Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2021

Exploring Improvements for Autonomous Traffic Management of Underground Mining Vehicles

Jacob Ljungberg

Master of Science Thesis in Electrical Engineering

Exploring Improvements for Autonomous Traffic Management of Underground Mining Vehicles

Jacob Ljungberg

LiTH-ISY-EX--21/5452--SE

Supervisor: **Pavel Anistratov**
ISY, Linköpings universitet
Gunnar Persson
Combitech AB

Examiner: **Jan Åslund**
ISY, Linköpings universitet

Division of Vehicle Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

A joint project between Combitech and Epiroc aims to further develop and improve autonomous underground block mining operations. Narrow tunnels and limited space impose a challenging task coordinating the vehicles. The current solution has worked well for specific cases, but newer projects impose more complex geometry where the simplicity of current method negatively impacts operation efficiency. This thesis approaches a reduced version of the problem to investigate new alternative methods that could be used to improve production rate for new mines.

In an attempt to reach the best possible solution, an effort is made to minimize decisional limitations of the traffic algorithm. This leads to identifying that improvement focuses around utilizing a path-finding algorithm which can perform space-time searches.

Instead of moving along a predetermined path, the vehicles can now move freely. Decision taking is based on a cost model where time, position, and movement are weighted such that the algorithm takes actions to minimize the cost, designed to complete agents' missions.

The custom built A* space-time search algorithm is a mix between different state-of-the-art applications. The end product is a mine simulator executing the new algorithm shown to vastly improve operational efficiency. Vehicles seemingly find creative ways to avoid collisions, improving the production rate with up to 49% for specific two agent cases. These results are for simple models and are seen as indications for potential improvement.

In conclusion, this study has been a first attempt in exploring possibilities for further development. Operational results have shown great future potential for a path finding solution. Further work extending past the scope is proposed to move towards a full scale implementation.

Acknowledgments

This thesis concludes my journey to receiving a Master of Science degree at Linköping University, and it has been an immensely exciting project from beginning to end. I have always found autonomous technology a fascinating subject and am thrilled to have gotten to do this project. I hope to get a chance to continue exploring the field in the future.

Even though I am the sole author on this thesis, my supporting team both at Combitech and the University have played a big part in the process. I want to direct my sincerest thank you to everyone who generously have spent time discussing ideas and solutions, helping me complete this thesis, specifically:

Jan Åslund who as the examiner has engaged in the material beyond the requirements of his role. Your experience and help has been greatly appreciated.

Pavel Anistratov as the University supervisor has been instrumental in both identifying problems and discussing solutions. You always found time to support and for your help I am truly grateful.

Daniel Engelson, who as my opponent has provided me with great feedback and whose perspective has been valuable.

Gunnar Persson, my external supervisor who gave me the warmest welcome to Combitech where he showed enthusiasm in the project accompanied by invaluable feedback and support. Your genuine engagement to discuss any aspects of the project has been greatly appreciated.

Simon Arkeholt, who together with the rest of the Combitech team all shared their solid expertise.

Lastly, I would like to thank my family and friends for all their support.

I am truly grateful to you all.

Linköping, Nov 2021
Jacob Ljungberg

Contents

Notation

Abbreviations

Abbreviations	Phrase
MAPF	Multi-agent pathfinding problem
RRT	Rapidly-exploring random tree

Notations

Notations	Description
a	Agent
e	Edge
g	Goal
G	Graph
st	Space-time
s	Starting point
$\mathcal{T}$	Reservation table
v	Vertex (alternatively node)

1 Introduction

This chapter presents background information, the goals, exclusions as well as a thesis outline.

1.1 Background

The nature of industry has been evolving towards process automation with increasing robotic presence in the recent years. Epiroc AB is a company incorporating autonomy in their solutions. They are a Swedish manufacturer of mining equipment using autonomous technology to aid customers performing tedious and repetitive tasks.

In partnership, Epiroc and Combitech have in recent years developed a solution for traffic management of autonomous loaders for mines. They have combined their technology and knowledge on how to systematise and develop traffic management solutions. The utilization of a driverless fleet of machines in a risky work environment has meant achievement of a new level of efficiency and safety. The solution is under continuing development by Epiroc and Combitech teams.

