CLASSICAL AND LEARNING-BASED MULTI-GOAL ORDERING AND PATH PLANNING FOR MOBILE ROBOTS

by
ABDULLAH ALLUS

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2025

CLASSICAL AND LEARNING-BASED MULTI-GOAL ORDERING AND PATH PLANNING FOR MOBILE ROBOTS

Approved by:

Prof. Dr. MUSTAFA ÜNEL ....
(Thesis Supervisor)

Assoc. Prof. Dr. KEMALETTİN ERBATUR ....

Asst. Prof. Dr. ERTUĞRUL BAYRAKTAR ....

Date of Approval: July 16, 2025

ABSTRACT

CLASSICAL AND LEARNING-BASED MULTI-GOAL ORDERING AND PATH PLANNING FOR MOBILE ROBOTS

ABDULLAH ALLUS

MECHATRONICS ENGINEERING M.Sc. THESIS, JULY 2025

Thesis Supervisor: Prof. Dr. MUSTAFA ÜNEL

Keywords: Mobile Robots, Path-Planning, Multi-Goal Pathfinding, A-Star, Transformers

In the field of autonomous mobile robotics, efficient and scalable solutions to the multi-goal ordering and path planning problem—where a robot must visit a set of spatially distributed goal nodes in the most optimal sequence—remain a significant challenge. In this study, we develop two approaches to tackle this important problem. The first one is a classical geometry-based approach and the second one is a learning-based approach that addresses the problem using either a traditional machine learning technique or a transformer-based method.

In the classical approach, we propose a novel ordering strategy based on a one-distance-two-angles paradigm, which reduces reliance on traditional distance metrics by incorporating geometric considerations to infer optimal visiting sequences. This ordering procedure is paired with an improved version of the A* algorithm that integrates principles from computer graphics to eliminate redundant zigzags and intermediate points often present in grid-based environments, resulting in smoother and more cost-effective paths without compromising computational efficiency. Additionally, we introduce two learning-based frameworks to predict goal visiting orders. The first is a traditional machine learning model trained on hand-crafted features derived from brute-force optimal solutions, capturing geometric patterns such as distances, angles, and spatial relationships. The second is a transformer model trained on features extracted using CNNs and Relational Transformers, along with geometric context-based features from optimal paths. Our experiments demonstrate that both models generalise effectively to unseen environments and large-scale scenarios, achieving high accuracy in reproducing near-optimal orders.

We conducted extensive evaluations on a variety of publicly available datasets and synthetic environments, benchmarking our proposed approaches against state-of-the-art algorithms. Results demonstrate that both the classical and learning-based methods outperform existing solutions in terms of distance cost, path smoothness, and computational runtime. The proposed methods also show strong scalability and reproducibility across different problem instances.

ÖZET

MOBIL ROBOTLAR İÇIN KLASIK VE ÖĞRENME TABANLI ÇOKLU HEDEF SIRALAMA VE YOL PLANLAMA

ABDULLAH ALLUŞ

MEKATRONİK MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, TEMMUZ 2025

Tez Danışmanı: Prof. Dr. MUSTAFA ÜNEL

Anahtar Kelimeler: Mobil Robotlar, Yol Planlama, Çoklu Hedefli Yol Bulma, A-Star, Dönüştürücüler (Transformers)

Otonom mobil robotik alanında, robotun mekânsal olarak dağılmış hedef düğümler kümesini en uygun sırayla ziyaret etmesi gereken çoklu hedef sıralama ve yol planlama problemlerine yönelik verimli ve ölçeklenebilir çözümler sunmak hâlen önemli bir zorluk olarak varlığını sürdürmektedir. Bu çalışmada, söz konusu probleme yönelik iki farklı yaklaşım geliştiriyoruz. İlk yaklaşım klasik, geometrik temelli bir yöntemken; ikinci yaklaşım, problemi ya geleneksel makine öğrenimi teknikleriyle ya da dönüştürücü (transformer) tabanlı yöntemlerle ele alan öğrenme tabanlı bir yaklaşımdır.

Klasik yaklaşımda, geleneksel mesafe metriklerine olan bağımlılığı azaltmak amacıyla, hedef ziyaret sıralarını geometrik açıdan çıkarımsal olarak belirlemeye imkân tanıyan “bir mesafe – iki açı” paradigmasına dayalı yeni bir sıralama stratejisi öneriyoruz. Bu sıralama yöntemi, ızgara tabanlı ortamlarda sıklıkla ortaya çıkan gereksiz zikzakları ve ara noktaları ortadan kaldırmak amacıyla bilgisayar grafiklerinden alınan prensipleri entegre eden, geliştirilmiş bir A* algoritmasıyla birleştirilmiştir. Böylece, hesaplama verimliliğinden ödün vermeden daha düzgün ve maliyet açısından daha verimli yollar elde edilmektedir. Buna ek olarak, hedef düğümlerin ziyaret sıralarını tahmin edebilen iki farklı öğrenme tabanlı çerçeve sunuyoruz.

İlki, cebirsel olarak elde edilen optimal çözümlerden türetilmiş, uzaklıklar, açılar ve mekânsal ilişkiler gibi geometrik desenleri yakalayan elle tasarlanmış öznitelikler ile eğitilmiş geleneksel bir makine öğrenimi modelidir. İkincisi ise, Evrişimli Sinir Ağları (CNN) ve İlişkisel Dönüştürücüler (Relational Transformers) kullanılarak çıkarılan özniteliklerle birlikte optimal yollardan elde edilen geometrik bağlam öznitelikleriyle eğitilmiş bir dönüştürücü modelidir. Yaptığımız deneyler, her iki modelin de daha önce görülmemiş ortamlara ve büyük ölçekli senaryolara etkili bir şekilde genelleme yapabildiğini ve optimal sıralamaya yakın sonuçları yüksek doğrulukla üretebildiğini göstermektedir.

Geliştirdiğimiz yaklaşımları, kamuya açık çok sayıda veri kümesi ve sentetik ortamlar üzerinde kapsamlı biçimde değerlendirdik ve mevcut en gelişmiş algoritmalarla karşılaştırmalı analizler gerçekleştirdik. Sonuçlar, hem klasik hem de öğrenme tabanlı yöntemlerin; mesafe maliyeti, yol düzgünlüğü ve hesaplama süresi açısından mevcut çözümleri geride bıraktığını göstermektedir. Ayrıca, önerilen yöntemlerin farklı problem örnekleri arasında güçlü bir şekilde ölçeklenebilir ve tekrarlanabilir olduğu da gözlemlenmiştir.

To my dearest and most cherished, my noble father, the honourable martyr Mohammad Zahran Aloush—this work is dedicated in loving memory and enduring gratitude. May Allah's mercy and blessings be upon him.

ACKNOWLEDGEMENTS

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

First and foremost, Alhamdulillah—all praise is due to Allah, by whose mercy and grace good deeds are completed. His blessings, guidance, and kindness have carried me through this journey, and without His support, none of this would have been possible.

I would like to express my deepest gratitude and heartfelt appreciation to my dear Hocam and thesis advisor, Prof. Mustafa Ünel. From the very first lecture I attended at Sabancı University—Autonomous Mobile Robotics—I was truly captivated by his clarity of teaching and depth of knowledge. As the weeks passed, my admiration only deepened. When I asked him to be my advisor during the second semester, he accepted me generously, despite my late request. Under his mentorship, I received unwavering support and invaluable guidance. He consistently encouraged me to pursue research, push my limits, and strive for excellence—laying the foundation for a future in academia. I am truly grateful for the wisdom he shared and the trust he placed in me.

I am also thankful to Assoc. Prof. Kemalettin Erbatur for serving on my thesis jury. I had the privilege of taking one of his courses during my studies, and I benefited greatly from his expertise and insights.

My sincere thanks also go to Assist. Prof. Ertuğrul Bayraktar for being part of my jury. I took two courses with him during my undergraduate studies, and it was in one of his classes that my journey in academic research and publication truly began. For that spark, I remain grateful.

To my close friend, colleague, and roommate, Abdulmoati Diab—you have been more than a friend; you have been a true brother. From the early days of our bachelor's programme to the final stages of our master's, your companionship has been a source of joy and strength. Thank you for sharing this long and rewarding journey with me.

I am also thankful to my colleagues and labmates in the Mechatronics Laboratory (ME1093 Lab): İlhami Osman Karakurt, Barış Almaç, Emre Yavaş, Harun Tolasa, and my elder brothers and PhD mentors—Shawqi Mohammed Farea, Mehmet Emin Mumcuoğlu, and Muhammed Zemzemoğlu. Your support, guidance, and friendship created a collaborative and inspiring environment that I was fortunate to be part of.

Lastly, and most importantly, I owe an immeasurable debt of love and gratitude to my mother and siblings. They were the light in my darkest moments, the safe haven I turned to in times of fatigue and uncertainty. Their unconditional love, encouragement, and prayers sustained me through every challenge. No words can ever capture the depth of what they mean to me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The field of robotics extensively researches and develops robotic autonomy to mitigate human error, enhance task accuracy, and optimise resource utilisation. The autonomy of mobile robots fundamentally hinges on robust paradigms of task planning, path planning, and motion planning Alatise & Hancke (2020); Davoodi, Panahi, Mohades & Hashemi (2015); Piazzi, Bianco & Romano (2007); Sanchez-Ibanez, Pérez-del Pulgar & García-Cerezo (2021). A primary challenge within path-planning involves identifying the least costly route between two specified positions on a given map, accounting for variables such as distance, terrain roughness, and travel time. The coordinates of the starting and goal positions are typically provided by either the user or another intelligent autonomous algorithm, while the map itself delineates the coordinates of existing structures and obstacles. In our study, we utilise a pre-processed map, where navigable areas are depicted in white and obstacles in black. Such maps are commonly derived from real-world workspaces, but can also originate from digital environments like games or design applications.

In numerous real-world scenarios, including warehouse automation Digani, Sabattini, Secchi & Fantuzzi (2015); Kumar & Kumar (2018); Poudel (2013); Wurman, D'Andrea & Mountz (2008), general logistics Fragapane, De Koster, Sgarbossa & Strandhagen (2021); Scholz-Reiter, Windt & Freitag (2004); Wurll, Fritz, Hermann & Hollnaicher (2018), and rescue missions in hazardous zones Murphy, Kravitz, Stover & Shoureshi (2009); Vasilyev, Kashourina, Krasheninnikov & Smirnova (2015), mobile robots demand complete autonomy for intricate path-planning, often maintaining a human-in-the-loop for safety and oversight. One such challenge is multi-goal path planning, where the robot must visit multiple equally prioritised goal nodes exactly once and then return to its origin, forming a closed loop Janoš, Vonásek & Pěnička (2021) as shown in Figure 1.1. This task underscores the critical need for resource efficiency, especially when minimising agents is possible or when operations are in dangerous environments. Our objective is to optimise this closed path by identifying the most efficient sequence for visiting goal nodes, thereby maximising resource utilisation and ensuring the most accurate execution of assigned tasks.

We introduce novel solution methodologies across two distinct frameworks: classical and learning-based. Both dynamically order goal nodes, striving to achieve a near-optimal visiting sequence that minimizes total distance cost in a given multi-goal scenario. Furthermore, our approaches are engineered for computational efficiency, a crucial aspect since many real-world situations can change rapidly, demanding swift adaptive responses.

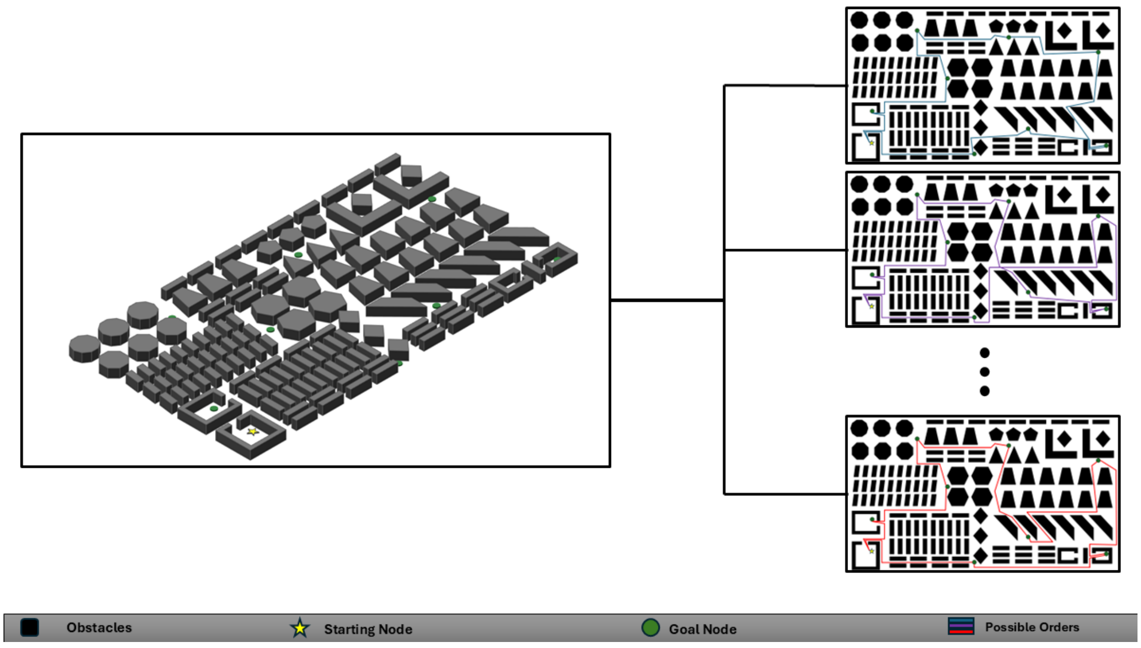


Figure 1.1 A 3D illustration that depicts a polymorphic, obstacle-filled workspace, featuring a complex environment with both obstacles and navigable regions. It showcases a multi-goal ordering and path planning problem, where a robot starts from a designated starting position, visits a set of goal locations, and then returns back to the starting position. The right-hand side presents 2D top-view maps, visually representing various possible visiting sequences for these goal positions.

At the heart of our proposed ordering approaches is an enhanced A* search algorithm, perfectly suited to the problem environments. We've improved the A* algorithm by incorporating essential image processing techniques to shorten the path during execution and post-prune it after generation. This enhancement minimises redundant nodes, manoeuvres, and abrupt changes, consequently reducing the overall distance cost.

To assess our proposed approaches, we performed extensive experiments on diverse maps with varying numbers of goal nodes. These evaluations allowed us to examine our algorithms' scalability, robustness, and reproducibility across unexpected and varied scenarios, including custom-designed maps, satellite port maps, and publicly available maps Bhardwaj, Choudhury & Scherer (2017); Sturtevant (2012). Moreover, we benchmarked our algorithms against current state-of-the-art solutions to underscore their potential as a competitive option in multi-goal path planning. Our results consistently show that our algorithms not only meet but frequently surpass the performance of existing methods, demonstrating their capability as contending approaches for ordering goal nodes in multi-goal path planning.

1.1 Problem Formulation

We hereby introduce the mathematical formulation of the multi-goal ordering and path planning problem:

The Multi-Goal Ordering and Path Planning Problem involves determining an optimal sequence of visits for a set of N goal nodes $\{G_1, G_2, ..., G_N\}$, starting and ending at a specified starting node S_0, where nodes are defined by their cartesian coordinates $\{(x_i, y_j) \mid i = 0, 1, ..., N\}$. For each sequence, a corresponding set of collision-free paths P_k must be generated connecting successive nodes. The primary objective is to minimise the total cumulative cost $\sum_{k=0}^{N} cost(P_k)$, subject to spatial constraints, obstacle avoidance, and kinematic constraints, path smoothness, within the given 2D map.

1.2 Thesis Contributions

Our work significantly advances the literature on multi-goal ordering and path planning, the A* algorithm, and the application of supervised machine learning and transformer models in this domain:

- First, we introduce a pre-processing framework for 2D workspace maps, designed for autonomous mobile robots. This framework involves generating mesh-like equidistant nodes, detecting obstacles, and constructing an efficient neighbourhood graph. This graph effectively represents the relationships between nodes, creating a suitable visual-graphical format for informed search algorithms like A*.
- Second, we propose a geometric enhancement to the A* algorithm utilising image processing techniques. The path generated by this improved A* is then refined through a similar post-pruning technique. This enhanced algorithm

demonstrates superior performance in reducing path distance and generating smoother, more robot-dynamics friendly paths with fewer turns and abrupt changes.

- Third, we present a novel classical algorithm for multi-goal node ordering, based on the guidance of a combination of local and global angles, aiming for a near-optimal visiting sequence. The order derived from our algorithm then undergoes post-pruning using modified, well-known algorithms. The resulting total paths, which visit all goal nodes, prove to be near-optimal, exhibiting minimal self-intersections and redundant manoeuvres.

- Fourth, we created two distinct datasets: one for 10-goal node scenarios and another for 7-goal nodes. For these datasets, the optimal visiting order of randomly selected goal node sets was determined either using an adapted Google's OR-Tools TSP solver or an enhanced brute-force technique.

- Fifth, we introduce a traditional machine learning-based method that dynamically orders goal nodes to approximate an optimal visiting sequence, thereby minimising overall distance cost. This supervised machine learning model is trained using meticulously designed, hand-crafted features, developed after extensive testing and observation of optimal paths obtained via enhanced brute-force methods across varying numbers of goal nodes and diverse map configurations. Crucially, our algorithm is optimised for computational efficiency, which is vital for real-world applications where environments can change rapidly, demanding swift adaptive responses.

- Sixth, we propose a transformer-based method for dynamic goal node ordering. This approach leverages a transformer model, trained on a generated dataset that utilises CNNs and relational transformers to produce information-rich, high-dimensional feature vectors for the goal nodes.

1.3 Thesis Outline and Organisation

The remainder of this thesis is structured as follows:

Chapter 2 offers a comprehensive literature review. It covers prominent path planning frameworks, environment representations, the fundamentals of robotic path planning, with a particular focus on the A* algorithm and its enhancements, and both classical and learning-based solutions for multi-goal ordering and path planning. The improved A* algorithm, which forms the core of our multi-goal ordering and path planning solutions, is introduced and thoroughly detailed in Chapter 3. Here, we elaborate on our enhancements to the algorithm, considering the working environment and experimental setup presented in the preceding chapter. Chapter 4 provides a detailed explanation of our proposed classical solution to the multi-goal ordering and path planning problem. This section introduces the novel "One-Distance-Two-Angles" paradigm and discusses the specific angular criteria chosen for ordering the goal nodes. In Chapter 5, we introduce two distinct Artificial Intelligence frameworks for tackling the multi-goal ordering and path planning problem: traditional machine learning and transformers. We delve into the preliminary assumptions, axioms, data generation processes, feature extraction methods, and the training and inference procedures used within each framework. Chapter 6 is dedicated to the verification and validation of our proposed approaches. We present various detailed experiments and comparative analyses, showcasing the results to clearly demonstrate the competence and effectiveness of our algorithms. Finally, Chapter 7 offers a summary of our study. It highlights the major contributions and key findings from our experiments and comparisons, while also discussing the limitations of our algorithms and outlining potential avenues for future work.

2. RELATED WORK ON MULTI-GOAL ORDERING AND PATH PLANNING

2.1 Fundamentals of Robotic Path Planning

Path planning is a fundamental aspect of mobile robot autonomy, broadly categorised into global and local path planning. The more research-intensive of the two is global path planning, which focuses on devising the shortest, most reliable path for a robot to reach a goal from a starting point while actively avoiding obstacles. This process relies on available environmental information, which can be represented in various ways. Common environment modelling methods include topological representations, graphical representations, which we will utilise in our work, and mixed representations. Researchers have long been dedicated to developing path planning techniques. Among the most widely adopted and effective approaches are cell decomposition techniques, sampling-based techniques, and graph-search techniques Alatise & Hancke (2020); Liu, Wang, Yang, Liu, Li & Wang (2023); Sanchez-Ibanez et al. (2021), the latter of which will be employed in this thesis.

2.2 Graphical Environment Representation and Path Planning

Graphical environment representation is a widely adopted method for digitally modelling physical workspaces, enabling the solution of various problems, including path planning. While diverse approaches exist for representing physical spaces using graphs, most entail designating navigable nodes to areas where a robot can traverse and non-navigable nodes to obstacles. These nodes, along with their attributes and relationships, are then stored within a neighbourhood graph . This structured representation allows algorithms to efficiently understand connectivity and identify barriers within the operational environment Masoudi & Fadel (2022); Wooden (2006). Various graph-based search algorithms are used to plan and find the shortest path. Informed search algorithms, such as Dijkstra's algorithm Alshammrei, Boubaker & Kolsi (2022); Dharmasiri, Kavalchuk & Akbari (2020); Dijkstra (2022); Liu, Lin, Yao, He, Zheng, Huang & Shi (2021); Qing, Zheng & Yue (2017); Wang, Yu & Yuan (2011) and the A* algorithm Erke, Bin, Yiming, Qi, Liang & Dawei (2020); Hart, Nilsson & Raphael (1968); Li, Hu, Wang & Du (2020); Tang, Tang, Claramunt, Hu & Zhou (2021); Wang, Wang, Qin, Wu, Duan, Li, Cao, Ou, Su, Li & others (2015); XiangRong, Yukun & XinXin (2021); Zhang, Chen & Zhang (2024), are predominantly employed due to their efficiency. Alternatively, some researchers utilize uninformed search algorithms like the RRT (Rapidly-exploring Random Tree) algorithm and its variants Feraco, Luciani, Bonfitto, Amati & Tonoli (2020); Hu, Cao & Zhou (2021); LaValle (1998); Mashayekhi, Idris, Anisi & Ahmedy (2020); Melchior & Simmons (2007), which are effective in high-dimensional spaces. More recently, there has been a growing focus on combining classical search algorithms with learning-based approaches to achieve more advanced and robust path-planning capabilities Gao, Ye, Guo & Li (2020); Sombolestan, Rasooli & Khodaygan (2019); Wang, Chi, Li, Wang & Meng (2020); Yu, Su & Liao (2020); Zhang, Cai, Yan, Yang & Hu (2024).

2.3 The A* Algorithm and Its Variants

A* is an informed search algorithm formulated for weighted graphs. Its primary objective is to discover the path with the minimal cost (e.g., least distance or shortest time) from a given starting node to a specified goal node within a graph. A* achieves this by maintaining a tree of paths originating from the starting node, incrementally extending these paths one edge at a time until the goal node is reached. In each iteration, A* strategically chooses which path to extend based on its current cost and an estimated cost to reach the goal. Specifically, it selects the path that minimises the function $f(n) = g(n) + h(n)$, where n represents the next node on the path. Here, $g(n)$ signifies the known cost from the starting node to n, and $h(n)$ is a problem-specific heuristic function estimating the cheapest path cost from n to the goal.

Researchers have long been interested in A*, dedicating considerable effort to developing improvements and variants suitable for diverse scenarios. For instance, Ju, Luo & Yan (2020) proposed an enhancement to the classical A* for situations where it fails to yield an optimal path, though their study lacked thorough experimental validation. Li et al. (2020) introduced an improved A* by expanding the neighbourhood definition to generate shorter paths with fewer inflection points and abrupt changes, but this increases computational complexity due to a larger search space.

Tang et al. (2021) presented a geometric enhancement to A* that addresses undesirable cross-path and sawtooth segments, simultaneously reducing expanded nodes, which shortens paths and decreases time complexity. Erke et al. (2020) developed an improved A* specifically tailored for autonomous land vehicles, addressing their unique navigational challenges. The study in XiangRong et al. (2021) proposed improvements to the classical A* algorithm to reduce memory consumption; however, they did not address the critical issue of reducing redundancies in the generated path that could hinder robot motion. Finally, Duchoň, Babinec, Kajan, Beňo, Florek, Fico & Jurišica (2014) introduced geometry-based modifications to the classical A* algorithm, primarily focusing on reducing its computational complexity, validating their improvements through experiments in SLAM robotic environments.

2.4 Multi-Goal Ordering and Path Planning

More complex path-planning tasks frequently arise in numerous real-world applications. Mobile robots operating in diverse environments like logistics hubs, warehouses, factories, libraries, museums, and parks often need to do more than simply find the shortest path between a start and a single goal. Instead, they must determine the most efficient visiting sequence for multiple goal nodes. This typically involves the robot starting at a specific location, visiting several designated goals, and then returning to its origin, all while minimising the total distance traveled. Solutions to this intricate problem in the literature generally fall into two primary categories:

- Classical approaches: Researchers in this domain address the problem using traditional methods such as dynamic programming, Traveling Salesperson Problem's (TSP) classical heuristics, expert systems, and geometry-based techniques. These methods rely on well-established algorithms and mathematical principles.
- Learning-based approaches: Here, researchers leverage artificial intelligence (AI) techniques. They develop various model structures trained on diverse datasets to determine optimal visiting orders, allowing the robot to learn complex relationships from data.

2.4.1 Classical Approaches

The multi-goal path planning problem extends beyond mobile robots; for instance, Wurll, Henrich & Wörn (1999) explored it in industrial robot settings, focusing on finding collision-free paths for robotic arms through a set of goal poses while minimising total path length. Our current study specifically addresses this problem within autonomous mobile robots.

