regions, the main benefit of using heuristics is not only obtaining a valid plan, but also reducing the final coordination cost. In this case, planning robots involved in fewer conflicts first appears to be advantageous, since simpler movements can be resolved before shared regions become heavily constrained by later reservations.

6.2.5.3 Scenario 3: Warehouse Regions to Docking Areas

The third large-scale scenario is shown in Figure 6.11. This scenario represents the return operation, where robots leave the warehouse regions and move back to the clustered docking areas on the left side of the factory layout. In contrast with the first scenario, the dense spatial configuration appears at the goal positions rather than at the initial positions, which makes the final stage of the planned paths particularly critical.

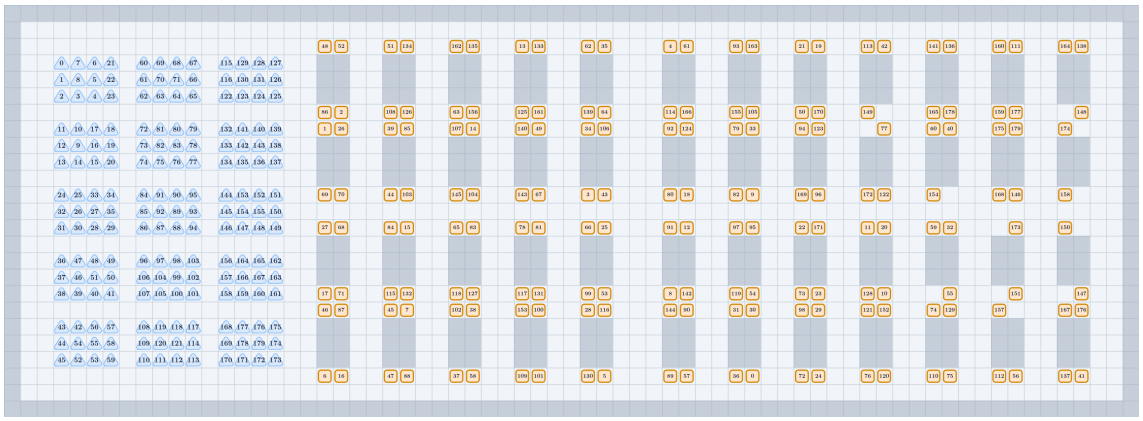


Figure 6.11: Large-Scale Scenario 3, where robots return from warehouse regions to clustered docking areas.

The baseline results in Table 6.15 show that the original random-ordering approach was again unable to produce any complete solution over 100 repetitions. Although the mean robot completion was 175.3/180, every random execution still failed to plan the full set of robots. This confirms that, when the goal region becomes highly congested, random priority assignments are not sufficient to resolve the complete large-scale planning instance.

Table 6.15: Comparison between baseline TEA* and heuristic TEA* for Large-Scale Scenario 3.

Method	Robots	Mean robot completion	Valid solutions	Excess cost (s)	Perm. time of total (%)	Best heuristic
Baseline TEA*	180	175.3/180	0/100	∞	–	–
Heuristic TEA*	**180**	**180/180**	**1/1**	**11371.94**	**1.2**	**Longest Path Last**

The heuristic-based version was able to compute a complete solution, with the best result obtained by *Longest Path Last*. This heuristic planned all 180 robots with an excess cost of 11371.94 s, while the baseline failed in every random repetition. Therefore, the main improvement in this scenario is again related to feasibility, since the heuristic ordering allowed TEA* to solve an instance where the random-ordering strategy consistently produced incomplete plans.

However, because this scenario contains a dense goal region, it is also relevant to compare this result with the heuristics from the *Surroundings (goal)* group. As shown in Table 6.16, these heuristics were also able to compute complete solutions for all 180 robots, but with substantially higher excess costs than *Longest Path Last*.

Table 6.16: Performance of goal-density heuristics in Large-Scale Scenario 3.

Heuristic	Robot completion	Excess cost (s)	Perm. time of total (%)
High Cluster Central Goal First	180/180	34899.40	0.01
High K-hop Density Goals First	**180/180**	**29958.06**	**0.05**
High Radius Density Goals First	**180/180**	**29958.06**	**0.44**

6.2.6 Comparative Analysis of Heuristic Performance

The results obtained in the controlled scenarios show that no single heuristic is consistently preferable across all planning conditions, since the best priority ordering depends on the structure of each planning instance, including spatial distribution, path interactions, and roadmap topology. The *Surroundings (pos)* heuristics were more relevant when the main difficulty was created by dense initial robot configurations, while the *Surroundings (goal)* heuristics were better aligned with scenarios where the goal positions were concentrated in a common region. Similarly, *Conflict* heuristics were more effective when the solution cost was dominated by interactions between robot paths, whereas *Topological* heuristics were more suitable for scenarios involving narrow corridors or low-connectivity regions.

The *Exploration Conflicts Pressure* heuristic provides a complementary corrective mechanism based on feedback from incomplete TEA* executions. In the Initial Position Density Scenario and in the Topological Constraint Scenario, this feedback was sufficient to recover complete solutions after other heuristics failed, showing that reservation-based rejections can help identify robots that were negatively affected by previous orderings. However, the heuristic did not produce the best solution cost in these cases, indicating that its main contribution is improving feasibility rather than optimizing the final plan. In the Goal Density Scenario, the same mechanism was not sufficient to recover a complete solution, suggesting that the accumulated rejection statistics do not always capture the critical ordering relation required when the main difficulty is created by a dense goal region.

The large-scale factory operation scenarios reinforce this observation under more demanding planning conditions. In Scenario 1 and Scenario 3, the random baseline failed to produce any complete solution over 100 random priority permutations, while the heuristic-based version was able to plan all 180 robots. In Scenario 2, the baseline obtained complete solutions in only 54 out of 100 repetitions, and the best heuristic reduced the excess cost from 19456.72 s to 17354.28 s, corresponding to a 10.8% improvement. These results indicate that heuristic-based prioritization improved both the probability of obtaining a complete solution and the final plan quality in the evaluated cases where valid solutions were achievable by random orderings.

The cost of computing the priority permutation has different relevance depending on the scale of the planning instance. In the smaller controlled scenarios, this percentage can be relatively high because TEA* itself is often very fast, making the heuristic computation time more visible in the total planning time. This can be observed, for example, in Tables 6.4 and 6.10, where the permutation time reaches 43.7% of the total planning time for *Low Average DoF Path First*. However, this is less problematic in small instances because the absolute computation times remain very low. In the large-scale scenarios, the situation changes because the TEA* planning time grows much more significantly with the number of robots than the permutation computation time, making the latter a very small fraction of the total execution. This is shown in Tables 6.13, 6.14, and 6.15, where the selected heuristics account for only 0.1%, 1.3%, and 1.2% of the total planning time, respectively.

The topological heuristics were typically among the most expensive to compute, mainly because they require DoF information along the candidate paths. However, this overhead may be reduced by precomputing a lookup table with the DoF value of each roadmap vertex, so that these values only need to be retrieved during permutation generation.

Although the *Roadmap distance* heuristics were not the main focus of any controlled scenario, they still obtained competitive results in some cases. This suggests that simple distance-based priority rules can be useful when path length is correlated with the difficulty or impact of a robot movement. However, because they do not directly model spatial density, expected conflicts, or topological constraints, their behavior is more dependent on the particular geometry of each instance.

Overall, the evaluation supports the use of multiple prioritization strategies instead of relying on a single fixed heuristic. Each heuristic captures a different aspect of the planning problem, and its usefulness depends on the structure of the scenario. This motivates the heuristic usage mechanisms introduced in the following chapter, where different priority orderings are explored within the same planning framework.

Chapter 7

Parallelization and Heuristic Dispatching Framework

The results presented in the previous chapter showed that no single prioritization heuristic dominates across all tested scenarios. Instead, the most effective priority ordering depends on the structure of each planning instance. This motivates the parallel evaluation of multiple heuristic-guided orderings, with the goal of increasing the likelihood of finding better solutions within the available planning time.

This chapter builds on a previous multi-threaded TEA* prototype, which explored multiple robot priority permutations in parallel using OpenMP [66]. In this dissertation, that idea is reorganized into a more structured parallel planning framework, where parallel execution is encapsulated in a dedicated planner class and combined with the heuristic ordering methods developed in Chapter 5. Instead of selecting robot priority permutations through lexicographic sampling, the proposed approach dispatches heuristic-generated priority orderings to independent TEA* instances, collects the resulting candidate plans, and returns the best valid plan to the Supervisor.

The chapter is organized as follows. Section 7.1 introduces the overall parallel planning approach, and Section 7.2 details the worker thread implementation and its synchronization and memory trade-offs. The heuristic dispatching mechanism is presented in Section 7.3, followed by the timeout and controlled termination logic in Section 7.4. Finally, Section 7.5 discusses the handling of redundant heuristics through runtime-based selection.

7.1 Parallel Planning Approach

The proposed architecture extends the TEA* planner by formalizing the multi-threaded planning idea previously explored in [66]. In that initial implementation, parallelism was used to evaluate multiple robot-solving orders in parallel, thereby reducing the dependence of TEA* on a single priority permutation. The framework developed in this dissertation preserves this principle, but replaces the previous lexicographic permutation selection strategy with heuristic-guided priority orderings and integrates the execution into a dedicated parallel planner structure.

As illustrated in Figure 7.1, the Supervisor sends a planning request to the Path Planner containing the current robot positions, their corresponding goals, and the maximum time available for planning, denoted by $I = \{(p_i, g_i)\}_{i=1}^{N}, T_{\max}$.

After receiving the request, the Path Planner initializes the execution and launches a set of worker threads. Each worker retrieves a heuristic from a shared heuristic pool, uses it to compute a priority permutation, and then runs TEA* independently using that ordering. Each execution therefore produces a candidate plan associated with one heuristic-generated priority ordering.

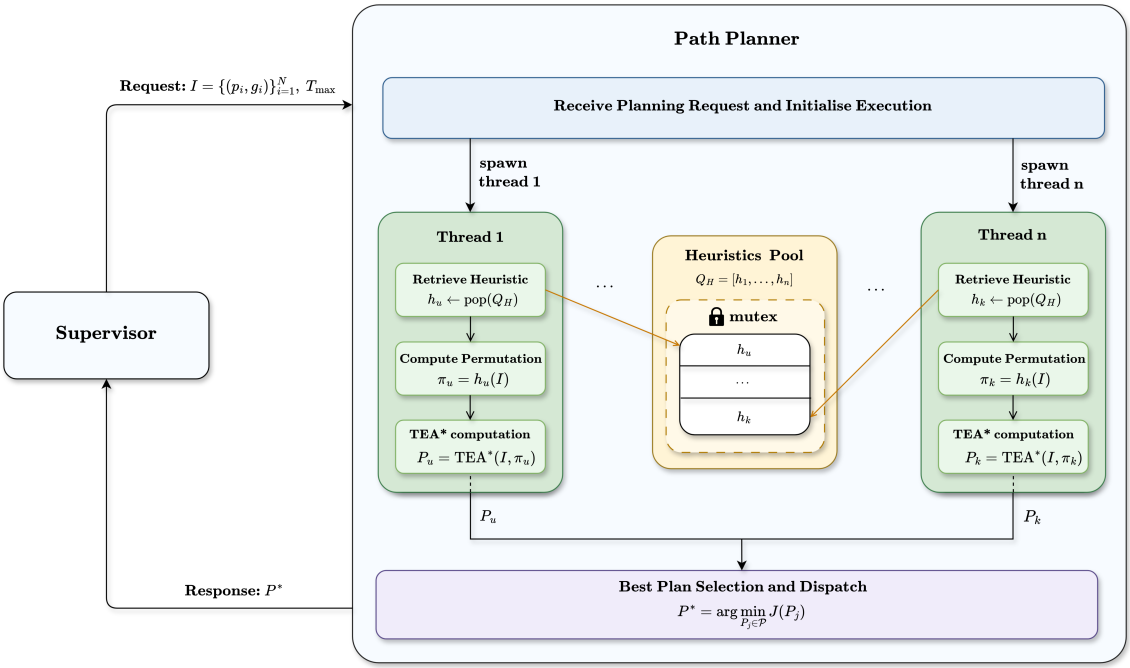


Figure 7.1: Overview of the proposed parallel planning framework.

For simplicity, Figure 7.1 represents the case in which the number of available heuristics is equal to the number of worker threads. In that simplified case, each thread retrieves one heuristic and produces one candidate plan. In the general case, however, these numbers do not need to be equal, as a thread may evaluate several heuristics during the same planning cycle while planning time is still available and heuristics remain to be tested.

During the parallel execution stage, each worker thread generates one or more candidate plans, which are collected in a set P and subsequently evaluated using the selected cost function J. Once all candidate plans have been evaluated, the plan with the lowest cost is returned to the Supervisor. In this work, $J(P_i)$ is defined as the sum of the individual costs of all robot paths in candidate plan P_i. Candidate plans that do not include valid paths for all robots are assigned an ∞ cost, as a plan is considered to have failed whenever at least one robot is unable to obtain a valid path.

The following sections describe the main components of the proposed framework in more detail, including the worker thread implementation, the heuristic dispatching strategy, the timeout and controlled termination mechanism, and the handling of redundant heuristics.

7.2 Worker Thread Implementation Details

The implementation of the parallel framework was based on the multi-threaded TEA* prototype presented in [66], where OpenMP was used to evaluate multiple robot orderings in parallel. In that initial implementation, however, the parallel execution logic was closely integrated into the Path Planner code. This made the approach suitable as a proof of concept, but less modular for the integration of the heuristic framework developed in this dissertation.

The redesigned implementation exploits the Object-Oriented Programming (OOP) structure of the existing planner. Since TEA* was already implemented as a class with its own internal planning state and methods, the parallel execution logic was reorganized around a dedicated TEA* Parallel Planner class, illustrated in Figure 7.2. This class encapsulates the parallel coordination logic and manages vectors of independent TEA* and A* instances. The TEA* instances are used by the worker threads to evaluate different priority orderings, while the A* instances support heuristics that require single-robot shortest-path computations. This structure keeps the original sequential behavior of TEA* inside each worker, while separating the parallel execution logic from the standard Path Planner implementation.