The mines in question use multiple autonomous vehicles, all hauling material in a shared underground area. Independent vehicles moving simultaneously through a network of narrow tunnels create coordination issues within the field of motion planning. To tackle the issue, the *traffic management system* is used which makes decisions on how the vehicles should move to ensure collision-free operation. The currently implemented traffic management solution for the vehicle-coordination is functional but limited.

With constant access to additional material to move, any improvements of the

vehicles' movement efficiency directly increases the mine's production rate.

1.2 The Vehicle

The traffic management system controls a fleet of loader-vehicles developed by Epiroc called *Scooptram ST18*, see Figure 1.1. The name refers to the 18 tonne bucket capacity. They are just over 11 meters long weighing around 50 tonnes. They have articulated steering, meaning the front and rear are split into two parts, jointed by a vertical hinge. Hydraulic cylinders control the articulation angle between the halves, which is how the vehicle is steered. The articulated steering means that the vehicle has similar driving characteristics in reverse as when moving forward. The vehicles can be operated from the on-board cabin or remotely by joystick.

Figure 1.1: *The Epiroc Scooptram ST18 loader used in the underground mine.*

1.3 Current State of Operation

The current system is used in two underground block caving mines: a mine producing both copper and gold denoted Alpha, and a mine producing iron denoted Beta. The mines utilize a mining method called block mining which involves loosening a large ore body, allowing it to progressively collapse under it's own weight. The broken rubble falls into constructed funnels and out onto a retrieval floor where operators can extract the material whilst being shielded from the shifting ore body above, see Figure 1.2. Any material removed from the piles is replaced by more rocks shifting down through the funnels. The operation will continue until the body of ore is extracted completely which for large ore bodies can be a multi-decadal operation.

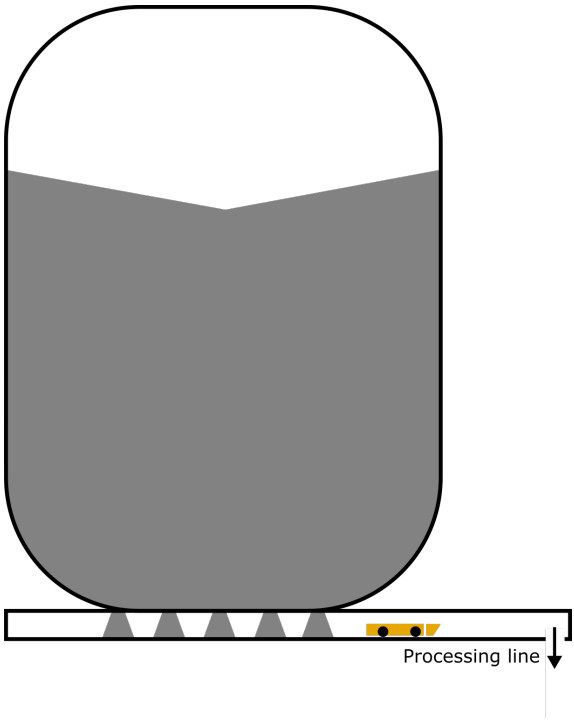

Figure 1.2: *A cross-section view of the block caving mine. A loader can be seen in yellow. The material is dumped at the end of the horizontal tunnel to proceed along the processing line.*

On the retrieval floor, the *Scooptram* loaders operate in semi-autonomous loading-dumping cycles. In Alpha, the cycles duration usually vary between 3 and 9 minutes. Due to the process complexity, an operator remotely controls the vehicle through the loading to fill the bucket. Once loaded, the operator engages autonomous mode where the algorithm takes control for the dumping sequence. The loaders transport the ore to a crusher into which they unload. The loaders operate on a single floor excavated just below the ore body. To reduce risk of a collapse, the pathways are just wide enough for one vehicle to pass. The block caving mine Alpha's geometry can be seen illustrated in Figure 1.3. Mine Beta is illustrated in Figure 1.4.

The autonomous system has been achieved by manually performing dumping cycles and simultaneously recording sensor data of both the surroundings and the vehicle itself. The sensor data is then used as a reference signal to replicate the movements through the vehicles control inputs. As long as the vehicle has an accurate approximation as to where it is, the system works as intended. Coinciding path recordings allows vehicle path transfers.