Saeed, Reforgiato Recupero & Remagnino (2021) tackled the mobile robot multi-goal problem with a boundary node method, executing in two steps: a genetic algorithm for sequence generation, followed by the boundary node method itself for inter-goal pathfinding. This paper, however, lacks extensive experimentation in highly complex scenarios with many goal nodes or complex obstacles. In Hongyun, Xiao & Hehua (2013), the authors combined a branch-detected algorithm for path planning with Traveling Salesman Problem (TSP) solutions for ordering within a grid-space environment. Their work, however, also lacked intensive experimentation to demonstrate the algorithm's efficiency across diverse scenarios and maps.

Janoš et al. (2021) introduces Space-Filling Forest (SFF*), a sampling-based planner designed to compute efficient, collision-free paths between multiple targets in obstacle-filled environments for multi-goal path planning. By growing multiple RRT*-like trees from each target and optimising interconnections, SFF* produces shorter target-to-target paths, leading to improved TSP solutions in terms of overall travel cost, but their experiments were limited to scenarios with a maximum of 20 goal nodes. Best, Faigl & Fitch (2016) presents a self-organising map (SOM) algorithm for multi-goal path planning in active perception and data collection tasks, where robots aim to maximise observation rewards within travel budget constraints. By jointly selecting nodes and sequencing sensing locations in overlapping continuous viewpoint regions, yet their experiments provided insufficient detail on handling map obstacles. Similarly, Faigl & Hollinger (2014) proposes a self-organising map (SOM) framework for solving variants of the multi-goal path planning problem, particularly in autonomous data collection scenarios involving stationary sensors. The approach accommodates flexible constraints such as sensor prioritisation and proximity-based data collection, outperforming traditional TSP heuristics in both simulated and real-world underwater monitoring applications, but did not explicitly address obstacle avoidance.

Devaurs, Siméon & Cortés (2014) proposed Multi-T-RRT, a multiple-tree extension of the Transition-based RRT algorithm, designed to address ordering-and-pathfinding problems by navigating continuous cost spaces without relying on symbolic task planners. The method is validated through comparisons with other planners and is successfully applied to an industrial aerial inspection task involving unordered waypoint traversal. However, their experiments did not emphasise closed-path scenarios and lacked thorough scalability analysis. Pěnička, Faigl & Saska (2019) tackled the Physical Orienteering Problem (POP), a variant tailored for multi-goal data collection where a robot must maximise rewards by visiting a subset of targets within a limited travel budget and an obstacle-filled environment. The proposed VNS-PRM* algorithm combines variable neighbourhood search with a progressively refined PRM* roadmap, achieving high-quality, collision-free paths suitable for applications such as UAV-based data collection in urban settings. Huang, Tan, Lee, Desaraju & Grizzle (2023) presented an anytime iterative framework that jointly addresses multi-objective path planning and destination ordering using a combination of an informable multi-directional RRT* algorithm and a hybrid solver integrating enhanced cheapest insertion and genetic algorithms. The system efficiently handles complex waypoint-based tasks in real-world driving scenarios and demonstrates scalability through multi-threaded implementation and evaluation on large graphs, but their work largely focused on large-scale maps without directly addressing critical issues like obstacle avoidance and robot dynamics.

In our prior work Allus, Diab & Bayraktar (2024), we introduced an A*-based multi-goal ordering and path planning algorithm. However, it suffered from requiring multiple A* executions within the ordering procedure and was restricted to local optimization, neglecting global map aspects. Consequently, we developed a new angle-based ordering paradigm that leverages global geometrical information and eliminates the need for multiple A* executions by using angles instead of distances, thereby overcoming these limitations.

We published our classical algorithmic framework in Allus & Unel (2025) under the title "Angle-based multi-goal ordering and path-planning using an improved A-star algorithm".

2.4.2 Learning-Based Approaches

Given that the multi-goal ordering problem in robotics path planning can be seen as a variation of the Travelling Salesperson Problem (TSP), we'll now discuss relevant learning-based solutions from the literature.

In Min, Bai & Gomes (2023), the authors propose UTSP, an unsupervised learning framework. This framework employs a Graph Neural Network (GNN) to predict edge probabilities for TSP, subsequently using local search to construct the final route. The work in Mele, Gambardella & Montemanni (2021) introduces a machine learning-based constructive heuristic for TSP. It leverages candidate lists to shrink the solution space and guide edge selection, allowing scalability to very large instances. By focusing the machine learning model on confirming likely optimal edges, this method demonstrates strong generalization and competitive performance against classical heuristics, even for problems with thousands of cities.

Joshi, Laurent & Bresson (2019) presents a non-autoregressive deep learning approach for solving the Euclidean TSP, utilizing Graph Convolutional Networks (GCNs) and parallelized beam search. This method outperforms recent autoregressive models in terms of solution quality, inference speed, and efficiency, though it still lags behind traditional Operations Research solvers. Lastly, Bresson & Laurent (2021) adapts the Transformer architecture to solve the TSP using reinforcement learning, eliminating the need for ground truth solutions during training. This approach achieves state-of-the-art performance on TSP benchmarks, significantly reducing the optimality gap compared to previous learned heuristics.

These learning-based approaches, while primarily developed for TSP, offer valuable insights and methodologies that can be adapted and extended to address the multi-goal ordering challenges in robotic path planning, particularly when dealing with complex or large-scale scenarios where traditional methods might struggle with computational efficiency or optimality.

3. IMPROVED A* ALGORITHM

The A* algorithm serves as the fundamental framework for all path planning strategies employed in this work. It was chosen due to its classification as an informed graph search algorithm, which allows it to efficiently guide the search process using heuristic information. Given that the input map is assumed to be well-structured and appropriately scaled—and will subsequently be converted into a graph-based representation—the A* algorithm proves to be the most suitable choice for our application. Its balance between optimality and computational efficiency makes it especially effective in environments with predictable geometric properties.

3.1 Graphical Environment Setup

This section outlines the process of converting an input map from an image format into a graphical representation suitable for navigation algorithms.

The workspace environment represents a real-world 3D space, heavily populated with obstacles that restrict a robot's movement. Figure 3.1 demonstrates the appropriate input format, which is created by transforming a polymorphic, obstacle-dense 3D workspace model into a 2D top-view map. This image-based format depicts obstacles in black and navigable areas in white. Both types of regions are represented using a mesh grid of equidistant nodes, with each node assigned a specific navigability flag (non-navigable for obstacles, navigable for free space). Therefore, each obstacle is defined by a collection of non-navigable equidistant nodes. It is crucial to meticulously determine the 2D map's scale [meter/pixel] to ensure that our algorithms' output can be smoothly integrated with a real-world robot's localisation system. While this 3D-to-2D transformation typically involves computer vision and image processing techniques, our algorithms are not limited to maps derived from physical environments; they are equally compatible with maps generated from virtual settings, such as games and graphical user interfaces (GUIs).

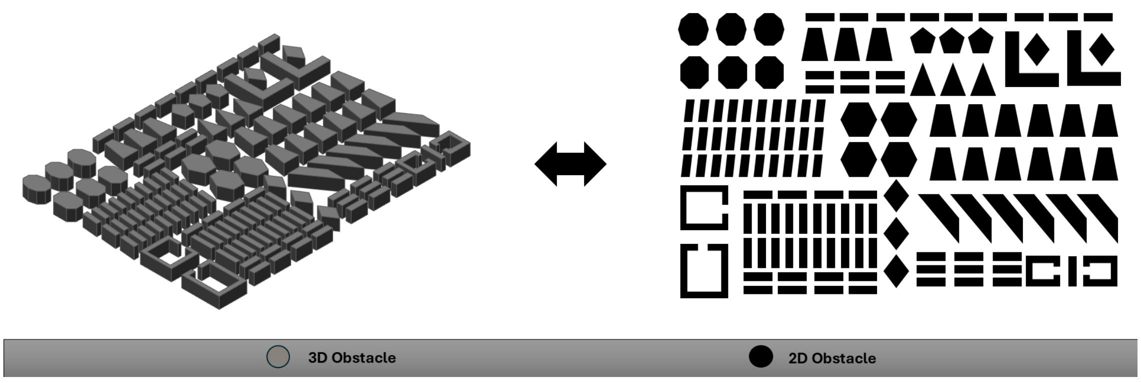
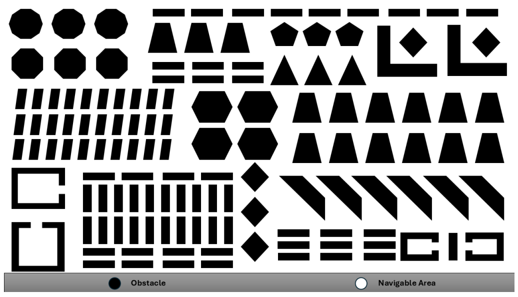
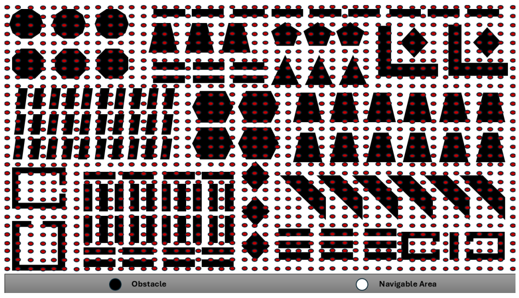


Figure 3.1 Transformation from a polymorphic, obstacle-dense 3D environment to a 2D top-view map. The map is presented in the precise format required for seamless compatibility with our proposed path planning algorithms.

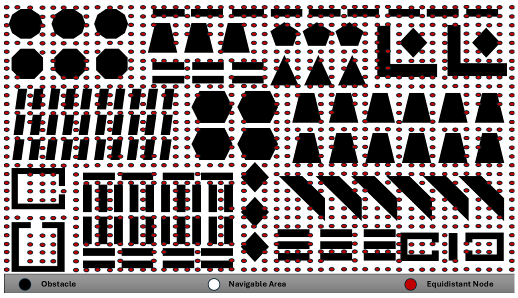
To prepare a 2D input map for navigation algorithms, especially A*, it is crucial to convert its image format into a suitable graphical representation. This involves storing the image's grayscale values as a matrix. Subsequently, down-sampling this matrix is necessary to lower computational costs in later stages, thereby boosting the algorithm's overall efficiency. Figure 3.2 illustrates the entire pre-processing stage, detailing steps from the initial raw 2D map to a final pre-processed version ready for path planning and search algorithms.



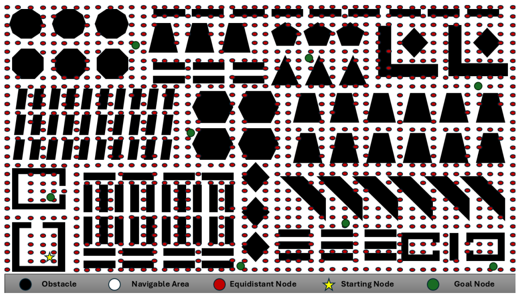
(a) An image that shows the initial processing of the input map to ensure compatibility with our proposed algorithms.



(b) A mesh of equidistant nodes is generated, with spacing defined by the user.



(c) Illustration of the filtering process, where nodes overlapping with obstacles are removed.



(d) A neighbourhood relation graph is constructed, connecting equidistant and user-defined nodes.

Figure 3.2 A figure that summarises the Pre-processing Stage's key steps: Equidistant Nodes Generation, Obstacle Detection, and Neighbourhood Graph Generation.

3.1.1 Equidistant Nodes Generation, Obstacle Detection and Equidistant Nodes Filtering

Figure 3.2b illustrates the creation of an equidistant node grid based on a user-defined spacing value, $\boldsymbol{\Delta}$. It is crucial that $\boldsymbol{\Delta} \leq d_{min}$, where d_{min} is the smallest dimension of the smallest obstacle on the map. This grid, $N = \left\{(x_i, y_j) \mid x_i = i \cdot \Delta,\ y_j = j \cdot \Delta,\ 1 \leq i \leq \frac{w}{\Delta},\ 1 \leq j \leq \frac{h}{\Delta}\right\}$, acts as a down-sampled matrix of the workspace (with w as width and h as height). Each vertex, (x_i, y_j), is an equidistant node vital for defining potential robot paths.

The choice of $\boldsymbol{\Delta}$ involves a trade-off between accuracy and computational complexity. The number of equidistant nodes, $|N| = round(\frac{w \cdot h}{\boldsymbol{\Delta}^2})$, is inversely related to $\boldsymbol{\Delta^2}$. A smaller $\boldsymbol{\Delta}$ increases node count, enhancing obstacle representation but also raising computational load for search algorithms. Conversely, a larger $\boldsymbol{\Delta}$ reduces computation at the cost of accuracy. To balance this, we typically set $\boldsymbol{\Delta} = d_{min}$, ensuring sufficient obstacle precision with minimal computational burden. A sensitivity analysis shows that halving $\boldsymbol{\Delta}$ quadruples nodes, significantly increasing complexity, while doubling $\boldsymbol{\Delta}$ quarters nodes, boosting efficiency but reducing accuracy. This highlights the importance of an optimal $\boldsymbol{\Delta}$ for precision and efficiency.

Each equidistant node is then evaluated by its grayscale intensity. Nodes above a user-defined threshold are marked as navigable, while those below it are flagged as obstacles. Figure 3.2c demonstrates this filtering, ensuring only nodes above the threshold are considered navigable for path-planning. For more precise obstacle definition, connected components analysis Suzuki & others (1985) can detect obstacle contours, providing exact boundary information for integration into our framework.

These preprocessing steps significantly boost the reliability and effectiveness of our navigation algorithms by ensuring thorough obstacle comprehension without excessive computational demand, and by restricting all subsequent navigation computations exclusively to navigable nodes.

3.1.2 Neighbourhood Relations' Graph and User-Defined Nodes

One of the most important preparations for the execution of search algorithms is to define the neighbourhood relationships between equidistant nodes, converting the input map into a graphical representation suitable for navigation algorithms. Typically, each node (x_i, y_j) can have up to eight neighbours: four directly adjacent (cardinal directions) with an edge cost of $d_{edge} = \Delta$, and four diagonally adjacent with a diagonal cost of $d_{diag} = \sqrt{2} \cdot \Delta$. Nodes bordering obstacles are assigned an infinite neighbouring cost to prevent traversal. Additionally, edge cases like map boundaries will naturally have fewer than eight neighbours. Figure 3.3a visually demonstrates these neighbourhood relationships among equidistant nodes, with the algorithmic specifics detailed in Algorithm 1.

Algorithm 1: Define Neighbourhood Relations For Equidistant Nodes

```
Input  : N, $\Delta$
Output: N
i ← 0;
j ← 0;
foreach row in N do
    foreach node in row do
        $\Delta \leftarrow \infty$ if not node.isNavigable else 1;
        neighbours ← $[(i,j)\mathbf{for}\, i\, \mathbf{in}[-1,0,1]\mathbf{for}\, j\, \mathbf{in}[-1,0,1]\mathbf{if}(i,j) \neq (0,0)]$;
        foreach (di, dj) in neighbours do
            distance ← $\Delta$ if di == 0 or dj == 0 else $\sqrt{2} \cdot \Delta$;
            node.addNeighbour(N[row + di][col + dj], distance);
return N;
```

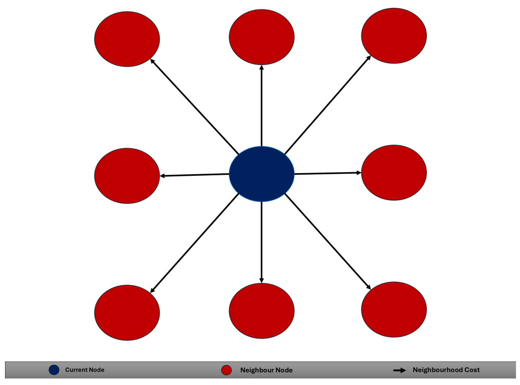
User-defined nodes offer a mechanism to manually integrate specific points of interest or constraints into the graph, thereby complementing the automatically generated equidistant nodes. These nodes, typically serving as starting or goal positions, establish their neighbourhood relationships through a brute-force search within a maximum allowable neighbourhood cost, $MANC = \sqrt{2} \cdot \Delta$. This process ensures connectivity exclusively to navigable neighbours, as illustrated in Figure 3.3b. Equation 3.1 is then used to calculate the connection cost based on their spatial proximity:

$$d_{UD} = \sqrt{(x_i - x_{UD})^2 + (y_j - y_{UD})^2} \tag{3.1}$$

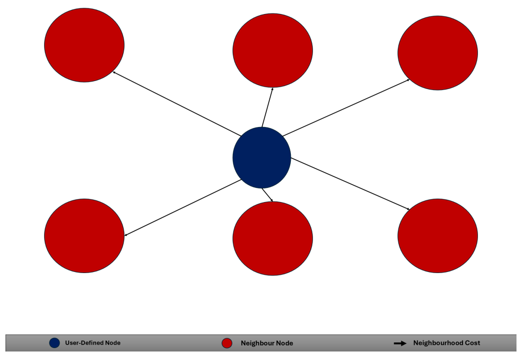
Algorithm 2 details the process for generating these nodes according to user specifications. Each node's navigability is assessed by comparing its grayscale value against a defined obstacle threshold. If a node is determined to be navigable, it is added to the navigable nodes vector (AN[0]); otherwise, it is assigned to the obstacle nodes vector (AN[1]).

Algorithm 2: Generate User Defined Nodes

```
Input  : GI, OT, UDNC, MANC, AN
// GI: Grey Image, OT: Obstacle Threshold, UDNC: User-Defined Nodes Coordinates, MANC: Maximum Allowable
   Neighbouring Cost, AN: All Nodes where AN[0] is the set of navigable nodes and AN[1] is the set of
   non-navigable nodes.
Output: GUDN
// GUDN: Generated User-Defined Nodes
GUDN ← [];
foreach nodeCoordinate in UDNC do
    if GI.getpixel(nodeCoordinate) < OT then
        isNavigable ← False;
    else
        isNavigable ← True;
    tempNode ← Node(nodeCoordinate, isNavigable);
    if tempNode not in AN[0] and tempNode not in AN[1] then
        if isNavigable then
            tempNode.addUserDefinedNeighbour(AN[0], MANC);
            AN[0].append(tempNode);
        else
            tempNode.addUserDefinedNeighbour(AN[1], MANC);
            AN[1].append(tempNode);
    GUDN.append(tempNode);
return GUDN;
```



(a) Neighbourhood structure of equidistant nodes, where each node can have up to 8 neighbouring connections based on equidistant spacing.



(b) Neighbourhood structure of user-defined nodes, established using a brute-force search approach with a maximum of 8 potential neighbours.

Figure 3.3 Comparison of neighbourhood graphs: (a) uses equidistant spacing to define neighbouring relations, while (b) applies a brute-force search to determine the neighbours of user-defined nodes, both allowing up to 8 connections.

3.2 The A* Algorithm

The A* algorithm serves as the central component of our path-planning methodology, tasked with finding the shortest route between two nodes using predefined costs and heuristics, as illustrated in Figure 3.4. In our adapted version, we have enhanced the classical A* to seamlessly integrate with our input map's graphical representation. This involves utilizing the neighbourhood relations graph to efficiently calculate node-to-node costs. Our chosen heuristic is the Euclidean distance, which is particularly effective as it naturally accommodates diagonal movement, a crucial capability in environments where such traversals between nodes are permissible.

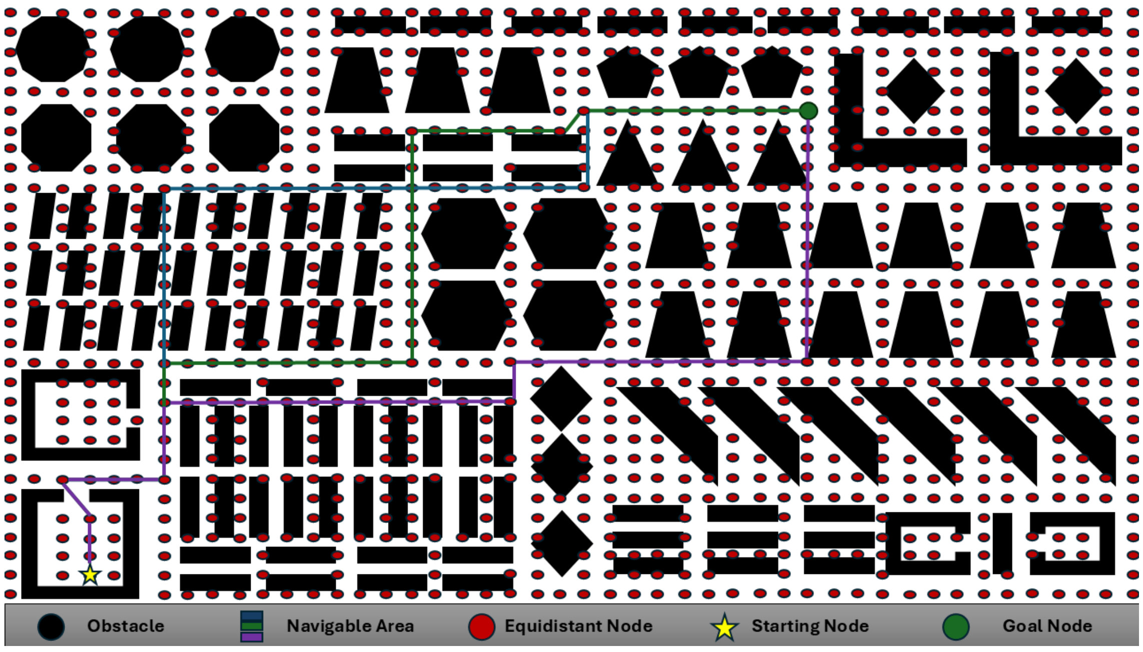


Figure 3.4 An illustration of several possible paths from a starting node to a goal node, successfully navigating around obstacles. The core function of the A* algorithm is to identify the shortest path among all viable options.

As depicted in Algorithm 3, the A* algorithm initiates its search from a starting node towards a goal node, evaluating potential paths using calculated costs and heuristic estimates. Crucially, if a path for the same node pair has been previously computed, it is retrieved from memory. This memory retrieval step significantly speeds up computation and conserves resources, especially vital in static environments like warehouses or well-structured engineered spaces. In such settings, after repeated traversals, frequently used paths can be directly accessed from memory, bypassing recalculation.

Algorithm 3: Improved A* Algorithm

```
Input  : SN, GN, GPs, GI, IPP
// SN: Starting Node, GN: Goal Node, GPs: Generated Paths, GI: Grey Image, IPP: Is Post Pruned
Output: PC
// PC: Path and Total Cost
PFPC ← GPs.get((SN, GN), None);
// PFPC: Previously Found Path and Cost
if PFPC ≠ None then
    return PFPC;
openSet ← [SN], closedSet ← [];
g ← {}, g[SN] ← 0;
parents ← {}, parents[SN] ← SN;
subPath ← None;
while length of openSet > 0 do
    n ← None;
    foreach temp in openSet do
        if n == None or g[temp] + calculateHeuristics(temp, GN) < g[n] + calculateHeuristics(n, GN) then
            n ← temp;
    if n == GN or length of n.neighbours == 0 then
        pass;
    else
        foreach (m, cost) in n.neighbours.items() do
            if m not in openSet and m not in closedSet then
                openSet.append(m);
                parents[m] ← n, g[m] ← g[n] + cost;
            else
                if g[m] > g[n] + cost then
                    g[m] ← g[n] + cost, parents[m] ← n;
                    if m in closedSet then
                        closedSet.remove(m);
                        openSet.append(m);
    if n == None then
        return None;
    if subPath ≠ None then
        n ← subPath[0][0], parents[goalNode] ← n, g[goalNode] ← g[n] + subPath[1], n ← GN;
    if n == GN then
        path ← [];
        while parents[n] ≠ n do
            path.append(n);
            n ← parents[n];
        path.append(SN);
        TC ← g[GN], pathCopy ← path.copy();
        pathCopy.insert(0, GN);
        path.reverse();
        // TC: Total Cost
        if IPP then
            path ← postPrunePath(path);
        GPs[(GN, SN)] ← [pathCopy, TC], GPs[(SN, GN)] ← [path, TC], PC ← [path, TC];
        return PC;
    openSet.remove(n);
    closedSet.append(n);
    rho ← $\sqrt{((GN.coordinates[0] - n.coordinates[0])^2 + (GN.coordinates[1] - n.coordinates[1])^2}$;
    if isLineObstacleFree(n, GN, GI) then
        subPath ← [[n, GN], rho];
```

3.2.1 Time and Space Complexity of The A* Algorithm

Let V represent the number of navigable nodes on the map, and k be the average number of neighbors per node. In a standard A* application, the worst-case time complexity involves visiting all nodes. The main loop runs $O(|V|)$ times. Each iteration requires finding the node n with the minimum $f(n)$, which contributes $O(|V|)$ to the complexity, and iterating through n's neighbours to update information, contributing $O(k)$. Thus, the approximate overall complexity for A* is $O(|V|^2 + |V| \cdot k)$.

However, since the number of neighbors per node is bounded (8 in our implementation), the term $O(|V| \cdot k)$ becomes effectively constant relative to V.Therefore, the overall time complexity of the A* algorithm can be approximated as $O(|V|^2)$.

Regarding space complexity, the algorithm utilizes several data structures—including the open set, closed set, cost dictionary, parent dictionary, and the reconstructed path—each storing at most $O(|V|)$ elements in the worst case. Consequently, the overall space complexity of the A* algorithm is $O(|V|)$.