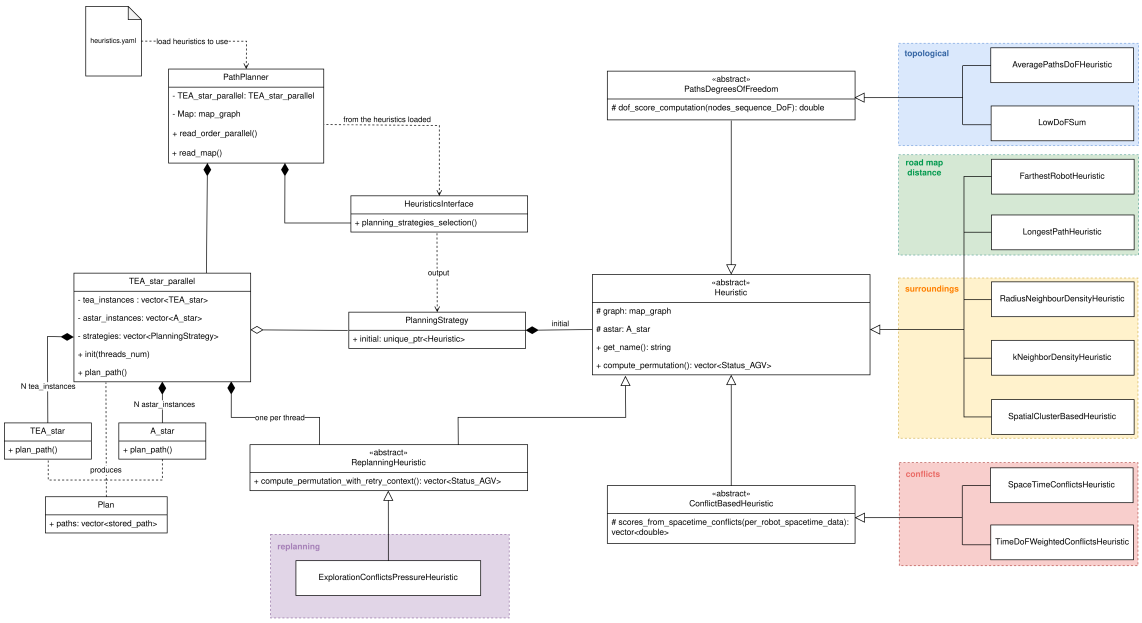


Figure 7.2: UML Class Diagram representing the implemented parallel planning framework and its main components.

Different thread-based parallelization options may be considered for this type of implementation, including *OpenMP*, *Intel TBB*, and the C++ standard threading library. These approaches rely on kernel-level threads that can be scheduled on multiple processor cores, but they expose different programming models. *Intel TBB* provides flexible task-based parallelism, while the C++ standard threading library offers explicit thread management through *std::thread*.

Since the previous prototype already relied on *OpenMP*, this choice was re-evaluated before developing the reorganized framework. The workload considered in this dissertation remains

naturally expressed as a set of independent worker executions, each evaluating one priority ordering at a time. Therefore, the additional control provided by alternatives such as *std::thread* or *Intel TBB* was not necessary for the implemented design. For this reason, *OpenMP* was adopted in the redesigned framework, as it provides a simple and sufficiently expressive interface for launching the main worker threads while avoiding explicit management of thread creation, joining, and scheduling.

During each planning cycle, the worker executions are independent but not equivalent. As each worker evaluates a different priority ordering, each TEA* instance may produce a different sequence of reservations, node expansions, and path rejections. To avoid interference between parallel executions, the mutable internal state of TEA* is kept private to each planner instance. This includes structures such as the 3D costmap, where the space-time reservations of the planned robot paths are stored, as well as the lists used during the search.

This design reduces the need for synchronization during the TEA* search itself. Sharing these structures between threads would require frequent mutex-protected accesses in some of the most computationally intensive parts of the algorithm, increasing implementation complexity. More importantly, workers would repeatedly block each other while accessing shared planning state, which may largely offset the performance benefit of executing multiple TEA* instances in parallel. However, the main drawback is the additional memory usage, as each TEA* instance maintains its own 3D costmap, which can be large because it consists of T roadmap representations, where T is the maximum number of planning steps considered. Therefore, using multiple worker threads implies storing this structure multiple times, making the memory cost proportional to the number of TEA* instances. This overhead can become significant for large maps or large values of T.

For this reason, only data that remains constant during planning is shared between instances. For example, the precomputed map cost lookup table is shared, as it is read-only and can be safely accessed by multiple threads without synchronization. This provides a compromise between thread isolation, which avoids frequent synchronization during the search, and memory efficiency, by sharing only the data that can be safely reused across workers.

Overall, the implementation combines the simplicity of *OpenMP* for launching the main worker threads with explicit synchronization mechanisms only for the few shared structures that require controlled access. This design keeps the core TEA* algorithm unchanged, while enabling several priority orderings to be evaluated concurrently through independent planner instances.

7.3 Heuristic Dispatching Mechanism

The previous multi-threaded TEA* prototype showed that evaluating multiple robot priority orderings in parallel can reduce the dependence of TEA* on a single ordering [66]. Building on that idea, the framework developed in this dissertation uses the prioritization heuristics introduced in Chapter 5 to generate more informed orderings. The dispatching mechanism was therefore designed to assign these heuristic-generated orderings to worker threads while avoiding the simultaneous evaluation of highly redundant heuristics.

As the heuristic set is organized into different families, the objective is to promote a balanced use of these families during parallel execution. This increases the diversity of the generated priority orderings and reduces the probability that several threads evaluate very similar planning alternatives. The overall dispatching flow executed by each worker thread is illustrated in Figure 7.3.

To achieve this, the available heuristics are stored in a shared priority queue. Each heuristic is associated with a heuristic group, and each group has a counter indicating how many heuristics from that group have already been selected during the current planning cycle. The priority queue is ordered according to these group counters, so that heuristics belonging to the least-used groups are given higher priority.

The set of heuristics inserted into the pool is defined through the *heuristics.yaml* file, represented in Figure 7.2. This configuration file specifies which heuristics are available during the planning cycle and associates each one with a heuristic group. Within each group, the relative order of the heuristics follows the order defined in the YAML file. The dynamic component of the dispatcher is therefore applied mainly at the group level, since heuristics belonging to the same group tend to use similar criteria to generate priority orderings.

Whenever a worker thread needs a new heuristic, it accesses the shared queue inside a critical section protected by a mutex and pops a heuristic from the group with the lowest usage counter. The selected heuristic is removed from the queue during this operation, preventing two threads from evaluating the exact same heuristic during the same planning cycle. After the heuristic is selected, the counter of the corresponding group is incremented, and the queue is reordered accordingly.

This mechanism encourages the simultaneous evaluation of heuristics from different families. Instead of assigning several heuristics from the same family to different threads at the same time, the dispatcher tends to distribute the available computational resources across the available strategies.

As depicted in Figure 7.3, if the shared heuristic queue becomes empty, the worker can no longer retrieve a heuristic from the pool. In this case, if that worker has not yet obtained a valid candidate plan, an additional attempt is performed using the *Exploration Conflicts Pressure* replanning heuristic. The statistics required by this heuristic are accumulated locally by each worker across its previous planning attempts, namely the number of reservation-based rejections and the number of expanded nodes already mentioned in Subsection 5.7.1. This heuristic is only used after all regular heuristic attempts of that worker have failed, as it tends to reverse the priority order in an informed way.

Although a predefined static ordering with manually alternated heuristic families may provide a similar behavior in the current implementation, the main advantage of the implemented mechanism is the flexibility it provides for extending the scheduling policy. In future extensions, the dispatcher may prioritize heuristics according to the remaining planning time or even the quality of candidate plans obtained by previous orderings.

After leaving the critical section, the worker thread evaluates the selected heuristic by generating the corresponding priority ordering and computing the associated TEA* paths. If the regular heuristic queue is empty, the worker either returns the best available plan, when it has already found a valid one, or performs an additional attempt with the *Exploration Conflicts Pressure* heuristic.

The worker only repeats this procedure after checking whether the current planning cycle is still active, according to the timeout and termination logic described in the following section.

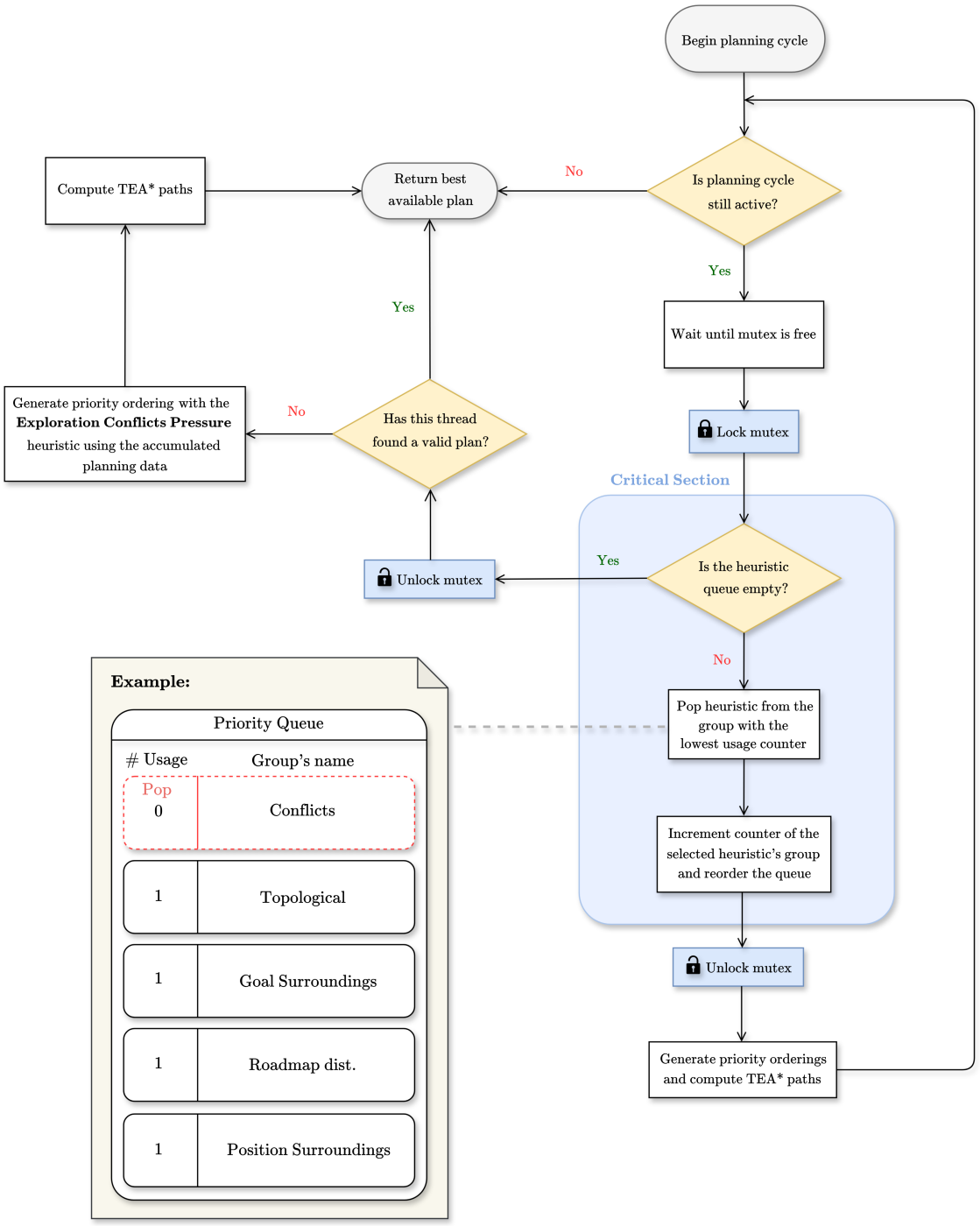


Figure 7.3: Heuristic dispatching flow executed by each worker thread.

7.4 Timeout Handling and Controlled Termination

The parallel planner was designed to operate under a maximum planning time $T_{\max}$, which represents the time available to compute and return a valid plan after a planning request is received. In the intended system behavior, this value is provided by the Supervisor and may be computed dynamically in future versions according to the current execution state of the robots. For instance, the Supervisor may estimate $T_{\max}$ from the remaining time before the most urgent robot reaches its next relevant node. On the Path Planner side, the timeout mechanism is already supported, although in the current implementation $T_{\max}$ is provided as a static parameter.

Immediately before launching the worker threads, a global timer is initialized. To monitor the elapsed planning time, an additional lightweight monitoring thread is launched alongside the worker threads. During the planning cycle, this thread periodically compares the elapsed time with $T_{\max}$, sleeping between consecutive checks in order to reduce computational overhead and avoid unnecessary CPU usage. Once the elapsed time exceeds the maximum planning time, the monitoring thread sets a shared atomic boolean flag to indicate that the timeout has been reached.

However, the timeout does not immediately force all workers to stop. Returning no plan is undesirable when no valid solution has yet been found, so the termination condition also depends on the existence of at least one complete candidate plan. For this reason, another shared atomic flag is used to indicate whether a valid plan has already been obtained. The first worker thread that finds a complete plan, containing valid paths for all robots, sets this flag.

The planning cycle is therefore considered inactive only when both of the following conditions are satisfied:

- The maximum planning time $T_{\max}$ has been reached;
- At least one valid candidate plan is available.

To allow running TEA* instances to stop in a controlled manner, the termination condition is checked at the beginning of each TEA* step. This point was chosen because the step operation is the most frequently executed operation throughout the search process, making it a natural location to test whether the current planning attempt should continue. When the termination condition becomes true, each worker detects it at the beginning of a subsequent step and interrupts its current planning process safely, allowing local variables and internal planner state to be closed consistently before the thread returns.

This mechanism allows the planner to use the available time to continue searching for better candidate plans after the first valid solution is found, while still ensuring that the computation can be interrupted once the time budget has expired. As a result, the framework does not stop at the first feasible plan, but instead uses the remaining time to improve the final selection whenever possible.