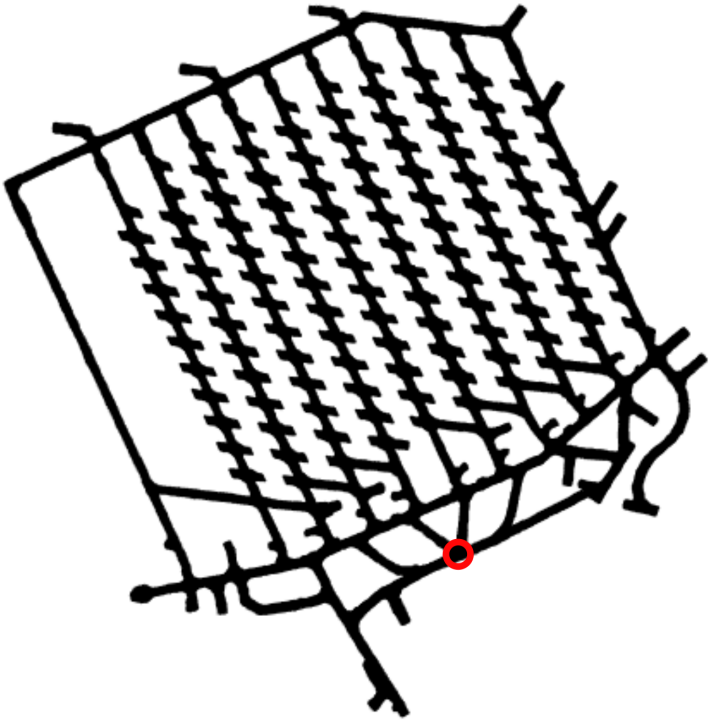

Figure 1.3: *An illustration of Alpha. The black shape is the tunnel network where loaders can move. Loading is performed in the pockets along the vertical pathways. Dumping is performed at the marked location.*

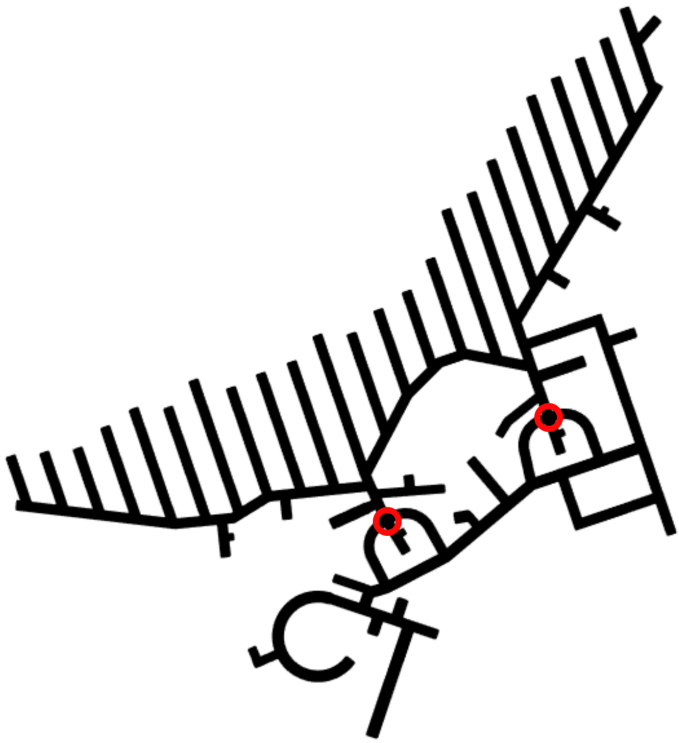

Figure 1.4: *An illustration of Beta. The black shape is the tunnel network where loaders can move. Loading is performed at the end of the vertical pathways. Dumping is performed at the marked locations.*

1.3.1 Current Coordination Method

The current method to coordinate the vehicles uses a static zone partitioning of the environment. Each zone is limited to a single agent at a time. Transferring zones requires the next zone to be free. If the next zone is occupied, the vehicle will stop and wait for access.