3.2.2 Straight Line Navigation Enhancement

Our primary enhancement to the A* algorithm focuses on maximising the use of straight-line segments. In other words, the algorithm should reduce the number of unnecessary zigzags in the generated paths. When our improved A* is tasked with finding the shortest path between two nodes, say $[(x_S, y_S), (x_G, y_G)]$, its initial step is to check for obstacle-free connectivity using Bresenham's line algorithm Bresenham (1977). If no obstacles are detected—meaning all pixels along the line from $[(x_S, y_S), (x_G, y_G)]$ are white (navigable)—the algorithm directly returns this straight line as the shortest path.

However, if an obstacle is encountered, the classical A* algorithm proceeds as usual. Yet, with each new node added to its closed list, our improved algorithm re-checks for obstacle-free connectivity to the goal node. As long as obstacles persist, the classical A* continues its operation. But, critically, once an obstacle-free connection is found between the current node and the goal node, the remaining path is determined to be a straight line. This strategic shortcut significantly reduces the number of iterations required by the classical A* algorithm, thereby enhancing the generated path in terms of overall cost and computational efficiency.

3.2.3 Post-Pruning The Generated Path

The inherent nature of the A* algorithm, operating on discrete graphical representations of workspaces, often results in redundant path segments. Because the algorithm is confined to navigating between a maximum of 8 neighbouring nodes, it can execute necessary but inefficient manoeuvres, increasing overall path costs. This limitation frequently forces A* to zigzag between equidistant nodes to circumvent obstacles, producing paths with many intermediate nodes that are often superfluous.

Therefore, we implement a post-pruning process to eliminate two categories of redundancies:

- **Distance redundancies**: These stem directly from the A* algorithm's neighbour-based movement constraints.
- **Useless intermediate node redundancies**: These refer to unnecessary nodes that do not alter the fundamental structure of the path.

To address useless intermediate nodes, we remove all path nodes lying on a straight line between two key nodes. Specifically, for any three consecutive path nodes, (x_k, y_k), (x_{k+1}, y_{k+1}), and (x_{k+2}, y_{k+2}), if they satisfy the collinearity condition:

$$(x_{k+2} - x_k)(y_{k+1} - y_k) = (y_{k+2} - y_k)(x_{k+1} - x_k) \tag{3.2}$$

then the intermediate node (x_{k+1}, y_{k+1}) is deemed redundant and is removed. This iterative process is applied to all consecutive triplets along the path.

Additionally, we tackle distance redundancies by checking for shorter, obstacle-free path segments using Bresenham's line algorithm Bresenham (1977). For any pair of non-consecutive nodes (x_k, y_k) and (x_l, y_l), if Bresenham's algorithm confirms a clear path between them, the direct straight Euclidean line: $d_l = \sqrt{(x_l - x_k)^2 + (y_l - y_k)^2}$ replaces all intermediate nodes originally between (x_k, y_k) and (x_l, y_l). This ensures the most direct path and eliminates unnecessary zigzagging, leading to a more efficient trajectory.

By combining these two strategies—removing unneeded intermediate nodes and substituting redundant zigzag patterns with straight segments—the final path is significantly optimised in terms of both node count and total distance.

3.2.3.1 Time and Space Complexity of The Post-Pruning Process

Let V be the number of navigable nodes and P the number of nodes in the returned path. The post-pruning process involves checking each pair of nodes in the returned path, which takes $(O(P^2))$ time. Each check uses Bresenham's algorithm, having a complexity linear to the distance between points, $(O(P))$. Consequently, the worst-case time complexity of the post-pruning procedure is approximately $O(P^3)$. Therefore, the overall time complexity for the improved A* algorithm can be approximated as $O(|V|^2 + P^3)$ in the worst case. However, in typical scenarios, P is considerably smaller than $|V|$, so the overall complexity of the improved A* algorithm remains roughly $O(|V|^2)$.

In terms of space complexity, the algorithm maintains the pruned path, which in the worst case could store all P nodes from the original path, resulting in an $(O(P))$ space requirement. Additionally, the function uses only a few auxiliary variables for indexing, which do not significantly contribute to the space complexity. Therefore, the overall space complexity of the post-pruning procedure is $(O(P))$.

4. CLASSICAL MULTI-GOAL ORDERING AND PATH PLANNING

This chapter introduces our novel classical angle-based approach for efficiently ordering multiple goal nodes. We start by detailing our fundamental assumptions and offering geometric insights into our methodology. Following this, we lay out the foundational principles guiding our algorithmic design. Finally, we present the comprehensive formulation of our innovative angle-based ordering paradigm.

4.1 Preliminaries and Assumptions

Discovering the precise optimal solution for an ordering problem with N_G goal nodes is computationally demanding, exhibiting a factorial complexity of $N_G!$. While this is manageable for small values of N_G, computational limitations quickly arise in larger problems. Here, even advanced technologies might take days to compute the optimal solution using brute-force methods. In our experiments, an enhanced brute-force enumeration was only feasible for up to 13 goal nodes; beyond this, computational constraints made it prohibitive. To analyse the structure and behaviour of optimal paths, we conducted extensive experiments using enhanced brute-force techniques across various goal node distributions and maps.

A crucial insight from these experiments reveals that in optimal paths, the next goal node,$(x^*_{G_{i+1}}, y^*_{G_{i+1}})$, chosen from the current goal node, $(x^*_{G_i}, y^*_{G_i})$, is typically one of the two nearest goal nodes in terms of actual traversed distance (not simply Euclidean distance). These are the potential next goal nodes: $(x^{\text{Pot}}_{G_{i+1}}, y^{\text{Pot}}_{G_{i+1}})$. This aligns intuitively with human navigation, where proximity often dictates immediate choices. This finding highlights the effectiveness of dynamically ordering goal nodes, structuring the algorithm to iteratively select the best next goal based on the current node's context within the overall expected path.

Dynamically ordering nodes means the algorithm's structure relies on a current goal node, (x_{G_i}, y_{G_i}), and a set of potential next goal nodes, $(x^{\text{Pot}}_{G_{i+1}}, y^{\text{Pot}}_{G_{i+1}})$. This same ordering process repeats up to N_G times to find a near-optimal sequence. Essentially, we order one goal node at a time.

While dynamic ordering offers advantages, it introduces a "cost of locality". By concentrating on immediate neighbour goal nodes during the ordering process, we risk compromising global path optimality. To counter this, our approach incorporates concepts designed to infuse the dynamic ordering procedure with global insights. These concepts form the foundational pillars of our proposed methodology, which we'll detail in subsequent sections.

Essentially, our methodology opts for ordering one goal node at a time, basing each decision on the current goal node's context. This ensures that while local proximity is considered, we simultaneously strive to maintain global path coherence. This balanced approach aims to leverage the efficiency of local decision-making while integrating strategic insights to improve overall path quality.

4.2 Closed Path Concept and Centroid Utilisation

A core global concept in our method is the closed path formation. The robot begins at a starting node, (x_{G_0}, y_{G_0}), visits all specified goal nodes, $[(x_{G_1}, y_{G_1}), ..., (x_{G_{N_G}}, y_{G_{N_G}})]$, and then returns to the starting node, $(x_{G_{N_G+1}}, y_{G_{N_G+1}}) = (x_{G_0}, y_{G_0})$. This forms a complete loop and provides the geometric intuition for our approach, especially through the use of the centroid of the goal nodes.

Our analysis of optimal paths derived from brute-force revealed that well-ordered sequences often create a circular pattern around a central point. This led us to use the centroid of the goal nodes, (x_{G_C}, y_{G_C}), as a crucial reference. The centroid serves as a guide for the incremental, circular traversal of nodes. We follow a counterclockwise convention, though clockwise is equally viable.

The centroid for N_G goal nodes and the starting node is calculated using their global coordinates as:

$$(x_{G_C}, y_{G_C}) = \left(\frac{\Sigma_{i=0}^{N_G} x_{G_i}}{N_G + 1}, \frac{\Sigma_{i=0}^{N_G} y_{G_i}}{N_G + 1}\right) \tag{4.1}$$

This calculation gives an approximate centroid based on the discrete sample points (the goal nodes and starting node). Even as an approximation, it provides a dependable basis for our strategy.

In short, the closed path concept introduces two vital global factors for our dynamic ordering: the starting node (which doubles as the final node) and the centroid (which steers the overall circular movement). This strategy helps us find a near-optimal and closed path for efficient robot navigation through all goal nodes.

4.3 Utilising Angles for Efficient Goal Node Ordering

Effectively combining local and global ordering strategies demands computational efficiency, especially for both offline and online applications. Given the presence of obstacles, solely relying on distance measurements would necessitate repetitive and computationally expensive A* algorithm executions, which is impractical. To address this, we developed the One-Distance-Two-Angles strategy.

Since our assumptions indicate the next optimal node is typically one of the two closest, this method minimises A* algorithm calls. It uses A* only once to identify the three nearest nodes to the current node based on distance cost. Among these, the farthest node is designated as a temporary goal, while the other two are considered potential next nodes. The centroid coordinates and the starting node coordinates are crucial global ordering factors, as they help determine relevant angles for the ordering process.

The One-Distance-Two-Angles strategy reduces reliance on distance calculations for next-node selection, favouring angles, which are significantly more time-efficient. Instead of three full distance measurements, this approach uses one set of distances (between the current node and its three nearest neighbour goal nodes). Additionally, it incorporates two sets of angles that relate the current node to the possible next nodes, based on distinct global and local criteria. These criteria are captured by three specific angles: α, β and γ.

Figure 4.1 visually represents the angles α, β and γ. These angles are crucial for assessing the relationships between the current node and potential next nodes, all in reference to specific key points.

- **Angle** α measures the relationship between the current node and each potential next node relative to the centroid. This offers a crucial global geometric perspective for node ordering by establishing a consistent measure against the fixed centroid. As discussed in Section4.2, an optimal path tends to revolve around the centroid, making it a reliable reference for selecting the next node in each iteration. It is computed via the dot product of vectors:

$$\alpha = \arccos\left(\frac{\mathbf{r}\cdot\mathbf{s}}{|\mathbf{r}|\cdot|\mathbf{s}|}\right) \tag{4.2}$$

- **Angle** β defines the relationship between the current node and each possible next node with respect to the starting node. This provides an additional global geometric perspective on node ordering, establishing a measure against the absolute starting node. Given that an optimal solution ultimately concludes at the starting node (as detailed in Section4.2), this ensures the starting node remains a consistent reference for selecting the next node at each iteration. It is calculated using the formula:

$$\beta = \arccos\left(\frac{\mathbf{t}\cdot\mathbf{u}}{|\mathbf{t}|\cdot|\mathbf{u}|}\right) \tag{4.3}$$

- **Angle** γ captures the relationship between the current node and each possible next node concerning the temporary goal node. This angle provides vital local geometric information for the ordering algorithm by establishing a direct measure between the potential next nodes and the temporary goal. This allows the algorithm to intuitively estimate the direction or trend of the upcoming goal nodes. It is computed as:

$$\gamma = \arccos\left(\frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}|\cdot|\mathbf{w}|}\right) \tag{4.4}$$

The vectors used in these computations are established as follows:

$$\mathbf{r} = (x_{G_i} - x_{G_C}, y_{G_i} - y_{G_C}), \quad \mathbf{s} = (x^{\text{Pot}}_{G_{i+1}} - x_{G_C}, y^{\text{Pot}}_{G_{i+1}} - y_{G_C})$$

$$\mathbf{t} = (x_{G_i} - x_{G_0}, y_{G_i} - y_{G_0}), \quad \mathbf{u} = (x^{\text{Pot}}_{G_{i+1}} - x_{G_0}, y^{\text{Pot}}_{G_{i+1}} - y_{G_0})$$

$$\mathbf{v} = (x^{\text{Pot}}_{G_{i+1}} - x_{G_i}, y^{\text{Pot}}_{G_{i+1}} - y_{G_i}), \quad \mathbf{w} = (x^{\text{temp}}_{G_{i+3}} - x_{G_i}, y^{\text{temp}}_{G_{i+3}} - y_{G_i})$$

Here, (x_{G_i}, y_{G_i}) are the coordinates of the current node, $(x^{\text{Pot}}_{G_{i+1}}, y^{\text{Pot}}_{G_{i+1}})$ refer to the coordinates of the two possible next goal nodes, and $(x^{\text{temp}}_{G_{i+3}}, y^{\text{temp}}_{G_{i+3}})$ represents the coordinates of the temporary goal node, which is the farthest of the nearest three nodes.

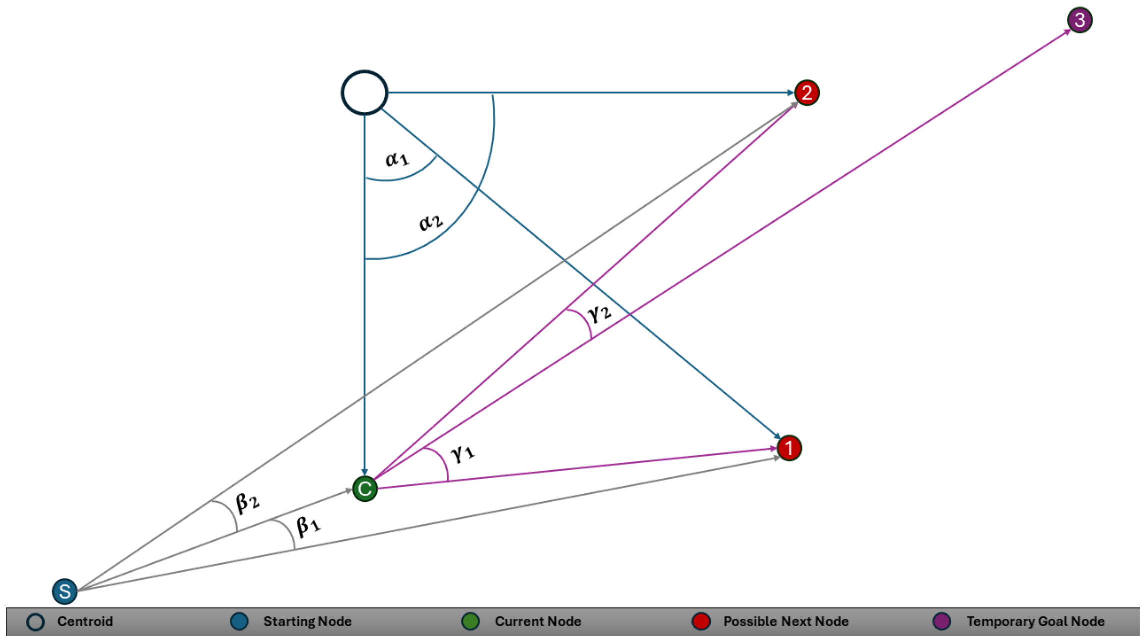


Figure 4.1 A visual illustration that demonstrates the key angles used in our angle-based ordering strategy. It practically shows how to calculate α, β and γ in a real-world scenario, highlighting their roles in determining the relationship between the current node, potential next nodes, and various reference points. Reproduced from Allus & Unel (2025)

The intuition behind using angles stems from their proportionality to distances between nodes, as governed by the law of sines. This relationship allows angles to offer a computationally efficient way to assess node relationships, eliminating the need for repeated A* algorithm executions. Consequently, the One-Distance-Two-Angles strategy substantially lowers computational complexity while preserving the effectiveness of the ordering process.

4.4 Euclidean-Based Heuristic Ordering

To manage the computational complexity of ordering goal nodes based on their proximity, we propose a heuristic approach utilising Euclidean distances before engaging the A* algorithm for distance-based ordering. This strategy is founded on our observation that, in dynamic multi-goal node ordering, the next optimal goal node is typically one of the two nearest nodes to the current node, even when considering environmental obstacles.

In practical terms, this heuristic initially orders all goal nodes based on their Euclidean distance from the current node: $[(\hat{x}_{G_{i+1}}, \hat{y}_{G_{i+1}}), ..., (\hat{x}_{N_G}, \hat{y}_{N_G})]$ *where* $|(\hat{x}_{G_{i+k}} - x_{G_i}, \hat{y}_{G_{i+k}} - y_{G_i})| < |(\hat{x}_{G_{i+k+1}} - x_{G_i}, \hat{y}_{G_{i+k+1}} - y_{G_i})|$ *for* $k = 1, ..., N_G - i - 1$. This preliminary step offers a swift, obstacle-agnostic approximation of proximity. Subsequently, the more computationally intensive A* algorithm computes the actual shortest path distances, factoring in obstacle avoidance, for the full list: $[(x_{G_{i+1}}, y_{G_{i+1}}), ..., (x_{G_{N_G}}, y_{G_{N_G}})]$.

The core innovation lies in validating the Euclidean ordering against the A* ordering for the first three nodes in the list: $[(x_{G_{i+1}}, y_{G_{i+1}}), ..., (x_{G_{i+k}}, y_{G_{i+k}})]$ *where* $k = 1, 2, 3$. If the sequence of these three nodes matches between both approaches $[(x_{G_{i+1}}, y_{G_{i+1}}), ..., (x_{G_{i+k}}, y_{G_{i+k}})] = [(\hat{x}_{G_{i+1}}, \hat{y}_{G_{i+1}}), ..., (\hat{x}_{G_{i+k}}, \hat{y}_{G_{i+k}})]$ $\forall k$, it confirms they are indeed the nearest nodes, even considering obstacles. However, if a discrepancy occurs, where a node $(\hat{x}_{G_{i+k}}, \hat{y}_{G_{i+k}})$ from the Euclidean order does not correspond to $(x_{G_{i+k}}, y_{G_{i+k}})$ from the A* order, that Euclidean node $(\hat{x}_{G_{i+k}}, \hat{y}_{G_{i+k}})$ is replaced by the next node $(\hat{x}_{G_{i+k+1}}, \hat{y}_{G_{i+k+1}})$ in the Euclidean sequence. Specifically, for some k, if $(x_{G_{i+k}}, y_{G_{i+k}}) \neq (\hat{x}_{G_{i+k}}, \hat{y}_{G_{i+k}})$, the substitution occurs, and this validation iterates until $(x_{G_{i+k}}, y_{G_{i+k}}) = (\hat{x}_{G_{i+k}}, \hat{y}_{G_{i+k}})$ *for* $\forall k$.

Algorithm 4 illustrates these details. This process ensures goal nodes are optimally ordered, reconciling both Euclidean proximity and A* search results, with necessary adjustments for obstacle presence.

Algorithm 4: Get Euclidean-Based Heuristic Ordering

```
Input  : CN, GNs
// CN: Current Node, GNs:  Goal Nodes
Output: OGNs
// OGNs:  Ordered Goal Nodes
OGNs ← [];
// OGNs:  Ordered Goal Nodes
for each node in GNs do
    append [node, Euclidean distance from CN to node] to OGNs;
sort OGNs by the second element (distance);
if length of OGNs ≥ 3 then
    counter ← 3;
    while counter > 0 do
        for each node in OGNs do
            update the second element with A* cost from CN to node;
            counter ← counter - 1;
            if counter == 0 then
                A*OGNs = sort OGNs by the second element (cost);
                if A*OGNs is not OGNs then
                    counter ← 1;
return OGNs;
```

Despite its efficiency, the Euclidean-based heuristic ordering has certain limitations. Its accuracy is influenced by the ratio of obstacles to navigable area, obstacle connectivity, and the uniformity of traversal costs. When obstacles are sparse (low obstacle-to-navigable-area ratio), the heuristic is more effective, increasing the likelihood of its ordering aligning with A*'s. Similarly, if obstacles are widely spaced rather than densely connected, the heuristic remains more reliable. However, when costs extend beyond simple distance, additional heuristics may be necessary to account for these variations. In such limiting scenarios, the worst-case impact is reduced heuristic efficiency, ultimately increasing reliance on A*-based ordering.

Nevertheless, this heuristic significantly reduces the number of full A* algorithm executions, accelerating the overall ordering process. By initially leveraging Euclidean distances, we streamline the identification of potential next nodes and optimise subsequent A* calculations for obstacle-aware pathfinding. This dual-step approach balances computational efficiency with pathfinding accuracy, particularly in scenarios where exhaustive search methods like brute force are impractical due to computational constraints.

4.5 Angle-Based Ordering

Our ordering paradigm is built upon the earlier assumptions concerning the distances to potential next nodes. In each iteration, the algorithm selects one of the two closest nodes, $(x_{G_{i+1}}^{\mathrm{Pot}_{1,2}}, y_{G_{i+1}}^{\mathrm{Pot}_{1,2}})$, to be the next ordered node. This selection is primarily guided by the angles illustrated in Figure 4.1.

Based on this, we propose two distinct ordering approaches:

4.5.1 Sequential Angle-Based Ordering (SABO)

Our first approach, Sequential Angle-Based Ordering (SABO), relies on a sequence of conditions based on the magnitudes of the angles previously discussed. As shown in Algorithm 5, for a closed path, the α angle is most indicative of path closeness. Ideally, within a closed path formed by a set of nodes, the next node chosen from the three nearest options should have the smallest α angle. Therefore, our primary ordering condition is:

$$\alpha_1 < \alpha_2 \rightarrow (x_{G_{i+1}}, y_{G_{i+1}}) = (x_{G_{i+1}}^{\mathrm{Pot}_1}, y_{G_{i+1}}^{\mathrm{Pot}_1}) \tag{4.5}$$

If this initial condition isn't met, we then consider the second angle criterion, γ. Although γ is a local measure, it offers crucial information about how much the potential next nodes deviate from the temporary goal node.

$$\alpha_1 > \alpha_2 \text{ AND } \gamma_1 > \gamma_2 \rightarrow (x_{G_{i+1}}, y_{G_{i+1}}) = (x^{\text{Pot}_1}_{G_{i+1}}, y^{\text{Pot}_1}_{G_{i+1}}) \tag{4.6}$$

Here, $(x^{\text{Pot}_1}_{G_{i+1}}, y^{\text{Pot}_1}_{G_{i+1}})$ deviates more from the temporary goal and is positioned after $(x^{\text{Pot}_2}_{G_{i+1}}, y^{\text{Pot}_2}_{G_{i+1}})$ within the presumed closed path. Since both angles for $(x^{\text{Pot}_1}_{G_{i+1}}, y^{\text{Pot}_1}_{G_{i+1}})$ are larger, it suggests this node is an outlier to the closed path. Thus, we opt to visit it next to prevent a detrimental self-intersection later in the ordering process.

Should the second condition not be satisfied, we use the third angle criterion, β, as a tie-breaker. Angle β helps evaluate angle α because both are global criteria. The key difference is that α uses the centroid as its origin, while β uses the starting node.

$$\alpha_1 > \alpha_2 \text{ AND } \gamma_1 < \gamma_2 \text{ AND } \beta_1 < \beta_2 \rightarrow (x_{G_{i+1}}, y_{G_{i+1}}) = (x^{\text{Pot}_1}_{G_{i+1}}, y^{\text{Pot}_1}_{G_{i+1}}) \tag{4.7}$$

$$\alpha_1 > \alpha_2 \text{ AND } \gamma_1 < \gamma_2 \text{ AND } \beta_1 > \beta_2 \rightarrow (x_{G_{i+1}}, y_{G_{i+1}}) = (x^{\text{Pot}_2}_{G_{i+1}}, y^{\text{Pot}_2}_{G_{i+1}}) \tag{4.8}$$

These four conditions collectively define our first angle-based ordering approach.

Algorithm 5: Get Sequential Angle Based Ordered Goal Nodes

```
Input  : ASN, GNs, GPs, CNCs, GI, CP
// ASN: Absolute Starting Node, GNs: Goal Nodes, GPs: Generated Paths, CNCs: Centroid Node Coordinates, GI: Grey Image, CP:
   Closed Path
Output: OGNsO
// OGNsO: Ordered Goal Nodes Output
SNH ← ASN, GNsH ← GNs, OGNsO ← [];
// SNH: Starting Node Holder, GNsH: Goal Nodes Holder
while length of GNsH > 0 do
    OGNs ← [];
    // OGNs: Ordered Goal Nodes
    OGNs = getEuclideanBasedHeuristicOrdering(SNH, GNsH)
    holder ← first element of OGNs;
    if length of OGNs > 2 then
        Calculate Angles alpha1&2, beta1&2, gamma1&2
        if alpha1 > alpha2 and gamma1 < gamma2 and beta1 > beta2 then
            holder ← second node;
    else if length of OGNs < 3 and CP then
        CTASN ← A* cost from SNH to ASN;
        // CTASN: Cost To Absolute Starting Node
        append [ASN, CTASN] to OGNs;
        Calculate Angles alpha1&2, beta1&2, gamma1&2
        if alpha1 > alpha2 and gamma1 < gamma2 and beta1 > beta2 then
            holder ← second node;
            append second node to OGNsO;
            append first node to OGNsO;
        else
            append first node to OGNsO;
            append second node to OGNsO;
        remove first node from GNsH;
        remove second node from GNsH;
        if CP then
            insert ASN at the start of OGNsO;
            append absoluteStartingNode to OGNsO;
        return OGNsO;
    append holder to OGNsO;
    SNH ← holder;
    remove holder from GNsH;
return OGNsO;
```

4.5.2 Combined Angle-Based Ordering (CABO)

The main limitation of sequential angle-based ordering is that it assesses potential next nodes and their relationships with factors like the centroid and temporary goal individually. We believe that combining these criteria to choose the next node could be more efficient. Thus, leveraging the proportionality between angles and distances (via the law of sines), we propose the following combined angle condition:

$$\alpha_1 + \gamma_2 > \alpha_2 + \gamma_1 \text{ AND } \alpha_1 > \alpha_2 \rightarrow (x_{G_{i+1}}, y_{G_{i+1}}) = (x^{\text{Pot}_2}_{G_{i+1}}, y^{\text{Pot}_2}_{G_{i+1}}) \tag{4.9}$$

Let us say that angles α_1, α_2, γ_1 and γ_2 are proportional to distances a_1, a_2, g_1 and g_2 respectively as shown in Figure 4.1. The condition can be interpreted as follows, the node that minimises the combination $\alpha_N + \gamma_M$ is to be selected as the next node. This is proportional to minimising the combination of $a_N + g_M$ which keeps locally minimizing the path sections seeking an overall global minimization for the whole path. Keep in mind that we still use the global criterion of angle α

as a parallel condition to optimise the assumed closed path. if the condition is not satisfied, then we resort to selecting $(x_{G_{i+1}}^{\text{Pot}_1}, y_{G_{i+1}}^{\text{Pot}_1})$ as the next node.