7.5 Handling Redundant Heuristics

The results presented in the previous chapter showed that *Radius Neighbor Density* and *K-hop Neighbor Density* often produce equivalent or very similar priority orderings. Running both heuristics may therefore lead to redundant TEA* executions. For this reason, the selection between them was based on one of their main differentiating factors, the estimated computation time. This criterion is directly motivated by their different computational behaviors, described in Equations 5.18 and 5.28.

For this purpose, linear regression models were fitted for both heuristics. The theoretical complexity analysis had already identified the main runtime trends expected in each case, making more complex modeling approaches unnecessary. The quality of the fitted models was evaluated using the regression metrics defined in Appendix B, namely the coefficient of determination R^2 and the Mean Absolute Error (MAE).

7.5.1 Runtime Estimation for Radius Neighbor Density

To estimate the computation time of the *Radius Neighbor Density* heuristic, an experimental dataset was collected by measuring its runtime for different numbers of robots. The tested values were $N = \{1, 5, 10, 20, 50, 100, 150, 200, 300, 400\}$, and, for each value of N, the heuristic was executed 50 times. The measured execution times were then used to fit different regression models.

The choice of candidate models was guided by Equation 5.18. Accordingly, different regression models were tested using combinations of N, N^2, and $N \log N$. Table 7.1 summarizes the obtained results.

Table 7.1: Regression models tested for estimating the computation time of the *Radius Neighbor Density* heuristic.

Model	θ_0^{rad}	θ_1^{rad}	θ_2^{rad}	$\mathbf{R^2}$	**MAE (ms)**
$T_{\text{radius}}(N) = \theta_0^{\text{rad}} + \theta_1^{\text{rad}} N$	-6.225741	0.155011944	–	0.913134	6.518654
$T_{\text{radius}}(N) = \theta_0^{\text{rad}} + \theta_1^{\text{rad}} N^2$	0.487353	0.000339267	–	0.990712	1.297389
$T_{\text{radius}}(N) = \theta_0^{\text{rad}} + \theta_1^{\text{rad}} N + \theta_2^{\text{rad}} N^2$	**0.413287**	**0.001414811**	**0.000336415**	**0.990718**	**1.282169**
$T_{\text{radius}}(N) = \theta_0^{\text{rad}} + \theta_1^{\text{rad}} N \log N + \theta_2^{\text{rad}} N^2$	0.467381	0.000089134	0.000338135	0.990713	1.293423

As expected, the models containing the N^2 term achieved the best results, clearly outperforming the model based only on N. This behavior is consistent with the complexity expression derived in Equation 5.18, where the dominant term is associated with the quadratic growth of the number of robot comparisons.

Among the models containing a quadratic term, the best result was obtained by combining N and N^2, leading to the following runtime model:

$$T_{\text{radius}}(N) = \theta_0^{\text{rad}} + \theta_1^{\text{rad}} N + \theta_2^{\text{rad}} N^2. \tag{7.1}$$

Although the complexity expression in Equation 5.18 also includes an $N \log N$ component, the difference between the $N + N^2$ and $N^2 + N \log N$ models was almost negligible. This is consistent with the fact that the dominant runtime behavior is captured by the N^2 term, while the remaining contribution can be sufficiently approximated by the linear term over the tested range of robot numbers.

Figure 7.4 shows the fitted curve obtained with the selected model, which captures the non-linear increase in computation time as the number of robots grows, consistently with the quadratic dominant term identified for this heuristic.

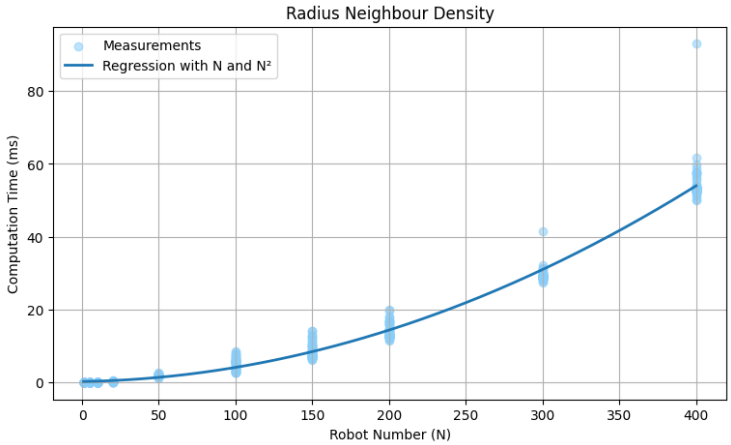


Figure 7.4: Regression model fitted to the computation time of the *Radius Neighbor Density* heuristic.

7.5.2 Runtime Estimation for K-hop Neighbor Density

For the *K-hop Neighbor Density* heuristic, the runtime estimation had to account for the expansion process performed when computing the topological neighborhood of each robot. In this case, the computation time is mainly affected by the total number of node expansions performed during the K-hop searches.

The total number of expansions performed during one execution of this heuristic is mainly affected by the average DoF of the roadmap vertices, the neighborhood depth k_{hops}, and the number of robots N. Varying the roadmap degree would require creating new maps or modifying existing ones only for this analysis, so this parameter was kept fixed. The dataset was therefore generated by varying k_{hops} and N, using $k_{\text{hops}} = \{1,2,3\}$ and $N = \{10,25,50,100,150,200\}$. For each combination, the heuristic was executed 20 times, and the corresponding computation times were recorded. The tested regression models considered the measured total number of node expansions, denoted by E, together with terms depending on the number of robots. Table 7.2 summarizes the obtained results.

Table 7.2: Regression models tested for estimating the computation time of the *K-hop Neighbor Density* heuristic.

Model	θ_0^{khop}	θ_1^{khop}	θ_2^{khop}	$\mathbf{R^2}$	**MAE (ms)**
$T_{\text{khop}}(N,E) = \theta_0^{\text{khop}} + \theta_1^{\text{khop}} E$	0.814326	0.002516826	–	0.851778	1.176020
$T_{\text{khop}}(N,E) = \theta_0^{\text{khop}} + \theta_1^{\text{khop}} N + \theta_2^{\text{khop}} E$	1.432990	-0.013862	0.002977	0.880253	1.076947
$T_{\text{khop}}(N,E) = \theta_0^{\text{khop}} + \theta_1^{\text{khop}} N \log N + \theta_2^{\text{khop}} E$	**1.310290**	**-0.002688199**	**0.003005392**	**0.884116**	**1.058590**

The best-fitting model for the *K-hop Neighbor Density* heuristic combines the $N \log N$ term with the total number of node expansions E:

$$T_{\text{khop}}(N,E) = \theta_0^{\text{khop}} + \theta_1^{\text{khop}} N \log N + \theta_2^{\text{khop}} E. \tag{7.2}$$

This result is consistent with Equation 5.28, as the expansion term E captures the cost of computing the k-hop neighborhoods, while the $N \log N$ term models the remaining operations dependent on the number of robots, including the sorting step. This explains why the model using both terms outperformed the alternative based only on E.

Figure 7.5 shows the fitted regression plane obtained with the selected model, using a three-dimensional representation due to the dependence of the estimated computation time on two input variables, $N \log N$ and E.

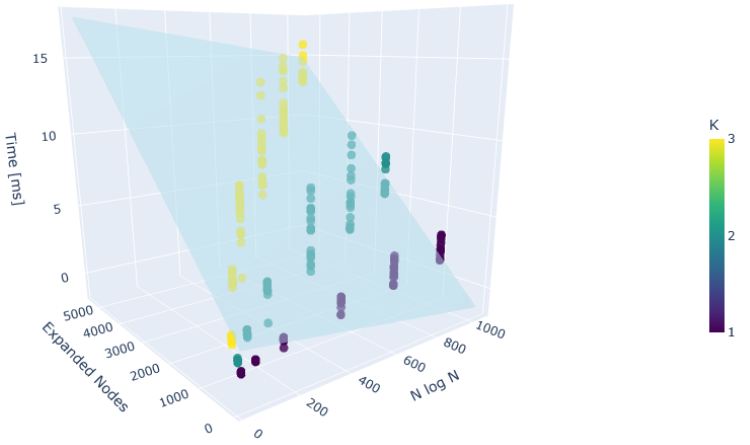


Figure 7.5: Regression model fitted to the computation time of the *K-hop Neighbor Density* heuristic.

To use this model before executing the heuristic, the total number of node expansions E must be estimated from the parameters of the current planning instance. In practice, the neighborhood depth K is fixed for a given run, and high values of K are not typically useful in this context, as they rapidly expand the considered neighborhood and may reduce the locality of the density estimate.

For this reason, the expansion effort was approximated from the number of vertices reachable within k_{hops} hops, previously defined in Equation 5.21. Assuming an average roadmap degree of $DoF = 4$, this provides a practical estimate of the number of vertices explored around each robot.

Although the recursive implementation may expand some vertices more than once, this becomes unlikely for the small values of k_{hops} considered here, making the approximation sufficient for runtime prediction.

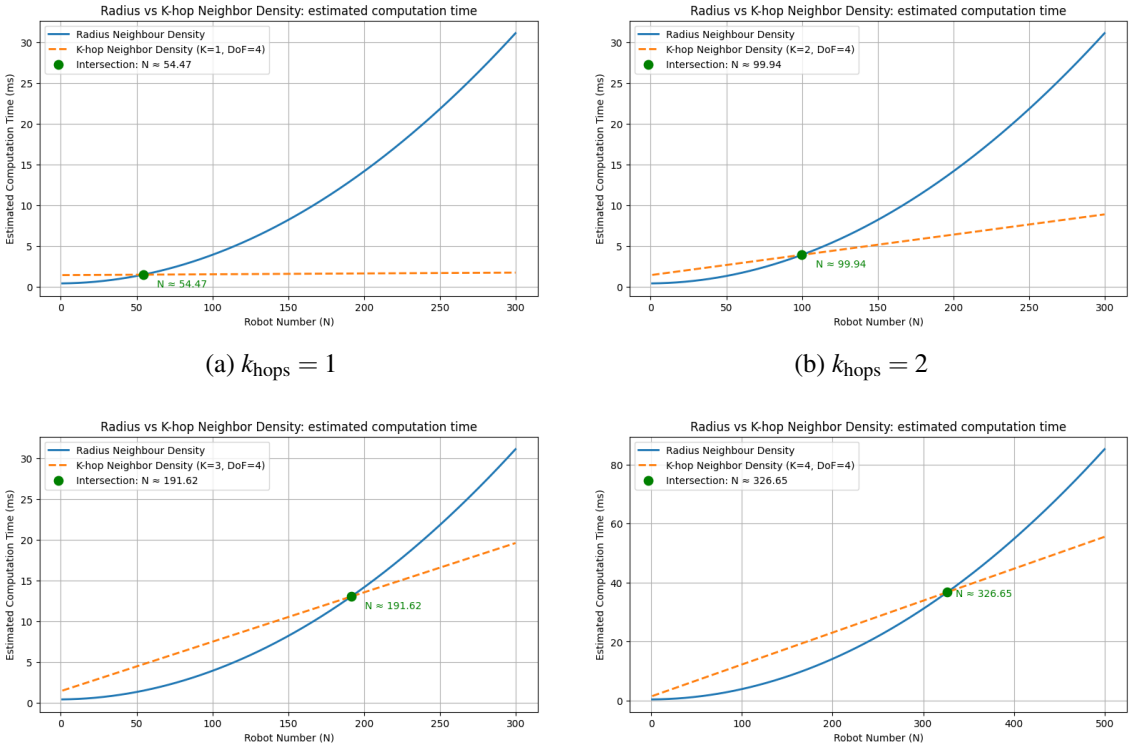
As the K-hop expansion is performed for each robot, the total expansion estimate used by the runtime model is given by

$$\hat{E} = N \cdot N_K = N \cdot \left(1 + 2k_{\text{hops}}(k_{\text{hops}} + 1)\right), \quad DoF = 4. \tag{7.3}$$

This estimated value $\hat{E}$ is then provided to Equation 7.2, allowing the computation time of the *K-hop Neighbor Density* heuristic to be predicted before executing it.

7.5.3 Estimated Runtime Comparison

After fitting both runtime models, their estimates were compared to select the faster heuristic for each planning instance. Although the *K-hop Neighbor Density* model also depends on $\hat{E}$, this term becomes a function of N once DoF and k_{hops} are fixed, as shown in Equation 7.3. Therefore, both models can be evaluated as functions of N and compared directly. Figures 7.6a, 7.6b, 7.6c, and 7.6d show the estimated runtime comparison for different values of k_{hops}, assuming $DoF = 4$. The intersection between the two curves indicates the approximate robot count at which both heuristics are expected to have the same computation time.



(a) $k_{\text{hops}} = 1$ (b) $k_{\text{hops}} = 2$

(c) $k_{\text{hops}} = 3$ (d) $k_{\text{hops}} = 4$

Figure 7.6: Estimated computation time comparison between *Radius Neighbor Density* and *K-hop Neighbor Density*, considering different values of k_{hops} and DoF $= 4$.

The intersections occur approximately at $N = 54.47$, $N = 99.94$, $N = 191.62$, and $N = 326.65$ for $k_{\text{hops}} = 1, 2, 3$, and 4, respectively. This increasing threshold is consistent with the fact that larger k_{hops} values require more neighborhood expansions, delaying the point at which the *K-hop Neighbor Density* heuristic becomes more efficient than the radius-based alternative.

Thus, for each fixed value of k_{hops}, the corresponding intersection defines the selection threshold used by the planner. Below this threshold, the *Radius Neighbor Density* heuristic is expected to be faster, while above it the *K-hop Neighbor Density* heuristic becomes preferable due to the quadratic growth of the radius-based model. Therefore, before launching the worker threads, both fitted regression models are evaluated for the current planning instance, and only the heuristic with the lowest estimated computation time is inserted into the heuristic pool.

7.6 Summary

This chapter presented the parallel planning framework developed around TEA*. The proposed framework builds on a previous multi-threaded TEA* prototype, where parallel execution was used to explore multiple robot priority permutations [66]. In this dissertation, that idea was redesigned into a dedicated TEA* Parallel Planner class and extended with the heuristic-based ordering methods developed in Chapter 5. Each worker thread runs an independent TEA* instance, producing candidate plans that are later compared according to the selected cost function.