With stopping being the only coordination method, it is crucial that the vehicles do not end up in situations where two vehicles wait for each other which implies a lockup. This problem was addressed by making the entire common area connecting the loading drifts a single zone. This means that any potential lockups are avoided due to any areas where lockups are possible are limited to one vehicle at a time. Another necessity is that no vehicles are assigned loading spots in the same drift zone.

This method is in operation and has been shown to work well for systems where the common zones are small. However, upcoming projects such as Beta involves larger systems where an increased number of drifts are connected to the same common zone, see Figure 1.5. This increases path coupling between the vehicles where a larger portion of each vehicles path generally coincides with others. Here, a more sophisticated system is needed for efficient operation.

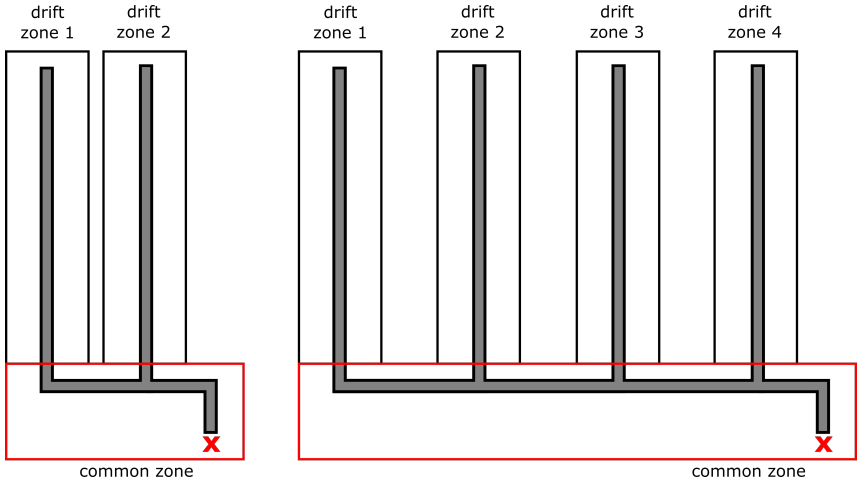

Figure 1.5: *Two case depictions with widely different zone ratios. The left model is less challenging for coordination.*

1.4 Goals

The main goal of this thesis is investigating trajectory planning improvements to increase the mine's rate of production through a reduction of time spent avoiding conflicts between the agents. An interesting aspect to implement is enabling

a more efficient operation by allowing vehicles to decide between all available dump sites.

The algorithm must be able to handle plan changes, meaning reversing machines to clear the way if necessary. As the algorithm will be used in real-time, the solutions must be produced quickly to minimize issues. The work will be completed from a conceptual standpoint, where methods are tested and evaluated in a separate system to reduce time spent on implementation into the existing system.

1.5 Exclusions

Operation safety is of high priority for the mining operation. This means the the vehicles should behave predictably. To allow adjusting vehicle behaviour, the sole approach has been decided to be model based algorithms.

To ensure focus on the agents' paths, vehicle dynamics will mostly be excluded. These properties are not essential to a path finding problem, nor projected to issue any large altercations when included.

1.6 Thesis Outline

What you as a reader can expect to encounter in this thesis.

- **Chapter 2 - Problem Analysis** is an approach to describe the problem scenario in detail. The chapter covers how the vehicle behaviour would theoretically improve the operations efficiency and what future problems that behaviour would produce.
- **Chapter 3 - Motion Planning Theory** introduces the general path planning problem along with a survey for related research on algorithms. Some initial discussion about the methods are used to extract valuable parts from different papers.
- **Chapter 4 - System and Environment Model** describes the discretization needed to apply a path planning algorithm to the scenario.
- **Chapter 5 - Path Planner** approaches the motion aspect where an algorithm for planning each agent is developed.
- **Chapter 6 - Results** describes what testing reveals about the potential improvement of the new approach.
- **Chapter 7 - Discussion** takes an approach to evaluate the method including strengths and weaknesses. Some final discussion about bridging the gap induced by the problem reduction.

2

Problem Analysis

Finding an appropriate solution requires understanding the problem which will be attempted here.

2.1 Mining Operation - Continuation

The mining operation is mentioned in Chapter 1 but to adequately understand the mine operation a further description is required.