Imagine angles α_1, α_2, γ_1 and γ_2 are proportional to distances a_1, a_2, g_1 and g_2, respectively, as depicted in Figure 4.2. This condition suggests selecting the node that minimises the combined sum $\alpha_N + \gamma_M$. This is proportional to minimising the combination $a_N + g_M$, which allows for local path section minimisation while striving for overall global path minimisation. Note that we still use the global criterion of angle α as a parallel condition to optimise the assumed closed path. If the primary condition is not met, we default to selecting $(x_{G_{i+1}}^{\text{Pot}_1}, y_{G_{i+1}}^{\text{Pot}_1})$ as the next node.

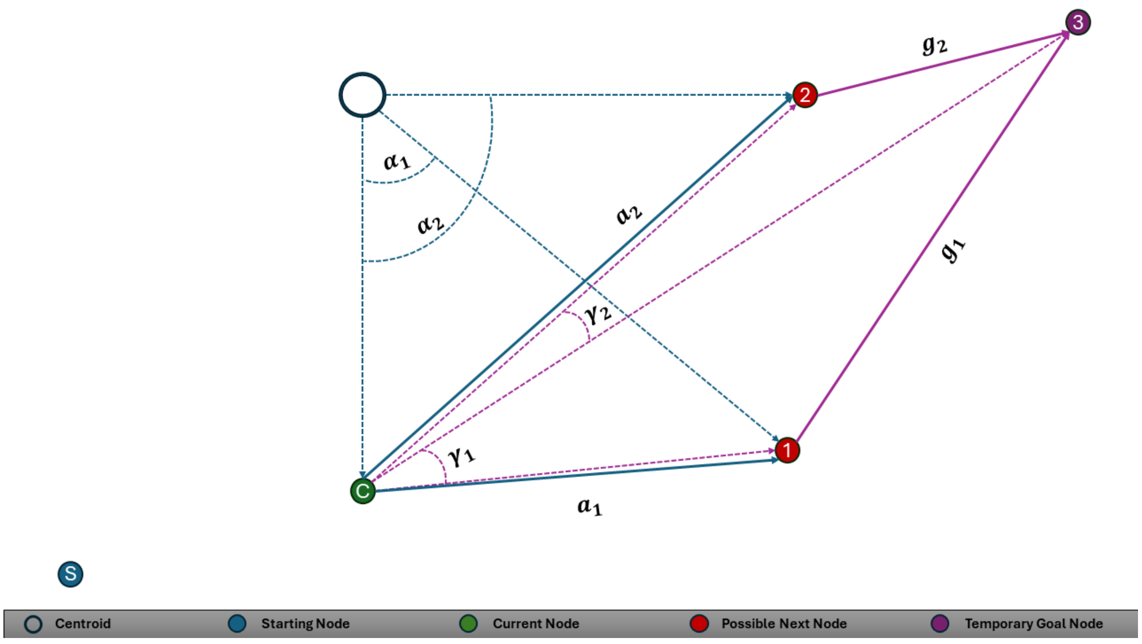


Figure 4.2 A visualisation of the Combined Angle-Based Ordering (CABO) that illustrates the angles and their proportional distances, helping to clarify our ordering concept. Reproduced from Allus & Unel (2025)

It's vital to highlight that both proposed ordering approaches are highly computationally efficient, as they only require simple geometrical calculations.

4.5.3 Time and Spcae Complexity Analysis

Let N denote the number of goal nodes to be ordered by either the Combined Angle-Based Ordering (CABO) or Sequential Angle-Based Ordering (SABO) algorithms.

Both algorithms commence with a Euclidean-based heuristic ordering. In this phase, goal nodes are ordered by their Euclidean distance to the current node. This involves calculating N Euclidean distances, which takes $(O(N))$ time, followed by sorting these distances, contributing $(O(NlogN))$ to the complexity. Thus, the total approximate complexity for this initial stage is $O(NlogN)$.

In the second stage, the A* algorithm is called up to 3 times per iteration to determine the actual obstacle-aware distances for the nodes ordered in the preceding stage. As detailed in Section3.2, the complexity of A* is $A = O(|V|^2 + P^3)$. The A*-based reordering typically stabilises after a small number of iterations k, which can range from 3 in the best case to N in the worst case. This results in a complexity of $O(3kA)$ for this stage.

The final stage updates the ordered list based on CABO or SABO's specific conditions. The calculations for these conditions have a constant complexity of $O(1)$. Therefore, the complexity of each iteration is primarily governed by the Euclidean-based ordering and A*-based reordering, leading to $O(NlogN + 3kA)$. Since the outer while loop runs N times, the overall complexity is approximately $O(NlogN + 3k(|V|^2 + P^3))$, which simplifies to$O(N^2logN + 3kN(|V|^2 + P^3))$.

If path post-pruning (discussed in Section3.2) is disabled, the $O(P^3)$ term is removed, simplifying the overall complexity to $O(N^2logN + 3kN|V|^2)$. Generally, the relative magnitudes of N, $|V|$ and P determine the dominating term. For large graphs, where $|V|$ and P are significantly larger than N, the term $O(3kN(|V|^2+P^3))$ becomes dominant. Under typical conditions, where P is considerably smaller than $|V|$, the dominating term further reduces to $O(3kN|V|^2)$. This highlights the critical importance of optimising the A* algorithm's adaptation to reduce the overall complexity of our proposed algorithms.

In terms of space complexity, the algorithm maintains the ordered list of goal nodes, which, in the worst case, stores all N nodes, contributing $O(N)$. Additionally, temporary variables like startingNodeHolder, goalNodesHolder, and intermediate ordered lists are stored, all proportional to N. The A* algorithm itself has a worst-case space complexity of $O(|V|)$ due to node information storage (as detailed in Section3.2). Since A* is invoked multiple times, the space complexity of the entire algorithm remains dominated by A*'s storage requirements, resulting in an overall worst-case space complexity of $O(|V| + N)$.

4.5.4 Ensemble Angle-Based Ordering

Given the notable efficiency of both proposed angle-based ordering algorithms, we suggest employing an ensemble approach to further enhance the resulting pixel cost of the path. This involves running both the Sequential Angle-Based Ordering (SABO) and Combined Angle-Based Ordering (CABO) variants in parallel. The algorithm then selects the output that yields the lowest overall path cost. While this method marginally increases the time needed to order the goal nodes, it proves to be both efficient and superior in terms of optimising pixel cost and maintaining favourable time complexity, as our upcoming experiments will demonstrate.

4.5.5 Post-Pruning Ensemble Angle-Based Ordering for Improved Path Costs

A major contributor to increased path costs in ordered goal nodes is the presence of self-intersections. Our inspection of optimal paths confirms they contain no self-intersections. To tackle this, we developed a post-pruning algorithm designed to reduce these self-intersections in the outputs of our proposed ordering algorithms. Our approach leverages an improved version of the well-known 2-Opt algorithm Chen, Zhou, Tang & Luo (2017). However, the main challenge with the standard 2-Opt algorithm is its computational expense: it requires evaluating the entire path cost after each swap, which is impractical for large-scale ordering problems due to its reliance on the A* algorithm.

To lessen this computational load, we propose two modifications:

- Modified 2-Opt with Euclidean Path Segments: This version performs swaps directly on the path generated by our angle-based algorithm. This path comprises ordered nodes and intermediate path nodes, all connected by Euclidean segments. Our key modification is to use a heuristic Euclidean distance cost check after each swap, instead of the computationally expensive A* distance cost check. This is similar to the approach used in our Euclidean-based heuristic ordering. Since many Euclidean swaps might be invalid due to obstacles, we added a crucial check using Bresenham's line algorithm Bresenham (1977) to verify obstacle-free connectivity on the Euclidean line connecting nodes involved in a swap.
- Modified 2-Opt on Ordered Nodes Only: This variant performs swaps solely on the set of ordered goal nodes, excluding intermediate path nodes. This offers the potential for more aggressive and effective pruning. However, it carries the risk of being misled by the Euclidean distance cost check since obstacle existence isn't directly considered in the swap validation. This post-pruning method proved effective and superior to the first approach for maps where the

total obstacle area doesn't exceed approximately 40% of the entire map's area. Algorithm 6 outlines the steps for implementing this approach.

Algorithm 6: Modified 2-Opt Post-Pruning

```
Input  : nodes, GI
// GI: Grey Image
Output: bestOrder
bestOrder ← nodes;
, improved ← True;
while improved do
    improved ← False;
    for $i \leftarrow 1$ to length of bestOrder - 2 do
        for $j \leftarrow i + 1$ to length of bestOrder - 1 do
            if $j - i == 1$ then
                - ; // Consecutive nodes, skip
            if $i > 0$ and $j <$ length of bestOrder - 1 then
                if not isLineObstacleFree(bestOrder[$i-1$], bestOrder[j], GI) or not
                  isLineObstacleFree(bestOrder[i], bestOrder[$j+1$], GI) then
                    -
            newOrder ← bestOrder;
            reverse the elements from $i$ to $j$ in newOrder;
            if calculatePathLength(newOrder) < calculatePathLength(bestOrder) then
                bestOrder ← newOrder;
                improved ← True;
return bestOrder;
```

These modifications enable efficient post-pruning of ordered goal nodes, preventing misdirection by the Euclidean distance heuristic and avoiding computationally expensive, repeated A* distance calculations. These refined approaches effectively minimise self-intersections, ultimately leading to improved path costs without excessive computational expense. The proposed algorithms are grounded in a robust mathematical intuition for selecting the specified angles and their conditions. As a result, they offer a near-optimal solution with a minimised optimality gap for any given multi-goal ordering and path-planning problem, as demonstrated in the results chapter.

5. LEARNING-BASED MULTI-GOAL ORDERING AND PATH PLANNING

This chapter introduces our learning-based approach to efficiently tackle the multi-goal ordering and path planning problem. We developed two distinct learning frameworks, each with unique pre-assumptions, data collection strategies, feature extraction methods, and model structures. The first framework is a traditional supervised machine learning classification framework, which uses hand-crafted features to dynamically predict the next node in a sequence. Conversely, the second framework is transformer-based, predicting the next node using features extracted via a combination of convolutional neural networks (CNNs) and relational transformers. We will begin this chapter by outlining the fundamental assumptions, geometric principles, and foundational concepts that guided the design of both models.

5.1 Data Insights and Dynamic Ordering

In our experiments, brute-force enumeration proved practical for up to 13 goal nodes; beyond that, computational limits became restrictive. To manage the complexity of standard brute-force, we used a heuristic-driven brute-force technique, which was significantly faster. We ran extensive tests with this improved method across various goal node distributions and maps, analysing optimal path structures and behaviours to find common patterns.

A key finding from these experiments is that in optimal paths, the next chosen node from a current node is usually one of the two closest nodes in terms of actual traversed distance, not just Euclidean distance. This aligns with human intuition, where immediate proximity often guides navigation choices. This insight supports the idea of dynamically ordering nodes: structuring the algorithm to iteratively pick the next optimal goal node based on its context relative to the current one.

When we say "dynamically ordering," we mean the algorithm is structured around the current node and a set of potential next goal nodes. The ordering process repeats for each chosen node, essentially building the closed path one goal node at a time.

However, dynamic ordering brings a "cost of locality". Focusing on immediate neighbours during ordering risks sacrificing global path optimality. To counter this, our approaches integrate global insights into the dynamic ordering process. These concepts form the bedrock of our methodology, which we detail in later sections.

In essence, our methods advocate for ordering goal nodes one by one, based on the current node's context. This balances immediate proximity considerations with the need for overall path coherence, aiming to blend the efficiency of local decisions with strategic insights to improve the path's quality.

5.2 Traditional Machine Learning Framework

In this section, we propose a traditional machine learning-based method designed to dynamically order goal nodes, aiming to approximate an optimal visiting sequence and, consequently, minimise overall distance costs. Our traditional model is trained using hand-crafted features, which were meticulously developed after extensive testing and observation of optimal paths. These optimal paths were generated through brute-force techniques across various numbers of goal nodes and diverse map configurations. Furthermore, our algorithm is optimised for computational efficiency, a critical factor for real-world applications where dynamic environments necessitate rapid, adaptive responses.

5.2.1 Distance-Related Features

Figure 5.1 highlights the distances central to our model. We incorporate local distances, which are the actual path lengths between the current node and each of its two potential next nodes. These local distance features define the immediate relationship between the current node and its prospective successors and are computed efficiently using the improved A* algorithm at each iteration, adding no extra computational overhead.

Furthermore, we use the centroid to derive global distance features. These include the distances from the current node to the centroid, and from each potential next node to the centroid. These global features are Euclidean distances, intentionally disregarding obstacles, as their purpose is solely to indicate proximity to the centroid rather than representing navigable paths.

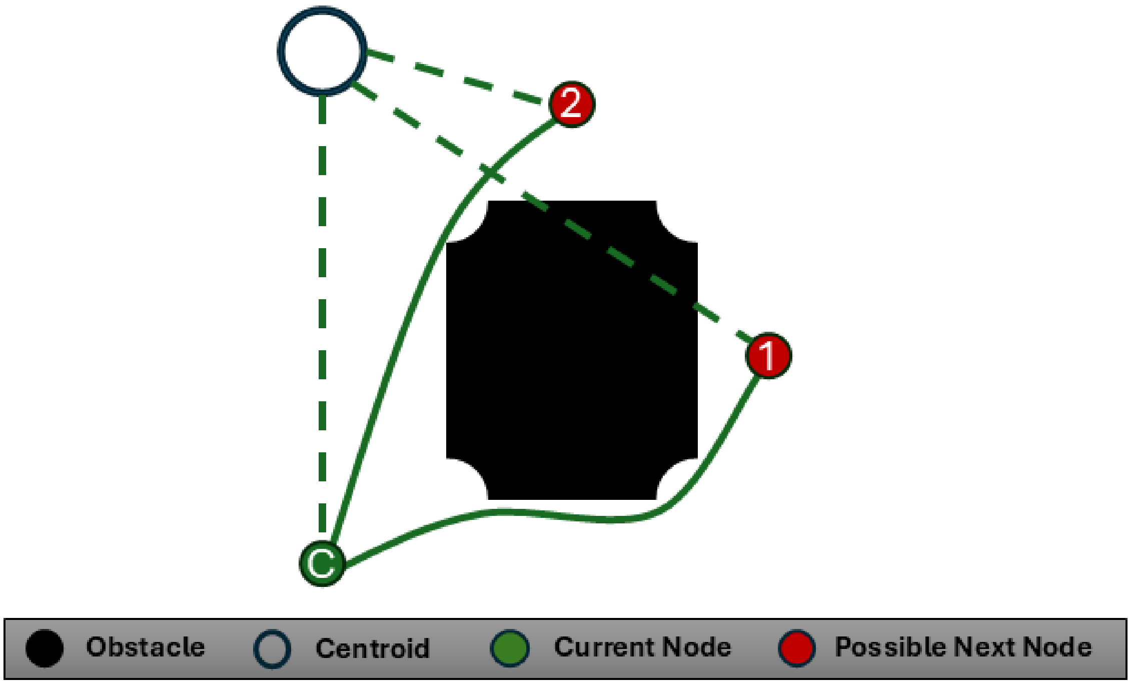


Figure 5.1 A visualisation that illustrates the distances used in our model's structure. Dashed lines represent Euclidean distances, while continuous lines denote distances calculated using the improved A* algorithm.

5.2.2 Angle-Related Features

To effectively combine local and global ordering techniques, we need a computationally efficient approach, especially for both offline and online applications. Since obstacles on the map make relying solely on distances impractical due to repeated, intensive A* algorithm executions, we developed a distance-angle relational strategy to boost efficiency. This strategy uses A* sparingly to find the three nearest nodes to the current node based on distance cost. From these, the farthest node becomes the temporary goal node, while the other two are potential next nodes. The centroid coordinates and the starting node are also vital global factors that help determine the relevant angles for ordering.

Figure 5.2 displays angles α, β, γ, and ζ. Angle α shows the relationship between the current node and each potential next node relative to the centroid, offering a global perspective for ordering. Angle β illustrates the relationship between the current node and each potential next node relative to the starting node. Angle γ captures the relationship between the current node and each potential next node with respect to the temporary goal node, providing crucial local information. Lastly, angle ζ represents the relationship between the starting node and each potential next node, independent of the current node, aiding in assessing preferred positions within an assumed optimal closed path.

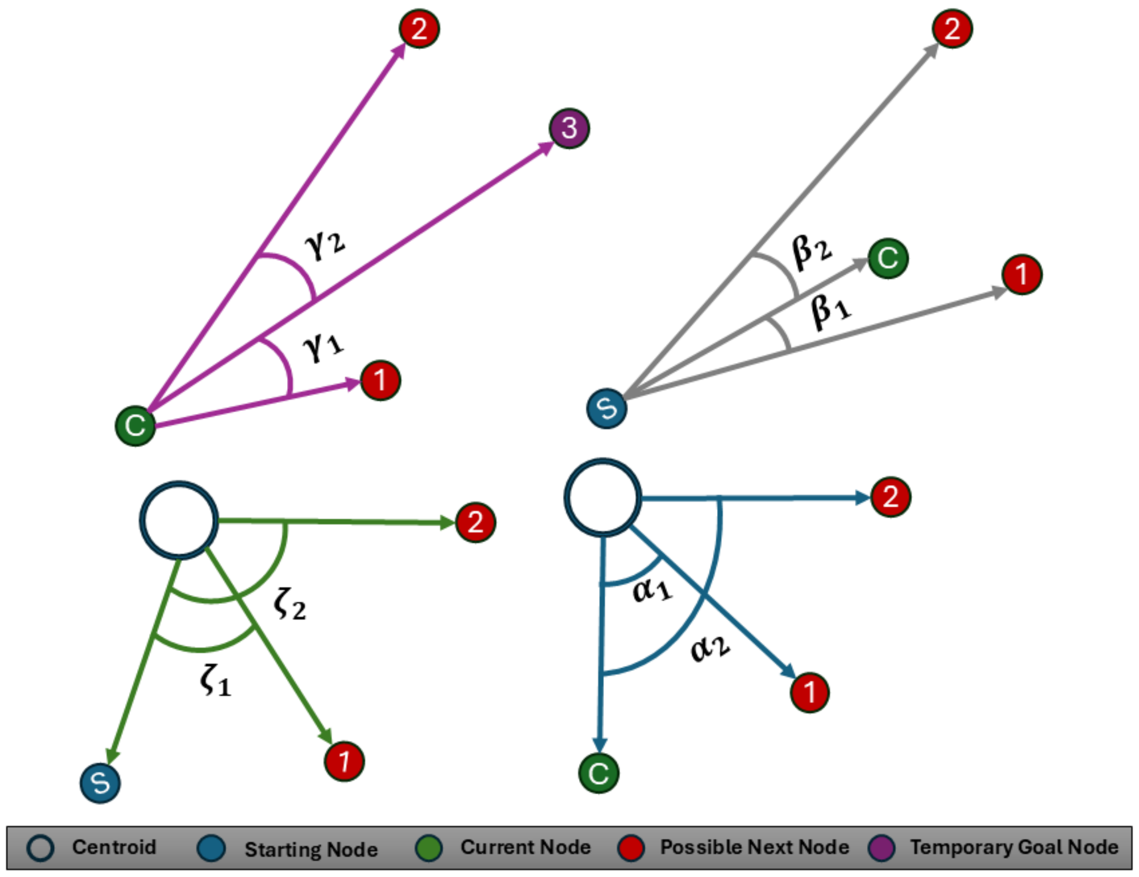


Figure 5.2 A visual illustration that shows the angles central to our proposed model's structure, depicting how each angle is formed by necessary factors in various scenarios.

The reason we use angles is their proportionality to distances between nodes, as explained by the law of sines. This allows angles to offer a computationally efficient way to assess node relationships without needing to run the A* algorithm repeatedly. As a result, our distance-angle strategy significantly cuts down on computational complexity while keeping the ordering process effective.

5.2.3 Model Structure

We approach our problem as a classification task. First, the algorithm dynamically identifies the three nearest nodes to the current node using an initial Euclidean-based heuristic. Next, it computes both global and local features for the current node and its potential next nodes. These features are then fed into our model, which classifies the potential next nodes as either optimal or not.

The model outputs a probability indicating how likely each node is to be the next in the sequence. If both nodes are classified as potentially optimal, the one with the higher probability is chosen. If only one node is classified as the next, it is selected directly, and the other is disregarded. This iterative process continues until only two goal nodes remain unordered. At this point, a brute-force approach determines the optimal order for these final two nodes, which is computationally trivial (since $2! = 2$). Finally, the starting node is appended to complete the path, ensuring a closed loop.

Figure 5.3 provides a flowchart of this ordering procedure, illustrating the steps from the initial unordered input to the formation of the final closed path.

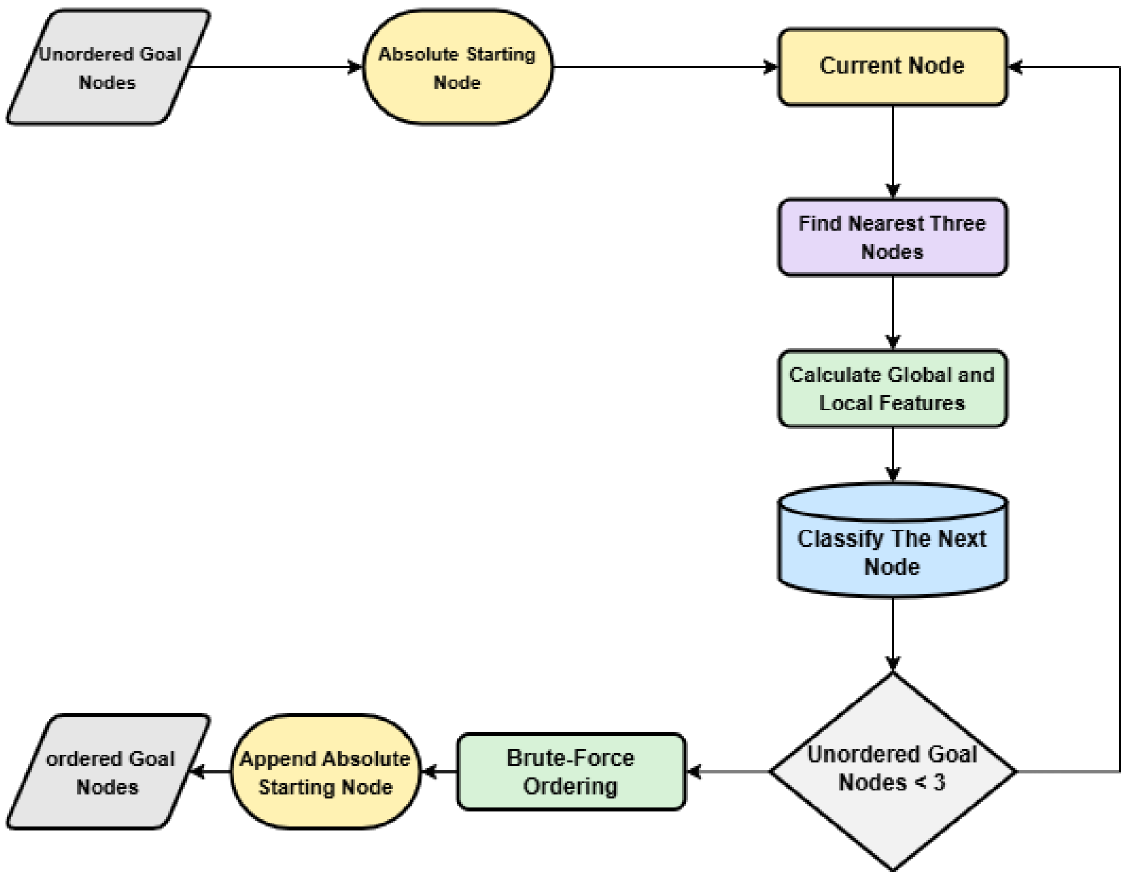


Figure 5.3 A flowchart that illustrates the proposed learning-based multi-goal ordering algorithm, showing the complete process from receiving unordered goal nodes to outputting the final ordered sequence.