OpenMP was adopted in the redesigned framework to launch the main worker threads, while synchronization is limited to shared structures such as the heuristic pool. The mutable state of each TEA* instance remains private to its worker, avoiding frequent locking during the most computationally intensive parts of the planning process. This increases memory usage, since each TEA* instance maintains its own mutable planning structures, including the 3D costmap used to store space-time reservations. Nevertheless, this trade-off was considered acceptable because it reduces contention between threads and helps preserve the benefit of evaluating multiple priority orderings concurrently.

The chapter also detailed the heuristic dispatching strategy used to assign priority orderings to the worker threads. The proposed mechanism relies on a shared priority queue organized by heuristic groups, promoting a balanced use of the available heuristic families and reducing the probability of evaluating redundant orderings in parallel. The *Exploration Conflicts Pressure* heuristic was also integrated as a fallback strategy when a worker exhausts the regular heuristic pool without finding a valid plan.

The timeout and controlled termination mechanism was then described, showing how the planner uses the available planning time without stopping immediately after the first valid solution is found. Finally, a regression-based selection mechanism was introduced to handle redundancy between the *Radius Neighbor Density* and *K-hop Neighbor Density* heuristics. Since these heuristics often produce equivalent or very similar priority orderings, only the alternative with the lowest estimated computation time is inserted into the heuristic pool.

Chapter 8

Performance Evaluation of the Parallel Planning Framework

This chapter evaluates the performance of the proposed parallel planning framework on the three large-scale scenarios with 180 robots introduced in Chapter 6. The objective is to assess how the combination of heuristic priority orderings and concurrent TEA* executions affects the response time and solution quality of the planning process when compared with the original sequential TEA* planner using random priority permutations.

Section 8.1 describes the experimental setup, baseline, thread configurations, and evaluation metrics. Section 8.2 presents the results for the three large-scale scenarios, while Section 8.3 discusses the main trends observed across scenarios.

8.1 Experimental Protocol

The experimental protocol compares the sequential random-ordering baseline with the proposed parallel framework. All experiments were executed using the hardware and software configuration reported in Table 6.1, and the heuristic parameters were kept equal to those used in the heuristic evaluation experiments, summarized in Table 6.2. This ensures that the observed differences result from the planning strategy and from the degree of parallelism, rather than from changes in the experimental configuration or heuristic parameter tuning.

The sequential baseline evaluates random robot priority orderings with TEA* until either a complete valid solution is found or the execution limit $T_{\max} = 15\,\mathrm{s}$ is reached. This value was chosen as a practical upper bound, since even a 15 s planning interval is already long for an online multi-robot coordination system. In this baseline, the timeout limits the start of new planner executions, rather than forcibly interrupting TEA* instances that have already started. Therefore, the reported planning time may slightly exceed $T_{\max}$ when the final TEA* execution begins before the timeout but terminates after it. For each scenario, this process is repeated over 100 independent runs and, in each successful run, the time required to obtain the first valid solution and its corresponding cost are recorded. Runs that fail to find a complete valid solution within $T_{\max}$ are counted separately and

excluded from the mean time and cost calculations, so the reported averages are computed only over successful runs.

The proposed framework was evaluated using 1, 3, 5, and 6 worker threads. The heuristic pool considered in these experiments included the goal-density heuristics *High Cluster Central Goal First* and *High K-hop Density Goals First*, the initial-position density heuristics *High Cluster Peripheral First* and *Low K-hop Density Positions First*, the roadmap-distance heuristics *Longest Path First* and *Longest Path Last*, the topological heuristics *High Average DoF Path First* and *Low Average DoF Path First*, and the conflict-based heuristics *More Conflicts First*, *Less Conflicts First*, *More Time-DoF Weighted Conflicts First*, and *Less Time-DoF Weighted Conflicts First*.

Some heuristics evaluated in Chapter 6 were not included in the pool to reduce redundant priority orderings. The *Low DoF Sum First* and *Low DoF Sum Last* heuristics were excluded because the average-DoF variants showed better results in the previous evaluation while capturing a similar topological criterion. Similarly, *Farthest Robot First* and *Farthest Robot Last* were not included because their main advantage is their very low permutation computation time. In the large-scale scenarios considered here, however, the permutation time already represents only a very small fraction of the total planning time, as shown for example in Table 6.15, where it accounts for only 1.2% of the total execution. Since *Longest Path* also considers the roadmap structure through the individual path length, it was preferred over the simpler *Farthest Robot* heuristic.

The corresponding radius-based variants, *High Radius Density Goals First* and *Low Radius Density Positions First*, were also included as candidate heuristics. However, for each Radius/K-hop pair, the planner selected only one variant at runtime according to the runtime selection mechanism described in Section 7.5. Since these experiments use $k_{\text{hops}} = 2$ and 180 robots, the comparison in Figure 7.6 indicates that the *K-hop Neighbor Density* variant is expected to be faster than the radius-based alternative above approximately 100 robots. Consequently, although both variants were available for selection, the K-hop variants were selected at runtime in all three large scenarios.

For each run, both the initial ordering of heuristic groups in the priority queue and the ordering of heuristics within each group were randomized, preventing the evaluation from manually favoring any specific heuristic family for a given scenario. After this initialization step, the dispatching mechanism follows the group usage counters described in Chapter 7, prioritizing heuristics from less used groups throughout the planning cycle.

Each parallel configuration was also executed 100 times per scenario, with each run recording the time to the first valid solution, the cost of the first valid solution, the time at which the best solution was found, the cost of the best solution, and the time required to complete the evaluation of the heuristic pool. In this context, the best known solution corresponds to the lowest-cost solution identified for the same scenario in the heuristic evaluation of Chapter 6. Therefore, the time to the best solution measures how quickly the parallel framework reaches the best solution previously obtained by the individual heuristic analysis.

As in the heuristic evaluation of Chapter 6, the reported cost is the excess trajectory cost relative to the sum of the optimal individual robot costs, rather than the raw coordinated cost. For the

sequential random baseline, the reported cost is the mean excess cost over the successful random runs, and incomplete runs are excluded from the mean-cost calculations.

In addition to the mean values reported in the tables, Empirical Cumulative Distribution Functions (ECDFs) are used to complement the analysis of planning times across the 100 runs. These curves are reported for the time to the first valid solution and for the time to the best known solution, allowing the consistency and convergence behavior of each configuration to be compared beyond the average values alone.

8.2 Parallel Framework Results

8.2.1 Scenario 1

The first large-scale scenario, described in Section 6.2.5.1, involves movements from docking areas toward warehouse regions. As shown in Table 8.1, the sequential random-ordering baseline was unable to obtain a valid solution in any of the 100 runs, which is consistent with the heuristic evaluation reported for the same scenario in Section 6.2.5.1, where random priority orderings were shown to be ineffective for this problem instance. In contrast, all heuristic-guided configurations solved every run, showing that the developed framework increased the valid-run rate from 0/100 to 100/100 in this instance.

Table 8.1: Parallel framework results for Large-Scale Scenario 1 over 100 runs.

Configuration	Valid runs	Mean time to first valid (s)	Mean first excess cost (s)	Mean time to best solution (s)	Mean time to complete pool (s)
Sequential random TEA*	0/100	–	–	–	–
Sequential heuristic-guided TEA*	100/100	2.88	20127.58	5.19	10.56
Heuristic-guided TEA*, 3 threads	100/100	1.93	20120.68	2.97	5.34
Heuristic-guided TEA*, 5 threads	100/100	1.74	20099.97	2.49	4.51
Heuristic-guided TEA*, 6 threads	100/100	1.91	20099.97	2.71	3.85

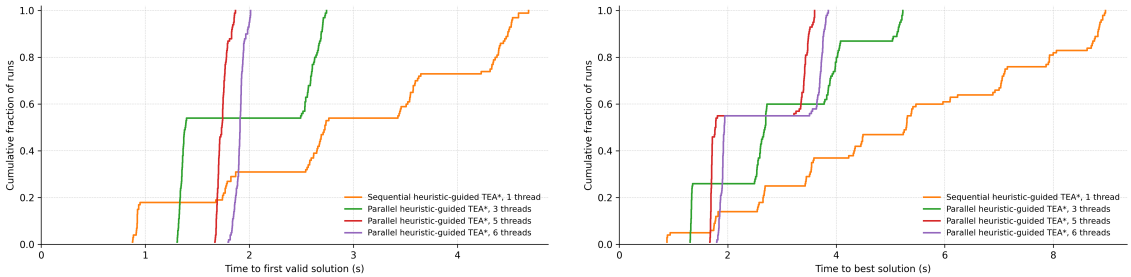
The timing distributions are presented in Figure 8.1. Since the sequential baseline did not produce any valid solution, it is omitted from the ECDF of the first valid solution. In these ECDFs, horizontal segments correspond to time intervals during which no additional run reaches the considered metric, meaning that the planner is still evaluating additional priority orderings before another run succeeds.

For the first valid solution, the single-threaded curve shows that the first sequential heuristic evaluation can sometimes be sufficient, but only in a small fraction of runs. In most executions, additional heuristic evaluations are required, as shown by the horizontal segments spread over a wider time interval. Depending on the initial heuristic ordering, the framework may therefore require between one and roughly five sequential evaluations before finding a valid solution.

Using three workers changes the upper part of the distribution more clearly than the earliest part. The first successful runs are slightly slower than in the single-threaded case, but the higher cumulative fractions are reached earlier, showing that most runs succeed within the first one or

two parallel batches. With five and six workers, the curves become more concentrated, and a valid solution is obtained within the first parallel evaluation batch in every run.

The time to the best known solution follows a stronger version of the same pattern. The single-threaded execution can require up to roughly ten sequential heuristic evaluations, while the parallel configurations reduce the number of evaluation batches required to reach the best known result.



(a) Time to first valid solution.

(b) Time to best known solution.

Figure 8.1: ECDFs of planning times in Scenario 1 over 100 runs.

8.2.2 Scenario 2

The second large-scale scenario, introduced in Section 6.2.5.2, represents movements between warehouse regions. Unlike Scenario 1, the sequential random-ordering baseline solved all 100 runs, which allows a direct comparison between the baseline and the proposed framework. This behavior is consistent with the heuristic evaluation of the same scenario, where random priority orderings were shown to be more effective than in the other large-scale instances. Even so, the results in Table 8.2 show that the heuristic-based configurations improve the quality of the first valid solution, while the multithreaded configurations also reduce the time required to reach the best known solution.

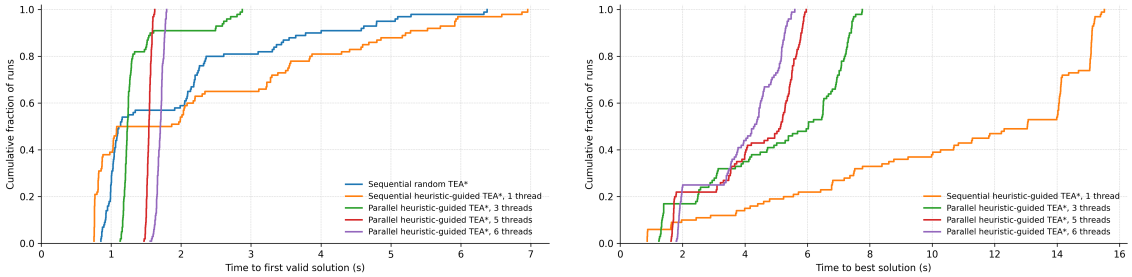
Table 8.2: Parallel framework results for Large-Scale Scenario 2 over 100 runs.

Configuration	**Valid runs**	**Mean time to first valid (s)**	**Mean first excess cost (s)**	**Solution-time speedup**	**Mean time to best solution (s)**	**Mean time to complete pool (s)**
Sequential random TEA*	100/100	1.92	19348.93	1.00	–	–
Sequential heuristic-guided TEA*	100/100	2.32	18908.07	0.83	10.52	15.17
Heuristic-guided TEA*, 3 threads	100/100	1.38	18123.41	1.39	5.01	7.82
Heuristic-guided TEA*, 5 threads	100/100	1.54	17487.68	1.25	4.26	6.41
Heuristic-guided TEA*, 6 threads	100/100	1.70	17599.87	1.13	3.88	6.07

The timing distributions in Figure 8.2 follow the same general pattern observed in Scenario 1, but this is the only large-scale scenario in which the sequential random baseline can be directly compared with the heuristic-guided configurations. Although the baseline obtains valid solutions in all runs, Table 8.2 shows that all heuristic-guided configurations improve the first-solution excess cost. In terms of response time, the single-threaded heuristic-guided configuration produces some

early valid solutions, but also has a longer tail in the ECDF, resulting in a higher mean time than the baseline. The multithreaded configurations reduce this effect, with three workers obtaining the lowest mean time to the first valid solution.

Increasing the number of workers beyond three does not improve the mean time to the first valid solution in this scenario. However, for the best known solution, higher thread counts remain beneficial, since more heuristic alternatives are evaluated within each planning batch and the best known result is reached earlier.



(a) Time to first valid solution.

(b) Time to best known solution.

Figure 8.2: ECDFs of planning times in Scenario 2 over 100 runs.

8.2.3 Scenario 3

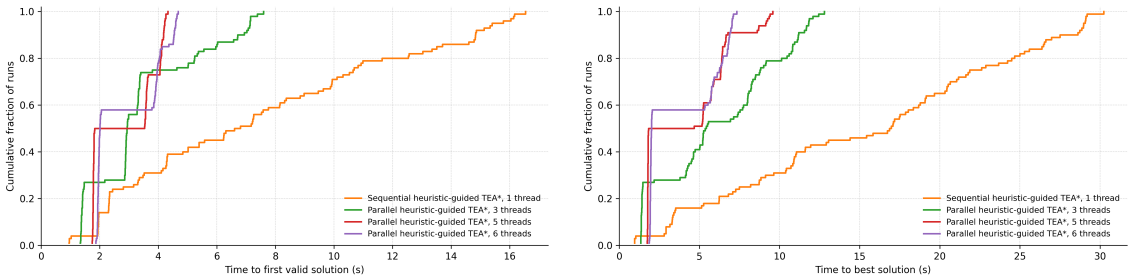
The third large-scale scenario, described in Section 6.2.5.3, considers movements from warehouse regions back to docking areas. This scenario was also shown to be highly restrictive in the heuristic evaluation, where random priority orderings rarely produced complete valid plans. The same behavior is observed in Table 8.3, where the sequential random-ordering baseline obtained valid solutions in only two of the 100 runs. The corresponding baseline averages are therefore computed only over these two successful executions and should be interpreted together with the low success rate. In contrast, every heuristic-based configuration solved all runs, as expected from the heuristic evaluation in Section 6.2.5.3.

