CRH*: A Deadlock Free Framework for Scalable Prioritised Path Planning in Multi-Robot Systems

James Heselden
HES15591313

School of Computer Science

College of Science

University of Lincoln

Submitted in partial satisfaction of the requirements for the
Degree of Masters of Science in Robotics and Autonomous Systems
in Computer Science

Supervisor Dr. Gautham P Das

CMP9140M
August 2020

Acknowledgements

I would like to thank the following people:

Gautham P Das, my supervisor for guiding me through this process, without your support, criticism, and expertise, this project would not have achieved what it did.

Jisha George and Jack Stevenson, for helping me understand the requirements for this thesis, stay focused and stop worrying.

My parents, for helping me find clarity and serenity whilst writing this thesis.

My family, for showing amazing support for my completion of this degree.

Benjamin Williams, for providing a public MSc thesis template for UoL SoCS on Overleaf.

Abstract

Multi-robot systems are an ever growing tool which is able to be applied to a wide range of industries to improve productivity and robustness, especially when tasks are distributed in space, time and functionality. Recent works have shown the benefits of multi-robot systems in fields such as warehouse automation, entertainment and now agriculture.

This project aimed to tackle a problem in multi-robot navigation, in which robots within the system, while sharing a common work-space, are caught in deadlocked situations - they are unable to navigate to their targets, being blocked by one another. This problem can be mitigated by efficient multi-robot path planning. The project was focused around developing a framework for deadlock free, scalable prioritised planning, which offered customisability options for varying alternative methods to improve performance.

The results of the experiments proved the project was able to deliver a solution which was both scalable and deadlock free. The novel customisation options included continuous assignment, dynamic scoring, combinatorial heuristics and context dependency. Of which all but context-dependency showed to have beneficial impacts on the quality of the assignments and in turn the efficacy of the system as a whole by showing a decrease in the total delay in the task executions of the system.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Autonomous mobile robotic technologies have matured over the past two or three decades enabling their uses in many real-world applications. This has resulted in a push towards scaling up the system to large mobile robot fleets to improve the operational efficiency, especially when the tasks are inherently distributed in space, time or functionality. One of the primary requirements to ensure proper coordination between the robots in a fleet, sharing a common working space, is efficient path planning and path allocation. Without efficient sharing of space and path planning, the physical interference between the robots can have a detrimental effect on the fleet operations.

In 2019, a collaborative InnovateUK project (Autonomous robots to support fruit picking) was completed between University of Lincoln, Saga Robotics, and Berry Gardens Growers Ltd, with the use of the Thorvald robotic platform. This project was aimed at investigating the potential gains from using autonomy for moving picked fruit from fruit pickers within polytunnels, to a fixed collation point. A human picker would spend approximately 20% of their working time for transporting fruits. By having robots to aid the pickers with fruit transportation, this time can instead be better utilised for picking operations. One of the main observations from this project was the need for a "congestion free" fleet coordination software to enable deadlock free movement across the entire farm in order to reduce the task execution delays.

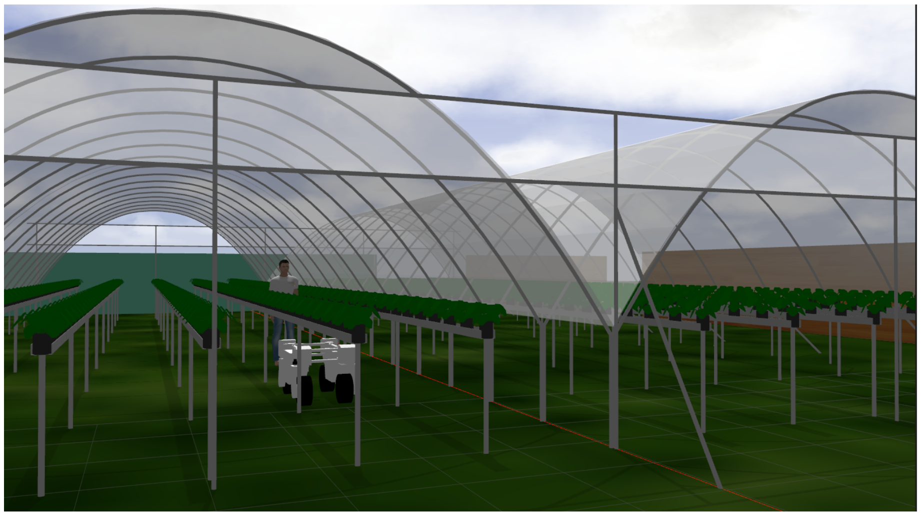


Figure 1.1: A view with human operator and robot from the Gazebo simulation of the strawberry polytunnels at the University's Riseholme campus.

This research, reported in this thesis, addresses the multi-robot path planning problem to enable congestion and deadlock free movement of the robots in agricultural farm environments like polytunnels, e.g. the strawberry polytunnels at the University's Riseholme campus (see Figure 1.1).

1.2 Aim and Objectives

The aim of the project is to develop a framework which offers a scalable, and deadlock-free implementation of prioritised planning. The project also aims to investigate various novel customisation options within prioritised planning systems designed to improve its performance.

This project is broken down into the following set of core objectives:

- Implement a scalable framework for the dynamic assignment of priorities.
- Implement a range of novel customisation options to improve the efficacy of the solution.
- Evaluate the efficacy of each of the novel customisation options.
- Evaluate the scalability of the solution with the optimal set of customisation options.

1.3 Rationale/ benefits

As the availability and accessibility of multi-robot systems grows, there is a paralleled increase in the resource requirements of coordinators for managing their routing. Prioritised planning is a resource inexpensive method to manage this growing demand.

Prioritised planning has a key issue, in that there is potential for robots to become unable to find a path when they are planning later then the other robots, this is referred to as a deadlocked situation.

This research aims to explore methods to guarantee deadlock free routes within a highly congested networks, along with exploring a variety of smaller-impact methods to improve the efficacy of the assignments.

1.4 Report Structure

The remainder of this thesis is broken down into six chapters, which detail the overarching principles and decisions behind how the project has been developed and the results obtained.

Chapter 2 of the thesis is the literature review. In this, the works which have inspired the project have been identified, and their contributions towards this project, are listed and critically reviewed.

The report is then followed with a methodological underpinning to the investigation in Chapter 3. Included in this chapter are the critical evaluations for the decisions made in project management, the rationale for the chosen toolsets, the research methods chosen to gather the results, and the systems and methods for performing an effective evaluation.

Chapter 4 describes the implementation itself, detailing the requirements for the implementation, along with the specifics of the implemented features, and the rationale behind both their inclusion and the manner in which they have been implemented.

The implementation is then followed by the evaluation in which the processes for the six sets of experimental evaluations are detailed, along with the their results, and analysis of their results in Chapter 5. This chapter is aimed at answering both the questions defined within the introduction, along with the further questions identified throughout the design and development. The results are summarised and the aim is evaluated within the conclusion which follows in Chapter 6.

The thesis is finished with the critical reflection in Chapter 7, which evaluates the success of the project, with details on what methodological processes guided the project to success, and which processes could have been improved on. This chapter ends with details on changes and amendments which could be implemented within future works with the project.

Chapter 2

Literature Review

There has been extensive research on Multi-robot systems and enabling efficient co-ordination among the robots within the system. There is some overlap in the focus areas of research between multi-robot systems and multi-agent systems. While multi-agent systems consider both physical (e.g. robots and humans)and software agents, multi-robot systems exclusively consider robots as the agents. Similarly, research in different other domains have similarities with research challenges in multi-robot systems. This chapter discusses the existing research which has guided this work starting from identifying the exact research objectives, to formulating the methodologies and evaluations.

2.1 Structure

The remainder of this chapter, is broken into six sections, beginning with an overview of multi-robot systems and multi-robot path planning. Firstly, the common challenges to MRPP are detailed, looking into the specifics of why the four main challenges of completeness, deadlocks, scalability, and communication are challenges, and what can be done to help manage them.

This is followed by a general description on the categories of MRPP approaches, including a brief description of how each method works within the two categories of coupling and centralisation.

A detailed analysis of prioritised planning then follows, detailing the common approaches within it, and how the approach has evolved since its creation. This section also looks into the varying methods which have been developed to enhance prioritised planning in managing the challenges mentioned in the second section.

The fifth section details some alternative approaches to prioritised planning which have been developed over the years which can be used in conjunction with prioritised planning to improve the quality and scalability of the solution.

2.2 Overview of Multi-Robot Systems

2.2.1 Multi-Robot Systems

Multi-Robot Systems (MRS) have seen a significant increase in their potential for use over the last decade. With higher quality automation and decreasing costs for robots, the potential for autonomous systems has soared. As these trends continue, more systems will see much more mass automation Rizk, Awad and Tunstel, 2019.

MRS have already seen critical development in warehouse automation such as in warehouses Fernandez Carmona, Parekh and Hanheide, 2019, agriculture Bechar and Vigneault, 2016, search and rescue and MRS have even seen promising development within entertainment, with the recent autonomous drone-based firework alternative displays such as in the Shanghai 2020 drone show for the New Year's celebrations.

MRS are becoming so widely used for two main reasons, reduced cost, and higher quality results. Just as in the industrial revolution, development in these sectors have become halted, where very little improvement could be implemented to optimise their internal strategies and processes, with the bottleneck for performance resting on human error and human exhaustion. Mass autonomy can be an effective combatant to this.

By replacing human-based roles, with tools more suited to the task, able to work with must less error and without fatigue, the throughput of the business can be increased along with the quality of the goods. Take for instance the agri-food industry, which has shown with the use of multi-robot systems, can not only increase the throughput of the logistics, but also has the potential to increase the quality of the goods by delivering them to cold storage quicker Das et al., 2018.

2.2.2 Multi-Robot Path Planning (MRPP)

A large factor impacting the effective deployment of MRS is Multi-Robot Path Planning (MRPP), that being, the ability to define paths each robot should take through the environment from its location to its respective goal, which both minimises the total distance each robot it taking through the network, and more importantly avoids all potential collisions with other robots.

This problem has been proven to be NP-Hard in multi-robot systems, as the joint state space grows exponentially as the complexity increases through increasing the number of robots, or the size of the map. This exponential growth is what restricts the scalability and thus deployment of MRS.

2.3 Challenges

2.3.1 Completeness

In an MRPP solution, the aim is to find a route which all robots can make use of in order to reach their targets. For any collection of robots, start locations and targets there is always an optimal collective route, that being the routes which each robot can take in order to minimise the total distance through the network. This optimal route can be defined by minimising the total distance, or the total time to complete the routes, or any other metric. The key principle is that there is some optimal assignment of routes which can be matched but cannot be beaten. Whilst it is a guarantee that such an assignment may exist, it is not always possible to find it. This is because as the size of the joint state space increases, the processing time to find the optimal route increases exponentially.

Completeness refers to the ability to guarantee a solution given enough computing time, i.e. ability to find a route from start to target if such at least one route exist. This is an issue which much of the research into MRPP is focused.

2.3.2 Deadlocks

Deadlock situations are a significant concern for MRPP solutions and deployments. Deadlocks refer to the situation in which a robot is unable to find a route through the environment because it is trapped due to planning inefficiencies or continuously given lower priority while conflicts are resolved. In the summary paper by Dewangan, Shukla and Godfrey, 2017, the authors designed an abstract concept detailing the causes of deadlocks which they used to evaluate the efficacy of a range of prioritised planning approaches. While their aim was to detail specifically prioritised planning issues, the theory can still be extended beyond. They summarised the causes of deadlocks into two key situations. In the first, the cause of the deadlock is the result of a high priority lying in the path required by a low priority robot, and in the second situation, the cause is a high-priority robot following behind a low priority robot.

Papers such as Cap et al., 2015, have attempted to resolve these types of issues by forcing agents to navigate around the initial movement areas of other agents. These methods, while simple, are effective in managing most situations of deadlocks.

2.3.3 Scalability

The largest challenge to successful deployment of MRS is scalability. The grand aim of MRS is to have many robots coordinating and working without issue for long periods of time independently. There are two main concerns with scalability: robot quantity, map size. As these three are increased, the state space increases with them at an exponential rate. As the complexity of the joint state space is increased, there is more data which requires processing and more situations which need evaluating. This leads to a slower processing system overall.

There are solutions to this problem, making use of distributed and decentralised approaches can help lift the burden of a central coordinator which would be handling all the processing by itself, but this comes with other scalability problems such as communication. Decentralisation is good; however, it is not enough alone. Distributed approaches can be adopted for localisation, managing topology updates and MRPP, however these often come with other caveats. MRPP is notorious for causing issues with scalability, especially with approaches which aim to find the complete route.

2.3.4 Communication

Communication is another challenge, within MRS which must be overcome. This issue is especially prevalent within decentralised and distributed MRPP approaches. Within these types of approaches, communication is key to not just the completion of the tasks, but also with regards to safety If communications slow down too much, there could be hazards as robots are moving around possibly in the same workspace as humans. While communication loss may be handled internally to prevent accidents, a loss of coordination between the robots would heavily limit the tasks they are able to perform.

Communication overhead scales linearly with the number of agents in a system when the agents are only required to communicate with the coordinator, however when robots are all communicating their locations in a distributed manner without the use of a central coordinator, this can result in a lot of communication processing for each agent, especially if the agents have to process a lot of information with each communication.

2.4 General Classifications of MRPP Algorithms

There are two classification systems which all MRPP approaches can be defined with. These are the local considerations which described how much communication goes between the agents, and the decision-making topology, which defines the number of processing units doing the planning.

2.4.1 Local Consideration

Within MRPP implementations, there are two types of approaches. These approach types, coupled and decoupled, describe how much consideration the agents take of one another in planning their routes.

Coupled

In a coupled approach, the agents would all be connected, with all routes calculated together simultaneously. Algorithms like the ones produced by Wagner and Choset, 2011; Wagner, Kang and Choset, 2012, take advantage of these types of approaches to identify the optimal route for all agents using a recoupling method.

Coupled approaches often take much more time to process than decoupled approaches as the state space must be explored to a greater depth, however they also guarantee much better routes, often with a guarantee of completeness. The caveat to this however is in the scalability. Coupled approaches, due to the exploration of the state space, are not able to scale well as the size of the state space increases exponentially with the number of agents.

Decoupled

Decoupled approaches work by ignoring the links and connections between agents, instead taking a higher use on conflict resolution. A decoupled approach often works with the agents each having their paths planned separately, then going through a more comprehensive conflict resolution phase. While decoupled approaches can guarantee a reasonable path, it is often not able to guarantee completeness, however in contrast to this, it is often able to guarantee a scalable solution, leading to decoupled approaches being used more in large scale MRPP applications.

Decoupled approaches, as the agents are planned separately, are very effective within distributed and decentralised decision-making topologies, they are also much better at adapting to changes in the environment, as there is much less processing required to replan routes.

2.4.2 Decision Making Topology

The decision-making topology defines the structure of the decision making within the planning, specifically what parts of the system are doing the path planning and the conflict resolution. There are three types of decision-making topology, centralised, decentralised and distributed.

Centralised

A centralised topology is focused around a single point, this single point, the co-ordinator, is responsible for all decisions within the system. Taking full advantage of this, a centralised approach to MRPP often is very simple to implement such as in Erdmann and Lozano-Pérez, 1987, with very little planning, and even less conflict resolution.

Centralised approaches do not scale well as the number of robots increases, as the centralised coordinator is quick to over burden as the state space increases. The coordinator will often be responsible for many processes beyond MRPP, such as localisation and managing topology updates.

Decentralised

A decentralised approach is the next simplest approach, taking advantage of multiple coordinators. There are two types of implementations decentralised approaches can use.

In the first, the coordinators work with groups of agents, in which a coordinator is responsible for say 10 total agents, conducting their own internal planning, and communicating any conflicts with a central coordinator, or a distributed structure of coordinators.

In the second, the distribution of the coordinators is topology based, in which the coordinators take responsibility for managing defined regions within the network. This is shown by Zhao, Liu and Ngoduy, 2019, where the decentralised coordinators work to manage defined junctions, taking control of autonomous vehicles which approach it and coordinating their movement through the junction.

Decentralised approaches help to manage the load from the central coordinator, but still struggle to keep up with the demand as the number of agents requiring management increase. There is a lot of communications overhead with these approaches, and a lot of uncertainty with the path as it is often not given the system approval before all coordinators are considered.

Distributed

Distributed approaches, similarly, to decentralised approaches, work without the use of a central coordinator for MRPP for the main processing unit, however it may use one for global conflict management. A distributed approach is usually also a decoupled one, in which agents are able to plan their full route independent of one another.

In a distributed setup, the main processing is done internally on the individual agents, this means that is scales well, as the number of agents also defines the number of processing units.