5.3 Transformer-Based Framework

In this section, we present a Transformer-based framework developed to dynamically establish the visiting order of multiple goal nodes, aiming to approximate the optimal sequence that minimises the total path cost in a multi-goal ordering and path planning scenario. As illustrated in Figure 5.4, the prediction pipeline begins with the extraction of feature representations using convolutional neural networks in combination with relational transformer modules. These initial features are computed once and held constant throughout the entire node-ordering process.

Following this, an iterative loop of length $N-1$ (where N is the total number of goal nodes) is initiated. In each iteration, context-dependent geometric features—such as the spatial relation of the current node to the remaining nodes—are dynamically computed and concatenated with the static features. These combined features are then used in a scoring phase, where the model selects the most probable next node in the sequence. This selection is based either on evaluating all remaining nodes or, for improved efficiency, limiting evaluation to the two closest nodes relative to the current one. The node receiving the highest score is chosen as the next in the sequence.

The model is trained on data generated using optimal solutions obtained from Google's OR-Tools and enhanced brute-force techniques across a diverse set of map configurations and varying numbers of goal nodes. The overall architecture is carefully designed to prioritise computational efficiency, ensuring the model remains responsive and robust in dynamic and potentially unpredictable real-world environments.

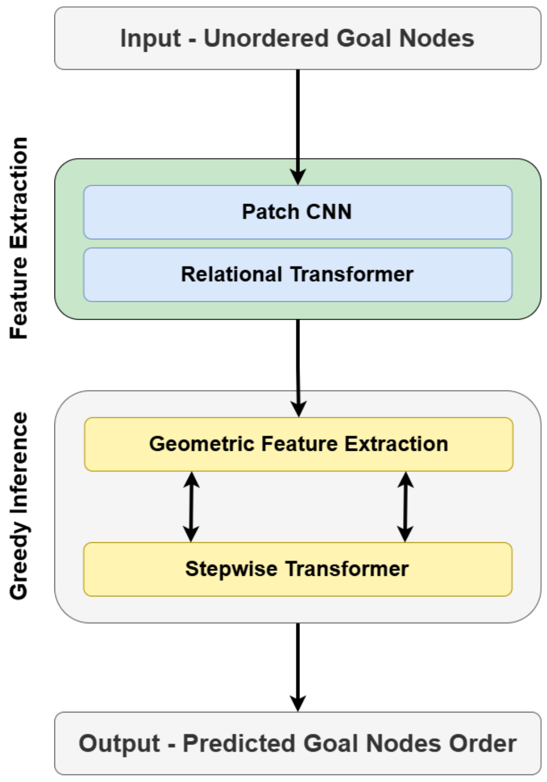

Input - Unordered Goal Nodes
Feature Extraction
Patch CNN
Relational Transformer
Greedy Inference
Geometric Feature Extraction
Stepwise Transformer
Output - Predicted Goal Nodes Order

Figure 5.4 An illustration of the transformer-based framework used to determine the visiting order of multiple goal nodes. The process begins with the extraction of static feature representations using CNNs and relational transformer modules. These features are combined with dynamically updated geometric information at each step to score candidate nodes. The node with the highest score is selected as the next in the sequence. This iterative procedure continues until all nodes are ordered.

5.3.1 CNN-Based and Relational-Transformer Features

To extract spatially-informed features (embedding vector) for each goal node, we employ a convolutional neural network (CNN) architecture, referred to as `PatchCNN`, which processes local image patches around goal node coordinates. The grayscale occupancy grid map is used as the input, from which square patches centred at each goal node are extracted and padded to accommodate a fixed receptive field. These patches are then fed into the `PatchCNN` as showsn in Figure5.5, a lightweight convolutional model composed of three convolutional layers with progressively increasing channel depths. The final output is a 62-dimensional feature vector for each node, capturing relevant local spatial patterns within its vicinity.

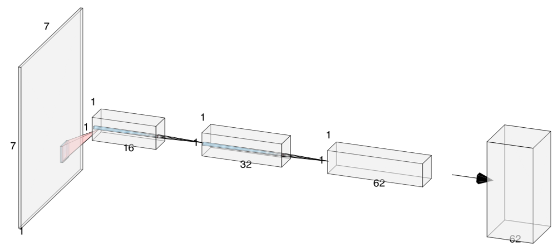


Figure 5.5 Architecture of the PatchCNN used for visual feature extraction. The network takes a local grayscale image patch of size 7×7 as input and processes it through three convolutional layers, a 7×7 filter for spatial compression, followed by two 1×1 convolutions for channel-wise transformations. The final output is a 62-dimensional feature vector representing the visual context of the patch.

To enhance the quality of the feature representations, we employ a `RelationalTransformer` module that augments the features previously extracted by the convolutional neural network. This module allows each goal node to attend to the entire set of goal nodes simultaneously, thereby capturing global relational context. Through self-attention mechanisms, the transformer updates each node's embedding by integrating information from its interactions with all other nodes in the set. This relational reasoning process equips the model with a richer understanding of the spatial and structural dependencies among goal nodes, and it does so in a manner that is invariant to the order in which nodes are presented.

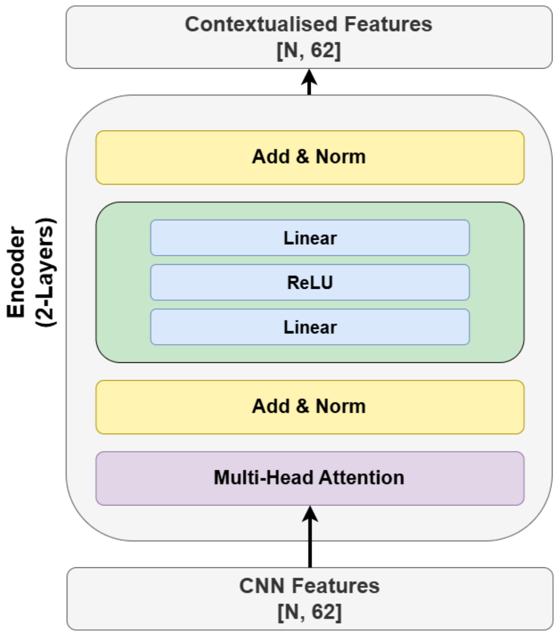


Figure 5.6 Architecture of the RelationalTransformer used for enriching the extracted visual features. The transformer takes the output feature vectors of the goal nodes provided by the PatchCNN network and works on contextualising them by attending to each goal node in the provided unordered set of goal nodes.

The final feature vector for each node is constructed by concatenating its transformed embedding with its normalised two-dimensional spatial coordinates. For the starting node, this feature vector is initialised as a zero vector, with only its coordinates explicitly inserted. To generate the training dataset, we create numerous problem instances by randomly selecting a starting node and a set of goal nodes from within the environment. The correct sequence of node visits (i.e., the ground-truth order) is computed using the Travelling Salesman Problem solver provided by Google's OR-Tools and an enhanced brute-force technique. For each goal node that is visited—excluding the starting node—we extract the local image context and encode it into a fixed-length feature vector.

Moreover, we enrich the constant CNN-derived features by appending geometric descriptors specific to each problem instance. These include pairwise relative distances, angular orientations of nodes relative to the centroid of the goal node set, and categorical quadrant-based location indicators. By combining these, each node is represented by a consistent 72-dimensional feature vector, which is then used as input to the transformer-based model introduced in the subsequent sections.

5.3.2 Hand-Crafted Features

In addition to the constant node embeddings extracted via CNN and refined by the relational transformer, we incorporate a set of problem-specific, hand-crafted geometric features to capture higher-level spatial relationships among nodes. These features are designed to provide additional context beyond local appearance, enabling the model to make more informed sequential decisions during path planning.

For each candidate node at a given decision step, we compute the following variables:

- Distance to current node: The normalised A* path cost from the current node to the candidate node.
- Distance to centroid: The Euclidean distance between the candidate node and the centroid of all goal and starting nodes, normalised by the map diagonal.
- Distance to starting node: The normalised A* path cost between the candidate node and the starting node.

In addition to distance-based metrics, we extract angular descriptors that provide information about the relative orientation of nodes:

- Angle α: The angle formed at the candidate node between the centroid, current node, and the candidate node.
- Angle β: The angle formed at the candidate node between the starting node, current node, and the candidate node.
- Angle ζ: The angle between the centroid, starting node, and the candidate node.

Each of these angles is normalised by dividing by 180°, ensuring values fall within a standardised range.

To further capture spatial layout, we assign quadrant labels to the starting node and each candidate node relative to the centroid of the goal configuration. These quadrant indices help encode coarse spatial location, which can aid the model in generalising across different node configurations.

By concatenating these eight hand-crafted features with the constant 64-dimensional features—comprising CNN-transformed appearance embeddings and raw coordinate values—we construct a comprehensive 72-dimensional feature vector for each decision step. These vectors form the basis of our dataset and serve as input to the learning framework for predicting the next node in the sequence.

5.3.3 Transformer Model Structure

We utilise a transformer-based architecture to sequentially predict the next node in a multi-goal ordering and path planning task, aiming to optimise the order of visits for minimal travel cost. This model, referred to as the *StepwiseNextNodeTransformer*, operates in an iterative manner: at each decision point, it receives a dynamically assembled set of candidate nodes and assigns scores to identify the most suitable next node to visit.

The architecture consists of two principal components: a transformer encoder and a feedforward scoring module. The encoder is composed of a stack of N transformer layers (with $N = 4$ in our implementation), where each layer incorporates 8 attention heads and a model dimensionality of 72. This encoder performs self-attention across the candidate nodes' feature vectors to capture the contextual relationships and spatial dependencies among them. The resulting embeddings are then passed to a feedforward multilayer perceptron (MLP), which generates a scalar score for each candidate node. The node associated with the highest score is greedily selected for visitation.

Formally, for a given set of input feature vectors $x_i \in \mathbb{R}^{72}$ corresponding to the candidate nodes at a particular step, the model computes:

$$score_i = MLP(TransformerEncoder(x_i)) \tag{5.1}$$

where the MLP denotes the feedforward scoring network. This process is repeated in a loop until all goal nodes in the environment have been visited.

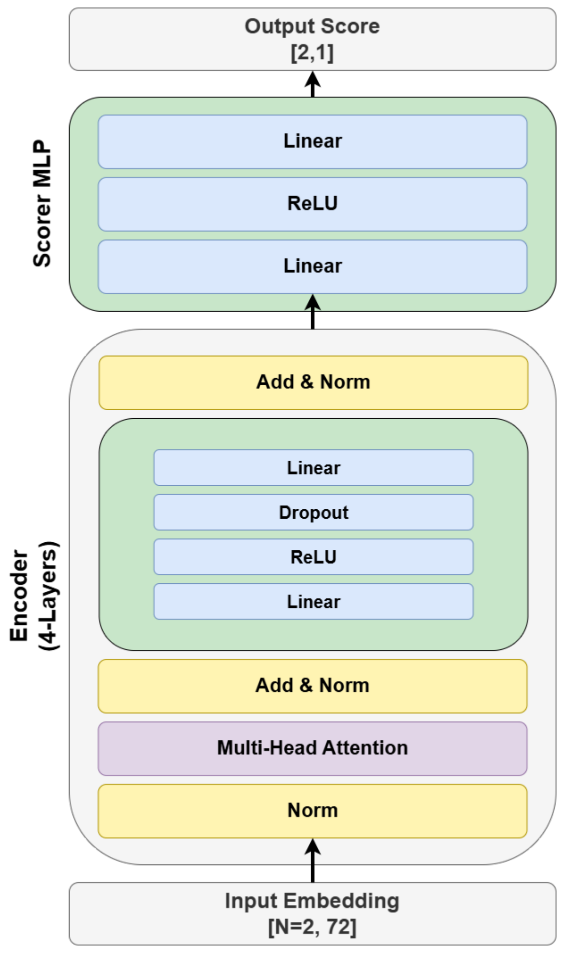


Figure 5.7 Architecture of the StepwiseNextNodeTransformer. At each decision step, a pair of candidate nodes is encoded using a 4-layer Transformer encoder to capture contextual interactions. The output embeddings are passed through an MLP-based scorer to evaluate each node. The node with the higher score is greedily selected as the next in the visitation sequence. This process repeats until all nodes are visited.

5.3.3.1 Goal nodes masking

Rather than providing all unvisited goal nodes to the transformer for scoring and selecting the highest-ranked one, an alternative is to supply only the nearest two goal nodes to the current node—effectively masking the rest, as previously discussed. This approach promotes local decision-making, potentially improving short-term choices, but at the cost of global ordering. Additionally, it decreases computational overhead, as only the features of the nearest two nodes must be identified at each step. Conversely, scoring all remaining goal nodes enhances the model's global awareness, but increases computational complexity and risks less accurate scoring due to the larger candidate pool. Ultimately, whether masking is applied should be determined based on empirical results.

5.3.4 Transformer Training

The transformer is trained using a custom dataset consisting of stepwise decision states extracted from synthetic problem instances. Each problem instance involves a fixed number of goal nodes (N), and is represented by a sequence of decision steps. Each step includes the 72-dimensional feature vectors of the current node and its remaining candidate nodes. The features include both static node-specific data and dynamically computed variables like distances and angular relations.

We construct a `TransformerTrainingDataset` where each sample contains:

- The starting node coordinates,
- The ground-truth sequence of node coordinates (optimal visitation order),
- A list of feature matrices corresponding to each step in the ordering process.

At each step, the input is a matrix of shape $[K, 72]$ where K is the number of candidate nodes at that step, and the target is the index of the optimal next node. The model is trained to maximise the log-likelihood of selecting the correct node among candidates using cross-entropy loss.

Data was aggregated from multiple synthetic datasets, amounting to over 1000 problem instances. The model is trained with mini-batch gradient descent using the Adam optimiser and an initial learning rate of 1e-4.

5.3.5 Goal Nodes Order Inference

During inference, we apply a greedy policy where the model selects the next node based on the highest score from the transformer output at each step. Starting from the initial node, we iteratively evaluate the remaining candidate nodes using the same 72-dimensional input feature representation as used in training.

The inference process incorporates a CNN model to extract fixed visual features from local patches around each node location. These features are then passed through a relational transformer to be enriched and contextualised. Then, they are combined with geometric information such as relative distances, scaled coordinates, angular relationships, and quadrant-based spatial labels to form the complete input vector. At each iteration:

1.1 The features of the remaining candidate nodes are computed.

1.2 The top 2 nodes with the smallest actual distances to the current node are selected (if this was the choice of the user).

1.3 The 72-dimensional dynamic features of these nodes are passed to the transformer.

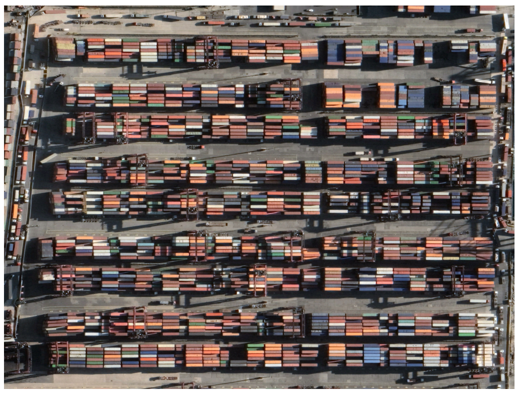
    The node with the highest score is chosen as the next node to visit.

This process repeats until all goal nodes are visited. The final node is always the starting node, closing the path into a tour. This inference pipeline is benchmarked against a traditional TSP solver, demonstrating competitive performance in both solution cost and computational efficiency.

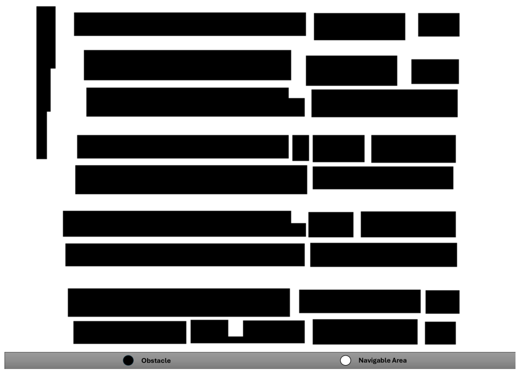
6. EXPERIMENTS, RESULTS AND DISCUSSIONS

This section thoroughly evaluates the efficiency, scalability, and reproducibility of the improved A* algorithm, our proposed classical and learning-based ordering algorithms, and other subsequent algorithms.

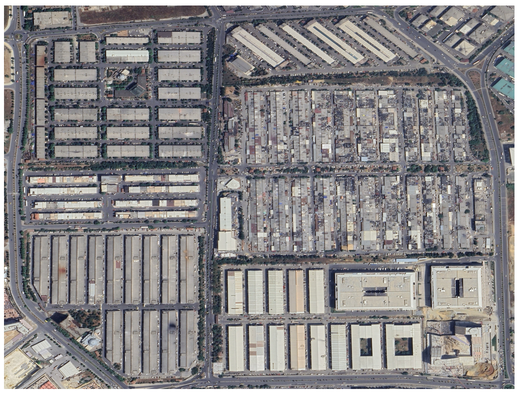
We will start by experimenting with the improved A* algorithm. Then, we will move on to the two main variants of our classical ordering algorithm, followed by our learning-based ordering algorithms—specifically, the traditional machine learning framework and the transformer framework. Figures 6.1 and 6.2 provide a brief visual example of the ordering algorithms' input and output.



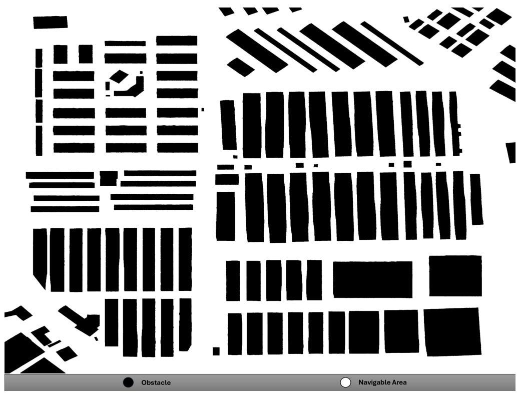
(a) A satellite view from Google Maps showing Ambarli Port in Istanbul, Turkey. The image highlights a dense arrangement of port containers, offering a realistic setting where mobile robots could perform logistical operations.



(b) A formatted approximation of the Ambarli Port map, prepared to be compatible with the proposed algorithms for conducting multi-goal path planning tasks.



(c) A Google Maps satellite image of the Ikitelli Sanayi industrial zone in Istanbul, Turkey. The region features various irregularly shaped industrial buildings, presenting a complex environment where mobile robots could perform advanced logistic tasks involving intricate manoeuvres.



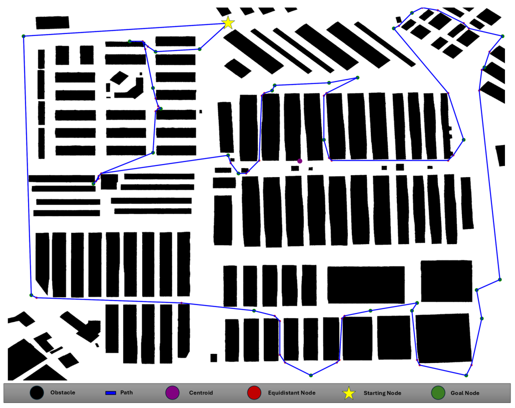
(d) A structured representation of the Ikitelli Sanayi industrial map, adapted for compatibility with the proposed multi-goal path planning algorithms.

Figure 6.1 Examples of real-world environments, such as ports and industrial zones, where fully autonomous robots can carry out complex multi-goal path planning tasks. Reproduced from Allus & Unel (2025)

We will assess the improved A* algorithm's capabilities across various scenarios and maps, including several publicly available ones. The ordering experiments are designed for a comprehensive evaluation of our algorithms' performance. We will also compare our approaches against state-of-the-art methods to contextualize their standing within current advanced ordering techniques. Finally, we will demonstrate the applicability of the paths generated by our algorithms using distinct mobile robots dynamics.

Obstacle
Path
Centroid
Equidistant Node
Starting Node
Goal Node

(a) Illustrative solution for a randomly distributed set of goal nodes on the Ambarli Port map.



(b) Illustrative solution for a randomly distributed set of goal nodes within the Ikitelli Sanayi industrial area.

Figure 6.2 Demonstration of multi-goal path-planning solutions applied to real-world environments. Each scenario features 30 randomly placed goal nodes along with a designated starting point. Reproduced from Allus & Unel (2025)

6.1 Experimental Setup

To ensure our experimental results are reliable, we are providing the details of the hardware and software environment we used. All experiments were run on a machine with Windows 11, featuring an Intel(R) Core(TM) i7-12650H processor (2.30 GHz) and 16.0 GB of RAM (15.7 GB usable). The system is a 64-bit operating system with an x64-based processor. We executed all experiments using Python version

3.12.2.

For a controlled and measurable environment, we primarily used the map shown in Figure 6.3 for our experiments. This map includes polymorphic obstacles, which allowed us to assess how well our approaches handle different obstacle scenarios. Table 6.1 details the map's specifications. The chosen spacing value meets the specified condition, and the listed number of nodes represents only the navigable ones.

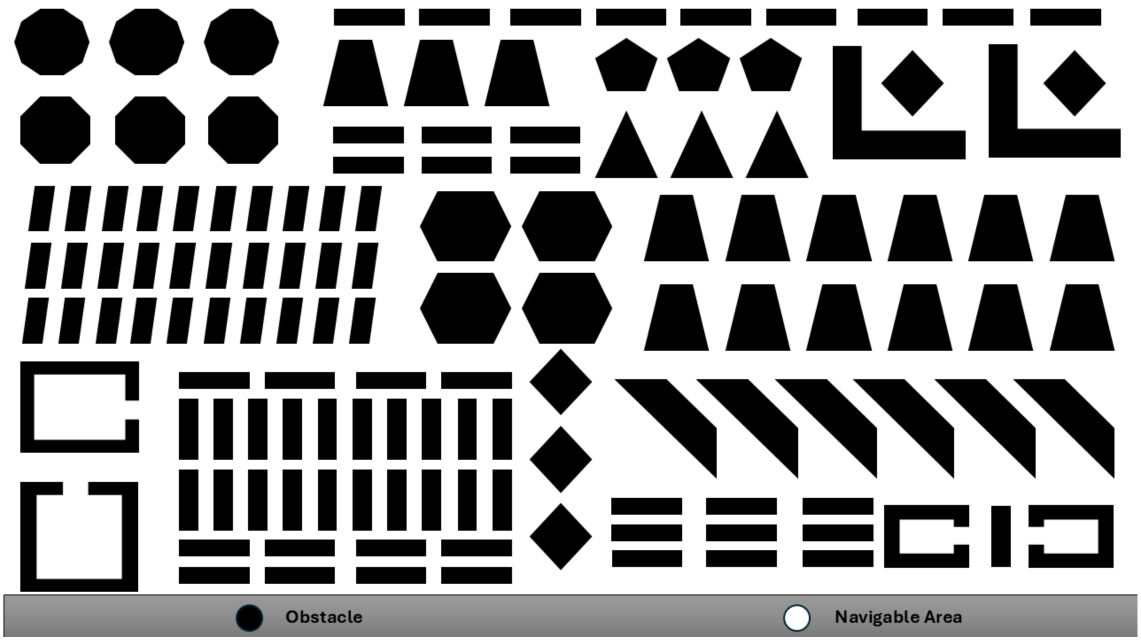


Figure 6.3 The primary experimental map, used for most of our experiments and comparisons. It adheres to the input format explained in previous sections.

The map and its specifications were crucial for creating a consistent and repeatable experimental setup. By keeping conditions uniform across all experiments, we ensured that our results directly reflected the algorithms' performance rather than variations in the testing environment. We selected a spacing of 40 pixels between nodes to achieve a high resolution of navigable paths while maintaining computational feasibility.

Table 6.1 The Adopted Experimental Map Specifications are the reference values for all experiments conducted on the main map. We maintain these consistent values across similar experiments to ensure reliability and consistency in our results. W: Map Width, H: Map Height, CS: Conversion Scale, S: Spacing, NN: Number of Navigable Nodes.

W [pixel]	H [pixel]	CS [m/pixel]	S [pixel]	NN
1280	720	1	10	9217

In short, the precise details of our experimental map, combined with our robust hardware and software setup, provided a reliable and consistent environment for evaluating our algorithms. The careful selection of map parameters and the controlled experimental conditions were key to obtaining valid and reproducible results.

6.2 Experiments on the Improved A* Algorithm: Efficiency, Scalability, and Reproducibility Analysis

In this section, we aim to confirm the efficiency of our improved A* algorithm by comparing it with some state-of-the-art algorithms. We will also assess its scalability and reproducibility by incorporating various maps that feature diverse obstacle distributions and topologies.

6.2.1 Comparison Between The Improved A* Algorithm and State-of-the-art Algorithms

Using the experimental setup detailed in Table 6.1, we ran experiments to confirm the superiority of our improved A* algorithm over the classical version.

Figure 6.4 displays examples of our improved A* algorithm's implementation within our ordering algorithm. The left side shows original paths from the classical A* algorithm, while the right side presents corresponding paths generated by the improved A* algorithm, which includes both online improvements and offline post-pruning.