Table 8.3: Parallel framework results for Large-Scale Scenario 3 over 100 runs.

Configuration	Valid runs	Mean time to first valid (s)	Mean first excess cost (s)	Solution-time speedup	Mean time to best solution (s)	Mean time to complete pool (s)
Sequential random TEA*	2/100	15.22	29453.67	1.00	–	–
Sequential heuristic-guided TEA*	100/100	7.32	24959.35	2.08	15.42	34.32
Heuristic-guided TEA*, 3 threads	100/100	3.41	24464.78	4.46	6.09	16.03
Heuristic-guided TEA*, 5 threads	100/100	2.83	21999.17	5.37	4.15	12.41
Heuristic-guided TEA*, 6 threads	100/100	2.89	19919.32	5.27	3.82	11.83

The ECDFs in Figure 8.3 show the same general trend observed in the previous scenarios, but with a stronger separation between configurations. Since the sequential random baseline produced only two valid runs, it is omitted from the first valid solution plot and the analysis focuses on the heuristic-guided configurations.

In this more restrictive scenario, the execution with a single thread remains widely spread over time, showing that several sequential heuristic evaluations may be required before a valid ordering is found. Adding workers reduces this variability, with the configurations using five and six workers reaching most successful runs substantially earlier than the configurations using one and three workers. For the best known solution, the separation is even clearer, since the executions with multiple threads reach high cumulative fractions much earlier and require fewer evaluation batches than the configuration with a single thread.



(a) Time to first valid solution.

(b) Time to best known solution.

Figure 8.3: ECDFs of planning times in Large-Scale Scenario 3 over 100 runs.

8.3 Overall Performance Discussion

The results across the three large-scale scenarios show that the proposed framework improved the practical behavior of TEA* planning under the adopted experimental setup through two complementary mechanisms, namely heuristic guidance and parallel evaluation. Heuristic guidance mainly increased valid-solution reliability, since the proposed robot sequences led to successful plans more consistently than random permutations. This is particularly evident in Tables 8.1 and 8.3, where the sequential random ordering baseline either failed to obtain any valid solution or solved only a very small fraction of the runs within the adopted execution limit, whereas all configurations guided by heuristics solved every run, consistently with the heuristic evaluation reported in Chapter 6.

Parallel execution reduces the time required to explore promising regions of the permutation space. With a single worker, the planner evaluates only one heuristic-derived sequence at a time, which avoids most parallel execution overhead but makes early performance strongly dependent on the first heuristics selected from the pool. As a result, the configuration with one thread can occasionally obtain an early valid solution, as visible in the ECDFs of Figures 8.1 and 8.3, while still showing more gradual curves and lower consistency across runs.

Increasing the number of worker threads was generally associated with a higher likelihood of evaluating a productive ordering alternative in the first planning batches, which is reflected in the reduction in the mean time to the first valid solution when moving from a single thread to three or five threads, as shown in Tables 8.1, 8.2, and 8.3. The most favorable configuration, however, depends on the scenario and on the specific metric. The five-thread configuration achieved the

lowest mean time to the first valid solution in Scenarios 1 and 3 (1.74 s and 2.83 s, respectively) and the lowest mean first-solution excess cost in Scenario 2, while matching the six-thread configuration for the lowest first-solution excess cost in Scenario 1 (20099.97 s for both), whereas in Scenario 2 the three-thread configuration reached the first valid solution fastest (1.38 s versus 1.54 s with five threads). These results indicate that, in the evaluated scenarios, broader parallel evaluation was generally associated with more consistent early solution discovery and lower first-solution excess costs, without a single thread count dominating every metric.

The ECDFs complement the mean values by showing how solutions accumulate over time. Rather than uniformly shifting the entire distribution toward shorter times, parallel execution mainly moves the higher cumulative fractions to earlier intervals. In the configuration with one thread, long horizontal segments indicate that several orderings may be evaluated before a useful one is found, whereas multiple workers evaluate several orderings within the same batch. Scenario 2 illustrates this behavior particularly well, since several heuristics from different groups can generate valid solutions and three workers are often enough to include a useful ordering in the first batches, explaining the lowest mean time to the first valid solution. However, the ECDF also shows that the configurations with five and six threads reach the full cumulative fraction within a shorter time interval, indicating a less dispersed distribution across all runs.

The gains, however, are not linear with the number of threads, since the configuration with six threads does not consistently improve the time to the first valid solution over the configuration with five threads and is slightly slower in some cases, as shown in Tables 8.1 and 8.3, as well as in the first valid solution ECDFs. This behavior can be explained by the additional cost of adding another worker, since although an extra worker allows one more priority ordering to be evaluated in the same planning batch, it also introduces scheduling overhead and additional synchronization points, including access to shared resources such as the heuristic queue.

Higher thread counts become more advantageous when considering the time to the best known solution and the time required to complete the heuristic pool. In Tables 8.2 and 8.3, the configuration with six threads reaches the best known solution fastest and completes the heuristic pool in the shortest mean time, while the ECDFs in Figures 8.2 and 8.3 show higher thread counts reaching large cumulative fractions earlier for the best known solution metric. Therefore, using more workers is particularly useful when the objective is not only to obtain a valid solution quickly, but also to evaluate more heuristic alternatives within a shorter planning interval.

Overall, in the evaluated large-scale scenarios, the proposed framework improved TEA* multi-robot planning by combining heuristic diversity with parallel execution. The heuristic orderings helped address the limitations of random priority sampling in the evaluated difficult scenarios, while the parallel architecture reduced the time required to test multiple promising orderings. The observed performance therefore results from the interaction between the quality of the heuristic priority orderings, the degree of parallelism used to explore them, and the computational resources available on the execution platform.

Chapter 9

Temporal Consistency and Redundant Replanning Mitigation through Batching

The previous chapters addressed the performance of the Path Planner by studying priority heuristics and by introducing a parallel planning framework capable of evaluating multiple heuristic-guided orderings within the available planning time. These contributions were evaluated using an isolated Path Planner testing framework, which made it possible to analyze the planning process independently from the rest of the coordination stack.

However, improving the planner alone does not fully determine the behavior of the complete CRIIS coordination system during integrated execution. When the full stack is executed, the Path Planner operates as part of a broader feedback loop involving the Navigation Handlers and the Supervisor. In this context, planning requests, robot state updates, and execution feedback may be generated within short time intervals, creating situations that are not captured by isolated planner tests.

This chapter therefore shifts the analysis from planning performance to coordination consistency. During integrated executions, temporal consistency issues were observed in the processing of planning requests and robot state updates, which can lead to redundant replanning operations. To mitigate these effects, batching mechanisms were introduced to group closely related planning requests and robot state updates before they are processed by the coordination pipeline.

9.1 Observed Temporal Consistency Issues

During planning and replanning operations, robots occasionally received paths whose initial segments no longer matched their current physical positions. This mismatch caused the navigation controller to observe a large error between the robot's current pose and the first target segment of the newly received path. When this error exceeded the controller's admissible threshold, the robot

would stop instead of continuing the planned motion, potentially compromising the consistency and safety of the overall coordination process.

A preliminary analysis revealed that these inconsistencies were strongly related to the way planning requests and robot state updates were processed by the Supervisor. More specifically, these operations could be triggered using outdated or temporally inconsistent robot states. This issue became particularly relevant because the planner could require a non-negligible amount of computation time, especially in large environments or high-density multi-robot scenarios. Consequently, the discrepancy between the planner's internal representation of the system and the real robot states could further increase before the generated plan was finally applied.

These observations motivated a deeper analysis of the temporal behavior of the coordination pipeline, with particular focus on message propagation delays, ROS callback queue processing, and replanning triggering mechanisms.

9.2 Temporal Analysis of the Coordination System

To better understand the origin of the inconsistencies described above, a temporal analysis of the coordination pipeline was conducted. Figure 9.1 presents a simplified representation of the most relevant events and temporal intervals involved in the interaction between the robots and the Coordination System.

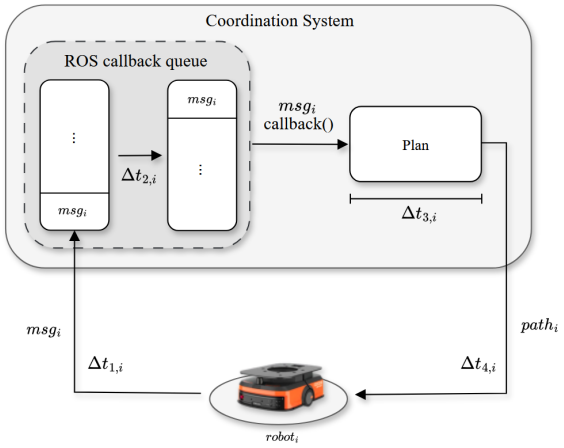


Figure 9.1: Simplified temporal flow within the Coordination System.

The abstraction focuses on the temporal intervals associated with message transmission, ROS callback queue processing, and planner execution. In particular:

- $\Delta t_{1,i}$ corresponds to the transmission delay between robot i and the Coordination System. The transmitted messages may correspond either to planning requests or to robot state updates. Both message types are relevant in this context, since planning requests directly trigger planning operations, while robot state updates may trigger replanning when inconsistencies in the robots' execution progress are detected;

- $\Delta t_{2,i}$ represents the waiting time of incoming messages in the ROS callback queue before being processed by the Supervisor;
- $\Delta t_{3,i}$ denotes the planner execution time required to compute a new plan after a planning or replanning request is processed;
- $\Delta t_{4,i}$ corresponds to the transmission delay associated with sending newly computed plans from the Supervisor back to robot i.

Since the experiments were conducted in a fully local simulation environment, with all ROS nodes executing on the same machine, inter-process communication delays were considered negligible with respect to planner execution time. Therefore, the dominant sources of temporal inconsistency were mainly associated with sequential ROS callback processing and planner execution, corresponding primarily to the intervals $\Delta t_{2,i}$ and $\Delta t_{3,i}$. The following subsections focus on these two intervals and describe the main limitations identified in the baseline Coordination System.

9.2.1 Redundant Planner Executions

One of the main issues identified in the baseline coordination system was the redundant execution of the planner when multiple robots issued planning requests within a short time interval. In the coordination framework, each robot can individually request a plan to reach a given goal vertex. However, in the original implementation, each planning request received by the Supervisor was immediately forwarded to the Path Planner, independently of whether requests from other robots had arrived at approximately the same time.

When a planning request was processed, the Supervisor requested the Path Planner to compute a coordinated plan for the set of robots requiring coordination at that instant. This set included the robots that had already submitted planning requests and had not yet reached their assigned destinations. Consequently, several planner executions may be triggered sequentially for what was effectively a single planning event, with the first planning execution considering only the first processed robot request, the second considering the first two requests, and this pattern continuing until the final execution.

Figure 9.2 represents the case where the planning requests $\{P_1, \ldots, P_n\}$ are sent by the robots within a very short time interval. These requests are inserted sequentially into the ROS callback queue and processed one at a time by the Supervisor. In the baseline implementation, each processed request immediately triggers a new planner execution, leading to a sequence of n planning processes.

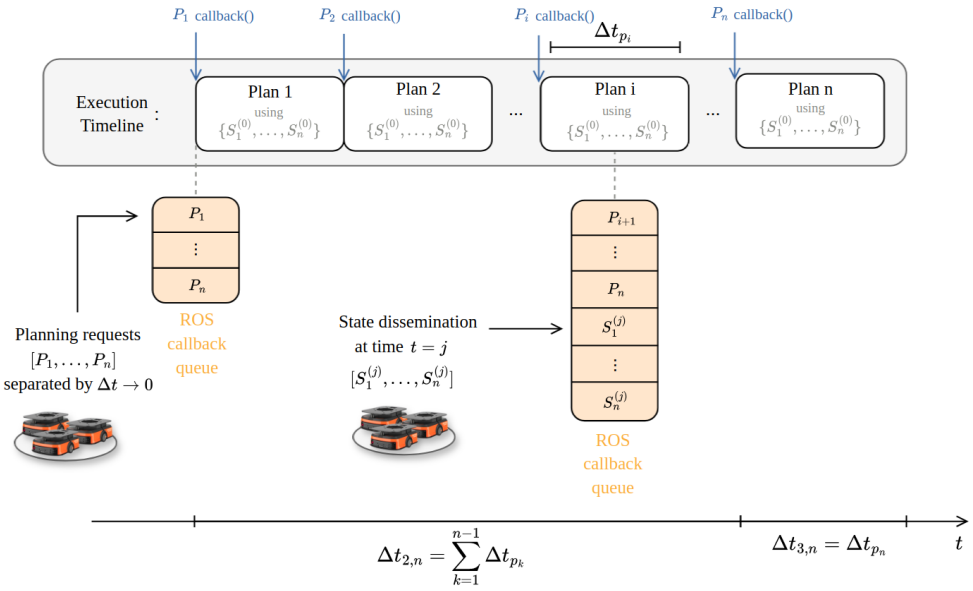


Figure 9.2: Sequential processing of nearly simultaneous planning requests, resulting in redundant planner executions.

During this sequence, Plan 1 is computed after processing only P_1 and, more generally, Plan i after processing $\{P_1,\ldots,P_i\}$. However, all these plans are computed using the same initial robot state set $\{S_1^{(0)},\ldots,S_n^{(0)}\}$, corresponding to the system configuration available before the first planning request was processed. Only Plan n considers the complete request set $\{P_1,\ldots,P_n\}$.

This behavior affected both computational efficiency and temporal consistency. The first $n-1$ planner executions provided limited practical value, since each intermediate plan was immediately superseded by a later execution considering a larger and more complete set of robot requests. At the same time, these redundant executions blocked the ROS callback queue, delaying the processing of newly received robot state updates. The same figure also shows that a new state dissemination may occur at time $t = j$ during the execution of planning process i, producing updated robot states $\{S_1^{(j)},\ldots,S_n^{(j)}\}$. However, if the corresponding state update messages are inserted after pending planning requests in the ROS callback queue, they remain unprocessed until the previous planner executions have finished.