2.5 Prioritised Planning

2.5.1 Overview

Priority assignment is a commonly used decoupled approach to MRPP, while it does not guarantee completeness by any means, it does guarantee optimality. Traditional prioritised planning, developed by Erdmann and Lozano-Pérez, 1987, defines it as the combination of two components, the motion planning system, and the priority schema. The motion planning system for each individual agent, plans the route, often using A*. The priority schema is what defines the ordering of the agents. In a traditional priority planning implementation, each agent will be assigned a score based on some heuristic, and then from high to low, the agents will each plan a route. The low priority agents which plan last will have more restrictive paths than the agents with the highest priority agents which planned on an empty map. So the low priority agents will have to plan around the high priority robots which are then represented as dynamic objects.

2.5.2 Prioritisation Schemas

Heuristics

The method for defining the ordering of the agents, is often referred to as heuristics. While simple approaches often work within prioritisation schemas, more complex heuristics can result in more effective orderings of the agents within the network.

The initial ordering for the priority schema developed for use by Erdmann and Lozano-Pérez, 1987, made use of the order the agents were added to the network as their heuristics. While this worked, it was terribly inefficient compared to the further research since, which has shown much better results with more optimal orderings.

In Van Den Berg and Overmars, 2005, there was significant heuristic-based development. The author made use of a heuristic which defined the ordering of the agents based on their distance from their respective targets. Where the agent which has the longest distance to travel was assigned the largest score, with all other agents being assigned respectively smaller scores. The concept behind the project was knowledge-based heuristics, which since, has not been explored too much. This method did have its disadvantages, with using the Euclidean distance to the target, systems which has large regions without any nodes to traverse to, had scores which did not reflect completely, the actual lengths of the routes the agents would have to take. Thus, an alternative implementation to this can be to use the optimal route length, which still retains the same underlying concept, but works with a much more appropriate reflection to more complex topologies.

Another alternative which relied on network knowledge was designed by Velagapudi, Sycara and Scerri, 2010, in which agents were assigned a priority based on how long it took them to find a route. In this, the intent was that agents which took a long time to find a route, would benefit more from planning in an emptier map, and should thus get a higher priority score. This approach like the Euclidean distance, worked well to support the prioritised planning. However due to varying hardware on different robots in a heterogeneous environment, this may prove to be unreliable, it could be improved by instead tracking the number of calls within the system or the number of task execution failures. It is worth noting that the focus on this paper was not in the heuristic but the accompanying distributed prioritisation algorithms which were designed to decrease the communication overhead for the implementation. In the first of the two algorithms, the communication between robots is limited so robots only communicate their intentions with a few defined robots, which in turn communicate with a few more, with the statistical hope that it would end up informing all the robots which required the information. The second algorithm developed was based around the idea of only informing the robots which would be affected by the path, about it. Both algorithms served to improve the communications overhead in fully distributed networks.

In Bennewitz, Burgard and Thrun, 2002, the author worked on a heuristic which took full advantage of the topology knowledge, this heuristic, which has been further titled Naive Surroundings, works to score agents based on how cluttered their workspaces are. The intent behind this is that agents which have a more cluttered workspace, would struggle more to find a route out of the workspace and may become trapped by other agents moving through it with higher scores. The approach counts the number of obstacles surrounding the agent in some defined radius and bases the scoring on that. It is termed naive as it does not consider the connection with the agents movement and its surrounding environment, this is in contrast to the Coupled Surroundings heuristic developed by Wu, Bhattacharya and Prorok, 2019 which only counts the obstacles which are effected by the movement of the agent.

The author of Wu, Bhattacharya and Prorok, 2019 worked to develop their own knowledge-based heuristic titled Path Prospects, in this, the agents are scored based on the number of surrounding paths which are available to them, calculated by counting the number of effective obstacles between them and their target through analysis of the homology classes of trajectories. They evaluated their approach against seven alternative knowledge-based heuristics, including their own approaches, along with some based on the work in Bennewitz, Burgard and Thrun, 2002, random assignments and a couple more. Their results showed for simulations comparing their results to the benchmarks set out, they were able to achieve a higher optimality with respect to completeness, which offered a good balance between make-span and flow-time.

Rescheduling

The biggest drawback to prioritised planning is completeness. Prioritised planning due to the manner in which agents are assigned, defines completeness as the optimal priority scheme for the agents which guarantees the optimal path for each agent through the network. The only way in order to guarantee completeness within prioritised planning originally, was, as described by Azarm and Schmidt, 1997, to perform a complete search through every combination of priority orderings. This however is a completely unscalable approach when applied to large MRS, as the number of priority combinations is a factorial of the number of agents in the network.

This is where rescheduling come into play, rescheduling is the modification of the priority scheme, in order to optimise small aspects of the generated orderings.

This was first explored within prioritised planning by Bennewitz, Burgard and Thrun, 2001, where they employed an approach making use of random rescheduling. In this, whenever a deadlock occurred, the priority scheme was randomised, and the plan completed again. Whilst simple, this was developed before the development and deployment of knowledge-based heuristics, so given the time, this design was appropriate and was successful at avoiding deadlocks in most situations when a deadlock was avoidable.

This work was later improved on by Bennewitz, Burgard and Thrun, 2002 which made use of a hill climbing search in which individual agents which had issues with their ordering resulting in deadlocks, would have their scores switched, and forced to replan. This hill climbing search decreased the total search space being explored to something far more reasonable then given by Azarm and Schmidt, 1997, and was still scalable, however took time to calculate all the new routes each time an amendment was made.

Another method was proposed in 2006 by 'Cooperative multi-robot path planning by heuristic priority adjustment' 2006. In this, the author proposed a more complex solution to rescheduling called continuous enhancement. In this, rescheduling works by allowing agents to modify their own score if they detect they will become blocked, unable to find a route through the environment. This, while still quite simple, was proven to work in many situations, and as the author states, is very robust with responding to dynamic changes within the environment.

This was simplified in 2018 by Andreychuk and Yakovlev, 2018, where instead of allowing agents to adjust their own score, the coordinator would take responsibility, awarding a score of the maximum allowed, when an agent fails to find a route. While very simple, this deterministic rescheduling is much simpler and decreases the total number of plans attempting to find a route. This works with the consideration that agents which would fail a route, may fail many routes due to the complexity of its local environment, thus would be better off with a higher priority. This follows the same mentality as the Naive Surroundings by Bennewitz, Burgard and Thrun, 2002.

Rescheduling was also evaluated under Ma et al., 2019, in which rather than reconfiguring only single priorities, the authors aimed to evaluate multiple configurations. While keeping the search space small, the implementation was designed to, identify when a deadlock was detected, and then find the agents within a certain radius of the deadlock and try to replan with all possible partial configurations of orderings for them. This method, while seemingly intensive, was able to perform similarly to alternatives, and was effective at improving the quality of the orderings.

A similar approach was taken in 2019 where Hu and Cao, 2019 made use of a method called priority tuning. In this approach, regardless of whether a deadlock occurred, once planning is completed, the agents with the least optimal routes would have their scores swapped with agents with slightly better scores, then the agents would all replan their routes with the new orderings. This would continue until there is no considerable change within the results. The implementation was somewhat similar to a combination of sorting algorithms and gradient descent. The authors also with this paper included an asynchronous approach to prioritised planning, which was employed to accelerate the tuning process. Their results show the novel method was able to achieve higher convergence scores in reasonable time.

2.6 General Approaches

2.6.1 Path Finding Algorithms

Most decoupled approaches to MRPP are split into two distinct sections, firstly is the motion planning in which a path is generated for each agent. This is then followed by the conflict resolution phase in which the paths are modified to avoid the conflicts. In Ferguson, Likhachev and Stentz, 2005, the authors developed a guide to assist with identifying the best approach for a given context along with a description of the four categories for heuristic-based motion planning.

Static Algorithms

Static algorithms are called such as once the optimal route is identified, the algorithms process is complete. Well-known algorithms such as Dijkstra, and A*, work to identify the shortest path to visit by iterating through edges connected from the start node, and calculating the distance from the start, with A* also calculating the distance to the target, and using this heuristic to identify the shortest route. These approaches can be taken as a forwards or backwards approach, by swapping the target and start nodes.

Replanning Algorithms

Replanning algorithms are designed to manage defined optimal routes, as the topology weights change without the requirement of restarting the search as would be required with static algorithms. The Dynamic A* (D*) Stentz, 1995 and D* Lite Koenig and Likhachev, 2002 algorithms are a small sample of some of the developed algorithms in this category. D* Lite is a modified version of a backwards A* approach. It works by performing replanning on small portions of the network when changes are identified, before propagating forwards towards the robot. It is designed in this way to improve efficiency by limiting the impact on states which are not impacted by the changes in the topology.

Anytime Algorithms

In anytime approaches, the goal is to identify a potential solution first, then improve on this initial solution over time. This type of approach means that the solution is generated quickly, but it comes with many drawbacks. The drawbacks include mainly the inability to respond to dynamic environments, where if weights are hanged, or a moving obstacle blocks a path, then the algorithm has to start from scratch, an example of this is the algorithm Anytime Repairing A* (ARA*) Likhachev, Gordon and Thrun, 2004. Anytime approaches work well for single robot path planning, when there is no coupling, however for conditions where many robots are restricted having to plan around one another this type of algorithm does not work well.

Replanning Anytime Algorithms

The replanning algorithm Anytime Dynamic A* (ADA*) Likhachev, Ferguson et al., 2005, makes use of elements from the anytime algorithm ARA*, and the replanning algorithm D* Lite to boost the efficiency in replanning around dynamic obstacles while avoiding. The elements of section reuse and invalidated section repair work to improve the limitations of the component algorithms when integrated together.

2.6.2 Topology Manipulation

Topology manipulation aims to reduce the complexity of the problem to be solved by simplifying and modifying the network structure to reduce the search space and thus increase the scalability.

Road-Map Decomposition

Road-map decomposition works to break apart the network into smaller manageable components which when done, allows for hierarchical planning to be completed, which increase solution quality by reducing the search space. This was developed by M. R. Ryan, 2008, developing a method to break the map into regions based on similarity with internal structuring.

This was further researched by Kavraki et al., 1996 with the goal of improving the planning effectiveness by integrating within the system, prioritised planning. In this, they were able to succeed in decreasing the planning requirements, outperforming both independently.

In M. Ryan, 2006, the author aimed to implement a similar approach to road-map decomposition, in which the map would be broken down into smaller sections based on their structures, such as rooms which were collections of multiple intertwined edges, and rooms which were collections of long isolated paths. This decomposition allowed more routing to be done in a smaller state space. The inspiration for this came from a research study on optimising for Sokoban puzzles Botea, Müller and Schaeffer, 2002 in which the player is to move boxes around warehouses to specific locations without being caught in deadlocks.

Other alternative implementations such as Zhao, Liu and Ngoduy, 2019 made use of junction definition where junctions within the network were isolated and managed independently to improve traffic flow. This approach was particularly suited to decentralised MRPP approaches, where each junction could be represented as an independent decision-making agent.

Probabilistic Planning Probabilistic planning takes advantage of topology building, rather than decomposition. In probabilistic planners, the aim is to identify routes through generating topologies through the network until a path is found. There are two main approaches with probabilistic planners, the first Probabilistic Roadmaps (PRM) Kavraki et al., 1996, are used by generating clusters within the network and growing the clusters until routes can be used to connect them at which point a route is found. Rapidly expanding random trees (RRT) LaValle and Kuffner, 1999, work by generating two expanding trees, from the start and end nodes, expanding them until they touch, at which point the path is defined.

2.6.3 Sub-Dimensional Expansion

Sub-dimensional expansion is a process in which agents will plan in a decoupled manner, then re-couple for regions where they are in proximity of one another. By doing this re-coupling, the implementation of this approach M*, developed by Wagner and

Choset, 2011. Works by decreasing the state space to a low-dimensional state space when agents have no risks around on another but increasing the state space when agents require more precise planning. This work shows it could work to improve the scalability of a solution effectively. This work was later improved on by Wagner, Kang and Choset, 2012 being redesigned and implemented to enhance the quality of probabilistic planners.

2.7 Findings

Through this analysis of existing literature, methods, and approaches to multi-robot path planning with prioritised planning, there were a few issues which seemed to arise quite frequently.

Much of the research does not include testing with actual robots, instead working entirely in simulation. Whilst this is mostly due to resource limitations, especially with testing large multi-robot systems, simulations do come with issues which should be addressed directly regarding the differences between the simulation and reality.

Batch assignment is a consistent concern, in which assignments are only discussed for an initial stage when the robots are all idle. This type of approach is how prioritised planning was originally designed around, where all robots would be idle, then their priorities would be assigned, and they would each plan their routes. This approach leads to agents waiting idle until all other robots are able to stay idle for the priorities to be recalculated after each agent completes their route.

Static scoring was another thing which was identified frequently. When priorities are assigned based on the distance to the target node, as the agents are moving through the network, if replanning is required, this would result in priority scores being used which are not reflective of the new locations of the robots within the system. This results in irrelevant historical information being used to prioritise the robots, which contrasts the purpose of using knowledge-based heuristics entirely.

Overall, through the review conducted, there was very little researched in the heuristics used for prioritised planning, with the majority of identified heuristics, included as an add on for other focuses and motivations of the papers. There is still a lot of research which can be conducted in this area as it has been explored so little.

Finally, all of the heuristics identified, are context-independent, in which the context of the implementation has no connection to the heuristic developed. This can lead to much of the knowledge of the context being neglected which could offer valuable insight to the prioritisation of the robots. This may simply be that research in this area is not as likely to be published as it is not as globally applicable, however is inclusion within the papers as an add on component would still be beneficial.

There is a lot available to investigate, and this project will aim in part, to investigate these areas as additional novel components.

Chapter 3

Methodology

3.1 Project Management

A key objective in the project is the design and development of a novel framework to support the implementation and testing of the hypotheses. In developing a novel framework, there are often issues which come up as requirements are discovered throughout the implementation. To support the development of the framework, this was taken into account and the idea was taken to begin the project with an exploration (or requirements gathering) phase in which the algorithms would be designed in an iterative process before being implemented in different simulations of differing complexities to evaluate.

To detail the overview of the project, the Gantt Chart in Figure 3.1 was put together, and was the main driving force behind the weekly task assignment.

A third of the way into the project, a project review was completed on the progress to make any necessary adjustments to the project plan taking into account the time estimates for the major tasks. During this review, a Lean approach was taken in which the later half of the project was restructured to take a stronger focus on the parts of the project which would contribute the most value. In this, the latter two simulations for evaluation were removed, and the first evaluation simulation was modified to make use of the development framework designed in the exploration phase. Lean approaches to project management are focused on shifting the focus from delivering multiple technologies riddled with fluff, to delivering value. Thus

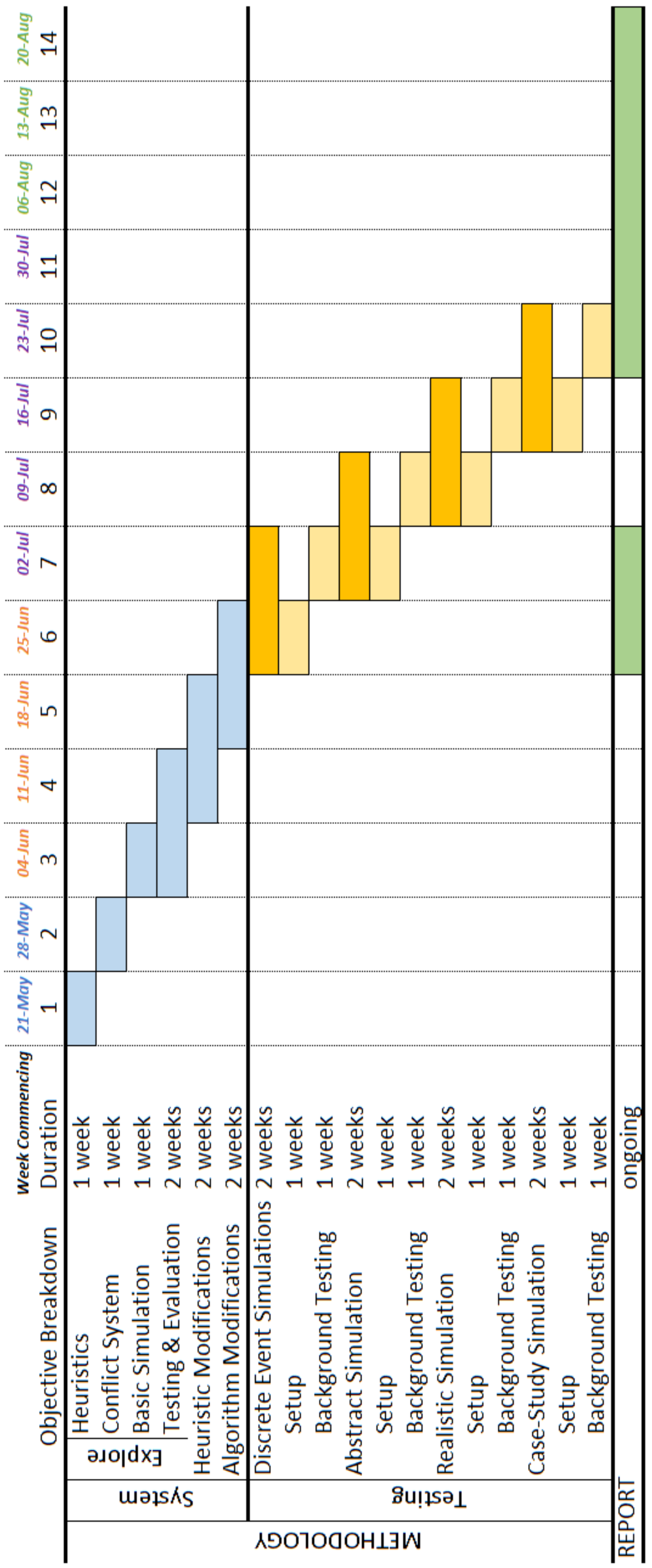


Figure 3.1: Gantt Chart used as a project plan to manage the overview of the project.

making use of it in the project review allowed the simplification and reevaluation of the important aspects of the research.