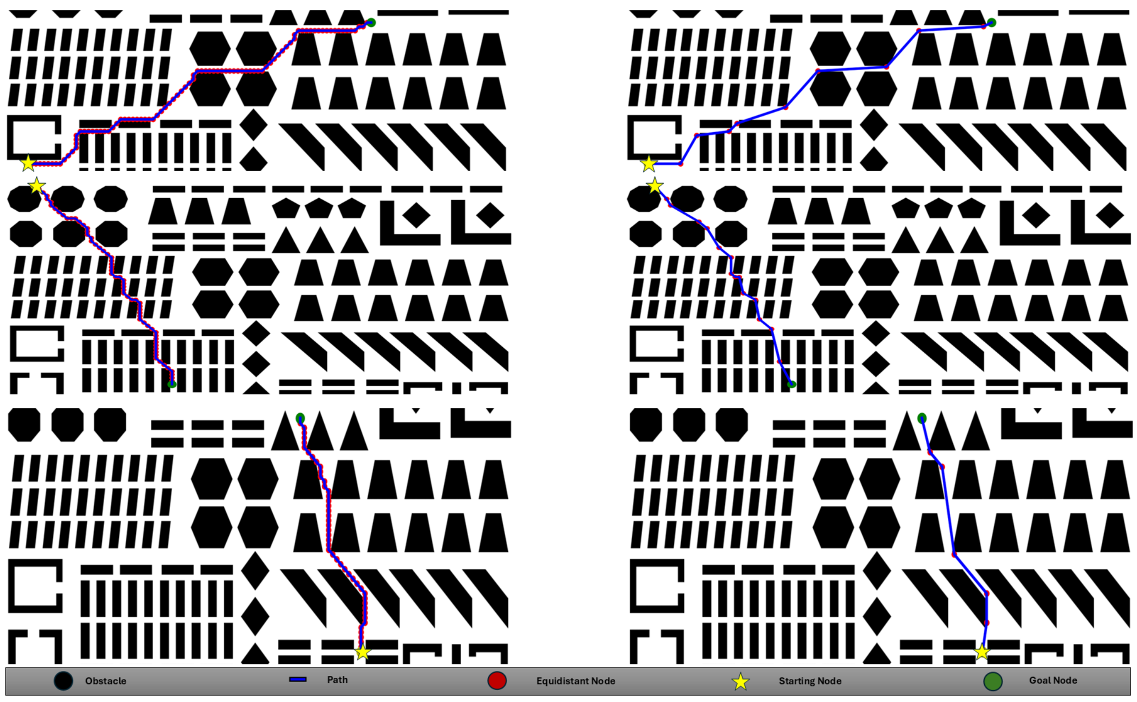


Figure 6.4 Visualisation of some experiments conducted on the improved A* algorithm. The left-hand side images show the classical implementation, and the right-hand side images display the improved implementation.

To quantify these improvements, we conducted 1000 experiments using randomly selected starting and goal nodes. Across these experiments, we calculated the average distances, average execution time, and average number of path nodes. Table 6.2 demonstrates the improved A* algorithm's superiority over other algorithms in terms of both the number of path nodes and pixel cost. Our improved version achieved a 3.828% reduction in pixel cost and an 89.44% reduction in the number of path nodes compared to the classical A* algorithm over these 1000 experiments. Furthermore, the improved algorithm significantly reduced turns, zigzags, manoeuvres, and abrupt changes, leading to smoother paths with fewer variations. This, in turn, makes the robot's motion planning task easier and more efficient.

The improved version did show a slight increase in average execution time compared to the classical A* algorithm. This marginal rise is expected, as the post-pruning process is applied after generating the classical A* output. However, this increase is negligible when weighed against the significant enhancements in average path cost, path smoothness, and the reduction in the average number of path nodes.

The benefits of using the A* algorithm or its variants over the Rapidly-exploring Random Tree Star (RRT*) algorithm are clear in such environments. A* and its variants are informed search algorithms that use complete knowledge of the graph, while RRT* and its variants are uninformed search algorithms that perform well in different scenarios. Although extended computation time and additional rewiring processes might improve RRT*'s average performance, we set a threshold significantly higher than A*'s average execution time. Even so, the RRT* algorithm still produced noticeably higher average path distances.

Table 6.2 Details of the Improved A* experiments. These experiments were conducted using 1000 randomly selected starting and goal nodes. We implemented and compared the classical A*, Bidirectional A*, Rapidly-exploring Random Tree Star, and Improved A* algorithms in terms of $\overline{NN}$: Average Number of Nodes, $\overline{D}$: Average Distance, $\overline{ET}$: Average Execution Time, $\overline{PI}$: Average Percentage Improvement.

Avg. of 1000 Paths	$\overline{NN}$	$\overline{D}$ [pixel]	$\overline{ET}[s]$
Classical A* Path	56.363	2540.788	0.202
Bi-A* Path	55.490	2544.337	0.044
RRT* Path	39.279	3827.900	1.515
Improved A* Path	5.952	2443.530	0.290
$\overline{PI}$	89.440%	3.828%	-

6.2.2 Improved A* Experiments On Publicly Available Maps

In this section, we are testing the scalability and reproducibility of our improved A* algorithm using several publicly available maps from Bhardwaj et al. (2017); Sturtevant (2012). These maps, provided in the correct input format, offer varying levels of complexity and environmental features. For each map, we set an appropriate spacing, performed the necessary pre-processing steps, and then randomly selected starting and goal nodes to apply our improved A* algorithm.

Figure 6.5 demonstrates that as long as the input format is suitable and the spacing condition is satisfied, the improved A* algorithm performs flawlessly across any map and environment. We used two categories of public maps:

- Maps with Dominating Obstacle Area
- Maps with Dominating Available Area

In both scenarios, our improved version consistently showed excellent performance in finding the shortest path between given nodes, proving its robustness and adaptability.

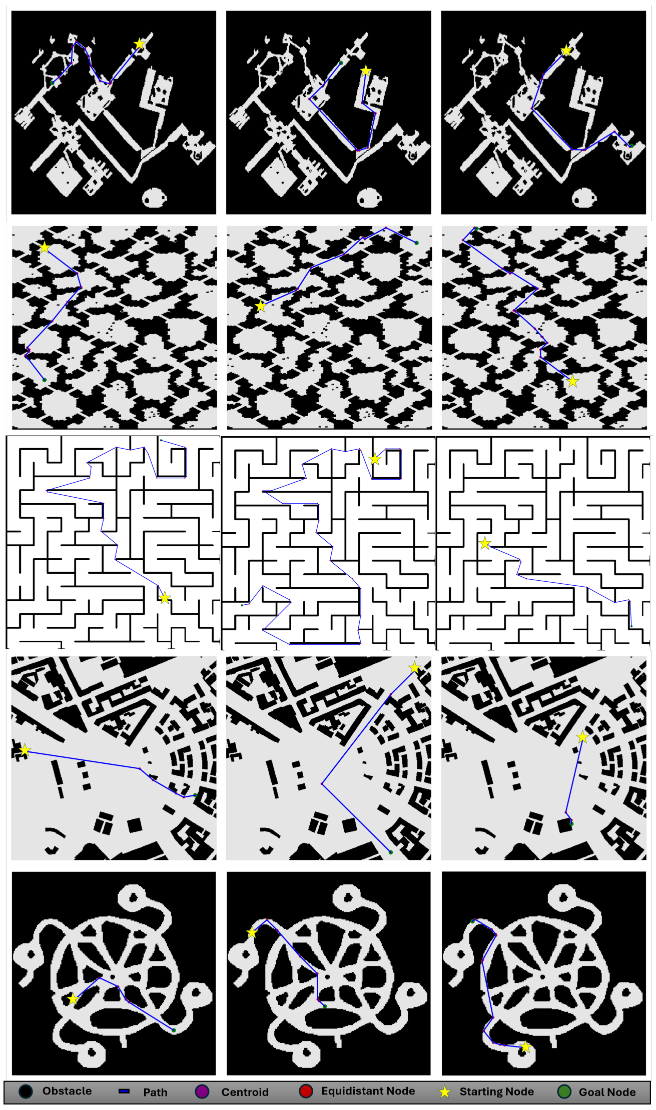


Figure 6.5 An illustration that demonstrates the scalability of the improved A* algorithm by applying it to various appropriate publicly available dataset maps. This is achieved by randomly selecting starting and goal nodes and specifying a suitable spacing value for each map. Reproduced from Allus & Unel (2025)

6.3 Experiments On The Proposed Classical Ordering Algorithm: Performance Evaluation, Efficiency, Scalability, and Reproducibility Analysis

In this section, we will evaluate the performance of our proposed classical ordering algorithm, focusing on its two individual variants and a combined ensemble approach. We will conduct several thorough experiments and comparisons, measuring key metrics like cost and time complexity.

6.3.1 Comparison Between Combined Angle-Based, Sequential Angle-Based Ordering and Optimal Ordering

For this experiment, we're testing both variants of our proposed ordering algorithm in a straightforward computational scenario involving five goal nodes. We'll then compare the goal node sequences generated by both algorithms against the optimal order found using the brute-force technique, which is computationally feasible for five goal nodes.

The results in Table 6.3 show promising outcomes regarding the average cost difference between our proposed ordering algorithm's variants and the absolute optimal cost over 100 experiments. As anticipated, the proposed algorithm's time complexity significantly outperforms the brute-force technique. However, testing with only five goal nodes is not enough to draw strong conclusions about the algorithm's overall performance. Therefore, we will increase the number of goal nodes in upcoming experiments to properly evaluate scalability and derive more meaningful insights.

Table 6.3 A table that compares the optimal ordering, combined angle-based ordering, and sequential ordering. It shows their average distance cost and execution time over 100 experiments with randomly selected starting nodes and sets of 5 goal nodes. $\overline{D}$: Average Distance, $\overline{ET}$: Average Execution Time, CABO: Combined Angle-Based Order, SABO: Sequential Angle-Based Order.

100 Exps on 5 GNs	$\overline{D}$ [pixel]	$\overline{ET}$ [s]
Optimal Order	11664.203	26.943
CABO	11931.566	0.829
SABO	12185.132	0.407

6.3.2 Experiments on The Scalability of The Proposed Classical Algorithms in Highly Computational Scenarios

To evaluate our proposed algorithm's capacity to handle highly computational scenarios, we conducted four experiments, each averaged over 100 iterations. These experiments involved varying numbers of goal nodes, from 15 in low-complexity environments up to 100 in high-complexity settings. In each iteration, both the goal nodes and the starting node were randomly selected from navigable areas. This random selection introduced diverse distributions of goal nodes and varied the starting node's position relative to both the goal nodes and the centroid. By performing 100 iterations, we aimed to capture as many different scenarios as possible and identify any outliers or unexpected behaviours. We compared our proposed algorithms' output to the nearest neighbor algorithm, a basic model for solving the ordering problem, and the MuGONA algorithm, an algorithm introduced by Allus et al. (2024) that uses only distance criteria to order the nodes, as finding the optimal order for a large number of goal nodes is computationally unfeasible. The nearest neighbour algorithm served as a baseline ordering algorithm for comparison.

The results in Table 6.4 show that both variants of our proposed algorithm significantly outperform both the nearest neighbour algorithm and the MuGONA algorithm in terms of time complexity. Regarding distance cost, both variants of our algorithm generally outperformed the other algorithms, except in the scenario with 50 goal nodes, where the Combined Angle-Based Ordering variant was surpassed. To achieve more consistently superior results, an ensemble of both variants was tested, as discussed in Subsection 6.3.3. The comparison with MuGONA highlights the efficiency of the One-Distance-Two-Angles strategy, demonstrating its advantage over solely relying on distance measurements for determining the order of goal nodes.

The Sequential Angle-Based Ordering variant outperformed both of the other algorithms in all scenarios concerning both distance cost and time complexity. The Combined Angle-Based Ordering variant came close to the nearest neighbour algorithm in terms of distance cost in two of the highly complex scenarios.

Table 6.4 Scalability experiments on both variants of the proposed algorithm. The experiments were conducted on 100 batches of randomly selected starting and goal nodes. The average quantities are calculated and displayed in the table. $\overline{D}$: Average Distance, $\overline{ET}$: Average Execution Time, NGNs: Number of Goal Nodes, NNO: Nearest Neighbour Order, CABO: Combined Angle-Based Order, SABO: Sequential Angle-Based Order.

Avg. of 100 Exps	$\overline{D}$ [pixel]	$\overline{ET}$ [s]	NGNs
NNO	19262.138	38.163	15
MuGONA	19144.219	13.854	
CABO	18444.863	7.521	
SABO	18531.764	8.351	
NNO	24777.373	96.970	25
MuGONA	24278.923	17.700	
CABO	24256.323	12.718	
SABO	23497.753	12.259	
NNO	34987.693	348.962	50
MuGONA	35134.617	36.411	
CABO	35310.502	32.607	
SABO	34028.898	32.236	
NNO	47751.578	1486.787	100
MuGONA	49464.362	98.752	
CABO	44541.858	39.651	
SABO	45254.434	68.737	

These experiments suggest that the sequential angle-based ordering is the most effective choice across all scenarios and conditions, while the combined angle-based ordering ranks second. It is superior to the nearest neighbour algorithm in low-complexity scenarios and nearly as effective in high-complexity scenarios in terms of distance cost. Figure 6.6 provides a visual sample of the conducted experiments with varying numbers of nodes. These experiments were performed on the raw outputs of the proposed algorithm. However, in upcoming sections, we will present the excellent post-pruned ensemble outputs, which remarkably outperform other algorithms in terms of both distance cost and time complexity. It is important to note that these experiments are averaged over 100 iterations; however, individual iterations may vary depending on the specifics of the scenario. Consequently, we will propose a solution to address this variability in the upcoming sections (Ensemble Angle-Based Ordering).

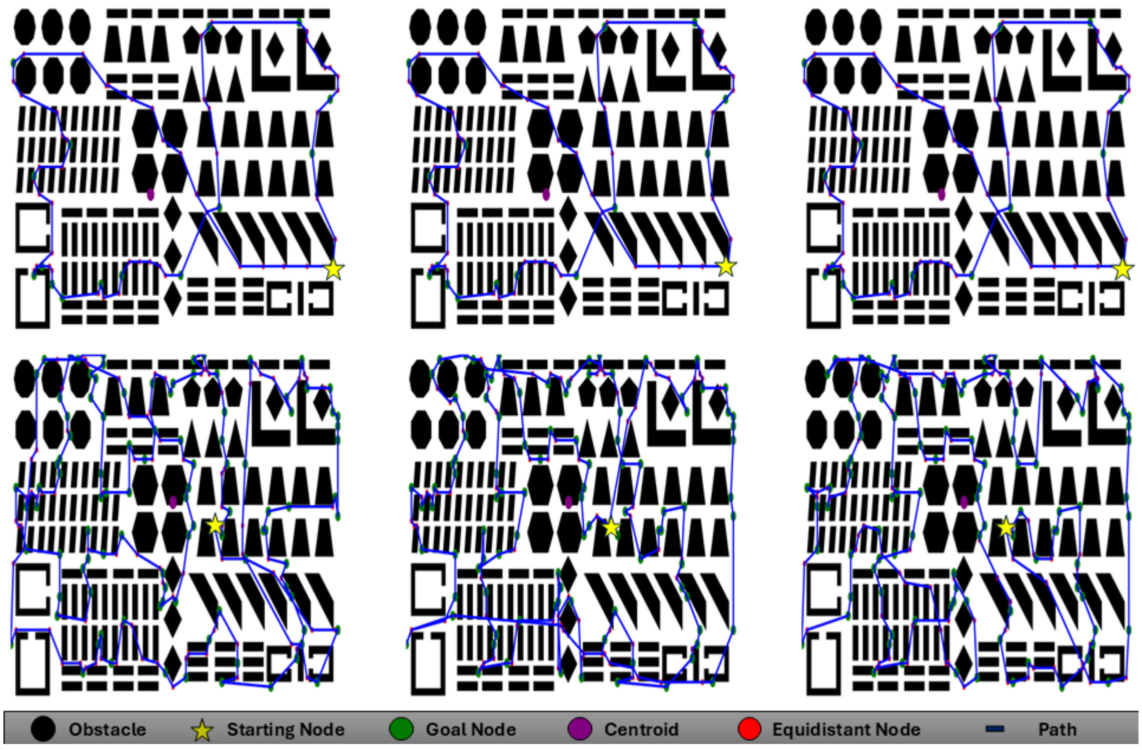


Figure 6.6 An illustration of the scalability of the proposed algorithm, showing its proper functioning in varying complexity scenarios with different numbers of goals (up to 100 goal nodes in this visual example). The first column displays the solution generated by the Nearest Neighbour algorithm, while the second and third columns show the results of the SABO and CABO variants of our proposed algorithm, respectively. The first row corresponds to a scenario with 50 goal nodes, whereas the second row illustrates a scenario involving 100 goal nodes.

6.3.3 Experiments On Angle-Based Ordering Ensemble and Post-Pruning

To confirm the effectiveness of our proposed modified 2-Opt algorithm for angle-based ordering post-pruning, we conducted 100-iteration experiments across five different scenarios, varying the number of goal nodes $N_G \in \{5, 10, 20, 50, 100\}$. These experiments were performed on a standard map where the obstacle area comprised less than 30% of the total area.

The results, presented in Table 6.5, show that the post-pruning algorithm consistently improved the distance cost in all scenarios. However, the improvement was relatively minimal with a low number of nodes and became more significant as the node count increased. This trend is because a higher number of nodes typically leads to more self-intersections in the overall path. Consequently, the post-pruning procedure is particularly beneficial in scenarios with a larger number of goal nodes, as it effectively reduces the occurrence of self-intersections, thereby improving the overall path cost.

Table 6.5 Experiments on the post-pruned ensemble angle-based ordering and comparisons with the normal ensemble to show the effect of post-pruning. $\overline{D}$: Average Distance, $\overline{ET}$: Average Execution Time, $\overline{IDC}$: Average Improvement in Distance Cost, NGNs: Number of Goal Nodes, EABO: Ensemble Angle-Based Order, PPEABO: Post-Pruned Ensemble Angle-Based Order.

Avg. of 100 Exps	$\overline{D}$ [pixel]	$\overline{ET}$ [s]	$\overline{IDC}$	NGNs
EABO	11543.783	3.458	2.228%	5
PPEABO	11286.625	0.248		
EABO	15883.275	3.340	5.857%	10
PPEABO	14953.017	0.116		
EABO	21598.737	7.438	7.851%	20
PPEABO	19903.008	0.176		
EABO	33693.100	35.864	10.377%	50
PPEABO	30196.603	1.386		
EABO	47512.601	122.363	11.637%	100
PPEABO	41983.669	11.815		

These findings emphasize the importance of the post-pruning algorithm, especially in complex scenarios with numerous goal nodes. They also highlight its capability to enhance path efficiency without incurring excessive computational costs.

For a clearer visual understanding of how post-pruning affects the output path of the proposed ensemble angle-based ordering algorithm, Figure 6.7 provides some extreme examples where the post-pruning procedure significantly reduced the total path distance cost of the ensemble algorithm's output.

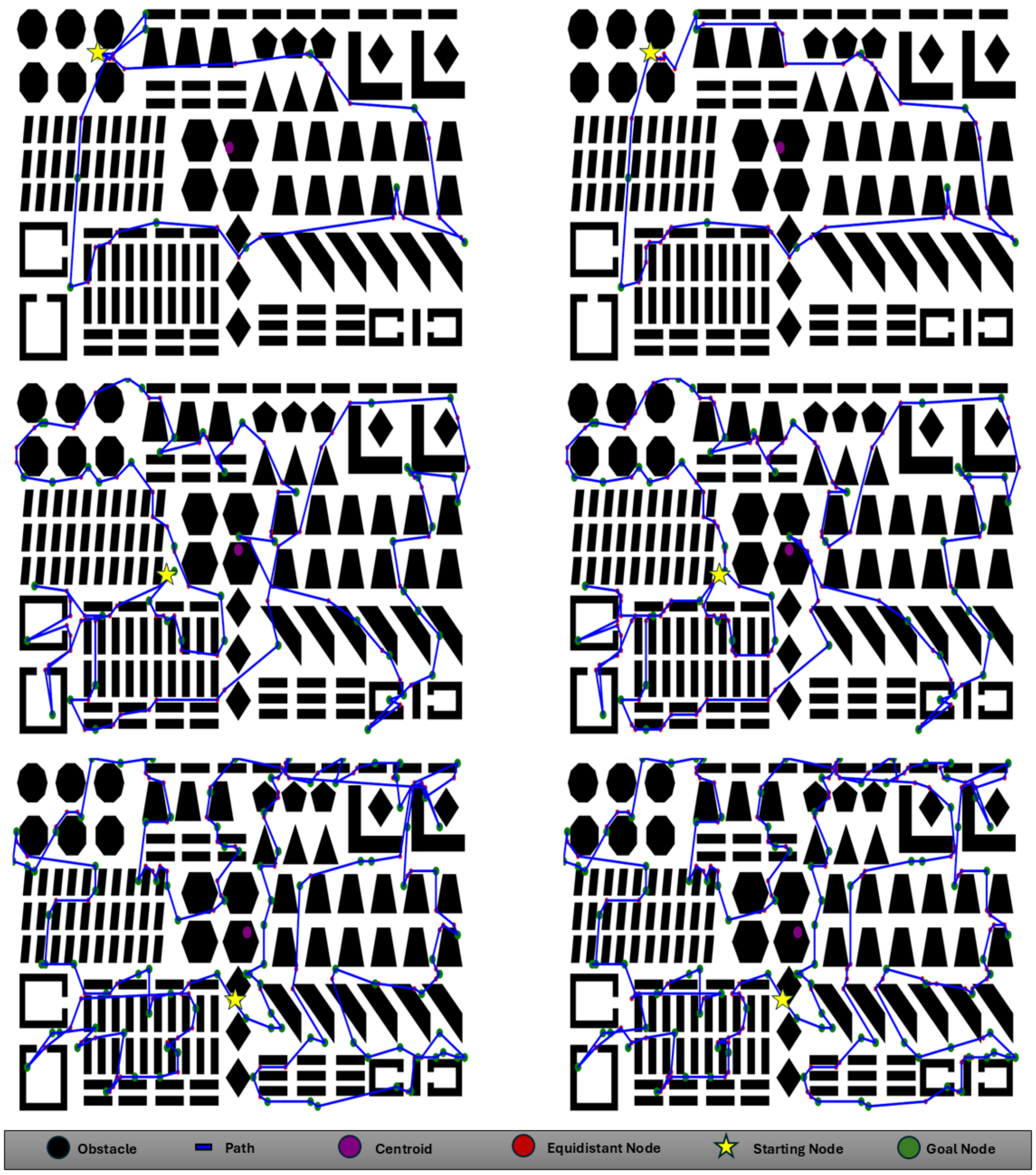


Figure 6.7 A visualisation of some severe post-pruning samples. The first column displays the ensemble angle-based ordered paths, while the second column shows the corresponding post-pruned ensemble angle-based ordered paths. The first row corresponds to a scenario with 10 goal nodes, the second row corresponds to a scenario with 50 goal nodes, whereas the third row illustrates a scenario involving 100 goal nodes.

6.3.4 Experiments On Publicly Available Datasets

To demonstrate the reproducibility of our proposed angle-based ordering algorithm, we tested it on various publicly available maps. Figure 6.8 showcases some of these experiments, featuring different numbers of goal nodes (10, 20, and 30). The results consistently show that our algorithm performs effectively on all maps, provided they adhere to the proper input format and meet the specified spacing condition.