This is particularly problematic because the robots associated with the first $i-1$ planning executions may already have received and started executing their newly computed paths. Consequently, their actual states at time $t = j$ may differ from the states $S^{(0)}$ used when those paths were computed.

In this scenario, the delay affecting the state information used by the final planning process can be approximated by the accumulated execution time of the previous planner calls:

$$\Delta t_{2,n} = \sum_{k=1}^{n-1} \Delta t_{p_k}, \tag{9.1}$$

where Δt_{p_k} denotes the execution time of the k-th planning process. This accumulated waiting time is then followed by the computation time of the final planning process itself:

$$\Delta t_{3,n} = \Delta t_{p_n}. \tag{9.2}$$

Assuming that the message transmission delays $\Delta t_{1,n}$ and $\Delta t_{4,n}$ are negligible in the local simulation environment, the total response time between the moment the first planning request is issued and the moment the final corresponding plan is delivered can be approximated as:

$$\Delta t_{\text{response},n} \approx \Delta t_{2,n} + \Delta t_{3,n}. \tag{9.3}$$

For the final planning process, this results in:

$$\Delta t_{\text{response},n} \approx \sum_{k=1}^{n-1} \Delta t_{p_k} + \Delta t_{p_n} = \sum_{k=1}^{n} \Delta t_{p_k}. \tag{9.4}$$

As a consequence, the time required to generate a plan that considers all robots can increase significantly. Moreover, the resulting plan might be computed using state information that no longer accurately reflects the actual positions of the robots in the system.

9.2.2 Planning with Temporally Inconsistent Robot States

A second issue identified in the baseline Coordination System was related to the way robot state updates were processed by the Supervisor. In the original implementation, whenever a robot sent a state update, the internal variables used to track that robot's progress were immediately updated. The updated progress was then compared with the progress of the remaining robots to determine whether a replanning operation should be triggered.

This behavior resulted from the coupling of two operations within the same ROS callback function, the update of the Supervisor's internal variables for the robot that sent the state message, and the advancement of the state machines responsible for monitoring the execution progress of the robots. As a result, each individual state update can immediately influence the global replanning logic.

The progress of each robot was represented by its current position along the assigned path, typically expressed in terms of the number of completed path steps and the percentage of progression within the current step. If the difference between the progress of two or more robots exceeded a predefined threshold, the Supervisor interpreted this as a coordination inconsistency and triggered a new replanning operation.

However, since robot state updates were processed sequentially in the ROS callback queue, this mechanism can lead to inconsistent replanning triggers. When the Supervisor processed the state update of a given robot, the states of the remaining robots might not yet have been updated with their most recent information. As a result, the progress comparison used to decide whether replanning was necessary would then be based on a partially updated system state.

As illustrated in Figure 9.3, Plan i is initially executed using the state set $\{S_1^{(m)},\ldots,S_n^{(m)}\}$, corresponding to an earlier instant $t=m$. At a later instant $t=j$, a new state dissemination occurs, producing updated state messages $\{S_1^{(j)},\ldots,S_n^{(j)}\}$ for the robots.

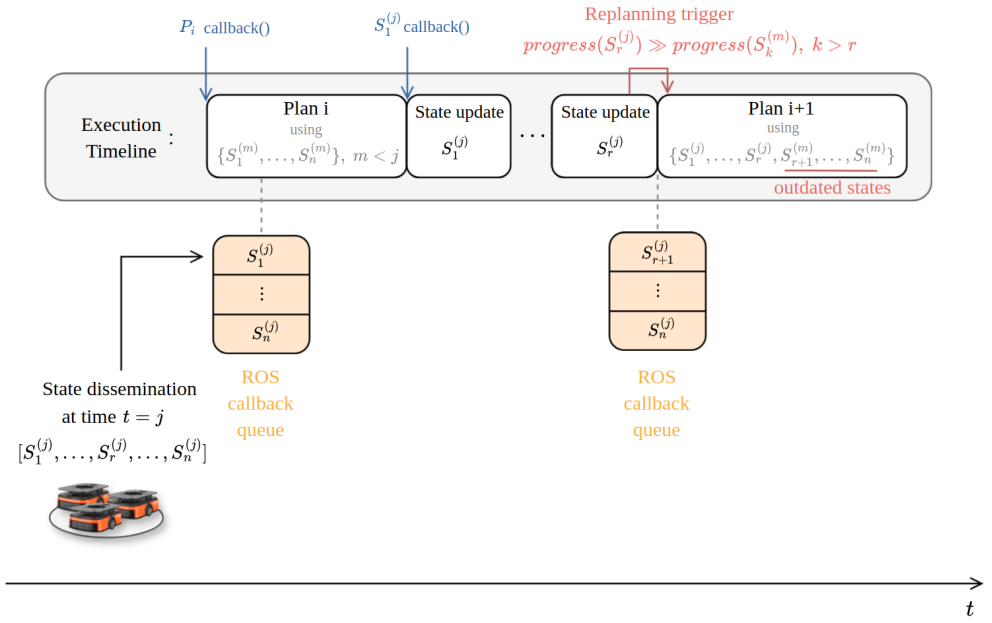


Figure 9.3: Replanning triggered from temporally inconsistent robot states.

For instance, after processing the callback associated with robot 1, the internal state of that robot is updated to $S_1^{(j)}$, while the states of the remaining robots might still correspond to older values. Later, when the callback associated with robot r is processed, the Supervisor evaluates the replanning condition by comparing the progress of robot r with the progress of the remaining robots. This evaluation can trigger a replanning operation if the progress difference exceeds the predefined threshold, represented in Figure 9.3 as $progress(S_r^{(j)}) \gg progress(S_k^{(m)})$, with $k > r$. However, at that moment, only part of the state dissemination has been processed. As a result, the new plan, denoted as Plan $i+1$, is then computed using a mixed state set composed of updated states for the robots whose callbacks have already been processed, $\{S_1^{(j)},\ldots,S_r^{(j)}\}$, and outdated states for the remaining robots, $\{S_{r+1}^{(m)},\ldots,S_n^{(m)}\}$.

This behavior can introduce two related problems:

1. The decision to trigger replanning can be affected by temporally inconsistent information, since the progress of one robot is compared against progress values from other robots that might still refer to older states, which can lead to unnecessary replanning operations.

2. If replanning is triggered immediately after processing a state update, the planner can be executed using robot states associated with different points in time. In this case, the generated plan is not based on a consistent global snapshot of the multi-robot system, but rather on a combination of updated and outdated agent states.

These limitations motivated the introduction of a batching mechanism, described in the following subsection, to decouple message reception from immediate planning and replanning decisions.

9.2.3 Proposed Batching Mechanism

To mitigate the temporal inconsistencies identified in the previous subsections, a batching mechanism was introduced for both planning requests and robot state updates. The main objective of this modification was to prevent the Supervisor from triggering planning or replanning operations based on individual messages processed in isolation.

Instead of forwarding each planning request directly to the Path Planner, the system first aggregates requests received close in time. Similarly, robot state updates are collected and processed together before the Supervisor evaluates whether replanning is required. This approach reduces redundant planner executions and allows replanning decisions to be based on a more temporally consistent representation of the multi-robot system.

The proposed architecture introduces two intermediate components between the robots and the Supervisor: the Robot Request Dispatcher and the Robot State Dispatcher. The Robot Request Dispatcher collects planning requests from multiple robots and forwards them to the Supervisor as a single batch. The Robot State Dispatcher performs an analogous role for robot state updates, ensuring that multiple state messages are processed together before progress comparisons are evaluated.

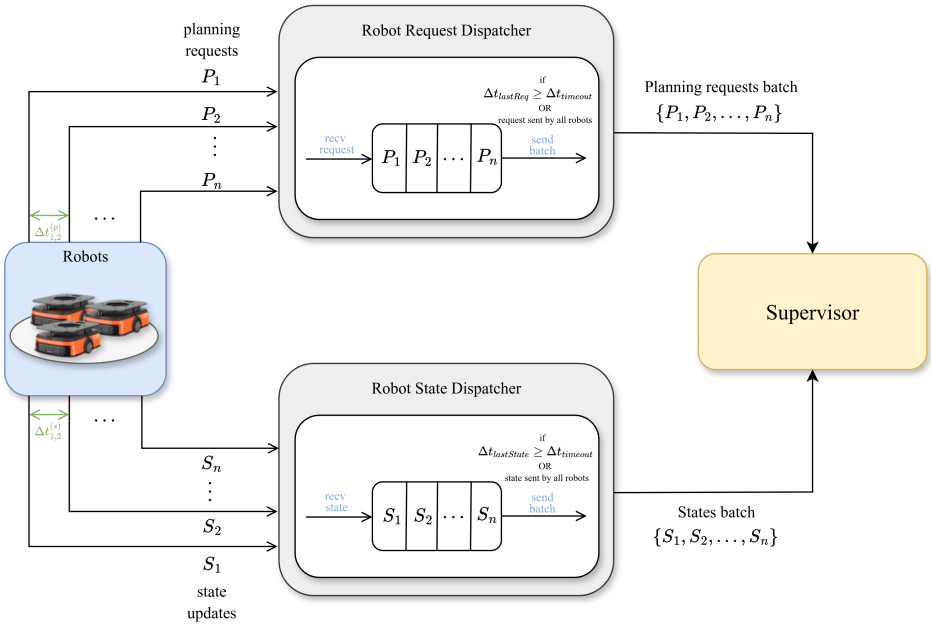


Figure 9.4: Proposed coordination architecture with batching mechanisms for planning requests and robot state updates.

In the architecture represented in Figure 9.4, planning requests are no longer processed independently as soon as they arrive at the Supervisor. Instead, the Robot Request Dispatcher accumulates

incoming requests until one of two conditions is satisfied:

1. Planning requests have been received from all robots in the system, ensuring that the complete set of requests is available for the next planning operation;
2. A predefined inactivity timeout, denoted as T_{timeout}, expires after the most recent planning request and is restarted whenever a new planning request is received.

A similar strategy is applied to robot state updates through the Robot State Dispatcher. In contrast to the baseline implementation, with the presented mechanism, the Supervisor can update the internal states of all robots represented in the batch before advancing the corresponding state machines and comparing robot progress values. This reduces the likelihood of evaluating replanning conditions using a mixture of recently updated and outdated states. Consequently, replanning decisions are based on a more coherent approximation of the global system state.

Overall, the batching mechanism introduces a controlled delay before processing planning requests and robot state updates. This delay is bounded by the timeout parameter, which must balance batch completeness and response latency. In this context, the bounded delay introduced by batching is preferable to the unbounded accumulation of redundant planner executions observed in the baseline implementation, while improving the temporal consistency of the planning pipeline.

9.2.4 Effect of the Batching Mechanism

The effect of the proposed batching mechanism is illustrated in Figure 9.5. In contrast to the baseline behavior, planning requests generated within a short time interval are first aggregated by the Robot Request Dispatcher before being forwarded to the Supervisor. As a result, the requests $\{P_1, \ldots, P_n\}$ are processed as a single batch, producing only one planner execution instead of a sequence of redundant planning processes.

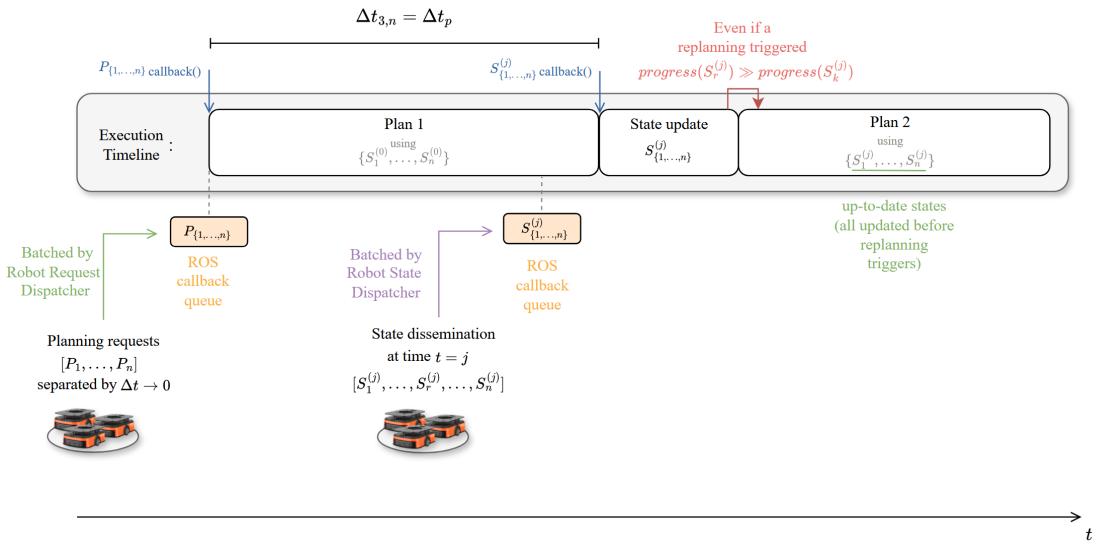


Figure 9.5: Execution flow after introducing batching mechanisms.

In this execution flow, the batched planning request triggers a single planning process, denoted as Plan 1, computed using the state set $\{S_1^{(0)}, \ldots, S_n^{(0)}\}$. Since the requests $\{P_1, \ldots, P_n\}$ are aggregated before being forwarded to the Supervisor, they are processed as a single planning event rather than as a sequence of independent callbacks. As a result, the $n-1$ intermediate planner executions observed in the baseline implementation are avoided.

The batching mechanism introduces a delay before the batch is forwarded to the Supervisor, denoted as Δt_{batch}. Since the inactivity timeout is restarted whenever a new request arrives, Δt_{batch} depends on the temporal distribution of incoming requests. In the worst case, if each new request arrives immediately before the inactivity timeout expires and the batch is completed only when the last robot sends its request, this delay is bounded by:

$$\Delta t_{\text{batch}} \leq (n-1) T_{\text{timeout}}. \tag{9.5}$$

Nevertheless, T_{timeout} is selected to be significantly smaller than the planner execution time, making this additional batching delay small in relation to the computational cost of planning. Once the batch is forwarded, the dominant remaining delay is the execution time of a single planning process, denoted by Δt_p.