Throughout the exploration phase, new requirements were added to the project to manage atypical behaviour caused by interactions between agents in unpredictable situations. To manage this, weekly meetings were set up to review the state of the project, and make minor adjustments to the timelines where necessary. These consisted of roughly hour long meetings in which the work completed in the previous week was discussed. This included discussions on such topics as, new issues which were discovered, how issues from the previous week were managed and ideas for modifications and alternative resolutions.

This weekly review was also coupled with a set of weekly objectives to aim to achieve as a minimum for the week, including such tasks as resolving certain bugs and issues, implementing new features, modifying existing features, and reviewing alternatives in the literature. This was managed on a week by week basis to ensure the project was able to adapt to the changing requirements of the implementation, along with managing the issues from underestimates for tasks laid out in the project plan at the start of the project.

As the project was to be completed mostly in solitude throughout the week, project management was a crucial part of maintaining a strong mental health. In projects where one feels overwhelmed by the amount of work, stress can take a large toll on the productivity of the project, causing a cycle of deterioration. An effective method to manage this is to ensure the workload of the project does not overbear the researcher. In practice, an effective method is to make use of a daily journal in which SMART targets are dealt at the start of the week. This ensures the daily responsibilities are managed responsibly and are fully adjustable to the complexities which arise from developing in a project were the requirements are changing quite frequently.

There are many other benefits to making use of a project diary as described by Dr Barbara Markway in her blog post in Psychology Today[1]. The diary was an

[1] `https://www.psychologytoday.com/gb/blog/living-the-questions/201808/6-reasons-why-you-should`

effective tool for logging the records of the project, detailing the rationale behind certain decisions throughout the project. Managing decision making throughout a long project such as this, can often lead to decisions made at the start of the project losing their meaning, thus causing time wasted reevaluating the rationale at a later date. With the use of the project journal, this was not an issue, as the rationale of each decision made was detailed throughout the process. The use of the journal also offered a platform to log ideas and methods of thinking which may have been forgotten otherwise. The use of the project journal offered significant improvements to productivity, time management and mental health.

3.2 Software Development

The project life-cycle was originally designed to be split into two defined sections, the first, referred to as the exploration phase, was aimed at designing the algorithmic framework for the project, alongside the identification of additional framework requirements. This was originally to be followed by a implementation and testing phase in which the developed algorithm and framework would be integrated into increasingly complex scenarios in order to evaluate the robustness of the solution.

The overarching development structure for this two-phase plan was rooted in the theory behind the waterfall methodology, where all of the requirements and design were to be completed first, before the implementation and testing were started. Completing it in this manner meant once the testing phase began, it could continue without any need for further development and work could then be focused on the documentation and write-up.

As with all projects involving a large portion of novel development, issues often arise from uncommon state spaces and predicting these issues is not always possible from the start of the project. This exploration phase was designed to manage this impact in a manner which did not impact the overall process of the project. In a traditional waterfall approach, all of the design would be completed before the project goes into a development phase, however this restricts the ability to adjust to the changing demands of the project. Thus this was not a feasible methodology for the exploration phase as new requirements would continuously arise during the system testing.

To manage the changing requirements, the exploration phase was treated separately from the waterfall phase which over-arched the project. The exploration phase instead, made use of an iterative waterfall methodology in which whenever a new requirement was identified, the project would go through a short design, development, and testing process to integrate the new feature or modification. In order to test the new modifications as part of the iterative waterfall, ad-hoc testing was incorporated, however as the system was rooted without randomness, unlike traditional ad-hoc testing, the testing completed here could be replicated before and after the change by setting the random seed.

3.3 Toolsets and Machine Environments

The aim of this project can be summarised to developing a framework to facilitate the implementation, and evaluation of novel features and methods for multi-robot prioritised planning. For this type of project, the programming language and toolsets chosen are key to the rapid development of the framework. Selecting a toolset which does not hit all of the requirements can result in a lot of needless struggles further down the line.

There are a number of requirements from the project which must be met by the toolsets in order to enable the project to go on without issue. For instance, to ensure the solution is able to scale up efficiently, the platform must enable the profiling of the code-base in order to isolate regions which are slowing the system. The chosen toolset must also have effective debugging tools, as the complexity of the implementation will likely incur many bugs and issues which go unnoticed and are hard to find the cause of. This is especially true with this project as the system is to be designed to work with Monte-Carlo simulation, and this can mean infrequent system states could cause issue. In order to further improve the scalability, having a non resource-intensive language would also be beneficial, as it would enable more testing completed in a shorter time and thus more reliable evaluations. In the event of having to go through the code to identify causes of issues, it is preferable to make use of a toolset which is syntactically clean, having excess code syntax can increase the difficulty in identifying the causes of issues, extra issues could occur with datatype conversions which could delay the project, thus typeless languages with clean workspaces would be preferable. The chosen toolsets should also have effective visualisation tools in order to easily be able to visualise the system and how it is working so as to ensure it is all working okay.

In terms of options, there are a couple which are noteworthy, Python would be the first choice as its ROS integration would be key to the further testing completed later in the project, however it is a very syntax-heavy language, and debugging, profiling and visualisation require the use of additional libraries, and keeping a clean codebase in python is quite difficult. An alternative would be to use MATLAB, while it is quite resource intensive during startup, it offers many tools to assist in development, including debugging tools, profiling tools, toolboxes with thousands of available functions which can be called without any importing of libraries. Both offer the ability to run object-oriented structures suitable for ROS integration, and both offer procedural methods effective for rapid ad-hoc testing, MATLAB is also typeless and has much more built-in customisation options.

MATLAB offers much more functionality than Python, being both a language and a workspace and much more can be done with MATLAB with a lot less effort, enabling rapid implementation and features deployment. There are three main benefits key to MATLAB which require deeper exploration, these features can each help facilitate the development and optimisation of the project in a manner which Python cannot match.

The large benefit to making use of MATLAB is the wide variety of **toolboxes available**, making full use of these toolboxes allows for the use of many small repetitive functions which would otherwise take time to implement. The well integrated nature of the toolboxes also allow for much of the common syntax to be ignored. Not having to include imports and package headings also allows the code to stay clean and simple, which is important in managing the complex nature of the algorithm in development. In Python, it is common place to make mass use of NumPy for managing arrays and matrices as this offers a wide variety of associated functions, however this same functionality is achievable in MATLAB, without the additional overhead and complexity of managing an external library. MATLAB also offers an extensive range of visualisation systems, where in Python, matplotlib.pyplot would be used offering similar functionality, but with much more overhead to achieve the same level of customisation available in MATLAB.

Some of the most ingenious tools in MATLAB however are in the **profiling systems**. Profiling is the act of dynamic program analysis, in which the complexity of a system and its parts are determined with regards to metrics such as memory usage, processing time or function call frequency. MATLAB offers tools specifically for this type of analysis which work by running the main script with a unique command. The script is run as usual, but with the internal calls measured and timed. When the script concludes (either through the system ending abruptly or ending in a controlled manner), the profiling window is opened, which includes a flame graph showing the time spent in each nested function. The window also shows, for each function, a breakdown of the total time spent in the function and the self time which excludes time spent in child functions. For each function called during the run, if clicked on, the window will show the total time spent on each line of code, which allows for easy understanding of where the system can be improved to run faster.

For a project such as this, in which the main aim regards scalability, ensuring slow implementations are avoided is key to evaluating the quality of the algorithm developed. While Python does have libraries available for effective profiling such as rkern's line profiler[2] and pprofile[3], none found do so in a way which is as elegant as how it is integrated into MATLAB, with many requiring a lot of code added within the script or additional setup outside, to monitor the processing information.

The most important requirement for the toolset choice is the **debugging tools**. Based on the descriptions offered by R.E.Pattis in 2002[4], there are a total of five categories of bugs, token errors, syntax errors, syntax constraint errors, execution errors and intent errors. The first four can be generalised together as syntactic errors, in which the error is caused by a single line of code having some clear issue, and in MATLAB all of these are identified dynamically as they are being programmed in, allowing for quick resolution. These come up highlighted in situations where a bracket is missing, or a keyword is spelled incorrectly, and underlined when the system thinks something might be going wrong, such as when a variable shares the name with a class property but has no class identification, or when a calculation may divide by 0. Intent errors can be labelled similarly as systemic errors, where the issue lies in the system design not making an important consideration and these are not so easily identifiable.

For a project in which there is novel development, there is always the risk of having

[2]`https://github.com/rkern/line_profiler`
[3]`https://github.com/vpelletier/pprofile`
[4]`https://www.cs.cmu.edu/%7Epattis/15-1XX/common/handouts/bugs.html`

unexpected issues arise due to intent errors. In these situations, it is beneficial to have a machine environment capable of allowing easy navigation through the running system to backtrack and identify the causes of these issues. The tools offered by MATLAB help greatly in managing this. There are two key tools which are a great help in a project such as this, the call-stack list and the conditional breakpoints.

Conditional breakpoints work by attaching some condition to a break-point, so when a certain criterion is met, the system will pause. In a project such as this, where bugs occur rarely due to certain uncommon state spaces causing issues, breakpoints can be set to only go off at certain time-steps in the system, or set to be called when an error occurs. A command included at the start of the main script "dbstop if error", can set the system to run in a manner which will call a break-point whenever the system errors or crashes. This can be extremely helpful in the identification and analysis of errors which occur late in the program execution. As the program will be designed to run for extended simulation times, if an error occurs after 5 hours of processing time and a conditional break-point is included, when the error occurs, analysis of the properties around the line which crashed, can be performed. This allows the issue to be identified within minutes, rather then trying to piece together the conditions to set a try catch, and then running the system for another 5 hours hoping the conditions are correctly identified. A common alternative recommended is to add many print statements to detail all the information which could be useful to identifying the cause of the issue, and then running the script again, however with many print statements logged to the console, this could slow a system greatly making it take probably closer to 6-7 hours to rerun.

The second tool the **call-stack list**, allows call stack navigation where nested functions can be navigated to identify issues at a higher level to where the issue occurred. This in complex projects is used more than any other debugging tool, as the location where the system crashes is often not the location where the issue is caused, but instead a symptom of an underlying intent error further back in the call-stack. Being able to navigate through the call-stack and query values is a very useful tool. Due to the synchronous method of implementation, there are many issues which could occur within the Agent class but be propagated through to the coordinator before being detected, in these situations, having the ability to navigate back through the call-stack to the Agent class to investigate causes is fundamental in the speedy resolution of the intent errors.

All of these tools help to simplify the process of detecting and resolving issues which are inherent in the algorithm but are rare in the frequency of their occurrence. while it is often the case to only focus on issues which are highly impactful or frequent, in designing a novel algorithm such as in this project, it is important to ensure the algorithm is robust and capable of handling all realistic situations, even those which rarely occur.

All things considered, making use of Python would allow for a more robust approach, allowing for simple and effective integration with existing technologies, especially for the evaluations, however due to the complex nature of the project and subsequent implementations, the rational choice is to use the toolsets which offer the most supportive tools to assist with this development. MATLAB offers an wide variety of tools to help save time and effort in the development of the technology.

3.4 Research Methods

To adequately evaluate the results, a lot of testing would be required to gather data. The implementation was designed with this in mind, specifically with the use of Monte-Carlo simulations in which random samples are used to select the agent targets. As the algorithm itself contains no randomness it can be classed as a deterministic system, thus the use of Monte-Carlo simulations enables large-scale testing as manually selecting targets is not required, and by including no randomness, beyond the target assignment, a random seed can be assigned for repeatability during evaluation. Beyond just the evaluation, this deterministic setup also works to allow for more effective debugging during development, as when issues occur, the code can be rerun with minor changes to test the effectiveness of solutions.

As there was restriction on the available hardware to run all robots simultaneously like it would be in a real-world setting, the decision was made to make use of a synchronous approach, so the hardware was able to handle the workload. This decision forced the project to abandon the use of ROS for the system manager, as ROS is predominantly designed to be an asynchronous communication management system. The original plan was to implement a ROS proof of concept towards the end of the project, however due to project restructuring described in Section 3.1, this was not considered to give enough value to the project to be justify the time required.

Using a synchronous system meant the simulation would be slower as it had to process much more information between each time-step. To counter this extra processing, along with other reasons such as evaluating the scalability, ensuring a wide range of state spaces were explored without issue, and to show more reliable results through larger sample sizes, an emphasis was placed heavily on minimising the performance waste. Minimising performance waste meant more tests could be completed in a shorter amount of time, allowing for more experimentation and in turn more reliable results. To minimise the performance waste, and enhance the quality of the resulting system, the decision was made to make use of Discrete Event Simulation (DES) on top of the Monte-Carlo simulations. With DES, less data is processed, and areas where random noise could come in, are removed, enabling more reliable results since the experimentation would not be hampered by the additional processing for unimportant components of a continuous simulation, and thus the repeatability could be enhanced by performing faster tests.

DES does come with its drawbacks, specifically related to the abstractness of the system. With a case-study based simulation, the simulation details can be very easily accessed and managed, but when that is removed, it becomes much more difficult to find those associations. Part of this project is aimed at looking into the context-dependent heuristics, however with such a lacking of context as in DES, then identifying the metrics to analyse and use to generate the heuristic values becomes much harder. The largest disadvantage however to the use of DES in this context is applicability, while the system may work in an abstract environment, there is no guarantee that it will work at all when context is added back, thus adding more work to the implementation of the system in practice, as a simulation closer to the case-study implementation would have to be developed and rigorously tested, making the mass of testing done in DES somewhat redundant.

To effectively implement the distributed approach while only using a synchronous implementation, the information processing had to be encapsulated within the agent class, to mimic how it would work in a continuous simulation. Thus all of the path planning was completely encapsulated, along with giving the agents their own instantiated copies of the maps, and having an interface class setup to copy the same ROS communication structure as in a pure ROS implementation. The system design, in the end, maintained the structuring of a distributed ROS system, while working in a synchronous manner. The ROSSim class was included for this reason, to help manage the transition from a synchronous approach to an asynchronous approach when the time comes to implement the approach in a more usable form.

Chapter 4

Design and Development

4.1 Requirements

The aim of the project was to develop a multi-robot path planning framework which offered a scalable, and deadlock-free implementation of prioritised planning with a variety of options to enable the user to customise the deployment.

To ensure the implementation was scalable, the algorithm was designed with the intent to be used in a distributed manner, this meant the scalability was not tied to the number of robots within the network, but instead tied only to the size of the map and the communication overhead from the coordinator. To ensure the coordinator communication overhead was not restrictive of the deployment, the goal was taken to reduce the amount of communication between the agents and the coordinator to a minimum. To ensure the motion planning in larger maps was not an issue, the motion planning algorithm chosen to base the project on was A*, an optimal path planning algorithm shown to be effective in large network sizes.

To ensure the implementation was deadlock-free, the decision was made to redesign the motion planning framework to better manage the situations in which potential deadlocks could occur and how to manage them.

To investigate the potential customisability options, this project aimed to investigate four options not previously investigated thoroughly. This being continuous assignment, dynamic scoring, context-dependent heuristics, and combinatorial heuristics.

4.2 Overview

The implementation can be found at: https://github.com/Iranaphor/CRH_Star

The main architecture of the implementation works in three stages, in the first stage, a target node within a topological map is assigned to an agent. This agent will download the latest copy of the topological map, which details the reservations each agent has made for their own routes, then it will conduct internal motion planning using an adapted A* to identify a route is is able to traverse, locking edges which cannot generate a higher CRH score then the reservations are stored with. It will then generate a list of reservations which will be passed to the coordinator.

The second stage is where the coordinator is then tasked with adding the reservations to the global map, then identifying any conflicting agents which require replanning. The coordinator then notifies each of the agents which had conflicts, about their reservations which were rejected.

The third stage is then where the agents with failed reservations perform a replanning, in which they attempt to find alternative routes through three methods of replanning from start, replanning from conflict, and replanning with delay. The shortest route from these three is identified and published to the coordinator.

The second and third stages are repeated until all agents have a defined route.