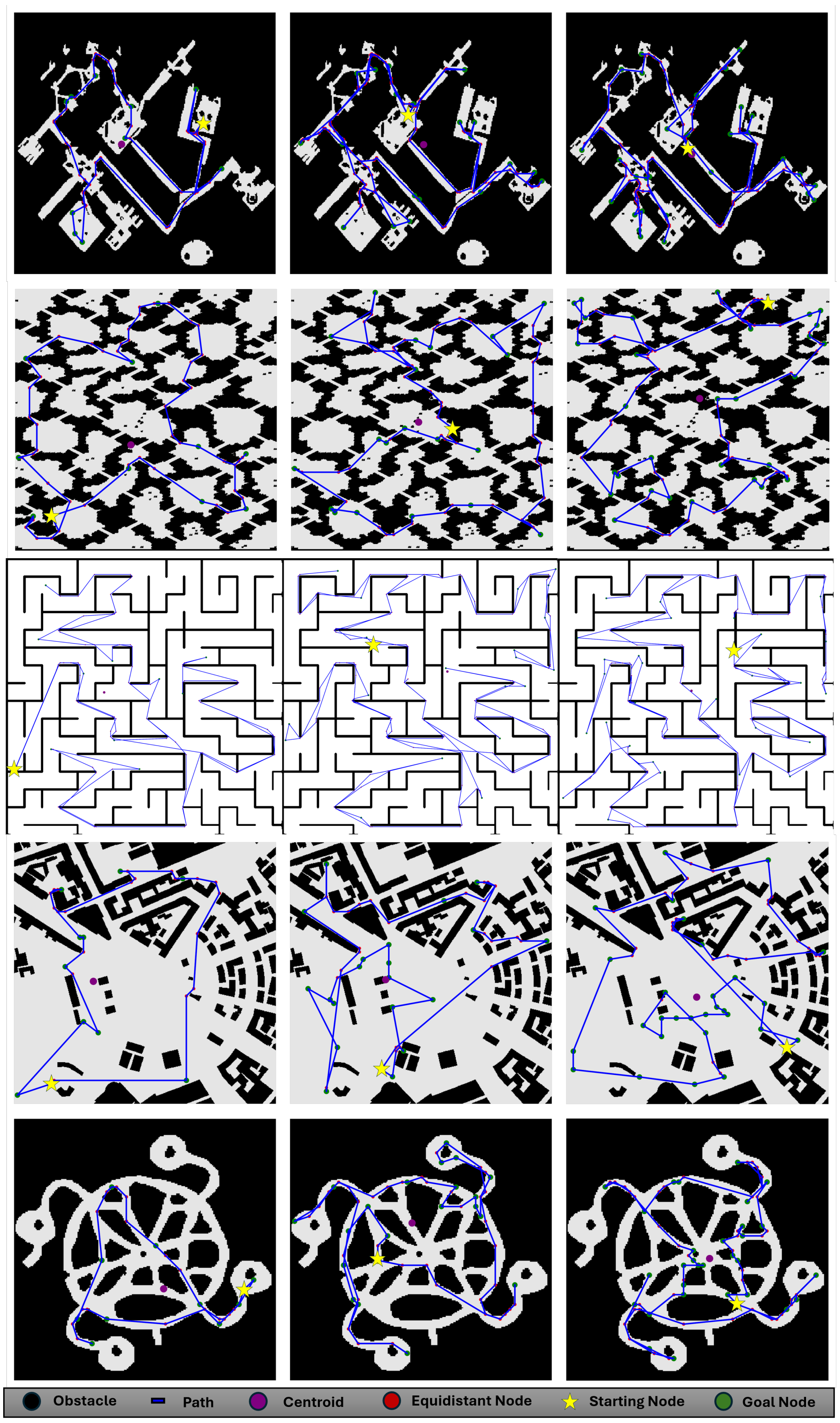


Figure 6.8 A figure that displays the visual outputs from some experiments conducted with the Ensemble Angle-Based Ordering Algorithm on publicly available dataset maps. These maps vary in complexity and the number of goal nodes. The first column illustrates results with 10 goal nodes, the second with 20, and the third with 30 goal nodes. Reproduced from Allus & Unel (2025)

6.3.5 Comparisons With the State-Of-The-Art Algorithms

In this section, we compare the ensemble algorithm, comprising both variants of our proposed angle-based ordering algorithm and a post-pruned version of that ensemble, against solutions obtained by Google's OR-Tools' adapted Traveling Salesperson Problem (TSP) solver and an enhanced brute-force technique, the nearest neighbour algorithm, and the MuGONA algorithm introduced by Allus et al. (2024). The solutions from Google's TSP solver and the enhanced brute-force technique are considered optimal reference solutions. Therefore, we calculated the percentage optimality gap between these optimal solutions and those provided by the other algorithms. The MuGONA algorithm, being one of the most recent ordering algorithms, has shown very strong results against several state-of-the-art ordering algorithms in its own experiments.

Table 6.6 Numerical results comparing the proposed post-pruned ensemble angle-based ordering with other state-of-the-art algorithms. These comparisons involved 100 iterations across four different goal node scenarios, except for the 100-goal node scenario, which was limited to 10 iterations. $\overline{D}$: Average Distance, $\overline{ET}$: Average Execution Time, $\overline{OG\%}$: Optimality Gap, NGNs: Number of Goal Nodes, NNO: Nearest Neighbour Order, EABO: Ensemble Angle-Based Order, PPEABO: Post-Pruned Ensemble Angle-Based Order.

Avg. of 100 Exps	$\overline{D}$ [pixel]	$\overline{ET}$ [s]	$\overline{OG\%}$	NGNs
Google TSP Solver	14742.469	32.611	-	10
NNO	16289.859	9.885	10.496	
MuGONA	16210.014	2.457	9.955	
EABO	15596.771	2.449	5.795	
PPEABO	15520.665	1.710	5.279	
Google TSP Solver	19900.676	133.380	-	20
NNO	23039.265	37.741	15.771	
MuGONA	23108.713	5.967	16.120	
EABO	22423.495	6.146	12.677	
PPEABO	22180.451	5.641	11.456	
Google TSP Solver	29399.812	838.138	-	50
NNO	34913.124	231.408	18.753	
MuGONA	35134.617	36.411	19.506	
EABO	34090.373	37.854	15.954	
PPEABO	33434.428	32.451	13.723	
Google TSP Solver	40113.300	3188.712	-	100
NNO	48542.393	848.058	21.013	
MuGONA	49464.362	98.752	23.312	
EABO	48382.892	123.693	20.616	
PPEABO	46832.874	127.893	16.751	

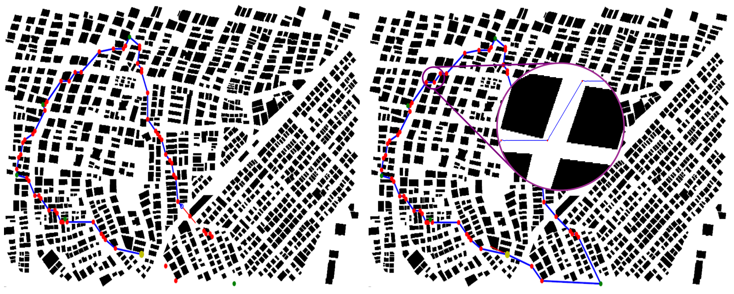
The results presented in Table 6.6 confirm the superiority of our proposed angle-based ordering algorithms in terms of both pixel cost and computational complexity across all scenarios and varying numbers of goal nodes. The only exception is the pixel cost of the TSP solver, which we treat as the optimal solution. Our proposed algorithm significantly reduced pixel cost and maintained a relatively small optimality gap, even in highly computational scenarios involving 100 goal nodes. Without a doubt, the time complexity of our proposed algorithm is the best among all compared algorithms.

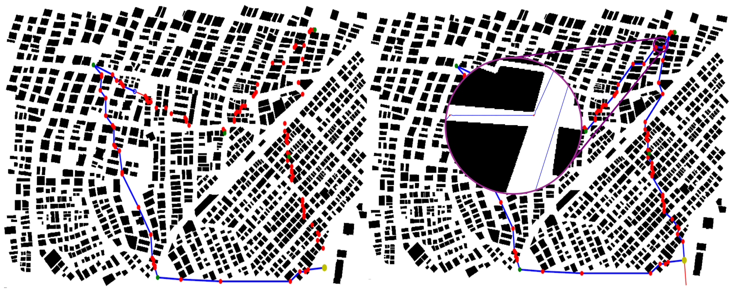
6.4 Experiments with Various Robot Dynamics

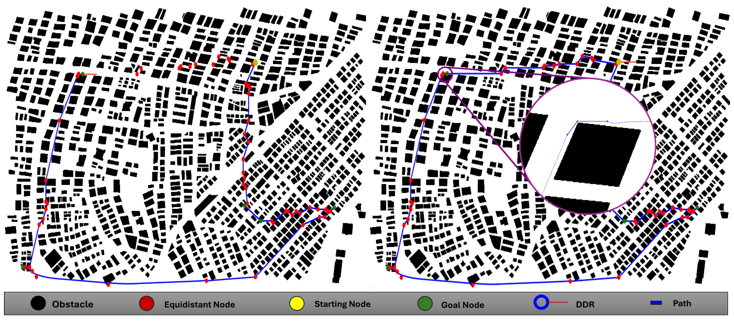
To confirm the robotic applicability of the paths generated by our proposed algorithms, we performed a series of experiments using a map from OpenStreetMap.org, processed with QGIS.org. These experiments involved three distinct dynamic models of mobile robots: a differential drive robot (DDR), a four-wheeled robot with zero-sideslip steering, and a three-wheeled omnidirectional robot.

The specifications for our simulated robots were as follows: The map scale was set to 1 pixel per meter, and the actuated wheels had a radius of 0.1 meter. For the differential drive robot, the wheelbase length was 0.5 meter. For the four-wheeled robot, the distances from the center of gravity to the front and rear wheels were 0.25 meter and 0.2 meter, respectively. The omnidirectional robot had a radius of 0.4 meter, with wheel angles set at $[0, 2 * \pi/3, -2 * \pi/3]$. All robots shared a maximum linear velocity of 3 meters per second and a maximum angular velocity of 1.5 radians per second. A geometric-based controller was implemented for the differential drive robot and four-wheeled robots, while a model predictive controller was used for the omnidirectional robot. The controller parameters were meticulously tuned to ensure near-optimal performance.

Figure 6.9 showcases several experiments validating our proposed algorithms' performance. In 6.9a, the simulated differential drive robot successfully tracks the generated path, with minor inaccuracies attributed to map scaling issues. Despite the lack of path smoothing, the differential drive robot effectively navigates sharp edges, as seen in the zoomed view, with minimal oscillations due to precise parameter tuning. Similarly, using a different set of nodes, the simulated four-wheeled robot navigates the designated path with ease. The zoomed view in 6.9b shows minimal oscillations at a sharp turn, further demonstrating the effectiveness of parameter tuning. Lastly, 6.9c illustrates the trajectory followed by an omnidirectional robot. While minor inaccuracies are observed due to the parameter tuning of the model predictive controller, the overall trajectory exhibits satisfactory performance, even at sharp corners and challenging manoeuvres, as depicted in the zoomed view. Collectively, these experiments validate the applicability of our proposed algorithms across various mobile robot dynamics, highlighting their robustness and encouraging their adoption in real-world mobile robotics scenarios.


(a) An image that illustrates a trajectory followed by a differential drive robot. The left side shows the simulated robot navigating the given path, indicating its position and heading. The right side displays the complete trajectory after navigation, highlighting control behaviours used for sharp turns.


(b) An image that illustrates a trajectory followed by a four-wheeled robot with zero-sideslip steering. The left side shows the simulated robot navigating the given path, indicating its position and heading. The right side displays the complete trajectory after navigation, highlighting control behaviours used for sharp turns.



(c) An image that illustrates a trajectory followed by a three-wheeled omnidirectional robot. The left side shows the simulated robot navigating the given path, indicating its position. The right side displays the complete trajectory after navigation, highlighting control behaviours used for sharp turns.

Figure 6.9 An illustration that shows the navigation procedure performed by simulated robots with three distinct dynamics. The paths are generated using the proposed algorithms on a map of a residential area in Istanbul, Turkey. The map, sourced from OpenStreetMap.org and processed with QGIS.org, displays the trajectories of the robots visiting 5 goal nodes and returning to the starting node, with each robot having a unique set of nodes. Reproduced from Allus & Unel (2025)

6.5 Experiments On The Proposed Learning-Based Framework: Performance Evaluation, Efficiency, Scalability, and Reproducibility Analysis

This section details the procedures for data generation, analysis, model development, evaluation, and case study results specifically for the supervised machine learning framework.

6.5.1 Data Generation

To train our models to identify patterns in optimal closed paths based on predefined features, we selected appropriately formatted maps and randomly assigned starting and goal nodes among the navigable points. Due to computational limitations, we capped the number of goal nodes at 10. We then used an enhanced brute-force technique and Google's OR-Tools' TSP Solver to find the optimal order for connecting these goal nodes and the starting node, aiming for the least costly closed path.

6.5.1.1 Traditional Machine Learning Framework Training Dataset

For the traditional machine learning framework, we used the enhanced brute-force method that starts with a Euclidean heuristic for initial ordering. We generate all possible permutations of the goal node sequence and compare their Euclidean distances, ignoring obstacles. If a permutation results in a lower Euclidean cost than the current minimum, we then calculate its actual cost, accounting for obstacles, using an improved A* algorithm. This process repeats for all permutations, ultimately identifying the optimal order while minimising reliance on the A* algorithm. Combining A* with Euclidean calculations significantly reduces the brute-force technique's computational complexity.

To ensure precise classification, we introduced several hand-crafted features, meticulously designed after extensive testing and observation of optimal paths. For each pair of potential next nodes, we defined a set of features and added boolean indicators to show if specific angles of the nearest potential next node were greater than those of the second node. This approach, along with quantitative features like angles and distances, provides descriptive features that effectively relate pairs of potential next nodes.

This data generation method yields $2 \cdot (N-2)$ data points per ordering scenario, where N is the total number of goal nodes.

Finally, to maintain a global perspective on the ordering process, we introduced a feature called *orderedNodesRatio*, which indicates the stage of the ordering process by referencing the position of the current node relative to the already ordered goal nodes.

6.5.1.2 Transformer Framework Training Dataset

To generate the training dataset for the transformer framework, we employed Google's OR-Tools TSP Solver to obtain exact optimal visiting sequences. For each problem instance, we first compute the exact A* distances between every pair of goal nodes, storing the results in a distance matrix. This involves $\frac{N \cdot (N-1)}{2}$ A* computations for N goal nodes, as distances are symmetric and only the upper (or lower) triangular part of the matrix is needed. The resulting matrix is then provided to the TSP solver to determine the shortest possible tour. Although this method is slower than the enhanced brute-force technique used previously, it guarantees optimality and allows us to diversify the data generation strategies, thereby enriching the training set with high-quality examples.

After determining the optimal visiting order for a given problem, we extract the CNN and relational transformer features for each goal node. Since these features are independent of the ordering steps, they are computed once and stored immediately. For a problem with N goal nodes, there are $N-1$ ordering steps. We treat each step as a distinct decision scenario and, for each, generate the corresponding step-specific features for every goal node. These are then combined with the previously extracted CNN and relational transformer features to form the complete input representation for that step.

Feature Normalization

Because feature generation occurred across different maps and scenarios, all features were normalised to the range $[0, 1]$ based on their corresponding maximum values. Angles were normalised by dividing by 180°, distances and coordinates by the maximum dimension of their respective map or the diagonal of the map, and the *orderedNodesRatio* by the total number of goal nodes.

6.5.2 Performance Comparison of Enhanced vs. Standard Brute-Force Techniques

Table 6.7 clearly demonstrates the significant advantages of our enhanced brute-force technique over the standard approach. This superiority is evident in both the number of A* algorithm executions and the overall time needed to find optimal orders. The standard brute-force method requires the A* algorithm to be run $(N+1)!$ times (where N is the number of goal nodes), causing the computation time to grow factorially as more goal nodes are added. In contrast, our enhanced brute-force technique strategically executes the A* algorithm only when a genuinely better order is likely. Each experiment was repeated 100 times with different starting and goal nodes, and the results presented are the approximate averages across these trials.

Table 6.7 A table that compares the number of A* algorithm executions and the time taken between Standard Brute-Force and Enhanced Brute-Force Techniques.

Avg. of 100 Experiments	**Goal Nodes #**	**Avg. A* Executions**	**Avg. Time [s]**
A* Brute-Force	5	6!	65
Euclidean-A* Brute-Force		30	1.5
A* Brute-Force	7	8!	3600
Euclidean-A* Brute-Force		90	5
A* Brute-Force	10	11!	-
Euclidean-A* Brute-Force		172	73

6.5.3 Traditional Machine Learning Model Selection and Evaluation

We generated **18,926** data points by ordering random sets of 10 goal nodes along with a starting node across various maps. This dataset was then split into training and test sets using an 80/20 ratio. We trained several models using grid search and 5-fold cross-validation. Afterward, each model was evaluated on the test set, and the one with the highest **F1** score was chosen for ordering goal nodes in new scenarios and for all subsequent experiments.

Table 6.8 presents the evaluation metrics for the top four best-performing models on the test set. While the differences are small, ensemble learning methods performed best overall. The **XGBoost** model, which achieved the highest F1 score, was selected as the primary model for the upcoming experiments. Figure 6.10 displays the normalized confusion matrix for the XGBoost model.

Table 6.8 Test set evaluation metrics for the top 4 models.

Test Set of 18,926 Data Points	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Logistic Regression	82.96%	79.53%	81.24%	80.38%
Decision Tree	83.52%	81.35%	79.95%	80.65%
Random Forest	85.63%	85.59%	80.01%	82.71%
XGBoost	85.74%	84.19%	82.23%	83.2%

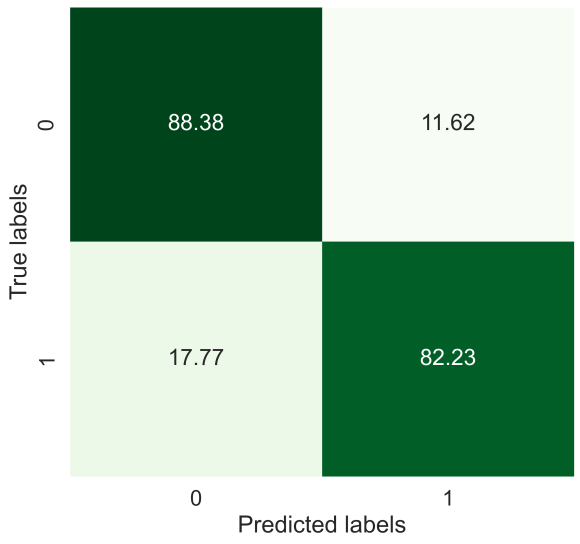


Figure 6.10 Normalised confusion matrix of the XGBoost model.

In 86% of the generated data, the next node was one of the two nearest goal nodes, which aligns with our methodological assumption. Therefore, the model's accuracy can be directly correlated with this percentage.

6.5.4 Traditional Machine Learning Model Evaluation on Workspace Maps

After training our model and evaluating its performance on test data, we conducted further tests on workspace maps. This involved determining the order of goal nodes and comparing the total path cost generated by our model against the optimal closed path cost. This step is crucial because evaluating the model on individual data points can be misleading; the multi-goal path-planning problem aims to optimise the entire path, not just the selection of the next node.

We performed this evaluation on two maps that were also used during the training process. These maps featured diverse obstacle shapes and sizes. Although these maps were part of the training data, we made sure the sets of goal nodes used in this evaluation were entirely distinct from those in the training and test datasets. Therefore, the results still provide valuable insights into the model's capabilities in novel scenarios.

Figure 6.11 shows comparisons between optimal and model-predicted goal node orders on two different maps. Due to computational limitations, the number of goal nodes in these experiments was limited to 10.

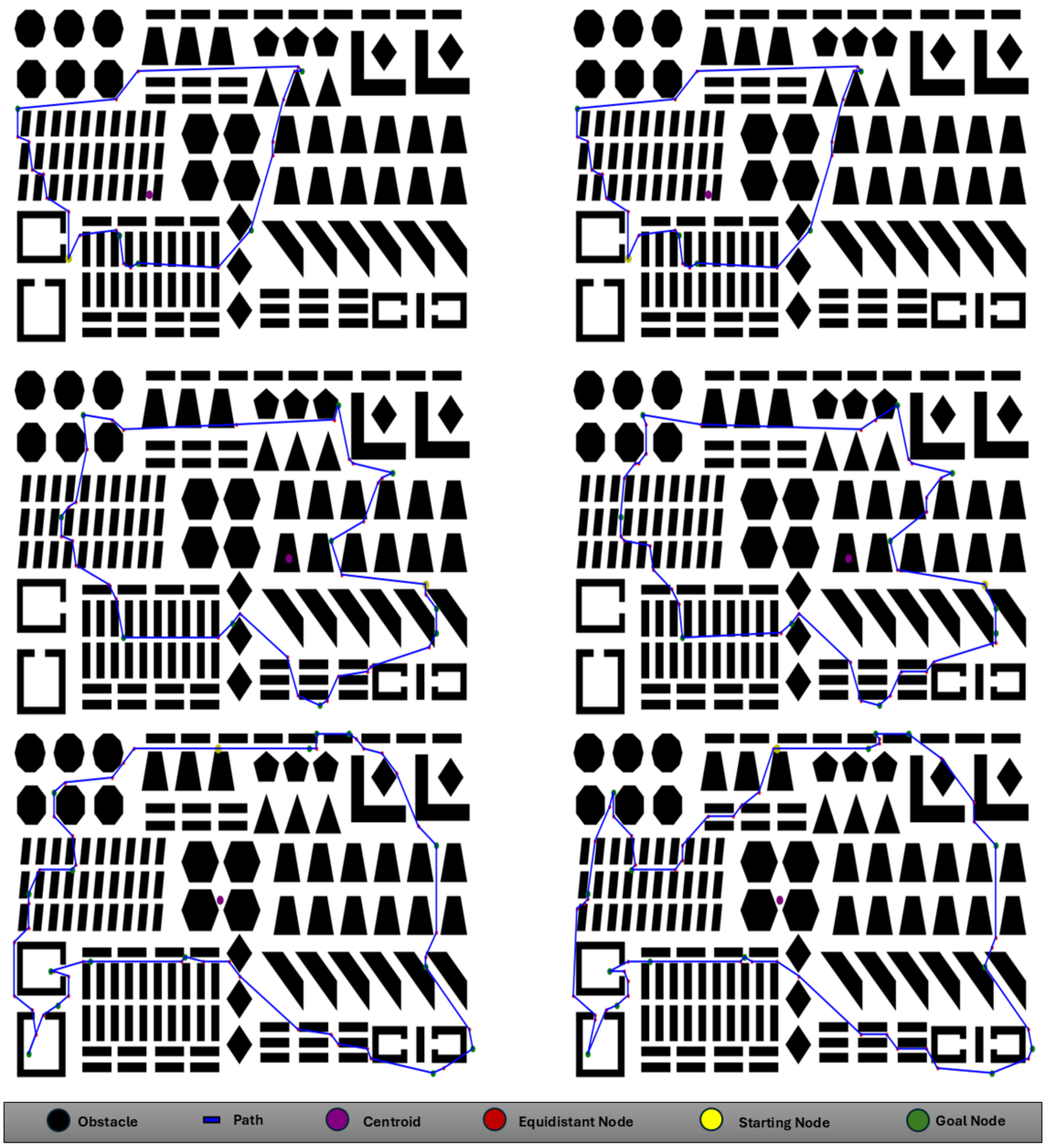


Figure 6.11 A figure that compares optimal and model-predicted goal node orders on the workspace map. The left column displays optimal orders, while the right column shows orders predicted by the model. The first row corresponds to a scenario with 5 goal nodes, the second row corresponds to a scenario with 10 goal nodes, whereas the third row illustrates a scenario involving 15 goal nodes.

Table 6.9 presents promising results regarding the accuracy of the learned orders. The increase in the average optimality gap as the number of goal nodes grows is because the model orders $(N-2)$ goal nodes, with the final two ordered using brute-force, as explained in the methodology section. Consequently, the optimality gap is smaller with fewer goal nodes, though this effect largely diminishes after seven goal nodes. In the worst case, the model's average accuracy is approximately 92.46%.

The model significantly improves ordering time because it only needs to calculate features to predict the next node, unlike the brute-force method, which computes all goal node permutations. This time improvement increases factorially as the number of goal nodes rises.

The consistency of results across maps of varying sizes indicates the model's robustness in diverse workspace settings.

Table 6.9 Comparison of optimal and learned orders on maps used during training.

Avg. of 100 Experiments	**Map Size [pixel]**	**Goal Nodes #**	**Avg. Optimal Cost [pixel]**	**Avg. Ordering Time [s]**
Big Map	(4800, 3840)	5	11346.09	5.523
		7	12736.91	9.501
		10	14744.88	98.377
Small map	(960, 720)	5	2157.115	2.354
		7	2465.553	3.659
		10	2880.93	86.937
Avg. of 100 Experiments	**Avg. Model Cost [pixel]**	**Avg. Ordering Time [s]**	**Avg. Cost Optimality Gap**	**Ordering Time Improvement**
Big Map	11626.823	3.468	2.47%	37.21
	13324.04	4.918	4.61%	48.24
	15857.287	9.437	7.54%	90.41
Small map	2217.628	1.529	2.81%	35.05
	2604.105	2.007	5.62%	45.15
	3087.085	3.72	7.16%	95.72

6.5.5 Transformer Evaluation

A total of **13,160** data points were generated by solving randomly sampled problems consisting of 10 goal nodes and a designated starting node across diverse map environments. The optimal orderings were obtained using Google's Operations Research Tools Travelling Salesperson Problem solver. The resulting dataset was then partitioned into training and testing subsets using a 90/10 split. We trained a transformer-based model and evaluated its inference performance using two distinct approaches:

- Selecting from all remaining unvisited goal nodes
- Selecting from only the two nearest unvisited goal nodes

In the first method, the model receives all currently unvisited goal nodes at each step. The transformer scores each one based on its features, assigning a probability that reflects the likelihood of it being the next node to visit. In contrast, the second approach first identifies the two closest goal nodes to the current node based on Euclidean distance. Only these two candidates are then passed to the transformer for scoring and selection. Both methods employed the same transformer architecture and training configuration and were trained for 50 epochs under identical hyperparameters.

Subsequent evaluation on the test set revealed a clear advantage for the second approach. Feeding only the nearest two nodes into the transformer led to significantly better performance across all key metrics, especially the **F1 score**, which is critical in imbalanced decision scenarios. As a result, this method was selected as the standard approach for goal node ordering in all further experiments.

Table 6.10 summarises the performance comparison between the two inference strategies on the held-out test set.

Table 6.10 Test set evaluation metrics for both transformer training and inference approaches.

Test Set of 13,160 Data Points	**Accuracy**	**Precision**	**Recall**	**F1 Score**
All remaining goal nodes	73.44%	100%	73.44%	84.69%
Nearest two goal nodes	94.69%	100%	94.69%	97.27%

6.5.6 Transformer Evaluation on Workspace Maps

After training our transformer model and evaluating its performance on test data, we conducted further tests on workspace maps. This involved determining the order of goal nodes and comparing the total path cost generated by the transformer against the optimal closed path cost found using the enhanced brute-force technique. This step is crucial because evaluating the transformer on individual data points can be misleading; the multi-goal ordering and path-planning problem aims to optimize the entire path, not just the selection of the next node.