Assuming that message transmission delays remain negligible in the local simulation environment, the total response time between the first planning request and the corresponding plan delivery can therefore be approximated as:

$$\Delta t_{\text{response}}^{\text{batch}} \approx \Delta t_{\text{batch}} + \Delta t_p. \tag{9.6}$$

This response time contrasts with the baseline behavior described in Equation 9.4, where the response time for the final planning process may accumulate the execution time of multiple redundant planner calls.

A similar improvement is obtained for robot state updates. When a new state dissemination occurs at time $t = j$, the updated states $\{S_1^{(j)}, \ldots, S_n^{(j)}\}$ are first aggregated by the Robot State Dispatcher and then forwarded to the Supervisor as a single batch.

With batching, state updates accumulated before the inactivity timeout expires are incorporated before the progress comparison is performed. Therefore, if replanning is triggered, the new plan is computed using a more coherent and up-to-date state set, $\{S_1^{(j)}, \ldots, S_n^{(j)}\}$, rather than a state representation made inconsistent by partially processed callback messages.

This mechanism therefore introduces a trade-off between temporal consistency and response latency. Since messages are accumulated before being forwarded to the Supervisor, planning requests and robot state updates are no longer processed immediately after reception. In the considered system, this additional delay is not expected to compromise coordination performance, since robot states are published at a higher frequency than that required by the Supervisor to make coordination decisions. If response latency became critical, more recent robot states may be estimated through interpolation between consecutive updates. However, this mechanism was not required in the evaluated system.

9.3 Batching Mechanism Evaluation

To evaluate the proposed batching mechanism, the complete CRIIS fleet coordination workflow represented in Figure 4.4 was executed with and without batching using 25 robots in the Stage simulator, which is lightweight and already integrated into the current system. A fleet with 25 robots was used because the integrated setup also includes the navigation and simulated execution workload associated with each robot, making it significantly more computationally demanding than the isolated Path Planner experiments. The experiment was triggered using an auxiliary test node already available in the system, which calls the path request service of each robot. For each configuration, 10 executions were performed, and the reported values correspond to the mean values obtained across these runs.

Although the implemented architecture includes both planning request batching and robot state update batching, the quantitative analysis focuses on planning request batching, since this mechanism produced the most visible impact on the aggregate execution metrics. The robot state dispatcher remains part of the proposed architecture as a temporal consistency mechanism, but its isolated effect did not produce a significant change in the measured aggregate metrics across these executions.

Table 9.1 summarizes the obtained mean results. The number of initial planner executions refers to the planner calls triggered by the first sequence of robot requests, before the complete fleet had received an initial coordinated plan. The total planning time corresponds to the sum of the execution times of all Path Planner calls. The maximum request response time is the largest interval between a robot request and the delivery of the corresponding plan, while the mission time is measured from the instant at which the first robot sends its request until all robots reach their final goals. The relative reduction is computed from the corresponding mean values.

Table 9.1: Mean effect of the batching mechanism on the integrated coordination execution with 25 robots over 10 runs.

Metric	**Without batching**	**With batching**	**Relative reduction**
Initial planner executions	25	1	96.0%
Total planner executions	37.5	18.6	50.4%
Total planning time (s)	35.72	21.87	38.8%
Maximum request response time (s)	19.62	3.76	80.8%
Mission time (s)	78.33	68.27	12.8%

Without batching, the 25 initial robot requests were processed independently, causing the Supervisor to trigger 25 initial planner executions before a plan considering the complete fleet was generated. With batching, the same requests were grouped into a single planning episode, reducing the number of initial planner executions from 25 to 1. This also reduced the mean total number of planner executions from 37.5 to 18.6 and the mean accumulated planning time from 35.72 s to 21.87 s.

The mean maximum request response time decreased from 19.62 s without batching to 3.76 s with batching. This shows that the bounded delay introduced by the batching mechanism is smaller than the accumulated delay caused by processing nearly simultaneous requests as independent planning events. The mean mission time was also reduced from 78.33 s to 68.27 s, indicating that reducing redundant planner executions improves the temporal behavior of the integrated coordination pipeline.

The mean number of replanning operations was 12.5 without batching and 17.6 with batching, obtained as the difference between the total number of planner executions and the initial planner executions. However, this difference is explained by the baseline configuration, where robots start moving while the redundant initial planning sequence is still being processed. As a result, later replanning operations occur when some robots are already closer to their goals, making a lower number of replanning operations in the baseline configuration expected.

An additional visualization of the executions is provided in Appendix A.2. The corresponding video shows the system running without batching and with batching, serving as an illustrative complement to the quantitative results.

9.4 Summary

This chapter analyzed temporal consistency issues observed during integrated execution of the complete CRIIS coordination pipeline. The main limitations identified were the redundant execution of the Path Planner when several planning requests arrived within a short time interval, and the possibility of triggering replanning decisions from partially updated robot state information. These issues were associated with the sequential processing of ROS callbacks and with the immediate execution of planning and replanning logic after individual messages were received.

To mitigate these effects, batching mechanisms were introduced for both planning requests and robot state updates. Planning request batching aggregates closely timed requests before invoking the Path Planner, while robot state update batching allows the Supervisor to update a more coherent set of robot states before evaluating replanning conditions. This introduces a bounded delay, but avoids the accumulation of redundant planner executions and improves the temporal consistency of the coordination pipeline.

The integrated system evaluation showed that the batching mechanism reduced redundant initial planning activity, total planner executions, accumulated planning time, request response time, and overall mission time. These results support the use of batching as a practical mechanism for improving the temporal behavior of the complete multi-robot coordination system without modifying the underlying TEA* planner.

Chapter 10

Conclusions and Future Work

10.1 Main Conclusions

This dissertation addressed centralized MRPP in a TEA*-based fleet coordination system. The work was motivated by the sensitivity of sequential prioritized planning to robot ordering, where different priority permutations may lead to substantially different outcomes in terms of feasibility, solution cost, and planning time. Since exhaustive exploration of all possible robot orderings is infeasible for large fleets, the dissertation investigated heuristic and parallel strategies for exploring more informative robot sequences within a practical planning time budget.

As a first step, the work established the theoretical and technical foundations required for the proposed approach, including MRPP and MAPF concepts, graph search, clustering, regression, and parallel computing. The existing CRIIS fleet coordination system was then analyzed as the baseline architecture, with particular focus on its centralized planning pipeline and on the role of TEA* as the underlying path planning method. This analysis showed that TEA* provides a practical mechanism for computing time-aware, conflict-free plans, but that its sequential structure makes the final solution strongly dependent on the selected robot ordering.

To address this limitation, several prioritization heuristics were designed and implemented. These heuristics were organized into different families, namely Roadmap Distance, Topological, Surroundings, Conflicts, and Replanning, according to the type of information used to compute robot priority scores. Their behavior was evaluated using an isolated Path Planner testing framework, which allowed the effect of each ordering strategy to be analyzed independently from the remaining coordination stack. The experimental results showed that no single heuristic was consistently dominant across all evaluated scenarios. Instead, each heuristic family was more effective under different spatial, topological, or conflict-related conditions, confirming the importance of considering multiple robot orderings rather than relying on a single fixed priority rule.

Based on this observation, a parallel planning framework was developed to evaluate multiple heuristic-generated orderings within the same planning cycle. In this framework, worker threads execute independent TEA* instances, each using a different robot sequence, while shared mechanisms manage heuristic dispatching, timeout handling, candidate plan selection, and redundant heuristic

filtering. This design preserves the underlying TEA* planner while extending the exploration of the ordering space, allowing the system to evaluate several candidate plans without replacing the existing centralized planning architecture.

Performance evaluation showed that the proposed parallel planner improved valid-solution reliability in the evaluated large-scale scenarios when compared with random priority selection. The results also showed that increasing the number of worker threads generally reduced the time required to obtain the first valid solution, although the gains were not strictly proportional to the number of threads. This confirms that parallel evaluation of heuristic alternatives can mitigate the ordering sensitivity of sequential TEA* planning, while also highlighting that the benefit depends on the scenario, the heuristic pool, and the available computational resources.

During integrated execution, the complete CRIIS coordination stack was also analyzed, revealing temporal consistency issues in the processing of planning requests and robot state updates. These issues can lead to redundant replanning operations and planning decisions based on incomplete temporal context. To mitigate this behavior, batching mechanisms were introduced for both planning requests and robot state updates. The integrated evaluation showed that these mechanisms reduced redundant planner executions and improved the temporal consistency of the coordination pipeline during the observed executions.

Dissemination of the work was also initiated through the submission of a paper to ROBOT2026, based on the heuristic prioritization study developed in this dissertation.

Overall, the results obtained throughout this dissertation show that the sensitivity of sequential TEA*-based planning to robot ordering was mitigated in the evaluated scenarios without abandoning the existing planner architecture. The proposed heuristics produced diverse and useful robot sequences, the parallel planner enabled several of these alternatives to be evaluated within the same planning cycle, and the batching mechanisms reduced redundant planner executions during integrated execution. Together, these contributions provide a practical extension of the baseline centralized fleet coordination system, with observed gains in valid-solution reliability, ordering flexibility, and temporal consistency in the evaluated MRPP setting.

10.2 Future Work

Several directions can be explored to extend the work developed in this dissertation. A first direction is the design of adaptive heuristic selection mechanisms. In the current approach, multiple heuristics are available and dispatched according to predefined rules, but the system does not explicitly infer which heuristic families are more suitable for each planning instance. Future work might use features extracted from the planning request, such as robot density, path overlap, graph connectivity, predicted conflicts, start-goal distribution, and previous planning outcomes, to prioritize the most promising heuristics before planning begins.

A second direction is the development of dynamic planning time allocation. In this dissertation, the parallel planner operates under a fixed maximum planning time. However, in an integrated coordination system, the available planning time is not only a fixed design parameter, but also

depends on the current execution state of the robots. For example, if a robot is currently traversing an edge, the useful time available for replanning is likely related to the time remaining until that robot reaches the next graph node. In this case, a new plan should ideally be available before the robot reaches that node, so that the following trajectory segment can be assigned without introducing unnecessary waiting or execution inconsistency. Future work can therefore define $T_{\max}$ dynamically from the minimum available decision time across the relevant robots, while also considering safety margins, communication delays, expected conflict level, fleet size, and the urgency of the planning request.

A further direction is the development of additional prioritization heuristics based on the heuristic families proposed in this dissertation. Future work might combine already explored criteria to design ordering strategies that adapt better to different planning instances. One possible example is the refinement of conflict-based heuristics. In this dissertation, estimated conflicts were computed from the optimal individual paths of the robots, planned without considering the remaining fleet. However, in a TEA*-based prioritized planner, robots often follow suboptimal trajectories due to space-time reservations introduced by previously planned robots. Therefore, future heuristics could estimate conflicts over a set of feasible suboptimal paths, rather than only over the individually optimal path, potentially providing a conflict estimate closer to the trajectories that are actually produced during coordinated planning.

In addition to designing new heuristics, future work could also optimize the computation of some of the existing ones. For instance, the heuristics that depend on the DoF of roadmap nodes currently require repeated access to topological information during score computation. This process could be improved by precomputing the DoF value of each roadmap node and storing it in a lookup table. Such a structure would allow DoF-based scores to be computed with lower overhead, particularly in large roadmaps or when several heuristic variants are evaluated within the same planning cycle.

Another relevant direction is the validation of the proposed mechanisms in additional integrated executions, including real-robot deployments or longer simulated missions, in order to assess their robustness under more diverse execution and replanning conditions.

Finally, future work might investigate tighter integration between planning, execution monitoring, and priority selection. Instead of treating each planning request independently, the system can use execution history, recurrent congestion regions, previous failed orderings, and robot-specific delay patterns to guide future planning cycles. This would move the framework toward a more adaptive coordination system, while preserving the practical advantage of using TEA* as the underlying planner.

References

[1] United Nations, *The Sustainable Development Goals*, https://sdgs.un.org/goals, Accessed: 2026-06-19.

[2] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Barták, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," *Proceedings of the 12th International Symposium on Combinatorial Search, SoCS 2019*, pp. 151–159, Jun. 2019, ISSN: 2832-9171. DOI: 10.1609/socs.v10i1.18510.

[3] P. Surynek, "An Optimization Variant of Multi-Robot Path Planning Is Intractable," vol. 24, AAAI Press, Jul. 2010, pp. 1261–1263, ISBN: 9781577354642. DOI: 10.1609/AAAI.V24I1.7767.

[4] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, Feb. 2015, ISSN: 0004-3702. DOI: 10.1016/J.ARTINT.2014.11.006.

[5] J. Yu and S. M. LaValle, "Planning Optimal Paths for Multiple Robots on Graphs," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2013, pp. 3612–3617, ISBN: 9781467356411. DOI: 10.1109/ICRA.2013.6631084.

[6] D. Silver, "Cooperative Pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, pp. 117–122, 1 Jun. 2005, ISSN: 2334-0924. DOI: 10.1609/AIIDE.V1I1.18726.

[7] D. M. Matos, P. Costa, H. Sobreira, A. Valente, and J. Lima, "Efficient multi-robot path planning in real environments: a centralized coordination system," *International Journal of Intelligent Robotics and Applications*, vol. 9, pp. 217–244, 1 Mar. 2025, ISSN: 2366598X. DOI: 10.1007/s41315-024-00378-3.

[8] J. Santos, P. Costa, L. F. Rocha, A. P. Moreira, and G. Veiga, "Time Enhanced A*: Towards the Development of a New Approach for Multi-Robot Coordination," in *Proceedings of the 2015 IEEE International Conference on Industrial Technology (ICIT)*, IEEE, 2015, pp. 3314–3319. DOI: 10.1109/ICIT.2015.7125589.

[9] T. Geft, "Fine-Grained Complexity Analysis of Multi-Agent Path Finding on 2D Grids," *The International Symposium on Combinatorial Search*, vol. 16, pp. 20–28, 1 May 2023, ISSN: 28329163. DOI: 10.1609/socs.v16i1.27279.