4.2.1 Communications

The general communication structure works with the use of two main connections between the agents and the coordinator. Both connections have the same message structure, passing a list of Reservation objects each which detail the information the coordinator requires to reserve a specific edge for a defined time period. The first connection, from agent to coordinator, passes information of a newly generated path or a replanned path. The second connection connects the coordinator back to an agent, passing the conflicted reservations details to the owner.

The only other connection within the system is the communication to individual agents to identify a target to navigate to. In here, the target locations are assigned randomly. However, in a future real-world implementation, these targets would be coming from a task allocation system.

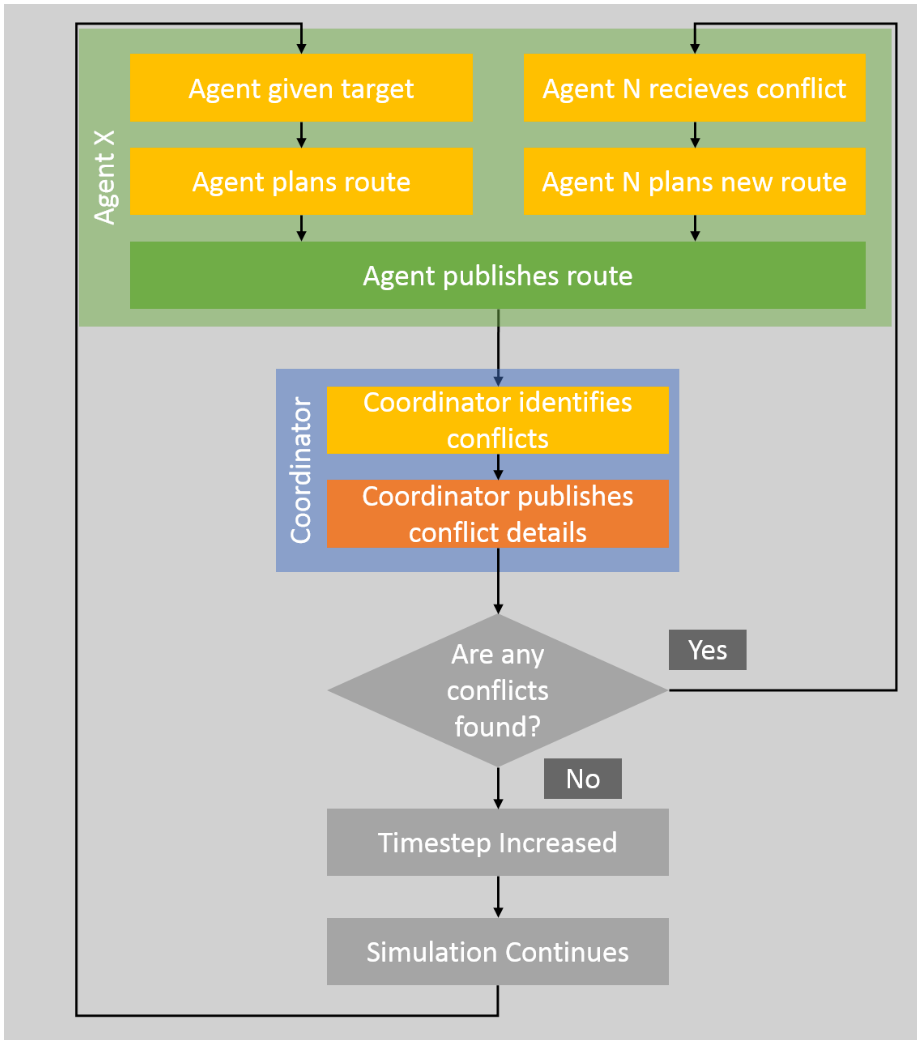


Figure 4.1: Flow chart detailing the communication structure between the coordinator and agents.

4.2.2 Planning

When a target is assigned to an agent, they generate a route through the network to the target, if such a route exists. If there is a route from the start position to the target position, the topology reservation details are then downloaded from the coordinator and the heuristic information is initialised as initial steps.

Once this is completed, the route is planned out making use of the adapted motion planning algorithm A* detailed in 4.3, utilising the downloaded topology reservation information to plan for a feasible path, which the robot can reserve for itself. Once the plan is generated, the route is then published to the coordinator.

4.2.3 Coordinator

When the coordinator receives a reservation list from an agent, it is tasked with processing it into the overlay, along with identifying any conflicts to notify the owners to replan.

This process is completed by first removing all existing reservations within the overlay belonging to the agent publishing the new reservation list. After which, the reservation list is iterated through, and for each reservation, any conflicts are identified and added to a list, and the new reservation is added to the overlay. If the reservation is a static reservation (used only to indicate extended delays), then it is ignored.

Once all reservations are iterated through, a list of the agents which originally published the conflicted reservations is generated.This new list is then iterated through, for each agent the full list of conflicts is compiled and published to it.

4.2.4 Replanning

The replanning structure is fairly straight forward operation within an agent, following a simple flow. Firstly the heuristic information is updated, and the total replans (used for evaluation purposes). Then the list of conflicts is analysed and the conflict which comes earliest in the reservation list (indicated by the position property), is identified. Only the earliest conflict is considered, as replanning from the earliest conflict would impact the remaining conflicts. Once the earliest conflict point is identified, the route is planned again. This is done using three methods of replanning, replanning from start, replanning from conflict and replanning with delay.

Once all three reservation lists have been calculated, the one with the shortest total time to reach the target node is selected and published to the coordinator.

Replanning from Start The first type of replanning works in the same manner as the initial planning, where the existing reservation list of the agent is ignored, and the agent generates a new plan from scratch from the start node.

Replanning from Conflict The second type of replanning used simply generates a new plan from the edge before the conflict point. Then combines this with the original reservation list before the conflict.

Replanning with Delay The third type of replanning calculated makes use of the delay functionality added for the deadlock management, in which the edge where the conflict occurred is given a delay till it is next available. This delay, along with the new path generated form that point, is combined into a new reservation list which is then returned.

4.3 A* Adaptations

4.3.1 A* Overview

A* is a commonly used motion planning algorithm, adapted from Dijkstra. Like Dijkstra, it works by assigning scores to each investigated node, detailing the shortest route along with the length of that route. However it is modified from Dijkstra where the score assigned to each investigated node is calculated as the combination of the optimal route length to that node, along with the Euclidean distance to the target node.

The algorithm, detailed in Listing 4.1 works as follows. When motion planning begins, the start node has its score added and its parent set to null, and is added to the OPEN list. The main loop is then begun in which the node in the OPEN list with the smallest score is chosen, and each of its neighbours is added to the OPEN list, along with their scores being calculated. If a neighbour is identified which is already in the open list, but a score from another direction is smaller then the one on record, then the parent is changed to the new one. Once the target node is reached, it is added to the path list, then repeatedly until the start node is reached, the parent node is investigated, and it is added to the list. By the end, the shortest path is generated with minimal exploration beyond the optimal route.

Listing 4.1: Default A* Algorithm

```
function [reservation_list, overlay] = astar_modified(...
        A, startID, targetID, overlay, start_time, generateScores)
    startNodeExt =  overlay.findNodeExt(startID);
    targetNodeExt = overlay.findNodeExt(targetID);
    [OPEN, CLOSED] = [];
    %Add the start node extension to the open list.
    OPEN = [OPEN, startNodeExt];
    %Define the time the agent will begin moving.
    startNodeExt.arrival_time = start_time;
    startNodeExt.departure_time = start_time;
    %Define end point for path generation
    startNodeExt.parent_id = -1;
    while True
        %Find smallest edge.
        currentNodeExt = A.findMin(OPEN);
        %Move currentNode from OPEN to CLOSED
        OPEN(OPEN==currentNodeExt)=[];
        CLOSED = [CLOSED, currentNodeExt.id];
        %If current point is at target, break.
        if currentNodeExt.id == targetNodeExt.id
            reservation_list = ...
                A.backprop_path_modified(targetNodeExt, overlay);
            return
        end
        %Determine costs for each neighbour to the current node.
        neighbour_list = overlay.findNeighbours(currentNodeExt);
        for neighbour_entry= neighbour_list
            neighbourNExt = neighbour_entry{1};
            EExt=overlay.findEdgeExt(currentNodeExt.id, ...
                neighbourNExt.id);
            %Calculate distance travelled and distance to target.
            g_cost = currentNodeExt.g_cost + EExt.edge.weight;
```

```
31              h_cost = A.pythag(neighbourNExt.node, ...
                    targetNodeExt.node);
32              f_cost = g_cost + h_cost;
33              %If new path is better, update cost and parent.
34              if any([f_cost<neighbourNExt.f_cost, ¬...
                    any(neighbourNExt==OPEN)])
35                  neighbourNExt.UpdateCosts(f_cost, g_cost, h_cost);
36                  neighbourNExt.UpdateParent(currentNodeExt.id);
37                  %Include node in search for smaller routes.
38                  if ¬any(neighbourNExt == OPEN)
39                      OPEN = [OPEN, neighbourNExt];
40                  end
41              end
42          end
43      end
44  end
```

4.3.2 Time Management

As the algorithm required the ability to reserve edges within specific time periods, the algorithm was amended slightly, so at the end of the algorithm when the costs are added to the neighbour, the reservation period is also added.

4.3.3 CRH system

To implement the dynamic prioritisation, it is required to generate scores dynamically through the network during the motion planning algorithm. These scores, referred to as Conflict Resolution Heuristic (CRH) scores, are used to dynamically define priorities for each edge investigated within the motion planning algorithm.

CRH scores are generated within an additional section to the main loop, just before the f_costs are generated in the original algorithm for A* on line 29 of Listing 4.1. In this section, shown in Listing 4.2, the edge under investigation is analysed to determine if a reservation exists within the time period requested. If a reservation does exist, then the locally generated CRH score is compared to the CRH score on record. The reservation is won if either the local CRH is greater then the stored CRH, or if there is no reservation within the time requested. If the reservation is lost, then the edge is marked as unavailable and the edge is added to the FAILED list, otherwise, the motion planning algorithm continues on as normal.

The CRH score is calculated locally by taking the combination of a defined subset of possible heuristics defined within the configuration file in either a static of dynamic collection. The static heuristics are calculated before the motion planning algorithm is called as they apply throughout the entire run. They are generated before the motion planning function is started, so as not to be calculated repeatedly within it wasting resources.

The combination of static or dynamic context-independent heuristics is then combined with the context-dependent heuristic before being returned for the conflict processing.

Listing 4.2: The method used to calculate the conflict resolution heuristic score.

```
function [success, CRH_Score] = CRH_BATTLE(A, currentNExt, ...
    neighbourNExt, targetNExt, EExt)
    %Return 1 if CRH edge is up for grabs (empty or CRH can win)
    %Return 0 if edge is unavailable (taken and CRH cant win)
    %Calculate time for use of edge.
    from_time = currentNExt.arrival_time;
    time_till = currentNExt.arrival_time+EExt.time_weight;
    %Check if edge is free at time required.
    conflicts_list = EExt.IsEmpty(from_time, time_till);
    %Calculate the CRH score to battle with.
    CRH_Score = A.getCRH(currentNExt, neighbourNExt, targetNExt, ...
        EExt);
    %Succeed if edge has no conflicts
    if isempty(conflicts_list)
        success = 1;
        return;
    end
    %Extract the local CRH score
    conflicting_CRH = 0;
    for R = conflicts_list
        conflicting_CRH = conflicting_CRH + R.CRH;
    end
    %Fail if conflicting CRH has reached maximum allowed.
    %if conflicting_CRH > A.maximumCRH
    %    success = 0;
    %    return;
    %end
    %This can only be used if the heuristics are normalised.
    %If local CRH score is smaller, return failure.
    success = (CRH_Score > conflicting_CRH);
end
```

4.3.4 Deadlock Management

Due to how paths can become unavailable due to reservations being unbeatable due to the inclusion of the CRH system, there exists the potential for an agent to become deadlocked, i.e. unable to find a path to its target that it can reserve.

In order to manage this and ensure all agents are able to find a route to their target, an additional system was included, referred to as the FAILED list, it works similar to the OPEN list. The OPEN list works by saving each edge which is unchecked but connected to the explored workspace, and with each iteration, selecting the edge with the most promise to take the agent to the target, to add to the explored workspace.

The FAILED list works in a similar way, but edges are added to the list only when they are not able to be traversed in which the agent is unable to use the edge due to failing the CRH battle. The point when the agent is deadlocked, is when the OPEN list has no more edges which can be investigated, at this point shown in Listing 4.3, the FAILED list is accessed, one edge with the best score is identified. The time the edge is next available is identified, and the delay to that time is logged on the node extension. The chosen edge is then removed from the FAILED list and added to the OPEN list, and the loop is continued.

By managing deadlocks in this manner, the motion planning algorithm is still able to retain the fundamental components proven to be successful in identifying optimal routes, while the deadlocks are quickly and quietly managed without issue. This method does incur delay each time a deadlock is managed, however this is preferable to the result of an agent not having a route, or being potentially delayed even more with alternative delays elsewhere.

Listing 4.3: The adaptation included within the motion planning algorithm for resolving deadlocks.

```
function [OPEN, FAILED] = findMinFailed(A, FAILED, CLOSED, OPEN, ...
    ¬, overlay)
    %Remove all nodes from FAILED if they have been closed ...
        elsewhere.
    FAILED(:,any(FAILED(1,:)==CLOSED'))=[];
    %Identify all FAILED edges with smallest f_cost into parent.
    minimal = FAILED(:,FAILED(3,:)==min(FAILED(3,:)));
    %Define target node
    T = overlay.findNodeExt(A.targetID);
    %Identify smallest edge with best f_cost into neighbour.
    %(for when multiple in FAILED have smallest f_cost)
    smallest = [0,0,0,0,Inf];
    for m = minimal %[neighbour_id; current_id; current_f_cost]
        %Identify nodes
        N = overlay.findNodeExt(m(1)); %neighbour Node
        P = overlay.findNodeExt(m(2)); %parent Node
        %Calculate costs
        EExt = overlay.findEdgeExt(P.id, N.id);
        g_cost = P.g_cost + EExt.edge.weight;
        h_cost = A.pythag(N.node, T.node);
        f_cost = g_cost + h_cost;
        %Idenfity smallest f_cost
        if f_cost < smallest(5) %Swap this out to just append, ...
            and find smallest later on
            smallest=[N.id, P.id, g_cost, h_cost, f_cost];
        end
    end
    %Identify nodes
    N = overlay.findNodeExt(smallest(1));
    P = overlay.findNodeExt(smallest(2));
    %Identify delay time and new arrival time
    EExt = overlay.findEdgeExt(P.id, N.id);
    PArrival = P.arrival_time;
    NArrival = PArrival + EExt.time_weight; %this must be calculated
    NewPDeparture = EExt.FindAvailablePeriod(PArrival, NArrival);
    PDepartureDelay = NewPDeparture - PArrival;
    %Update deadlock count
    EExt.deadlock_overcome = true;
    %Save new meta info
    N.UpdateCosts(smallest(3), smallest(4), smallest(5));
    P.departure_time = NewPDeparture;
    P.set_departure_times(N.id, NewPDeparture);
    N.UpdateTimes(P, EExt);
    %Append neighbour to OPEN list
    OPEN = [OPEN, N];
    %Remove neighbour from FAILED list
    FAILED(:,all(FAILED(1:2,:)==smallest(1:2)')) = [];
end
```

4.3.5 Further Modifications

Alongside the major additions and reworks to A*, completed to extend the motion planning algorithm to work with reservation overlays and path replanning, some minor changes were also made, to improve the computational efficiency. An early exit was also implemented before the CRH battle, where if a neighbouring node was already in the CLOSED list, then it wouldn't be skipped.

4.4 Reservation Management

4.4.1 Overlay & Extension Classes

In order to manage the reservations and extended capabilities of the nodes and edges, while still preserving the abstract structure of the nodes and edges for connection with external systems, the Extension classes and the Overlay class were created as wrappers for the Network, Nodes and Edges.

The Extension classes were initially designed to work only as wrappers for the transition to a ROS-based solution after the exploration phase was concluded so the system could connect with the topological_navigation ROS package developed by Fentanes et al., 2015 without issue, however after the project was restructured, they became effective for more than just that, also taking on the role as containers for complex interactions within the algorithm, the edges, and the nodes. The implementation was designed to be as simple as possible in order to improve clarity within the complex functions, so there was a large focus in the development to move as much code to external functions where they could be handled more effectively, such like building a toolbox. This resulted in many functions designed to identify information about the network, edges and nodes, being extracted to the methods of the extension classes.

4.5 Framework Facilities

Configuration File To ensure the ease of customisation for using and modifying the system, different configurable parameters and their values for the test cases are collated into a configuration file.

This configuration file is broken down into two parts, the first part defines meta priorities about the simulation itself, such as agent counts, map selection, etc, the second part defines the details of the simulation style, allowing for selection of dynamic scoring, batch processing, independent heuristic, etc. Key fields for these two parts are described in the following subsections.