To successfully perform the bucket actions at each destination, the vehicles must arrive in the correct orientation. Accessing some loading sites requires arriving in reverse, and for others the opposite, see Figure (2.1). The vehicles are limited to changing orientation in tunnel intersections where two-, and three point turns can be performed. When planning the paths, this aspect must be taken into consideration.

In Alpha, the number of active vehicles varies from time to time as they rotate in and out of maintenance and repair. Usual operation consists of between 2 and 3 vehicles. Other operations might differ in fleet size.

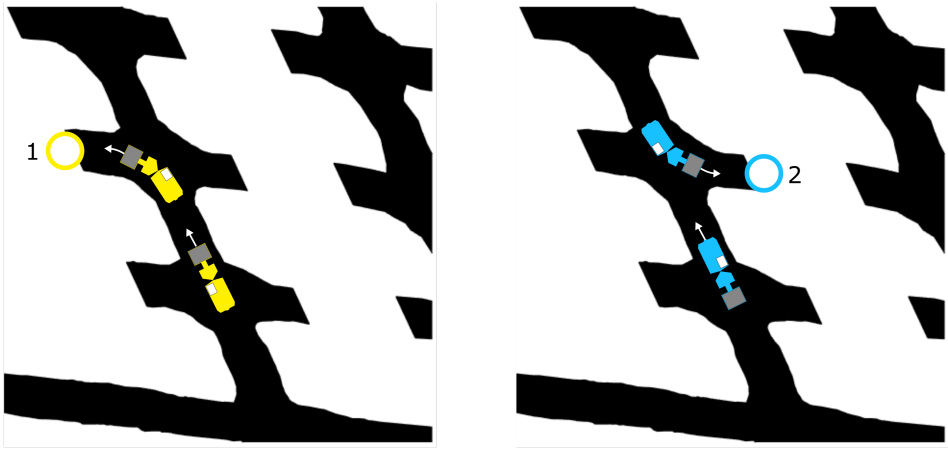

Figure 2.1: *The narrow tunnels requires many of the loading sites to be approach in reverse as seen for the blue vehicle.*

2.2 Vehicle Behaviour

Knowing how the situation with the mine and the vehicles looks, we will now focus on the vehicles and what can be expected from them. There will here be a case comparison and some initial discussion. The case is inspired by a current project under development by Combitech, mine Beta. The case has shown to give the current coordination method a hard time. Here, a long horizontal tunnel connects a large number of loading drifts. Consider two vehicles denoted a_1 and a_2 in operation where they repeatedly move material from the load piles to the dump site, see Figure 2.2. The vehicles are depicted as colored diamonds with their paths illustrated in respective color. As both vehicles set off at the same time from their respective position, an apparent conflict will arise at some point as both paths coincide. For simplicity, a_2 will have priority and selfishly perform its mission without interruptions. Vehicle a_1 will complete its mission whilst avoiding conflict.

2.2.1 Vehicular Coordination

Here, a few different types of vehicle behaviour will be manually tested and compared with a reduced model to identify which one the new method should capture.