[10] J. Gao, Y. Li, X. Li, K. Yan, K. Lin, and X. Wu, "A review of graph-based multi-agent pathfinding solvers," *Knowledge-Based Systems*, vol. 283, Jan. 2024, ISSN: 09507051. DOI: 10.1016/J.KNOSYS.2023.111121.

[11] A. Yang, Q. Niu, W. Zhao, K. Li, and G. W. Irwin, "An Efficient Algorithm for Grid-Based Robotic Path Planning Based on Priority Sorting of Direction Vectors," vol. 6329 LNCS, Springer, Berlin, Heidelberg, 2010, pp. 456–466, ISBN: 978-3-642-15597-0. DOI: 10.1007/978-3-642-15597-0_50.

[12] H. Ma, "Graph-Based Multi-Robot Path Finding and Planning," *Current Robotics Reports*, vol. 3, pp. 77–84, 3 Jun. 2022. DOI: 10.1007/s43154-022-00083-8.

[13] T. Lozano-Pérez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, 10 Oct. 1979, ISSN: 15577317. DOI: 10.1145/359156.359164.

[14] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, pp. 345–405, 3 Jan. 1991, ISSN: 15577341. DOI: 10.1145/116873.116880.

[15] J.-C. Latombe, *Robot Motion Planning*. Springer, 1991.

[16] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005, ISBN: 9780262033275.

[17] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010, ISBN: 978-0136042594.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, Sep. 2009, ISBN: 9780262033848. [Online]. Available: https://mitpress.mit.edu/9780262033848/introduction-to-algorithms/.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 2 1968, ISSN: 21682887. DOI: 10.1109/TSSC.1968.300136.

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1 Dec. 1959, ISSN: 0029599X. DOI: 10.1007/BF01386390.

[21] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley, 1984, ISBN: 9780201055948.

[22] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*, 2nd ed. Academic Press, 2020, ISBN: 978-0-12-818803-3.

[23] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, 1967, pp. 281–297.

[24] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. DOI: 10.1109/TIT.1982.1056489.

[25] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[26] A. J. Onumanyi, D. N. Molokomme, S. J. Isaac, and A. M. Abu-Mahfouz, "AutoElbow: An Automatic Elbow Detection Method for Estimating the Number of Clusters in a Dataset," *Applied Sciences*, vol. 12, no. 15, p. 7515, 2022. DOI: 10.3390/app12157515.

[27] A. Ng, *The K-means Clustering Algorithm*, CS229 Lecture Notes, Stanford University, Accessed: 2026-05-21, 2020. [Online]. Available: https://cs229.stanford.edu/notes2020spring/cs229-notes7a.pdf.

[28] E. Schubert, J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN," *ACM Transactions on Database Systems*, vol. 42, no. 3, pp. 1–21, 2017. DOI: 10.1145/3068335.

[29] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 6th ed. Wiley, 2021.

[30] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2018. [Online]. Available: https://pages.cs.wisc.edu/~remzi/OSTEP/.

[31] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Wiley, 2018, ISBN: 978-1119456339.

[32] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Pearson, 2015, ISBN: 978-0133591620.

[33] E. W. Dijkstra, "Cooperating Sequential Processes," in *Programming Languages*, F. Genuys, Ed., New York, NY: Academic Press, 1968, pp. 43–112.

[34] L. Siefke, V. Sommer, B. Wudka, and C. Thomas, "Robotic Systems of Systems Based on a Decentralized Service-Oriented Architecture," *Robotics*, vol. 9, no. 4, p. 78, 2020. DOI: 10.3390/robotics9040078.

[35] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, "Distributed coordination of a set of autonomous mobile robots," *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 480–485, 2000. DOI: 10.1109/IVS.2000.898389.

[36] M. Čáp, P. Novák, M. Selecký, J. Faigl, and J. Vokřínek, "Asynchronous Decentralized Prioritized Planning for Coordination in Multi-Robot System," in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3822–3829. DOI: 10.1109/IROS.2013.6696903.

[37] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1928–1935, 2008, ISSN: 10504729. DOI: 10.1109/ROBOT.2008.4543489.

[38] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, pp. 90–98, 1 1986, ISSN: 02783649. DOI: 10.1177/027836498600500106.

[39] Z. Wang, X. Zhao, J. Zhang, N. Yang, P. Wang, J. Tang, J. Zhang, and L. Shi, "APF-CPP: An Artificial Potential Field Based Multi-Robot Online Coverage Path Planning Approach," *IEEE Robotics and Automation Letters*, vol. 9, pp. 9199–9206, 11 2024, ISSN: 23773766. DOI: 10.1109/LRA.2024.3432351.

[40] Z. Wu, J. Dai, B. Jiang, and H. R. Karimi, "Robot path planning based on artificial potential field with deterministic annealing," *ISA Transactions*, vol. 138, pp. 74–87, Jul. 2023, ISSN: 00190578. DOI: 10.1016/J.ISATRA.2023.02.018.

[41] P. Caloud, W. Choi, J. C. Latombe, C. L. Pape, and M. Yim, "Indoor automation with many mobile robots," *IEEE International Conference on Intelligent Robots and Systems*, vol. 1990-July, pp. 67–72, 1990, ISSN: 21530866. DOI: 10.1109/IROS.1990.262370.

[42] J. Santos, P. Costa, L. Rocha, K. Vivaldini, A. P. Moreira, and G. Veiga, "Validation of a Time Based Routing Algorithm Using a Realistic Automatic Warehouse Scenario," *Advances in Intelligent Systems and Computing*, vol. 418, pp. 81–92, 2016, ISSN: 2194-5365. DOI: 10.1007/978-3-319-27149-1_7.

[43] P. Moura, P. Costa, J. Lima, and P. Costa, "A Temporal Optimization Applied to Time Enhanced A*," vol. 2116, American Institute of Physics Inc., Jul. 2019, ISBN: 9780735418547. DOI: 10.1063/1.5114225.

[44] A. Andreychuk, K. Yakovlev, E. Boyarski, and R. Stern, "Improving Continuous-time Conflict Based Search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11 220–11 227, 13 May 2021, ISSN: 2374-3468. DOI: 10.1609/AAAI.V35I13.17338.

[45] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5628–5635, 2011, ISSN: 10504729. DOI: 10.1109/ICRA.2011.5980306.

[46] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem," *Proceedings of the International Symposium on Combinatorial Search*, vol. 5, pp. 19–27, 1 2014, ISSN: 2832-9163. DOI: 10.1609/SOCS.V5I1.18315.

[47] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. K. Kumar, and S. Koenig, "Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, pp. 83–87, Jun. 2018, ISSN: 2334-0843. DOI: 10.1609/ICAPS.V28I1.13883.

[48] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103 574, 2021, ISSN: 0004-3702. DOI: 10.1016/j.artint.2021.103574.

[49] B. Muppasani, R. Dey, B. Srivastava, and V. Narayanan, *Towards Information-Optimized Multi-Agent Path Finding: A Hybrid Framework with Reduced Inter-Agent Information Sharing*, 2025. DOI: 10.48550/arXiv.2510.09469. arXiv: 2510.09469 [cs.MA].

[50] H. Zhang, M. Yao, Z. Liu, J. Li, L. Terr, S. H. Chan, T. K. S. Kumar, and S. Koenig, "A Hierarchical Approach to Multi-Agent Path Finding," *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, pp. 209–211, 1 Jul. 2021, ISSN: 2832-9163. DOI: 10.1609/SOCS.V12I1.18586.

[51] C. Leet, J. Li, and S. Koenig, "Shard Systems: Scalable, Robust and Persistent Multi-Agent Path Finding with Performance Guarantees," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 9386–9395, 9 Jun. 2022, ISSN: 2374-3468. DOI: 10.1609/AAAI.V36I9.21170.

[52] S. Wang, H. Xu, Y. Zhang, J. Lin, C. Lu, X. Wang, and W. Li, *Where Paths Collide: A Comprehensive Survey of Classic and Learning-Based Multi-Agent Pathfinding*, 2025. DOI: 10.48550/arXiv.2505.19219. arXiv: 2505.19219 [cs.AI].

[53] J. Hu, "A Survey on Reinforcement Learning-Based Multi-Agent Path Planning," *ITM Web of Conferences*, vol. 78, Jan. 2025. DOI: 10.1051/itmconf/20257801015.

[54] M. Everett, Y. F. Chen, and J. P. How, "Collision Avoidance in Pedestrian-Rich Environments With Deep Reinforcement Learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021. DOI: 10.1109/ACCESS.2021.3050338.

[55] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with Consistent Prioritization for Multi-Agent Path Finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7643–7650. DOI: 10.1609/aaai.v33i01.33017643.

[56] S. Zhang, J. Li, T. Huang, S. Koenig, and B. Dilkina, "Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding," *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, no. 1, pp. 208–216, 2022. DOI: 10.1609/socs.v15i1.21769.

[57] J. R. Heselden and G. P. Das, "Heuristics and Rescheduling in Prioritised Multi-Robot Path Planning: A Literature Review," *Machines*, vol. 11, no. 11, p. 1033, 2023. DOI: 10.3390/machines11111033.

[58] M. Bennewitz, W. Burgard, and S. Thrun, "Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots," *Robotics and Autonomous Systems*, vol. 41, no. 2, pp. 89–99, 2002. DOI: 10.1016/S0921-8890(02)00256-7.

[59] J. P. van den Berg and M. H. Overmars, "Prioritized Motion Planning for Multiple Robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2217–2222. DOI: 10.1109/IROS.2005.1545306.

[60] W. Wu, S. Bhattacharya, and A. Prorok, "Multi-Robot Path Deconfliction through Prioritization by Path Prospects," in *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9809–9815. DOI: 10.1109/ICRA40945.2020.9196813.

[61] A. W. ter Mors, "Evaluating Heuristics for Prioritizing Context-Aware Route Planning Agents," in *Proceedings of the IEEE International Conference on Networking, Sensing and Control*, 2011, pp. 127–132. DOI: 10.1109/ICNSC.2011.5874899.

[62] C. M. Clark, T. Bretl, and S. Rock, "Applying Kinodynamic Randomized Motion Planning with a Dynamic Priority System to Multi-Robot Space Systems," in *Proceedings of the IEEE Aerospace Conference*, vol. 7, 2002, pp. 3621–3631. DOI: 10.1109/AERO.2002.1035338.

[63] R. Regele and P. Levi, "Cooperative Multi-Robot Path Planning by Heuristic Priority Adjustment," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5954–5959. DOI: 10.1109/IROS.2006.282480.

[64] A. Andreychuk and K. Yakovlev, "Two techniques that enhance the performance of multi-robot prioritized path planning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2018)*, Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2177–2179.

[65] F. Cardoso, D. Matos, M. Brilhante, P. Costa, H. Sobreira, and C. Silva, "Enhancing Mobile Robot Navigation: A Graph Decomposition Submodule for TEA*," Apr. 2025, pp. 152–157. DOI: 10.1109/ICARSC65809.2025.10970181.

[66] J. Ribeiro, M. Brilhante, D. M. Matos, C. A. Silva, H. Sobreira, and P. Costa, "Parallel Path Planning for Multi-Robot Coordination," in *2025 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Funchal, Portugal: IEEE, 2025, pp. 78–85. DOI: 10.1109/ICARSC65809.2025.10970166.

[67] N. Murugesan, I. Cho, and C. Tortora, "Benchmarking in Cluster Analysis: A Study on Spectral Clustering, DBSCAN, and K-Means," in *Data Analysis and Rationality in a Complex World*, Springer Nature Switzerland, 2021, pp. 175–185. DOI: 10.1007/978-3-030-60104-1_20.

[68] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012. DOI: 10.1109/TCIAIG.2012.2197681.

[69] J. Li, *CBSH2-RTC: An Optimal Multi-Agent Path Finding Solver*, https://github.com/Jiaoyang-Li/CBSH2-RTC, Accessed: 2026-06-08.

Appendix A

Execution Videos

This appendix provides links to supplementary videos illustrating selected executions of the multi-robot coordination system.

A.1 Large-Scale Heuristic Execution

The video available at https://youtu.be/P9jD-t3u8gc is associated with the large-scale factory operation scenarios presented in Chapter 6, specifically the three 180 robot instances described in Section 6.2.5. It shows the planned robot movements for the "docking to warehouse", "warehouse to warehouse" and "warehouse to docking" configurations.

A.2 Batching Mechanism Execution

The video available at https://youtu.be/IMvVRX2669w shows the integrated coordination system running without batching and with batching. It is included as an illustrative complement to the batching mechanism evaluation presented in Chapter 9.

Appendix B

Evaluation Metrics

This appendix defines the main evaluation metrics used throughout this dissertation to analyze planning performance, regression quality, and the distribution of experimental results. The definitions are included to support the interpretation of the quantitative results presented in the evaluation chapters.

B.1 Mean Absolute Error

The MAE measures the average absolute difference between an observed value and the corresponding estimated or predicted value. Given a set of n observations, where y_i denotes the observed value and $\hat{y}_i$ denotes the estimated value, MAE is defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{B.1}$$

This metric is expressed in the same unit as the evaluated variable and provides a direct measure of the average magnitude of the error. In this dissertation, MAE is used when the objective is to quantify prediction or estimation error in an interpretable way, without distinguishing between positive and negative deviations.

B.2 Coefficient of Determination

The coefficient of determination, commonly denoted as R^2, measures how much of the variance in the observed data is explained by a regression model. It is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{B.2}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the observed values.

An R^2 value close to 1 indicates that the model explains a large proportion of the observed variance, while values close to 0 indicate limited explanatory power. Negative values may occur when the model performs worse than simply predicting the mean of the observed data. In this

dissertation, R^2 is used to assess the explanatory quality of regression models used for runtime analysis.

B.3 Empirical Cumulative Distribution Function

The ECDF represents the proportion of observations that are less than or equal to a given value. For a set of n observations, it can be written as

$$\hat{F}_n(x) = \frac{k}{n}, \tag{B.3}$$

where k is the number of observations with value less than or equal to x.

In this dissertation, ECDF plots are used to compare experimental runs by showing the percentage of runs that achieve a value below a given threshold. This is useful for metrics such as planning time and solution cost, since it shows the distribution of results instead of only average values.