4.5.1 Simulation Facilities

Simulation Structure To manage the simulation structure, the implementation has been set up to make use of a configuration file which is loaded at the time the simulation is started.

After that, the pointset is loaded in, and the network is built. Then the agents are each loaded into the simulation depending on their task type. It is at this point that the agents are each given an initial navigation target randomly generated.

Finally, the main cycle is begun, in which for a total number of targets defined from the configuration object, the timestep is moved forward to the next event, be it the next agent completing its task or the final agent completing its task depending on whether batch or continuous assignment is used, after which the next target is assigned to the selected agent.

Once all cycles have been completed, the evaluation metrics recording the number of replans, the delay, the agent id, the number of deadlocks, and the total time the agent took, are all returned from the function.

Map Selection The topological map of the environment can be selected from three maps that were extensively used for the simulation. Other maps can also be added with corresponding map definitions.

The first map is the **fork map**, seen in Figure 4.3, this is a small map designed for rapid testing, it consists of 53 edges, taking the shape of a single poly tunnel with six rows connected at the ends. Its size is key for analysing the effectiveness of algorithms in a small enclosed environment without having to run too many simulation trials to fully explore the conceivable state space.

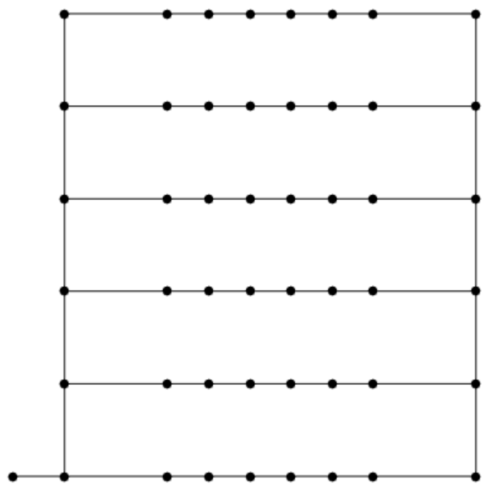


Figure 4.2: Network diagram for fork_map.

The second map is based on the research and experimentation centre in **Riseholme**, seen in Figure 4.3, this map is made up of 150 edges and is structured to make use of two polytunnels with the rows within the tunnels connected at both ends but with the two tunnels only connected from one side. The map has three bays for the robots charging stations on the left edge of the map, next to a common route that leads round the corner from the top of the map to the storage room. This is the map used for the evaluations, along with the fork map for evaluating scalability. The maps beyond this were too resource intensive for running simulations on the available hardware.

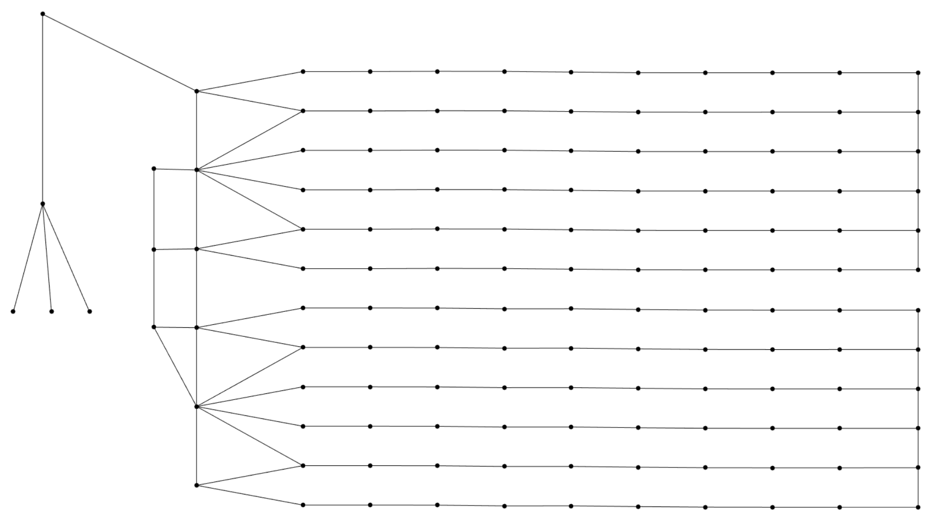


Figure 4.3: Network diagram for strawberry polytunnels at Riseholme campus.

The third map **chf__south__large**, is from an actual strawberry farm with 20 polytunnels with lengths ranging from 100m to 130 m, with a total of 2502 edges. While the simulation was run for small amounts on the fourth and fifth maps to ensure it worked in larger scenarios, there was no significant testing as the processing was too slow being a synchronous approach do to hardware restrictions.

Total Targets The meta section of the configuration file includes a field "total__targets". As the simulation runs, randomly allocated targets are generated for each agents when required. This field is used to define how many targets will be generated before the simulation ends.

Total Agents When the simulation begins, the meta field "total_agents" is used to define how many agents to instantiate. For the task-specific agents, the fields "total_logistics", "total_crop_monitoring", and "total_row_monitoring" can be used. Each of these fields define the total number of the task-specific agents to instantiate.

Agent Task Definition The three tasks defined within the system mimic three different types of tasks defined in the context of autonomous robot assisted berry production in polytunnels.

In this context, the main task is to assist the berry pickers who are within the polytunnels picking the berries. The task this robot has, is to navigate throughout the network to the pickers when requested, to have the picker load the crate of picked berries into the robot, then when full, to travel to the fixed collation points to deposit the berries. The "total_logistics" agents instantiated are tasked with this. In their target generation, they are given a node to travel to within the polytunnels, once reached, their internal load counter is increased, before being assigned another target. This is repeated until the agents have a load value of 5 (selected as an arbitrary number), at which point they will navigate to a node marked as a storage node, where their load is dropped to 0. Then the structure of the target assignment repeats.

While logistics is the main task of the robots in the context, there are also additional tasks. The robots also have tasks related to plant care, specifically plant monitoring. In these tasks, the robots are required to navigate across entire rows, monitoring the plants for bugs, watering them, shining UV lights and much more. The "total_row_monitoring" field within the meta section of the configuration file is used to define how many of these types of agents are included within the simulation. These agents select their targets differently to the logistic agents, by first selecting a random row, then navigating to the start node of that row. Once at the end, they select the far end of the row to navigate to. This target assignment is then repeated with randomly selected rows.

The final type of agent implemented into the system is the "total_crop_monitoring" agents. These agents mimic the functionality of a third type of robot in the context.

This third type of robot is tasked with moving around the network with a human agronomist, to perform monitoring on specified crop areas. As in the context, the agronomist would specify the edges of interest to the robot, and would often only take of interest, a few scattered edges at a time. Thus this agent within the simulation, randomly selects single edges within the network to navigate across, with the first target selected as one end of the edge, and the second target selected as the other end, before selecting a new edge to repeat.

4.5.2 Customisation Facilities

Continuous Assignment In traditional priority planning, target assignment structures work only once all agents are idle, where they will each be assigned a score and will each plan their route in an order based on that score. Once the first plans their route, they are treated as a dynamic object within the terrain for other agents to plan around. Only once all agents have reached their targets, are new targets set, as agents have defined scores which restrict when they are allowed to plan. This method of target assignment is fundamentally flawed with its ideas, as it causes many agents to sit waiting idle for others to complete their tasks before they can move on to the next target.

The counter to this method is referred to here as continuous assignment, in which agents are assigned new targets as they complete their existing ones. This works by making use of dynamic reservation assignment and replanning schema which allow the agents to replan should an agent with a higher score request edge access.

Both assignment types are included within the facilities offered by the framework as a toggle option within the configuration file under the field, "use_continuous_assignment".

Dynamic Scoring The traditional method of assigning score to agents is done by making use of a static score, in which a single score is assigned before the path is generated, and that score is used to make the required reservations.

This framework offers this ability, along with a novel approach called dynamic scoring, in which scores are adjusted based on relevant local information at each edge.

Take for instance, the Euclidean distance heuristic, as the agent moves towards its goal, the heuristic in a static scoring system would be the same for every edge, so at a single snapshot, the score for an edge would be based on historical data. However with dynamic scoring, any snapshot would be as relevant as the initial step, as the scoring would take into account the information at the time of the reservation. So the Euclidean distance heuristic would cause each subsequent reservation closer to the target, to have a smaller score in a dynamic scoring system.

This approach to prioritised planning is also customisable, with both being available through a toggle option within the configuration file "use_dynamic_scoring".

Context-Independent Heuristics In the system, the option is available to select a wide range of options for the independent heuristic used for the scoring of the CRH. It is worth noting that both static and dynamic CRH scoring can extended with new methods for additional heuristics. Below is a brief summary of how each implemented method works and the inspiration or intuition behind them.

Euclidean Distance: The static variant takes the direct distance of the agent's start position to its target, and the dynamic variant takes the direct distance from the each edge in the reservation to the target. This rewards the highest priority to the agent which is furthest from its goal. This is derived from the works by Van Den Berg and Overmars, 2005.

Optimal Path Length: This makes use of the same idea as Euclidean Distance, but in the context of polytunnels, if an agent was to move from the centre of one tunnel, to the same location on the next, it would have to move a distance nearly equal to the length of the tunnel, while Euclidean Distance would be really small. Optimal path length takes this world information into account, and awards a higher score to the robot which has the longest optimal path to its target. Like Euclidean distance, the static variant takes the optimal path from the start point, while the dynamic variant takes it from each node in the reservation. This is an adapted form of the works by Van Den Berg and Overmars, 2005 which mainly focused on Euclidean Distance working with metric maps.

Planning Time: This prioritisation schema which is derived from the paper by Velagapudi, Sycara and Scerri, 2010. It works by evaluating the processing time it takes to generate the route as its score, with the intuition that agents which take the longest time to plan, require the least cluttered work-space to find an optimal route thus the agent with the longest planning time gets the highest score. The static variant, like with the two distance-based prioritisation schemes takes planning time generating a path from the start to the target, whereas the dynamic variant works by taking the planning time from each node on the route to the target. Something to note with this schema; performance time is often variable in its consistency so repeatability may yield slightly different final results.

Agent ID: This schema makes use of the agent's unique id number, generated as the order the agents were added to the system, with the first of 10 agents given a score of 10, and each subsequent agent given a score 1 below. So the first agent added to the system gets priority over all others, and the final agent added to the system has no influence over the other agents. This is how the original conception of prioritised planing came with, by Erdmann and Lozano-Pérez, 1987.

Random: The static variant of this schema makes use of a single random number generated at when the agent is assigned a target to plan for. The dynamic variant generates a new random number each time an edge is attempted to be taken. This heuristic is designed to be the worst case scenario where randomly agents will be unable to take the routes they require.

No Replanning: This schema works by assigning every CRH score to the same value regardless of the agent, path, or surroundings. The static and dynamic variants are identical in their results. This essentially becomes first come first served to the edges. Once an edge is taken, it cannot be taken by any other agent.

While further heuristics were explored within the literature review, approaches such as Path Prospects and Naive Surroundings were not included due to time constraints and their complexity.

Context-Dependent Heuristics A context-dependent heuristic, is a measurement which can be used in a given scenario which may not necessarily work in all scenarios. This is in contrast to a context-independent heuristic which can be generalised across all scenarios.

For the context of the strawberry polytunnels, a context-dependent heuristic can be task importance as the differing tasks can have differing importance levels. For the context of a shop like IKEA, the context-dependent heuristic could be something like route preference, designed to keep customers from being inconvenienced. Each context or scenarios may have a single, or multiple, context-dependent heuristics.

Within the configuration file, the field "dependent_heuristic" identifies any selected context-dependent heuristics, that is, any heuristics which are designed for the specific scenario of berry-picking in polytunnels for this specific context. As this is only a rudimentary simulation working in a highly abstract environment to process Monte-Carlo simulations in a discrete event simulation environment, there were not many possibilities for context-dependent heuristics. As such, task-importance was the only heuristic included which fits into this category.

It works by assigning a static score to agents based on the task they are completing at the time of target assignment. This, for this scenario offers three potential scores, a score of 3 (the highest available) for an agent running logistics tasks, a score of 2 for agents running the single crop monitoring task, and the lowest score of 1, for any agents completing row maintenance tasks.

These scores were decided based on the requirements and inconveniences from the humans around the robots attempting to complete their own tasks. Where the logistics agents had both the responsibility of rapid delivery of fruit to the fixed collation points, along with the inconveniences for pickers waiting for the robot to collect the fruit, the crop monitoring agents has the inconveniences for the supervisor present. However as the row monitoring agents have no urgency with their task, or potential inconveniences for humans around them, they are awarded the lowest of scores.

4.6 Assumptions Made

In the development of this project, certain assumptions were made in order to simplify the development process as time restrictions applied. This section details the specific assumptions which were made, the rationale behind their exclusion, and how their absence may impact the efficacy of the solution. In the future, the path planning framework can be extended with these assumptions relaxed, and thereby increasing the complexity of the problem.

4.6.1 Collision Box Absence

In a more complex simulation, collision boxes would be factored in to mimic the real world more closely. However as the simulation developed was a discrete event simulation, collision was not considered. The aim was to define assignments, with the assumption that such assignments would prevent the collision from being required. In part, this allowed for speedier simulations as there was no physics engine taking processing resources, however it also meant if there were agents bumping into one another, it would go undetected.

This absence does not make much of an impact on the efficacy of the solution, only in terms of the scalability, in which the scalability may be increased with respect to a more complex simulation as less processing is required, however in the real world, physics simulation would not be included anyway so it does not have much impact through this.

4.6.2 Node and Edge Capacity

In keeping the simulation simple, yet still trying to keep some semblance of real-world requirements, the decision was made to give the nodes an infinite capacity, and the edges a capacity of 1. Nodes were given an infinite capacity with the assumption that nodes could have the space around them to allow robots to pass one another such as in junctions, while edges were given capacity of only 1, to restrict the movement of agents so they would show their ability to resolve conflicts, without having to add hundreds of agents into small simulation maps.

This design choice does restrict the ability of the solution, in that it does not recognise a crucial element of the situations it is planning for. By removing this as a consideration designed into the solution, the reliability of the solution for actual route processing is questionable.

4.6.3 Agent Movement Speed

To keep the simplicity required for discrete event simulation, all agents were assigned the same speed of 1 meter per second. While agents of the same type would in actuality have the same average speed, agents performing different tasks such as plant monitoring and logistics would potentially have differing speeds.
Having agents all move at the same speed restricts the amount of state spaces able to be explored, with such instances like overtaking not occurring at all. This restricts the accuracy of the solution for real-world use.

4.6.4 Random Noise

A key requirement to the effective use of Monte-Carlo simulations is that the simulation must be deterministic in its design, with any randomness being in controlled settings. This not only impacts the simulation itself, but also the reputability of the results. The simulation as it is implemented, allows for a single random seed to be passed into the configuration file, for the simulation to be repeated exactly as before. Introducing random noise to this through adding noise to the movement speed or localisation as it would be with a more realistic simulation, would distort the repeatability of the experiments.
This, like with the movement speed, also restricts the reliability of the solution for real-world use, as it further takes the simulation into unrealistic processing environments.

4.6.5 Battery Charging

The actual robots the project is based on, make use of a charging station to recharge their batteries after around 4km on a single battery or 8km on a double battery.

So the number of tasks they can perform is limited. This was ignored within the simulation with the decision made to give the agents an unlimited battery duration. This was chosen to focus more on the algorithm itself, as opposed to the limitations within the real world. As robotics would develop further, these restrictions would not apply as much as they do now. The exclusion of this feature does not pose too much impact on the results of the experimentation, as the battery on any other robot could differ anyway.

Chapter 5

Experiments and Evaluation

5.1 Experiment

The project aimed to develop a framework which offered a scalable, and deadlock-free implementation of prioritised planning which offered a variety of customisable options. The developed framework would then have to be tested for evaluating their efficacy. The evaluation process consisted of six sets of evaluations, the first four looked at the variation in performance of the framework with continuous assignment, dynamic scoring, context-dependence and combinatorial heuristics. The final two evaluations investigated the effectiveness of the framework for managing deadlocks and scalability.

5.1.1 Base vs Continuous Assignment

In commonplace batch assignment, the assignment of targets will only be done at a point when all agents are idle. Although this can improve the routing optimality, most often many robots would end up idle and will have to wait for all other robots to finish their tasks, rather then navigating to their next target. Some algorithms adapt to this by forcing the moving robots to consider the idle robot as obstacles and move around them. However, there has been little effort to utilise the robots' ability to replan or adapt their routes for the new system states.

The proposed approach of continuous assignment addresses these delays, by redesigning the method of target assignment so that agents can move to their next target as soon as they arrive at and complete their current task. This results in the structure of assignments in Figure 5.1(lower), compared to the batch assignment methods shown in Figure 5.1(upper).