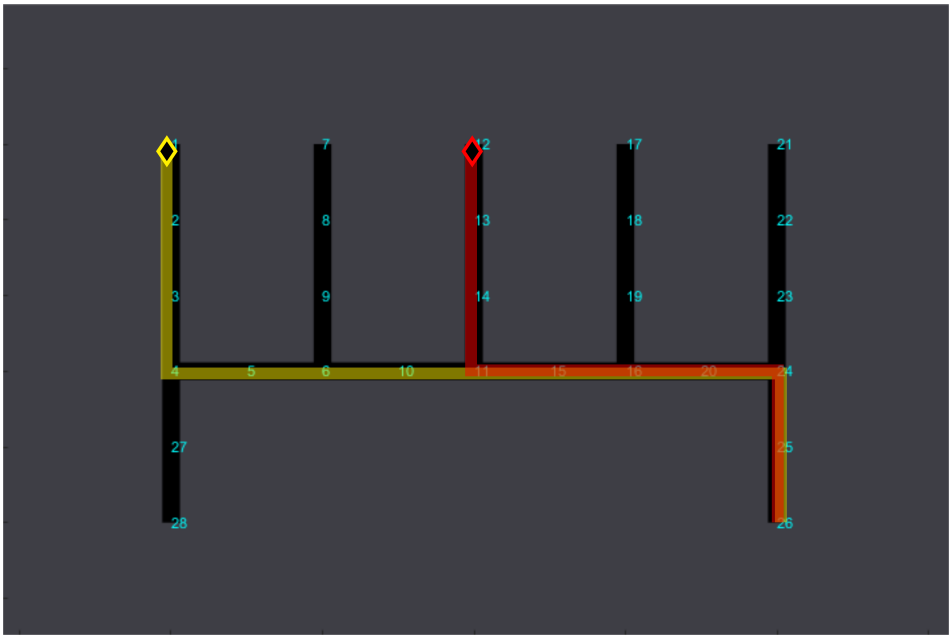

Figure 2.2: *An explanatory case depicting the coordination of two vehicles in a mine.*

A novel approach to quantify behaviour performance for this scenario is by comparing the cycle completion time. To get an insight into the efficiency, the cycle times will be compared between best case scenarios, where both vehicles have a free path. Reaching 100% efficiency means no conflict arose, which for this specific case is unreachable.

To use a simple model, assume the particle vehicle can either move one unit of distance or wait in place at each unit of time. The dumping process requires an extra time unit. The best case scenario for each individual agent is depicted in Table 2.1. By counting the steps, one can see that the shortest path of a_1 is 13 units long, whereas the shortest path of a_2 is 9 units long. Including return trip and the extra dumping duration we arrive at the following:

Table 2.1: *Shortest mission length for the case depicted in Figure 2.2*

Vehicle	a_1	a_2
Mission length	27	19

To enable comparison to the current method, the entire horizontal tunnel is assigned into a single-agent reservation zone. Since a_2 has been given priority, a_1 will be required to wait. The entire horizontal shaft is considered occupied as a_2 remains within it. Vehicle a_1 will wait at node 3 illustrated in the Figure 2.3,

until given access. This method requires a_1 spending 42 units of time before completing the cycle, which gives an efficiency of: $\frac{27}{42} = 64\%$.

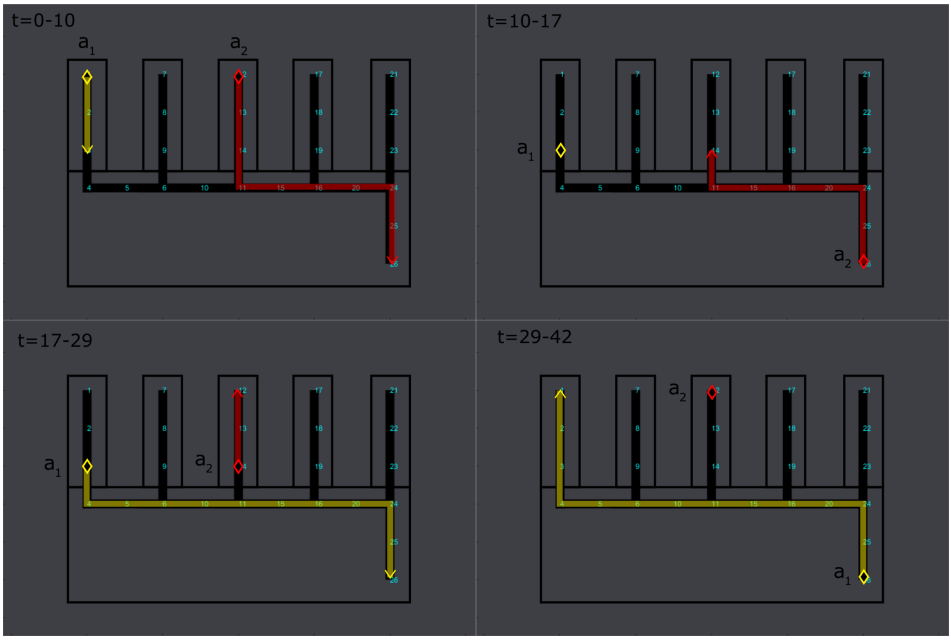

Figure 2.3: *An illustration of the solution for the current coordination method.*

In an attempt to manually figure out a solution with higher efficiency one can imagine that instead of waiting at node 3, a_1 could approach node 10 and instead wait at this point for a_2 to pass. This since this is the closest point along its path which does not overlap with the path of a_2. This would shorten the cycle from 42 units to 38 units, increasing a_1's time efficiency to $\frac{27}{38} = 71\%$, see Figure 2.4.