We performed this evaluation on a map that was used during the training process. This map featured diverse obstacle shapes and sizes. Although the map was part of the training data, we made sure the sets of goal nodes used in this evaluation were entirely distinct from those in the training and test datasets. Therefore, the results still provide valuable insights into the model's capabilities in novel scenarios.

Figure 6.12 shows comparisons between optimal and model-predicted goal node orders on the chosen map. Due to computational limitations, the maximum number of goal nodes in these experiments was limited to 10.

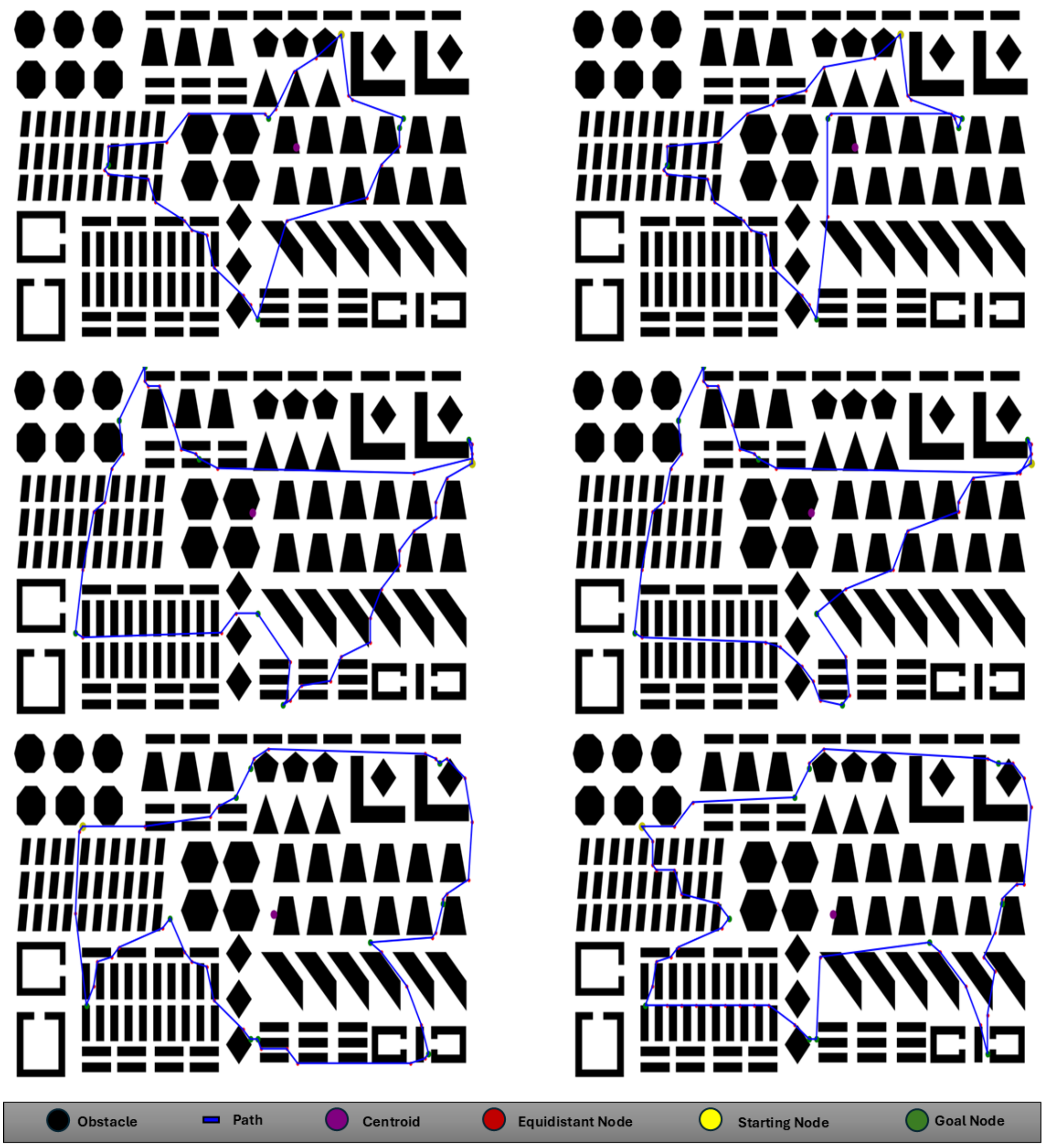


Figure 6.12 A figure that compares optimal and transformer-predicted goal node orders on a selected workspace map. The left column displays optimal orders, while the right column shows orders predicted by the model. The first row corresponds to a scenario with 5 goal nodes, the second row corresponds to a scenario with 7 goal nodes, whereas the third row illustrates a scenario involving 10 goal nodes.

Table 6.11 presents promising results regarding the accuracy of the orders predicted by the transformer. The increase in the average optimality gap as the number of goal nodes grows is because the model orders ((N-2)) goal nodes, with the final two ordered using brute-force, as explained in the methodology section. Consequently, the optimality gap is smaller with fewer goal nodes, though this effect largely diminishes after seven goal nodes. In the worst case, the model's average accuracy as presented in Table 6.10 is approximately 94.69%, which is significantly higher than all average accuracies of the models in Table 6.8, boosting the superiority of the transformer model over traditional machine learning models.

Nonetheless, the model suffers from the drawback of ordering time, as it requires calculating a huge number of features, some of which necessitate many Euclidean distance calculations and orderings, to predict the next node. However, it is still considerably better than the average ordering time of the enhanced brute-force ordering (not to mention Google's TSP solver that takes even more time, or the normal brute-force ordering that takes significantly more time). Additionally, since the ordering time needed by brute-force techniques increases factorially as the number of goal nodes rises, the improvement in ordering time achieved by the transformer becomes more evident with an increasing number of goal nodes.

Table 6.11 Comparison of optimal and transformer orders on maps used during training.

	Map Size [pixel]	**Goal Nodes #**	**Avg. Optimal Cost [pixel]**	**Avg. Ordering Time [s]**
Avg. of 100 Experiments	(4800, 3840)	5	11095.250	29.319
		7	12597.715	50.455
		10	14636.881	139.854
	Avg. Model Cost [pixel]	**Avg. Ordering Time [s]**	**Avg. Cost Optimality Gap**	**Ordering Time Improvement**
	11761.551	26.337	5.67%	10.171%
	13668.596	49.369	7.83%	2.152%
	16113.848	99.685	9.17%	28.722%

6.5.7 Experiments On Publicly Available Datasets

To confirm the reproducibility of our trained models, we ran a series of experiments on publicly available maps that were not used during the training process. Figure 6.13 displays two such maps from Sturtevant (2012). On these maps, we generated optimal orders for randomly selected goal and starting nodes and then used our trained model to order the same sets of goal nodes.

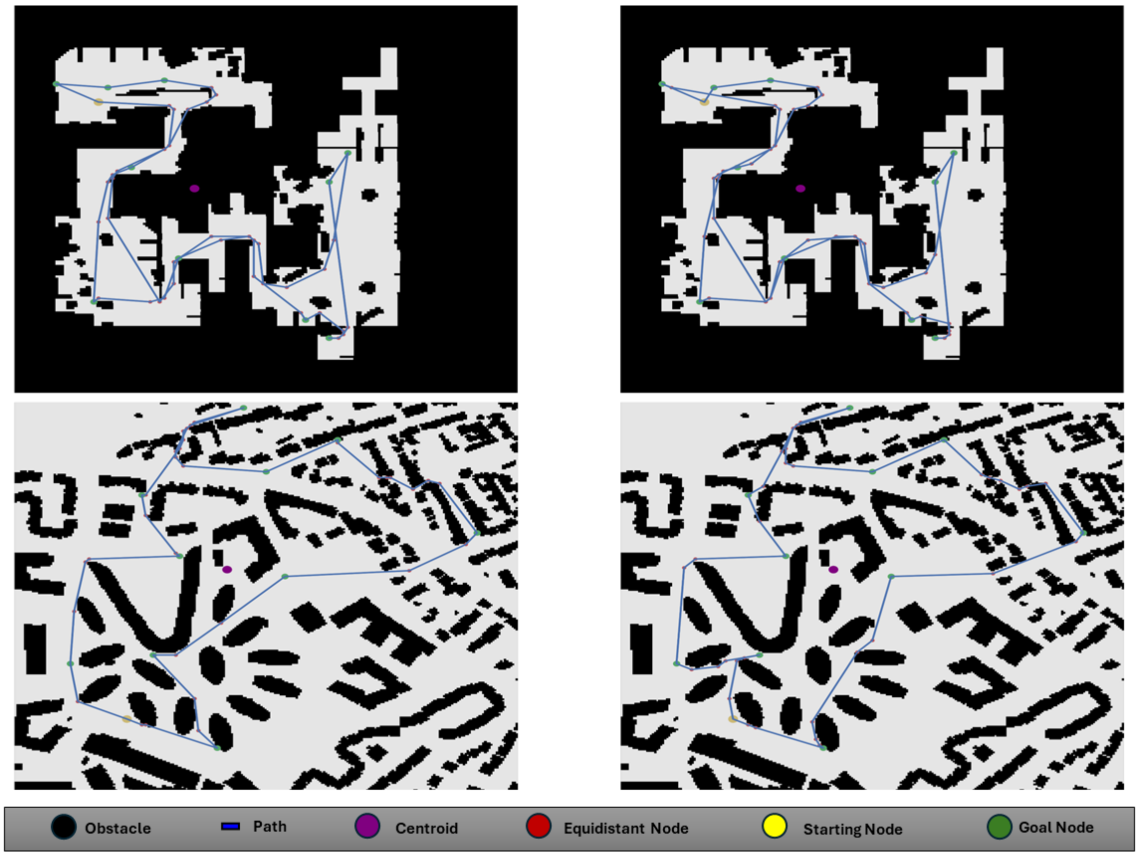


Figure 6.13 A figure that illustrates the evaluation of our trained model on publicly available maps. The left-hand column shows the optimal orders, while the right-hand column displays the corresponding model-predicted orders. Reproduced from Allus & Unel (2025)

Table 6.12 presents numerical results comparing the optimal orders to the model-predicted orders. Tested with three different sets of goal nodes, the results align with our earlier findings from the workspace maps section. Notably, the average cost optimality gap on these publicly available maps is even smaller than that on the training maps, which suggests a higher accuracy than our previously estimated 92.46%. Nevertheless, we continue to consider that estimate as the worst-case scenario.

Table 6.12 Comparison of optimal orders to model-predicted orders on publicly available maps.

Avg. of 100 Experiments	Map Size [pixel]	Goal Nodes #	Avg. Optimal Cost [pixel]	Avg. Ordering Time [s]
lt_undercityserialkiller	(1073, 1073)	5	3472.19	7.19
		7	3806.47	35.72
		10	4202.31	196.13
Paris_1_256	(1073, 1073)	5	2873.43	10.71
		7	3327.56	18.83
		10	3764.91	93.59
Avg. of 100 Experiments	**Avg. Model Cost [pixel]**	**Avg. Ordering Time [s]**	**Avg. Cost Optimality Gap**	**Ordering Time Improvement**
lt_undercityserialkiller	3516.45	3.04	1.27%	57.76%
	3905.09	4.50	2.59%	87.40%
	4348.11	7.74	3.47%	96.06%
Paris_1_256	2940.14	7.02	2.32%	34.48%
	3444.15	9.82	3.50%	47.88%
	4018.75	15.07	6.74%	83.90%

6.5.8 Experiments On the Scalability of the Learned Model

Due to computational complexity, our training process was limited to scenarios with a maximum of 10 goal nodes. However, we hypothesized that this limit was sufficient for the model to learn common patterns in optimal closed paths. To test the model's scalability, we ran experiments with more than 10 goal nodes and compared the learned orders to those generated by an adapted version of the 2-Opt heuristic Chen et al. (2017). As shown in Table 6.13, the average cost of the learned orders is almost identical to that of the 2-Opt orders, indicating that our model's scalability is competitive with state-of-the-art techniques.

Table 6.13 This table compares 2-Opt orders and learned orders to validate the scalability of the trained model.

Avg. of 100 Experiments	Map Size [pixel]	Goal Nodes #	Avg. Opt-Two Cost [pixel]	Avg. Model Cost [pixel]	Avg. Cost Difference
Small Map	(960, 720)	20	4263.12	4251.18	0.28%
		50	6585.61	6585.81	0.00%
		100	9020.72	9074.35	0.59%

7. CONCLUSION

In this work, we proposed a comprehensive set of solutions to the multi-goal path planning problem by integrating classical heuristics and modern learning-based approaches. We began by enhancing the classical A* algorithm to improve the quality of individual path segments. Our enhancement introduces a post-processing technique that applies image-based filtering—using methods such as Bresenham's line algorithm—to remove redundant intermediate nodes and eliminate unnecessary zigzag movements. This significantly improves the smoothness and realism of planned paths, especially in grid-based environments, without incurring additional computational overhead.

Following this, we introduced a novel ordering algorithm, the Angle-Based Multi-Goal Ordering and Path-Planning Using an Improved A-Star Algorithm (AMuGOPIA), which includes two distinct variants, an ensemble approach, and a post-pruning mechanism. These heuristics leverage geometric reasoning based on inter-node distances and angles to propose an efficient visiting order for multiple goals. Evaluated across various maps with different obstacle densities and goal node counts, AMuGOPIA consistently outperforms several state-of-the-art methods in terms of path cost, path smoothness, and runtime, positioning it as a reliable and computationally efficient solution for real-world robotic applications.

Building upon these algorithmic contributions, we further introduced a learning-based framework that leverages hand-crafted geometric features—such as distances, angles, and spatial relationships—to learn goal ordering patterns. This model successfully predicted near-optimal visitation sequences with a high average accuracy, and it generalised effectively to previously unseen environments and large-scale scenarios. Compared to traditional heuristics, the model consistently produced competitive or superior results.

To further improve ordering performance in more complex environments, we developed a Transformer-based learning framework that predicts the optimal visiting sequence of goal nodes. This model, trained on synthetic data derived from optimal

TSP solutions, integrates spatially-informed CNN features, relational transformer embeddings, and hand-crafted geometric descriptors—including distances, angles, and quadrant-based spatial indicators—into a 72-dimensional feature vector. The model operates in a stepwise fashion using a Transformer encoder and a feedforward scoring network to select the next node at each step. During inference, the framework demonstrates competitive performance compared to traditional solvers, while offering significant improvements in scalability, flexibility, and generalisation across unseen environments.

Together, these contributions present a hybrid pathway to robust multi-goal path planning, combining the interpretability and efficiency of classical methods with the adaptability of learning-based techniques. The modularity of the proposed framework allows for flexible integration into autonomous robotic systems that demand efficient, smooth, and intelligent navigation across dynamic and structured environments.

BIBLIOGRAPHY

Alatise, M. B. & Hancke, G. P. (2020). A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, *8*, 39830–39846.

Allus, A., Diab, A. M., & Bayraktar, E. (2024). Improving efficiency and cost of ordering algorithms in pathfinding using shell layers. *Expert Systems with Applications*, *238*, 121948.

Allus, A. & Unel, M. (2025). Angle-based multi-goal ordering and path-planning using an improved a-star algorithm. *Robotics and Autonomous Systems*, *190*, 105001.

Alshammrei, S., Boubaker, S., & Kolsi, L. (2022). Improved dijkstra algorithm for mobile robot path planning and obstacle avoidance. *Comput. Mater. Contin*, *72*, 5939–5954.

Best, G., Faigl, J., & Fitch, R. (2016). Multi-robot path planning for budgeted active perception with self-organising maps. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (pp. 3164–3171).

Bhardwaj, M., Choudhury, S., & Scherer, S. (2017). Learning heuristic search via imitation. In *Conference on Robot Learning*, (pp. 271–280). PMLR.

Bresenham, J. (1977). A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, *20*(2), 100–106.

Bresson, X. & Laurent, T. (2021). The transformer network for the traveling salesman problem. *CoRR*, *abs/2103.03012*.

Chen, X., Zhou, Y., Tang, Z., & Luo, Q. (2017). A hybrid algorithm combining glowworm swarm optimization and complete 2-opt algorithm for spherical travelling salesman problems. *Applied Soft Computing*, *58*, 104–114.

Davoodi, M., Panahi, F., Mohades, A., & Hashemi, S. N. (2015). Clear and smooth path planning. *Applied Soft Computing*, *32*, 568–579.

Devaurs, D., Siméon, T., & Cortés, J. (2014). A multi-tree extension of the transition-based rrt: Application to ordering-and-pathfinding problems in continuous cost spaces. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 2991–2996).

Dharmasiri, P., Kavalchuk, I., & Akbari, M. (2020). Novel implementation of multiple automated ground vehicles traffic real time control algorithm for warehouse operations: Djikstra approach. *Operations and Supply Chain Management: An International Journal*, *13*(4), 396–405.

Digani, V., Sabattini, L., Secchi, C., & Fantuzzi, C. (2015). Ensemble coordination approach in multi-agv systems applied to industrial warehouses. *IEEE Transactions on Automation Science and Engineering*, *12*(3), 922–934.

Dijkstra, E. W. (2022). A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy* (pp. 287–290).

Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path planning with modified a star algorithm for a mobile robot. *Procedia engineering*, *96*, 59–69.

Erke, S., Bin, D., Yiming, N., Qi, Z., Liang, X., & Dawei, Z. (2020). An improved a-star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems*, *17*(5), 1729881420962263.

Faigl, J. & Hollinger, G. A. (2014). Unifying multi-goal path planning for autonomous data collection. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 2937–2942).

Feraco, S., Luciani, S., Bonfitto, A., Amati, N., & Tonoli, A. (2020). A local trajectory planning and control method for autonomous vehicles based on the rrt algorithm. In *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, (pp. 1–6).

Fragapane, G., De Koster, R., Sgarbossa, F., & Strandhagen, J. O. (2021). Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *European Journal of Operational Research*, *294*(2), 405–426.

Gao, J., Ye, W., Guo, J., & Li, Z. (2020). Deep reinforcement learning for indoor mobile robot path planning. *Sensors*, *20*(19).

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hongyun, L., Xiao, J., & Hehua, J. (2013). Multi-goal path planning algorithm for mobile robots in grid space. In *2013 25th Chinese Control and Decision Conference (CCDC)*, (pp. 2872–2876). IEEE.

Hu, B., Cao, Z., & Zhou, M. (2021). An efficient rrt-based framework for planning short and smooth wheeled robot motion under kinodynamic constraints. *IEEE Transactions on Industrial Electronics*, *68*(4), 3292–3302.

Huang, J.-K., Tan, Y., Lee, D., Desaraju, V. R., & Grizzle, J. W. (2023). Informable multi-objective and multi-directional rrt* system for robot path planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*.

Janoš, J., Vonásek, V., & Pěnička, R. (2021). Multi-goal path planning using multiple random trees. *IEEE Robotics and Automation Letters*, *6*(2), 4201–4208.

Joshi, C. K., Laurent, T., & Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem. *CoRR*, *abs/1906.01227*.

Ju, C., Luo, Q., & Yan, X. (2020). Path planning using an improved a-star algorithm. In *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, (pp. 23–26).

Kumar, N. V. & Kumar, C. S. (2018). Development of collision free path planning algorithm for warehouse mobile robot. *Procedia computer science*, *133*, 456–463.

LaVallе, S. (1998). Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, *9811*.

Li, X., Hu, X., Wang, Z., & Du, Z. (2020). Path planning based on combinaion of improved a-star algorithm and dwa algorithm. In *2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, (pp. 99–103).

Liu, L., Wang, X., Yang, X., Liu, H., Li, J., & Wang, P. (2023). Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*, *227*, 120254.

Liu, L.-s., Lin, J.-f., Yao, J.-x., He, D.-w., Zheng, J.-s., Huang, J., & Shi, P. (2021). Path planning for smart car based on dijkstra algorithm and dynamic window approach. *Wireless Communications and Mobile Computing*, *2021*(1), 8881684.

Mashayekhi, R., Idris, M. Y. I., Anisi, M. H., & Ahmedy, I. (2020). Hybrid rrt: A semi-dual-tree rrt-based motion planner. *IEEE Access*, *8*, 18658–18668.

Masoudi, N. & Fadel, G. (2022). Solving three-dimensional path planning problem using a visibility-based graphical representation of the design space. *Journal of Mechanical Design*, *144*(8), 081704.

Melchior, N. A. & Simmons, R. (2007). Particle rrt for path planning with uncertainty. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, (pp. 1617–1624).

Mele, U. J., Gambardella, L. M., & Montemanni, R. (2021). A new constructive heuristic driven by machine learning for the traveling salesman problem. *Algorithms*, *14*(9).

Min, Y., Bai, Y., & Gomes, C. P. (2023). Unsupervised learning for solving the travelling salesman problem. *Advances in Neural Information Processing Systems*, *36*, 47264–47278.

Murphy, R. R., Kravitz, J., Stover, S. L., & Shoureshi, R. (2009). Mobile robots in mine rescue and recovery. *IEEE Robotics & Automation Magazine*, *16*(2), 91–103.

Piazzi, A., Bianco, C. G. L., & Romano, M. (2007). Splines for the smooth path generation of wheeled mobile robots. *IEEE Transactions on Robotics*, *23*(5), 1089–1095.

Poudel, D. B. (2013). Coordinating hundreds of cooperative, autonomous robots in a warehouse. *Jan*, *27*(1-13), 26.

Pěnička, R., Faigl, J., & Saska, M. (2019). Physical orienteering problem for unmanned aerial vehicle data collection planning in environments with obstacles. *IEEE Robotics and Automation Letters*, *4*(3), 3005–3012.

Qing, G., Zheng, Z., & Yue, X. (2017). Path-planning of automated guided vehicle based on improved dijkstra algorithm. In *2017 29th Chinese Control And Decision Conference (CCDC)*, (pp. 7138–7143).

Saeed, R. A., Reforgiato Recupero, D., & Remagnino, P. (2021). The boundary node method for multi-robot multi-goal path planning problems. *Expert Systems*, *38*(6), e12691.

Sanchez-Ibanez, J. R., Pérez-del Pulgar, C. J., & García-Cerezo, A. (2021). Path planning for autonomous mobile robots: A review. *Sensors*, *21*(23), 7898.

Scholz-Reiter, B., Windt, K., & Freitag, M. (2004). Autonomous logistic processes: New demands and first approaches. In *Proceedings of the 37th CIRP international seminar on manufacturing systems*, (pp. 357–362). Budapest.

Sombolestan, S., Rasooli, A., & Khodaygan, S. (2019). Optimal path-planning for mobile robots to find a hidden target in an unknown environment based on machine learning. *Journal of ambient intelligence and humanized computing*, *10*, 1841–1850.

Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 144–148.

Suzuki, S. et al. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, *30*(1), 32–46.

Tang, G., Tang, C., Claramunt, C., Hu, X., & Zhou, P. (2021). Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment. *IEEE Access*, *9*, 59196–59210.

Vasilyev, I., Kashourina, A., Krasheninnikov, M., & Smirnova, E. (2015). Use of mobile robots groups for rescue missions in extreme climatic conditions. *Procedia Engineering*, *100*, 1242–1246.

Wang, C., Wang, L., Qin, J., Wu, Z., Duan, L., Li, Z., Cao, M., Ou, X., Su, X., Li, W., et al. (2015). Path planning of automated guided vehicles based on improved a-star algorithm. In *2015 IEEE International Conference on Information and Automation*, (pp. 2071–2076). IEEE.

Wang, H., Yu, Y., & Yuan, Q. (2011). Application of dijkstra algorithm in robot path-planning. In *2011 Second International Conference on Mechanic Automation and Control Engineering*, (pp. 1067–1069).

Wang, J., Chi, W., Li, C., Wang, C., & Meng, M. Q.-H. (2020). Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, *17*(4), 1748–1758.

Wooden, D. T. (2006). *Graph-based path planning for mobile robots.* PhD thesis.

Wurll, C., Fritz, T., Hermann, Y., & Hollnaicher, D. (2018). Production logistics with mobile robots. In *ISR 2018; 50th International Symposium on Robotics*, (pp. 1–6). VDE.

Wurll, C., Henrich, D., & Wörn, H. (1999). Multi-goal path planning for industrial robots. In *International Conference on Robotics and Application.*

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, *29*(1), 9–9.

XiangRong, T., Yukun, Z., & XinXin, J. (2021). Improved a-star algorithm for robot path planning in static environment. In *Journal of Physics: Conference Series*, volume 1792, (pp. 012067). IOP Publishing.

Yu, J., Su, Y., & Liao, Y. (2020). The path planning of mobile robot by neural networks and hierarchical reinforcement learning. *Frontiers in Neurorobotics*, *14*, 63.

Zhang, D., Chen, C., & Zhang, G. (2024). Agv path planning based on improved a-star algorithm. In *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 7, (pp. 1590–1595).

Zhang, L., Cai, Z., Yan, Y., Yang, C., & Hu, Y. (2024). Multi-agent policy learning-based path planning for autonomous mobile robots. *Engineering Applications of Artificial Intelligence*, *129*, 107631.