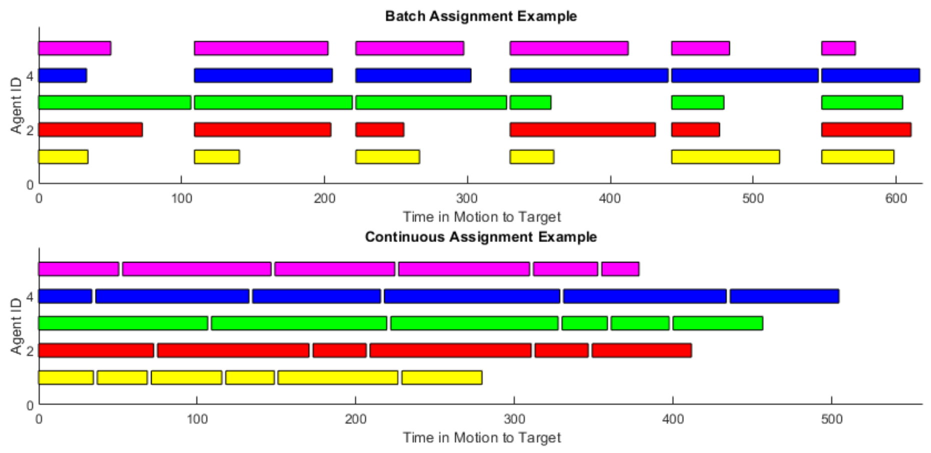


Figure 5.1: Graph showing how removing idle space between batches of routes can decrease final time to complete a set of tasks.

The result of this is robots would spend less time idle, and completing navigation to a total number of targets would be overall quicker. In order to measure the effectiveness of continuous assignment, the time to complete a defined number of paths can be compared across a system running batch and continuous assignment.

The evaluation consists of six tests. Each one consisting of 10 runs with 20 targets assigned to the agents, with five runs using batch assignment, and the remaining five using continuous assignment, the first two tests were run with 5 agents in the map, the second with 10 agents and the third with 20. For each run, the time at which the final target was reached was extracted and these were plotted into the graphs in Figure 5.2.

The expected result if the test was successful, would be that the continuous box-plots in the three subplots would be lower then the respective batch subplots. This, as seen in the graphs, is clearly evident, with all three continuous plots lower. While there is some overlapping the plots with fewer agents, as the number of agents increases, this difference all but disappears.

To ensure the feature is actually a success, a further test was performed in which the total number of targets was increased to 1000 for each run. The results of this

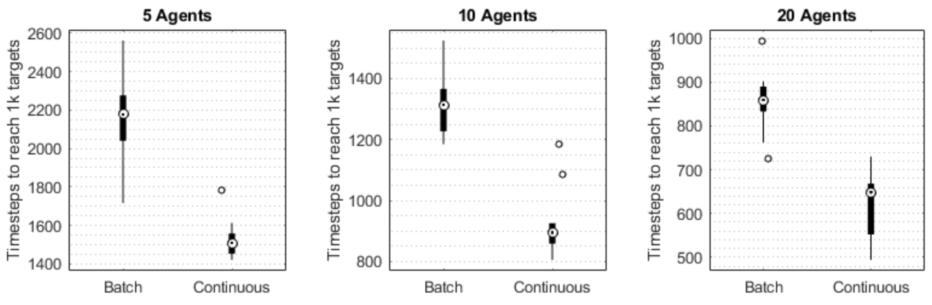


Figure 5.2: Boxplots showing distribution of final times with the use of batch and continuous assignment. Run with three quantities of agents in the network with 20 targets per agent.

can be seen in Figure 5.3, in which it is clearly evident that over a longer period of run time, the benefit of using continuous assignment becomes more obvious, leading to a 20% time reduction with 5 agents, a 38% time reduction with 10 agents, and a 47% time reduction with 20 agents.

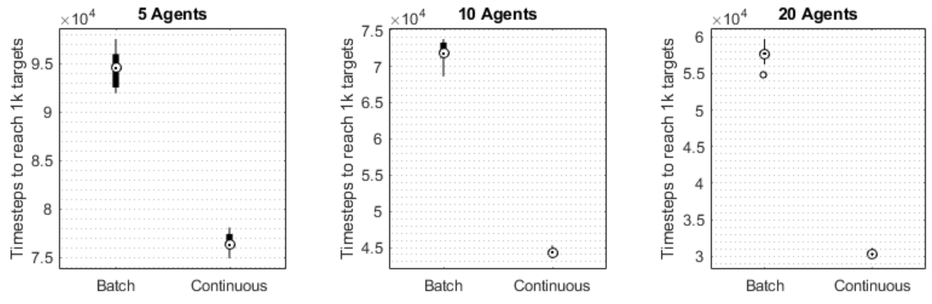


Figure 5.3: Boxplots showing distribution of final times with the use of batch and continuous assignment. Run with three quantities of agents in the network with 1000 targets per agent.

5.1.2 Base vs Dynamic Scoring

With traditional prioritised planning, normally the agents will each be given a single static score using some heuristic. For every reservation made, this score will be used, unchanged until the next point when all agents are idle and new times can be assigned. This results in agents taking reservations with scores which are based on historical data. The intuition behind this feature is that it would offer the system more relevant information about the agent and its location at any single snapshot.

Take for instance, using the optimal route length as a scoring metric, if the optimal route an agent could travel is to take four 5m edges through the graph to get to its target. Then in a static scoring approach, every edge would be assigned a score of 20 (given 1m means one point). As the agent moves through the system and across the first three edges to now be 5m away from its target. If a second agent would begin planning from that same point to go 5m beyond the first agents target, it would get a score of 10. In this snapshot of the scenario, the agent furthest from its goal would not get the priority, the heuristic is designed to take. With a dynamic scoring approach, the agent furthest from its goal would always get the priority as the scoring at each edge would be dependent on the distance to the target. Where for that first agent, the score used to reserve the first edge would be 20, then the second edge would be 15, then 10, then 5. So at the point of conflict, when the first agent has 5m left, the second agent would take priority having 10m remaining on its path.

In conventional prioritisation evaluations, the priorities are often assigned to minimise total combined distance or total battery expenditure of all agents as in Ma et al., 2019. This ability for a system to find the optimal assignments which minimise this metric is referred to as completeness, which is a large concern with prioritised planning systems.

To calculate the optimal complete plan, would take a tremendous amount of computational resources. Even with only 10 agents, the total number of priority configurations is calculated as 10! which equates to 3,628,800 total possible configurations and this is without factoring in the changing prioritisation schema in the dynamic implementation. In the dynamic setup, this value would be multiplied by the total number of edge battles in each path generated. Making it completely unfeasible for an evaluation metric.

There are methods designed to reduce this total number; the 2019 paper by Ma et al., 2019 worked by evaluating all possible priority configurations, swapping and replanning only when conflicts were caused, and others have tried similar methods; the 2019 paper on priority tuning Hu and Cao, 2019 worked in a similar manner, where priorities were assigned, then swapped to converge to an optimal assignment;

these have shown to reduce the complexity in static prioritisation schema. However in dynamic prioritisation schema, this is still not enough to make the processing time viable.

As comparing to the optimal complete path is not viable, the next best thing is to compare against the unrealistic best case (referred to as the delay), which is the score given by calculating the optimal route through the network for each individual agent, then comparing this with the length of the routes the agents actually took. As this is an unrealistic estimate, its values should be taken with a grain of salt, the values returned from this will always be large, however it is effective in comparing the quality of the dynamic scoring as an augmentation to static scoring.

In order to run this evaluation, 12 tests were completed, each test was run 5 times, the mean delay for each run was saved, where the delay is calculated for each target as the difference between the optimal time and the actual time to complete the route. The tests were divided into two groups, where the first six tested the results with 50 targets per run, and the second six used 100 targets per run. The tests were further divided with one half made use of static scoring and the other half with dynamic scoring, and were divided further again so three different heuristics were used for the assignments, making use of Euclidean distance, optimal route length, and planning time. For more information on how these work, see the Heuristics section in Design and Development.

The results obtained from this test can be seen in Figure 5.4, which shows in the two subplots, the distribution in average delay across the two groups. As is evident, every dynamic scoring test resulted in significantly smaller average delays by a quite significant amount, showing the effectiveness of dynamic scoring for improving the quality of the assignment.

5.1.3 Base vs Context Dependence

Most research focuses on abstract concepts which can be applied to any situation, rather then specific and minute methods to improve selected domains. This section of the evaluation aimed to introduce new metrics which are context specific. In selecting

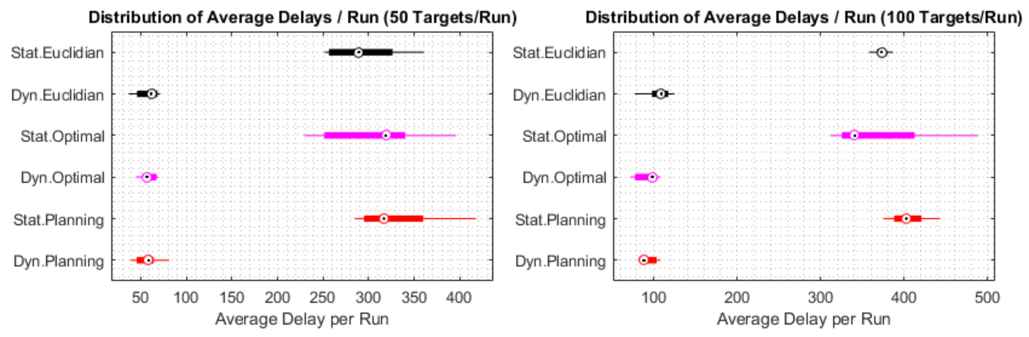


Figure 5.4: Box plots showing the distribution of average delays with experiments run with three heuristic variants with both dynamic and static scoring systems. Run with 50 targets per run (left) and 100 targets per run (right).

a heuristic, abstract heuristics make use of information which is guaranteed to always be present, making them independent from the context it is implemented around. This evaluation looks into the impact of building the heuristic from information directly related to the context as a method to augment the efficacy of the assignment.

To evaluate this effectively, three test groups were setup. Each group consisted of two tests, the first made use of only the context-independent heuristic of Euclidean Distance which is known to have flaws in the given scenario of polytunnels, while the second makes use of both the context-independent Euclidean Distance, alongside the context-dependent heuristic of task-importance. All test groups worked with 20 runs, with he first group having 50 targets per run, the second having 100, and the third using 200.

The agents have three possible tasks; the first is logistics where the robot must travel to five different crop nodes in the map where at each one the load is incremented before the robot ultimately travels to the storage node to reset the load before continuing again; the second task is row monitoring, where the robot will pick the start node of a random crop row in the map, then once reached it will traverse to the other end of the row, select a new row and repeat; the final task is crop monitoring, where a random edge in a random crop row is chosen to navigate to and traverse. The logistics, as in a real setting, is given the highest of scores as its completion must come first; the crop monitoring is given the second highest score as this task will often be done with a supervisor present whose time must not be wasted; and finally row monitoring is given the lowest score as it is a background process and does not have the same kind of time constraints as the other tasks. In these simulations, four agents are tasked with logistics, three are row monitoring and three are crop monitoring.

As with the evaluation for dynamic scoring, this evaluation is also tasked with measuring the priority assignments, thus it follows the same logic in making use of the delay for evaluating the effectiveness of the augmentation.

The results for this test can be seen in Figure 5.5. As evident in the results, there is very little difference between the categories, and further reruns of the test showed similar results, with relatively no difference between the delay including and excluding the context-dependent heuristic.

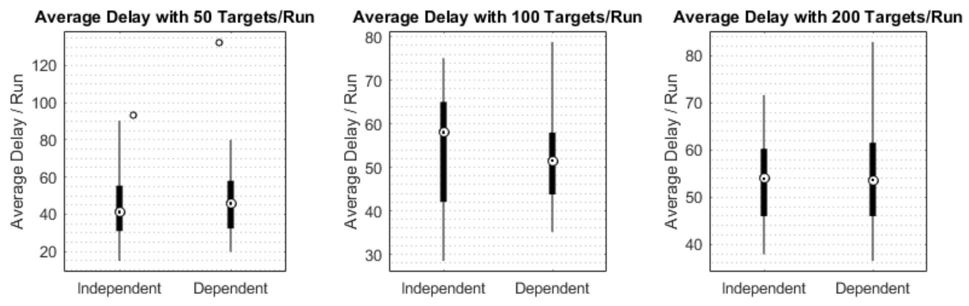


Figure 5.5: Box plots showing the distribution of average delays across three experiments comparing the effectiveness of context-dependent heuristics. Run with 50 targets per run (left), 100 targets per run (centre), and 200 targets per run (right).

These results show that the context-dependent heuristic chosen does not improve the efficacy of the assignments by any means, however this does not disprove the potential that results could amount from it. The Monte-Carlo simulations required to complete the implementation limits the available information with regards to the context to be randomly generated, thus any attempt to artificially replace this information would have issues with consistency with the real world. So while this specific experiment does not disprove the null hypothesis, further experimentation in future works where the simulation is closer to reality could show improvement.

5.1.4 Base vs Combinatorial Heuristics

In conventional prioritised planning implementations, it is commonplace to make use of a single heuristic to assign priorities. While this is acceptable and works well for the most part, this section of the evaluation looks to the potential gain from combining multiple heuristics together in order to get a more effective assignment.

The intuition behind this comes from stacked bar charts, where if one field ranks a record highly, while another field ranks the same record low, then combined it would be just average, while the second highest score from the first field could be just above average on the second field, resulting in it being the highest overall. In stacking heuristic scores, it would emphasise the consistently strong scores, and understate the consistently weak scores, which would lead to a strong importance on heuristics agreeing with one another, while disagreement would result in mediocre scores. Applying this to the prioritisation heuristics, it would allow for heuristics to be weaker independently while still resulting in a more effective priority schema.

For an example, see Figure 5.6 in which the graph shows how combining scores can change the assignment structure. In this graph, it is clear that Heuristic 1 would prefer to award the highest priority to Agent 6, as it has the largest blue bar. However Heuristic 2 has the opposing opinion about Agent 6, where it is awarded the lowest score. Due to the differing opinion regarding the agent, the score awarded is decreased compared to Agent 5 of which both agreed required a high priority. In contrast, both believed Agent 2 deserved a low score, and Agent 1, was given a lower score then Agent 2 through Heuristic 1, but due to the score from Heuristic 2 being so high, was given considerable preference over Agent 1.

The configuration of the evaluation makes use of three groups of experiments, in each one, a selection of heuristics are evaluated independently first, with 10 runs of 20 targets, followed by a final run of 20 targets in which the heuristics are combined. The first group makes use of Euclidean distance, optimal route length, and planning time, combined by taking the sum of the Euclidean distance and optimal route length, and multiplied by the planning time. These heuristics are combined in this manner in order to overcome the normalisation, and offer a way in which the heuristics can be

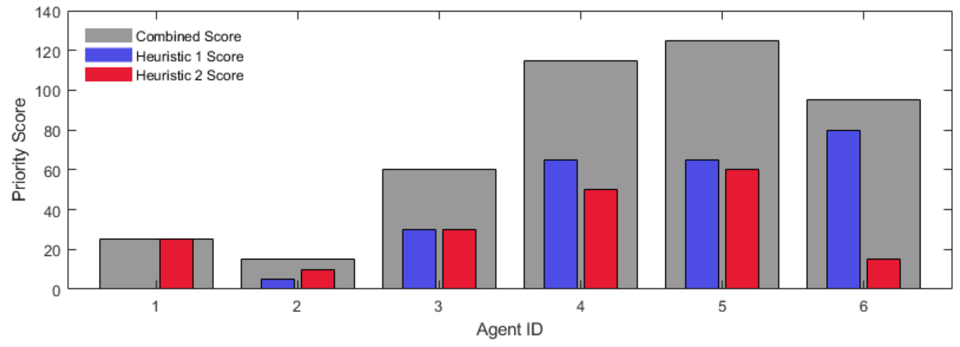


Figure 5.6: Graph showing an example of the potential of combinatorial heuristics (grey), as the combination of two independent heuristics (blue and red)

combined so they all have an impact, as planning time can be a very small number, it would likely be overshadowed by the other two. This same principle is also applied to the second and third groupings. In the second grouping, agent ID and optimal route length are evaluated independently and then combined through summation, and in the third grouping, Agent ID and Euclidean distance are evaluated independently then combined through summation.

Like with the evaluations for dynamic scoring and context-dependency, this evaluation also measures the delay against the unrealistic best case for the evaluation of the augmentation.

The results for this test can be seen in Figure 5.7, which show, for the three test groups, the individual distributions in black on the left, and the combined in blue on the right. The results for all three groups show promise, with the range of scores for the combinatorial heuristics being brought below the maximum score for any of the individual tests. While the mean is not decreased significantly if at all in any of them, the reduction from the maximum score achieved does show promise in the ability for combinatorial heuristics to wrangle the impact of worse metrics.

For instance, take the first test combining Euclidean distance, optimal route length and planning time. While the mean score for the combination is above the mean scores for the individual heuristics, its maximum score does not exceed the maximum scores for Euclidean distance and Planning time. This shows that by combining multiple heuristics together, the combination of their strengths can indeed be enhanced.