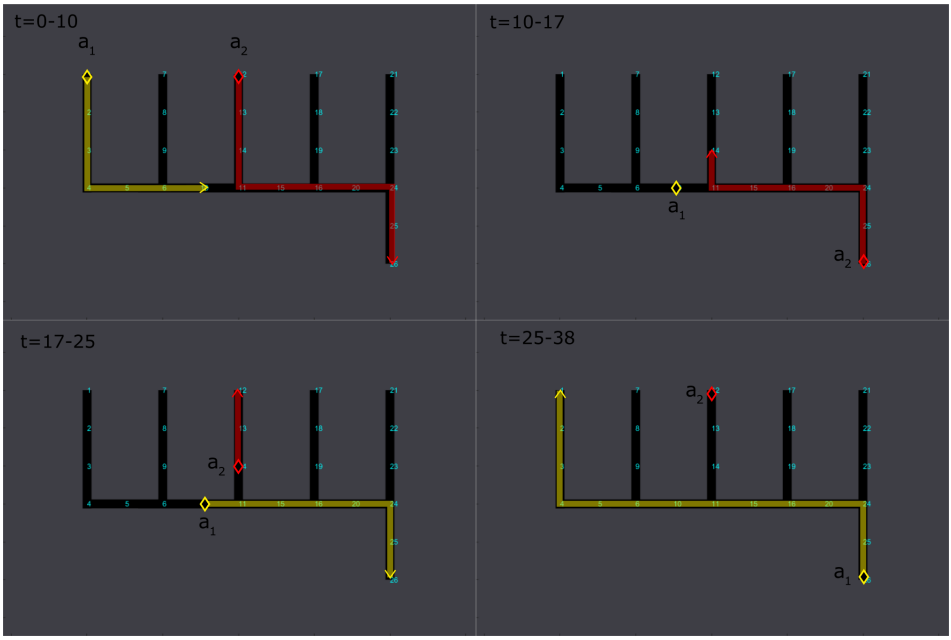

Figure 2.4: *An illustration of a method to coordinate two vehicles with a higher time efficiency compared to Figure 2.3 by disregarding the static zones and instead looking at the vehicles' path coupling.*

There are however solutions to this specific case producing even higher efficiency than having a_1 waiting at node 10. Vehicle a_1 can instead follow a_2 into the conflicted zone. As a_2 turns down to dump, a_1 has enough time to reach node 19 and avoid a conflict as a_2 returns. As a_2 passes by, a_1 can proceed with its mission, finishing its cycle sooner. This sequence can for a_1 be completed during 34 time units. This gives an efficiency of 79%, see Figure 2.5.

To reach the fully optimal solution (unknown efficiency value) the vehicles would all be selfless and always act in the best interest of the entire fleet. This is something which quickly becomes difficult to solve with pure intuition and instead requiring computing power. The results here indicate the possibility of allowing vehicles to deviate from the standard routes to chase optimality by smarter time utilization.