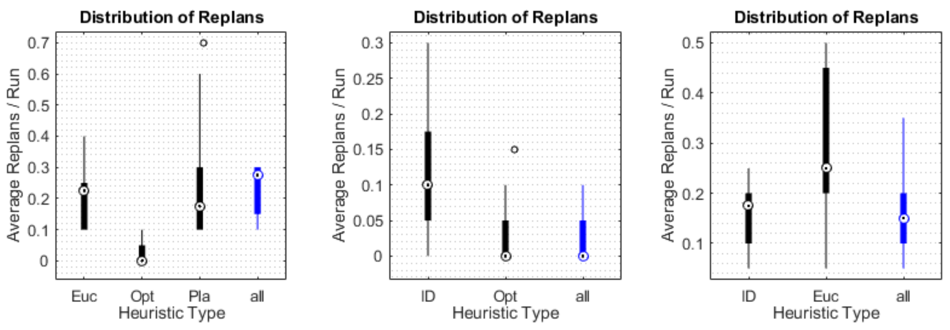


Figure 5.7: Boxplots showing the distribution of replans between single heuristics (black), and the combination of the heuristics (blue). Tested with three combinations of Euclidean Distance, Optimal Route Length, Planning time and Agent ID.

This same conclusion can be found with both of the other sets of results, however with the other set of results, there is the added bonus that the combination also is able to match or beat the average scores of the other groups.

5.1.5 Deadlocks

In a traditional prioritised system, there can be instances in which an agent is unable to get to its target. This can be for a number of reasons which are detailed earlier in this report (see Section 4.3.4). This implementation is designed to take this into account and manage the instances when a path is not available, this can be managed sometimes by increasing the priority score, thus forcing another agent to replan, and other times through adding a short delay to the agent so it waits idle at the reserved node so it can reach its target along the optimal route. As this is often a situation which is not managed properly or in a scalable manner, this portion of the evaluation aims to investigate how well the solution works.

This is evaluated by taking 50 runs of the simulation, and for each path generated within the run, logging the number of edges which had hit a deadlock situation and were overcome by the algorithm. The simulation itself is run in the smallest map, with 10 logistics agents and 1k total targets per run.

As the system is designed to handle every instance of a deadlock, the results in Figure 5.8 are only an indication of the occurrence of deadlocks.

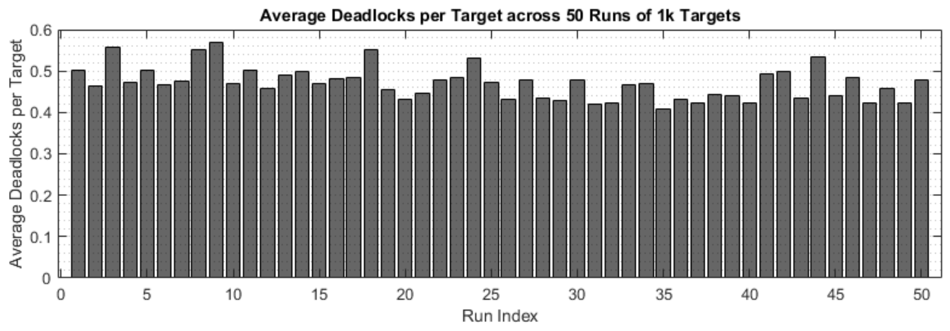


Figure 5.8: Bar chart showing the number of average number of deadlocks countered per run, across 50 runs of 1000 targets.

5.1.6 Scalability

The overarching theme of the project was rooted in scalability. While not the main focus, it is important for this to be included within the evaluation. In evaluating the scalability, it is important to consider two key aspects which can increase. This being the number of agents which increases the communication overhead and the amount of distributed and asynchronous planning, and the size of the map, which when increased, increases the memory requirement to run the system, along with communication data on the changes to the map, and the planning time increase as more of the map needs to be explored to navigate to the target.

In order to evaluate this system, it must first be broken down into core and extra processing components, where a core processing component is essential for any planning to take place, and extra processing component refers to components which are only occasionally run. In the algorithm it can be essentially broken down into two components, the first is the initial planning, and the second is the replanning. To generate any path to a target, the algorithm must perform the core planning, however the replanning is an area of extra processing which is only occasionally called. In a complete system, there would be no replanning, as the initial planning would already have the optimal assignments, thus the replanning can be ultimately classified as an extra processing step and thus a measure of the inadequacy of the planning schema. For this implementation, the total number of replans is calculated by the algorithm and is logged for analysis.

It is also worth mentioning here that in a distributed system, of which the system has been designed for, replanning is in no way cumbersome to the performance of the system, as the base for the algorithm is A* which is highly optimal for routing. However for evaluating the quality of the solution, it must be taken into account where there is excess processing in order to identify where improvements can be found.

In this evaluation, two tests are performed. For both tests, the distribution of replans is compared across 5 runs of 50 targets with the first test using the smallest map (containing only 53 edges), and the second test using the Riesholme map (containing 150 edges). Both are run with a varying number of agents, beginning with 1, then 5, 10, 25 and 50. The average number of replans per target is returned form the simulation, and the distribution of averages is measured for the resulting graph. The results can be found in Figure 5.9, where the average across the five runs, is plotted as a line graph through the centre of the grey region, while the grey region itself shows the range of values for each test from the minimum score to the maximum score.

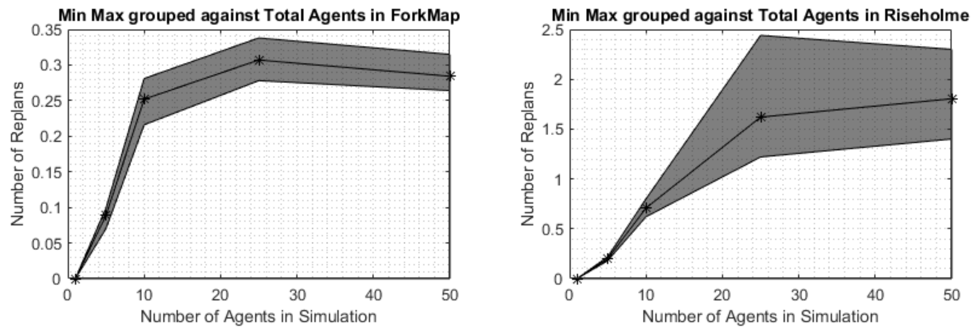


Figure 5.9: Grouped area charts showing the average number of replans against the number of agents int he simulation, grouped based on the minimum and maximum scores for each collection of runs. Completed within the fork_map (left) and the Riseholme map (right), for 1, 5, 10, 20 and 50 agents.

The results for both maps show the same general trend with a steep beginning to the curve as the number of agents increases, but both seem to level out around point where half the edges are filled, at which point the maximum value dips, with the number of agents increasing beyond the number of edges in the system the number of replans shoots up to around 1.5 replans per target, however this is not too relevant as the number of agents in any system would not be able to exceed the number of edges in the network as this would prevent any effective movement.

5.1.7 Edge Activity

The specific paths taken by each of the agents are also recorded by the simulation, included with the implementation, are tools to log reservation and conflict activities on all edges of the topological map, indicating the possible bottlenecks within the environment. The reservation activities and conflict activities for the Riseholme map are shown in Figures 5.10 and 5.11 respectively.

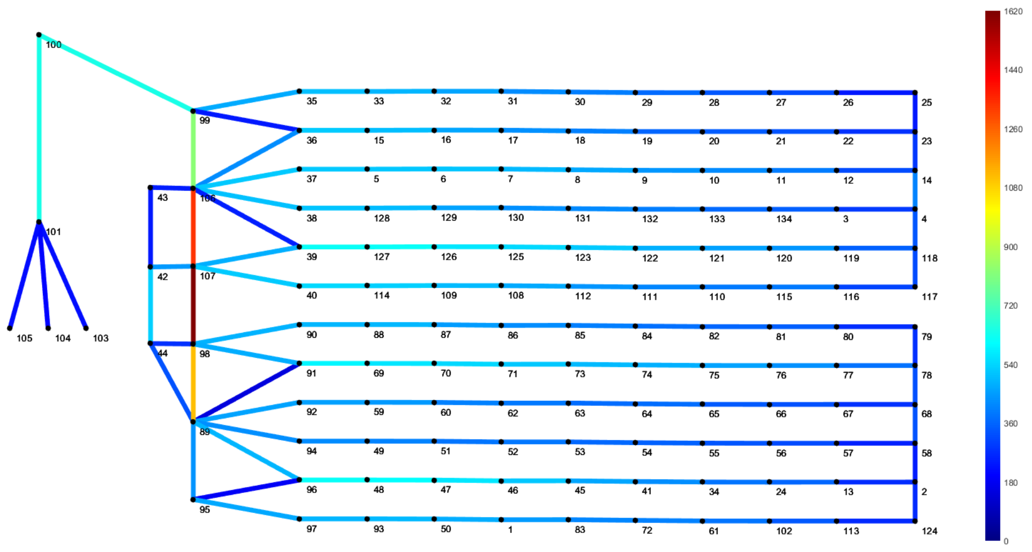


Figure 5.10: Network diagram for Riseholme with edges coloured to reflect the total number of reservations made for each edge across 5000 cycles. The numbers along the lines are node numbers.

Analysis these plots can help in environment restructuring operations to enable better multi-robot operations. From these graphs for Riseholme environment, show clearly, that the most active area within the network is the region just outside of the charging bays, having a very high activity with both the number of conflicts and reservations requested. If any area of the topology is to be addressed for bottlenecks, it should be this region. Although not highlighted by heavy activity, the route entering the food handling unit is also a bottleneck. This would have been more evident, if the robots would have been returning to the storage after reaching

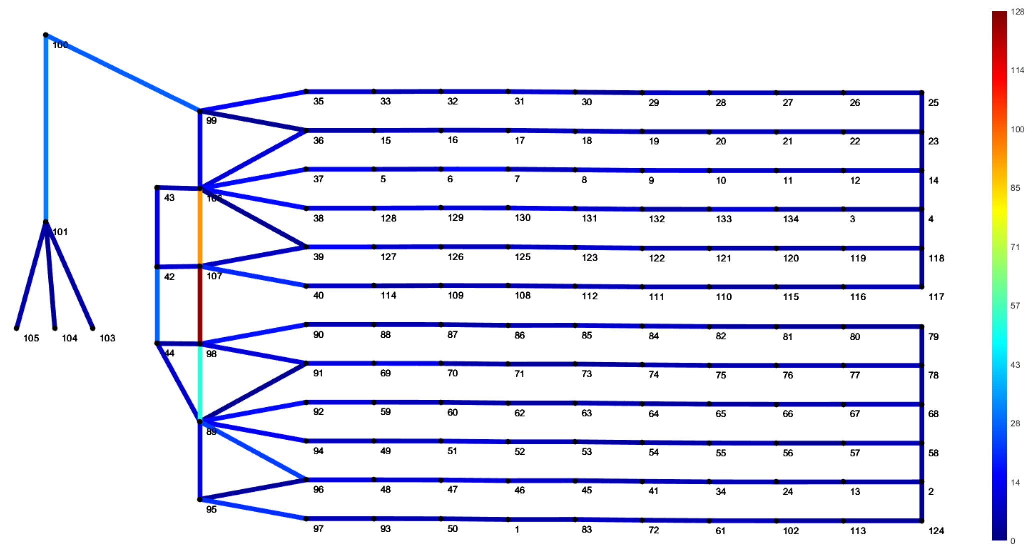


Figure 5.11: Network diagram for Riseholme with edges coloured to reflect the total number of conflicts occurring for each edge across 5000 cycles.. The numbers along the lines are node numbers.

each picker location (in this work each robot would visit five picker locations before going to storage for unloading). A simple restructuring option for such bottleneck is addition of a parallel lane.

Chapter 6

Conclusions

6.1 Conclusion

Within the introduction, the aim of the project was defined in two parts, the first to develop a framework which offered a deadlock free implementation of scalable prioritised planning, and the second to investigate potential improvements to prioritised planning to improve assignment quality.

The implementation was designed around these two aims, with the structuring and modifications to A* focused around improving quality of assignments, and to ensure it is deadlock free and to facilitate the investigation of potential improvements. The structuring of the planning and communication systems were designed around ensuring the implementation was scalable.

In total, four potential improvements were gleaned, with the first changing the manner in which targets were assigned, and the remaining three changing the manner in which priority scores were assigned. These four experiments were conducted by evaluating the total delay caused by the priority assignments, this delay being calculated as the difference between the time of arrival defined by the unrealistic optimal route, and the actual time of arrival.

The results gathered show how continuous assignment far outweighs batch assignment, with the benefits increasing as the simulation time increases, showing long-term operational benefits (Section 5.1.1). They also show how dynamic scoring can improve the quality of the assignments by removing the historic information about the agents paths generated with static scoring (Section 5.1.2). The results show how combinatorial heuristics have promise for managing wild assignments given by varying heuristics (Section 5.1.4). The fourth test showed that making use of the context-based heuristic chosen did not improve the results, however this could be simply due to the lack of context from the Monte-Carlo simulation (Section 5.1.3).

The results testing the first aim were both positive, showing how well the system handled deadlock situations, and showing how well the implementations scaled up as more agents were added.

To summarise further, the implementation has succeeded in being scalable and deadlock-free, and there is still promise for more potential improvements within prioritised planning assignments. Further work on this project will aim to investigate this.

Chapter 7

Discussion and Reflective Analysis

7.1 Project Reflection

MATLAB Debugger As described in the Toolsets section of the report (Section 3.3), MATLAB was in part chosen due to the extensive range of tools for debugging.

As the project involved a lot of complex interactions, managing these did not go without fault, with many points occurring early in development having minor issues as predicted. The tools were most often used for identifying small issues and mistakes while programming, however they showed their worth, when it came to more intricate error handling.

Around half way into the project, the system was being run for long periods of time to explore unknown state spaces, ensuring the system could handle effectively, all situations and setups encountered. During this period, there were numerous situations which caused issue tens of hours into the running of the system.

As the system made use of random seeds during this testing, when an issue arose causing system errors, it was possible to rerun the simulation to reach the same condition, however doing so meant there would be a lot of wasted time repeating tests to ensure minor amendments worked.

Setting a conditional break-point at the point where the issues occurred would have worked given a lot more knowledge about the cause of the issue, however to keep processing low, an effort was made to limit the amount of print statements, given how intensive a process they are.

This is where the command ‘dbstop if error’ came to be useful. This command, set at the top of the main script forced the system to enter a debugging mode at the point where the system would otherwise crash. By making use of this, along with other tools available such as the call stack, the code could be easily navigated in the debugging state. Making for an easily accessible environment to test changes within the issue environment before committing them.

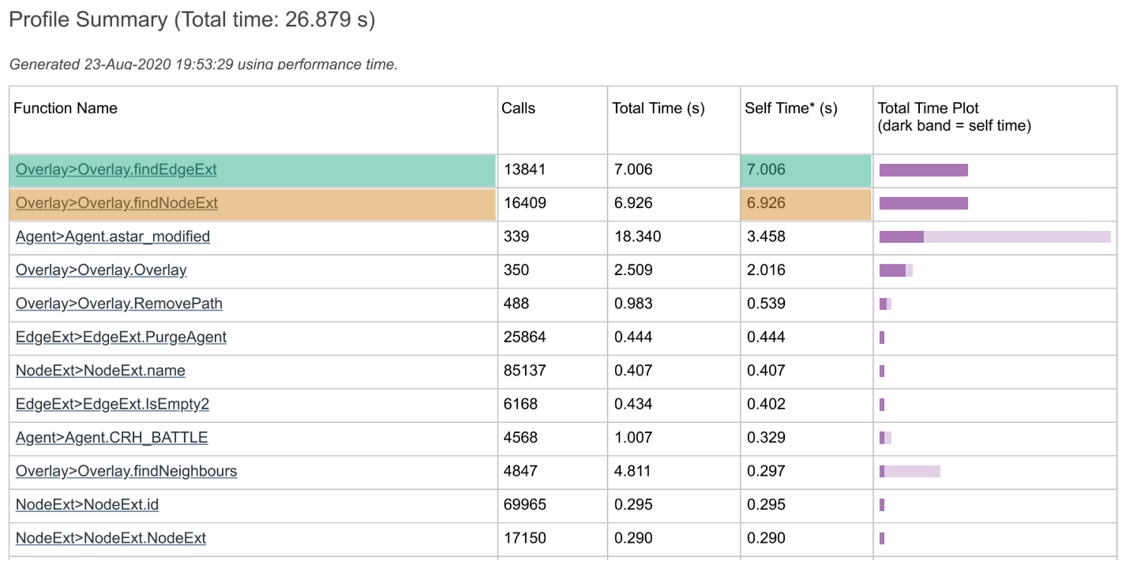
These tools were fully utilised throughout the development process and their use saved a lot of time which would otherwise be wasted rerunning the simulation to print out more information to the console.

In future projects, both with using MATLAB and other toolsets, it will be worth investing the time into gaining a better appreciation for the debugging tools available. MATLAB alone has many tools which have not been fully investigated.

MATLAB Profiler The MATLAB profiling tool was chosen to be used every so often during development to review the quality of the implementation and identify methods to simplify and improve the efficiency of the implementation.

It came to be very useful during development, where it was used to identify two small portions of code which were designed to find an edge in the full list of edges based on its two connected nodes, and to find a node based on its id in a list. These portions of code were called very frequently as they were used to move between reference identifiers and the objects they reference, being called multiple times within each loop of the motion planning algorithm. The two functions alone made up over 50% of the total processing time, shown in Figure 7.1. On top of this, the initial loading of a new map took up a significant amount of time also.

Making use of the profiler, enabled these resource-heavy regions to be identified without having to add lots of counters and timing circuits throughout the codebase. Along with some other minor amendments, these functions were reworked to be more efficient by replacing the for loops which searched the entire list of edges and nodes, and instead adopting a reference dictionary for identifying the location in the list in


Profile Summary (Total time: 26.879 s)

Generated 23-Aug-2020 19:53:29 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
Overlay>Overlay.findEdgeExt	13841	7.006	7.006	
Overlay>Overlay.findNodeExt	16409	6.926	6.926	
Agent>Agent.astar_modified	339	18.340	3.458	
Overlay>Overlay.Overlay	350	2.509	2.016	
Overlay>Overlay.RemovePath	488	0.983	0.539	
EdgeExt>EdgeExt.PurgeAgent	25864	0.444	0.444	
NodeExt>NodeExt.name	85137	0.407	0.407	
EdgeExt>EdgeExt.IsEmpty2	6168	0.434	0.402	
Agent>Agent.CRH_BATTLE	4568	1.007	0.329	
Overlay>Overlay.findNeighbours	4847	4.811	0.297	
NodeExt>NodeExt.id	69965	0.295	0.295	
NodeExt>NodeExt.NodeExt	17150	0.290	0.290	

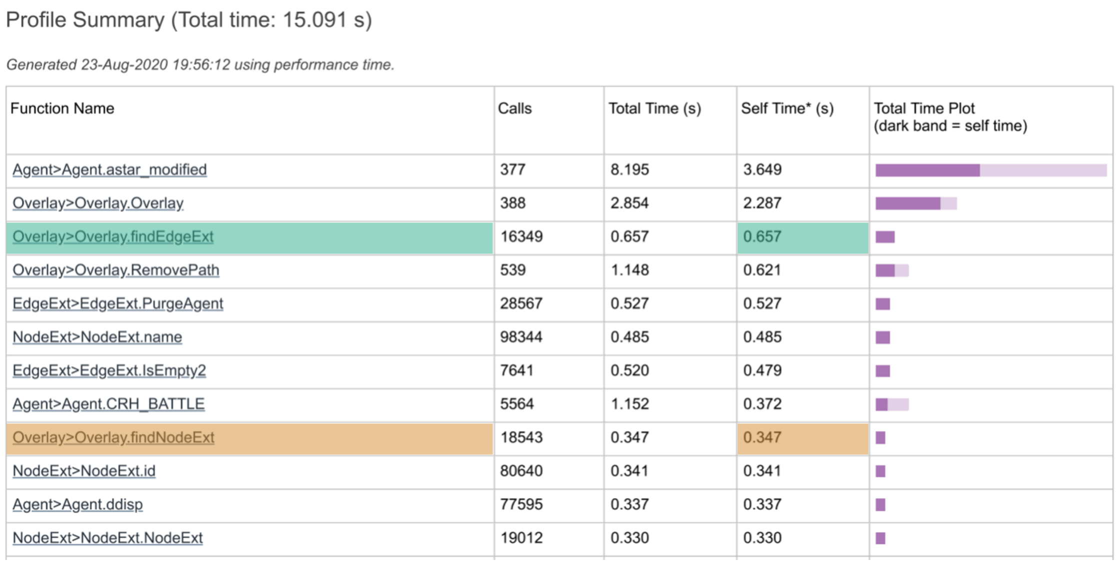
Figure 7.1: Profiler results detailing the processing time for the functions with the highest total processing time. Self time indicates time spent within the function excluding time spent in child functions. Slow variant for findEdgeExt and findNodeExt.

a vectorised form. With these changes, the total processing time was decreased by over 40%, shown in Figure 7.2.

While it was used every now and again, it was not used as often as initially planned. This meant that each time it was used, there was a lot of improvements which could have been missed as there were larger issues which overshadowed them.

The inconsistency of its use was simply caused by more important elements of the project taking priority, while the aim was to use the profiler frequently, there was no actual plan put in place to list when it should be used. However even with the limited use it was given, the most impacting elements of the resource processing were still handled effectively throughout the project.

This could potentially be improved on in future projects by integrating a more comprehensive log of the processing time of varying elements in a somewhat automated manner, perhaps through test-cases run whenever a commit is made. The tools to do such a thing are available within GitHub, along with external tools like Jenkins so this could serve to be very effective.


Profile Summary (Total time: 15.091 s)

Generated 23-Aug-2020 19:56:12 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
Agent>Agent.astar_modified	377	8.195	3.649	
Overlay>Overlay.Overlay	388	2.854	2.287	
Overlay>Overlay.findEdgeExt	16349	0.657	0.657	
Overlay>Overlay.RemovePath	539	1.148	0.621	
EdgeExt>EdgeExt.PurgeAgent	28567	0.527	0.527	
NodeExt>NodeExt.name	98344	0.485	0.485	
EdgeExt>EdgeExt.IsEmpty2	7641	0.520	0.479	
Agent>Agent.CRH_BATTLE	5564	1.152	0.372	
Overlay>Overlay.findNodeExt	18543	0.347	0.347	
NodeExt>NodeExt.id	80640	0.341	0.341	
Agent>Agent.ddisp	77595	0.337	0.337	
NodeExt>NodeExt.NodeExt	19012	0.330	0.330	

Figure 7.2: Profiler results detailing the processing time for the functions with the highest total processing time. Self time indicates time spent within the function excluding time spent in child functions. Fast variant for findEdgeExt and findNodeExt.

Project Diary The project diary was intended for note-taking throughout the project, keeping a daily record of the decisions and actions completed each day and to serve as a log for reviewing further into the project.

Following the description given within the methodology, the project plan, when fully utilised, worked well for maintaining the structure within the project, it was used not only for storing notes and task details, but also for sharing ideas, information and planning.

While in the beginning, during the exploration phase the project diary was used very frequently, as the project progressed it was used less often. There were periods during development when a lot of work was being completed and the diary was neglected. The poor use, boils down to self-discipline, where as the project diary was very disconnected to the requirements of the project, it was easy to dismiss it at times, however as it was neglected more and more, it became easier to dismiss.

In future projects, this will be improved by integrating it more into the requirements of the project. By creating an environment where the project diary is used more as an interactive tool, it would be harder to dismiss.This can be easily achieved with methods such as by making use of MS Sharepoint for interactive communication with the project supervisor. Doing this, it could become a more interactive tool not just for taking notes but also for general communication.

Project Plan The project plan was an effective tool for detailing the timescales of the project, especially once adapted when the lean approach was taken, it easily identified in a visually stimulating manner, the tasks and their respective duration's. This meant that with only a glance, it was clear where the project was with consideration to the time allocations assigned.

While the project's general structure was planned from the start, not enough care was taken to consider the specific details of the evaluation until around half way into the project. The project plan, along with the research methods section of the methodology, focused mainly on the timescales within the project, and the features of the project which would help to increase the quality of the evaluations, rather then the specific details for the effective data collection and evaluation of the results.

With future projects, this could be easily improved by also completing an experiment plan at the start of the project, a simple document which details the experimental design. While this was investigated in part through the elements mentioned within the research methods section of the methodology (Section 3.4), it was not investigated to enough depth. Including a short plan detailing the experiments at the start of the project, and completed in a single form, it would cement the foundations of the project and its development.

7.2 Further Works

As detailed in the design and development chapter of the report, there were quite a few assumptions (Section 4.6) were made and integrated within the development of the implementation.

These assumptions ranged from little impact elements such as agents being assumed to have infinite battery and random noise being included with movement, to node and edge capacity being neglected. Each of these assumptions made, were done so for varying reasons, however before being able to implement this proposed method onto a real set of robots, these assumptions would have to be managed.

In the project management section (Section 3.1 it details how after the mid-project review, the testing of the solution was reduced to focus more on the core elements of the project. Before implementing the solution to real robots, the simulations extending the detail should also be completed.

The results showed the effectiveness of the context-dependent heuristics as negligible at best. As the simplified simulation lacked a lot of the context required for context-dependency, the evaluation itself was highly restricted. A simulation which mimicked the real world more closely would have more opportunity for context-dependency to show. It would be interesting to see what could be gained from investigating this concept within autonomous systems in other contexts. Contexts such as shelf stacking robots, or warehouse robots dynamically retrieving shelving units could show promise for further investigation. This could be explores further by evaluating different metrics, for instance looking into how context-dependent heuristics can assist within homogeneous multi-robot systems and looking at how task types are prioritised differently.

The inclusion of these changes, amendments and implementation structures would shine more light on issues identified within the project. Further works should be mainly focused on extending the capabilities of the simulation, with importance given to firstly developing the node and edge capacity as this element of all the further improvements identified, takes the simulation furthest from reality.

It would also be worth investigating the proposed methods, with the use of more complex heuristics such as Path Prospects and Naive Surroundings explored in the literature review.

References

Andreychuk, Anton and Konstantin Yakovlev (2018). ‘Two techniques that enhance the performance of multi-robot prioritized path planning’. In: *Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS* 3, pp. 2177–2179. ISSN: 15582914. arXiv: 1805.01270 (cit. on p. 16).

Azarm, Kianoush and Guenther Schmidt (1997). ‘Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation’. In: *Proc. - IEEE Int. Conf. Robot. Autom.* 4.April, pp. 3526–3533. ISSN: 10504729. DOI: 10.1109/robot.1997.606881 (cit. on p. 15).

Bechar, Avital and Clément Vigneault (2016). ‘Agricultural robots for field operations: Concepts and components’. In: *Biosystems Engineering* 149, pp. 94–111. ISSN: 1537-5110 (cit. on p. 6).

Bennewitz, Maren, Wolfram Burgard and Sebastian Thrun (2001). ‘Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems’. In: *IEEE Int. Conf. Robot. Autom.* Vol. 1, pp. 271–276 (cit. on p. 15).

– (2002). ‘Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots’. In: *Rob. Auton. Syst.* 41.2-3, pp. 89–99 (cit. on pp. 14–16).

Botea, Adi, Martin Müller and Jonathan Schaeffer (2002). *Using abstraction for planning in Sokoban* (cit. on p. 19).

Cap, Michal et al. (2015). ‘Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots’. In: *IEEE Trans. Autom. Sci. Eng.* 12.3, pp. 835–849. ISSN: 15455955. DOI: 10.1109/TASE.2015.2445780. arXiv: 1409.2399 (cit. on p. 8).

‘Cooperative multi-robot path planning by heuristic priority adjustment’ (2006). In: *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 5954–5959. DOI: 10.1109/IROS.2006.282480 (cit. on p. 15).

Das, Gautham P. et al. (2018). ‘Discrete Event Simulations for Scalability Analysis of Robotic In-Field Logistics in Agriculture – A Case Study’. In: *ICRA 2018 Workshop on Robotic Vision and Action in Agriculture*. Brisbane (cit. on p. 7).

Dewangan, Ram Kishan, Anupam Shukla and W. Wilfred Godfrey (2017). ‘Survey on prioritized multi robot path planning’. In: *2017 IEEE Int. Conf. Smart Technol.*

Manag. Comput. Commun. Control. Energy Mater. ICSTM 2017 - Proc. August, pp. 423–428 (cit. on p. 8).

Erdmann, Michael and Tomás Lozano-Pérez (1987). 'On multiple moving objects'. In: *Algorithmica* 2.1-4, pp. 477–521 (cit. on pp. 11–13, 51).

Fentanes, Jaime Pulido et al. (2015). 'Now or later? Predicting and maximising success of navigation actions from long-term experience'. In: *Proceedings - IEEE International Conference on Robotics and Automation.* Vol. 2015-June. June. Seattle, WA, USA, pp. 1112–1117. ISBN: VO -. DOI: `10.1109/ICRA.2015.7139315` (cit. on p. 45).

Ferguson, Dave, Maxim Likhachev and Anthony Stentz (2005). 'A guide to heuristic-based path planning'. In: *Proc. Int. Work. Plan. under Uncertain. Auton. Syst. Int. Conf. Autom. Plan. Sched.* Pp. 1–10 (cit. on p. 17).

Fernandez Carmona, Manuel, Tejas Parekh and Marc Hanheide (2019). 'Making the Case for Human-Aware Navigation in Warehouses'. In: *Towards Autonomous Robotic Systems.* Ed. by Kaspar Althoefer, Jelizaveta Konstantinova and Ketao Zhang. Cham: Springer International Publishing, pp. 449–453. ISBN: 978-3-030-25332-5 (cit. on p. 6).

Hu, Biao and Zhengcai Cao (2019). 'Minimizing task completion time of prioritized motion planning in multi-robot systems'. In: *IEEE Int. Conf. Syst. Man Cybern.* Vol. 2019-Octob. IEEE, pp. 1018–1023 (cit. on pp. 16, 59).

Kavraki, Lydia E. et al. (1996). 'Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces'. In: *IEEE Trans. Robot. Autom.* (Cit. on pp. 18, 19).

Koenig, Sven and Maxim Likhachev (2002). 'Improved Fast Replanning for Navigation in Unknown Terrain'. In: *IEEE Trans. Robot.* 21.3, pp. 354–363 (cit. on p. 17).

LaValle, S. M. and J. J. Kuffner (1999). 'Randomized kinodynamic planning'. In: *Int. J. Rob. Res.* 20.5, pp. 378–400 (cit. on p. 19).

Likhachev, Maxim, Dave Ferguson et al. (2005). 'Anytime dynamic a*: An anytime, replanning algorithm'. In: *ICAPS 2005 - Proc. 15th Int. Conf. Autom. Plan. Sched.* Pp. 262–271 (cit. on p. 18).

Likhachev, Maxim, Geoff Gordon and Sebastian Thrun (2004). 'ARA *: Anytime A * with Provable Bounds on Sub-Optimality'. In: *Adv. Neural Inf. Process. Syst.*, pp. 767–774 (cit. on p. 18).

Ma, Hang et al. (2019). 'Searching with Consistent Prioritization for Multi-Agent Path Finding'. In: *Proc. AAAI Conf. Artif. Intell.* 33, pp. 7643–7650. ISSN: 2159-5399. DOI: `10.1609/aaai.v33i01.33017643`. arXiv: `1812.06356` (cit. on pp. 16, 59).

Rizk, Yara, Mariette Awad and Edward W. Tunstel (Apr. 2019). 'Cooperative Heterogeneous Multi-Robot Systems: A Survey'. In: *ACM Comput. Surv.* 52.2. ISSN: 0360-0300 (cit. on p. 6).

Ryan, Malcolm (2006). 'Multi-robot path-planning with subgraphs'. In: *Proc. 2006 Australas. Conf. Robot. Autom. ACRA 2006* (cit. on p. 19).

Ryan, Malcolm R.K. (2008). 'Exploiting subgraph structure in multi-robot path planning'. In: *J. Artif. Intell. Res.* 31, pp. 497–542 (cit. on p. 18).

Stentz, Anthony (1995). 'The Focussed D* Algorithm for Real-Time Replanning'. In: *Int. Jt. Conf. Artif. Intell.* August (cit. on p. 17).

Van Den Berg, Jur P. and Mark H. Overmars (2005). 'Prioritized motion planning for multiple robots'. In: *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, pp. 2217–2222 (cit. on pp. 13, 50).

Velagapudi, Prasanna, Katia Sycara and Paul Scerri (2010). 'Decentralized prioritized planning in large multirobot teams'. In: *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst.* Pp. 4603–4609 (cit. on pp. 13, 51).

Wagner, Glenn and Howie Choset (2011). 'M*: A complete multirobot path planning algorithm with performance bounds'. In: *2011 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 3260–3267 (cit. on pp. 10, 19).

Wagner, Glenn, Minsu Kang and Howie Choset (2012). 'Probabilistic path planning for multiple robots with subdimensional expansion'. In: *Proc. - IEEE Int. Conf. Robot. Autom.* IEEE, pp. 2886–2892 (cit. on pp. 10, 20).

Wu, Wenying, Subhrajit Bhattacharya and Amanda Prorok (2019). 'Multi-Robot Path Deconfliction through Prioritization by Path Prospects'. In: pp. 22–24. arXiv: `1908.02361`. URL: `http://arxiv.org/abs/1908.02361` (cit. on p. 14).

Zhao, Weiming, Ronghui Liu and Dong Ngoduy (2019). 'A bilevel programming model for autonomous intersection control and trajectory planning'. In: *Transp. A Transp. Sci.*, pp. 1–25 (cit. on pp. 11, 19).