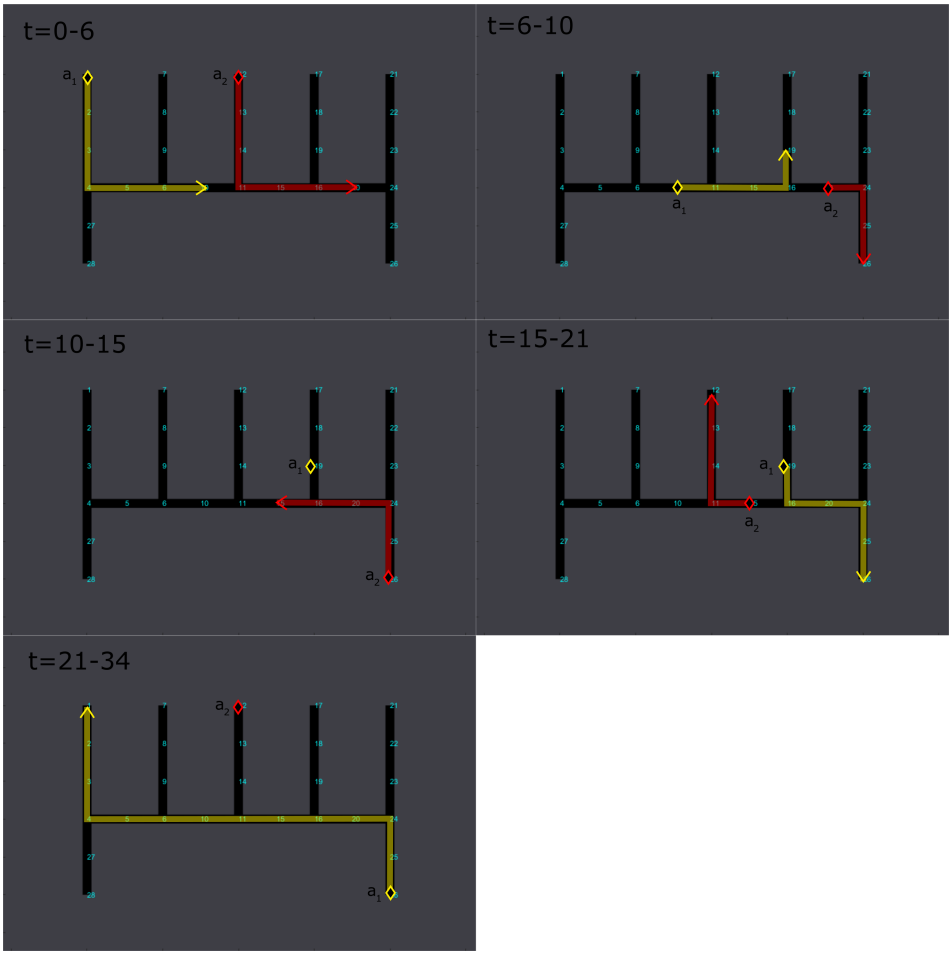

Figure 2.5: *An illustration of a method to coordinate two vehicles with a higher time efficiency compared to Figures 2.3, 2.4. Here, the standard paths are abandoned for better time utilization.*

Table 2.2: *Comparison of the a_1's time efficiency between different coordination behaviour*

	Current Solution	Behaviour 1	Behaviour 2
Time steps	42	38	34
Efficiency	64%	71% (+7%)	79%(+15%)

Remembering that this is a very specific case with arbitrarily determined dynamics and map structures, the obtained results are fairly preliminary. Adding more agents would increase complexity meaning a solution would not be as easy to

find through intuition alone. The mission of allowing two vehicles to complete one cycle deviates from total fleet productivity which is the true objective function of the problem. It is however a clear example of how more advanced path planning can increase time efficiency. Allowing vehicles to proceed through conflicted areas to find alternative spots to give way is an aspect which according to this evaluation should have a positive impact.

2.2.2 Evaluation of Alternative Goal States

Another aspect which ought to affect efficiency is each vehicle's choice of dump site, as some mines contain multiple dump sites. The current solution assigns each vehicle a specific dump site to visit, which in theory limits possibilities and in extent the efficiency. It is easy to imagine situations where deciding for a different dump site is time saving in comparison to always waiting for the assigned to vacate. Due to this, it would be interesting to consider giving vehicles the ability to freely choose which dump site to visit. Even if the nearest in most cases is the most efficient to use, allowing the decision to be based on a projected completion cost has interesting potential.

2.3 Initial Approach

With a vehicle behaviour outline it is possible to start discussing ideas. Early on, there were discussions about possibilities to improve on the static-zone method using a variety of potential approaches. One will be mentioned here to describe the trail of thought and motivate the final method of choice.

2.3.1 Smaller sections

A natural first question was since the large common zone was an issue, could it be partitioned into smaller sections? Subdividing the zones into smaller sections allows for more precise coordination based on the same logic of reservation. However, this imposes new problems which is the possibility of deadlocks where two vehicles with different destinations are facing each other and both waiting for the other to move, see Figure 2.6. To now simply reserve the zone a vehicle is currently in is no longer viable due to the possible lockups occurring. This means that finer zones require a higher level of coordination, but the approach has interesting potential.

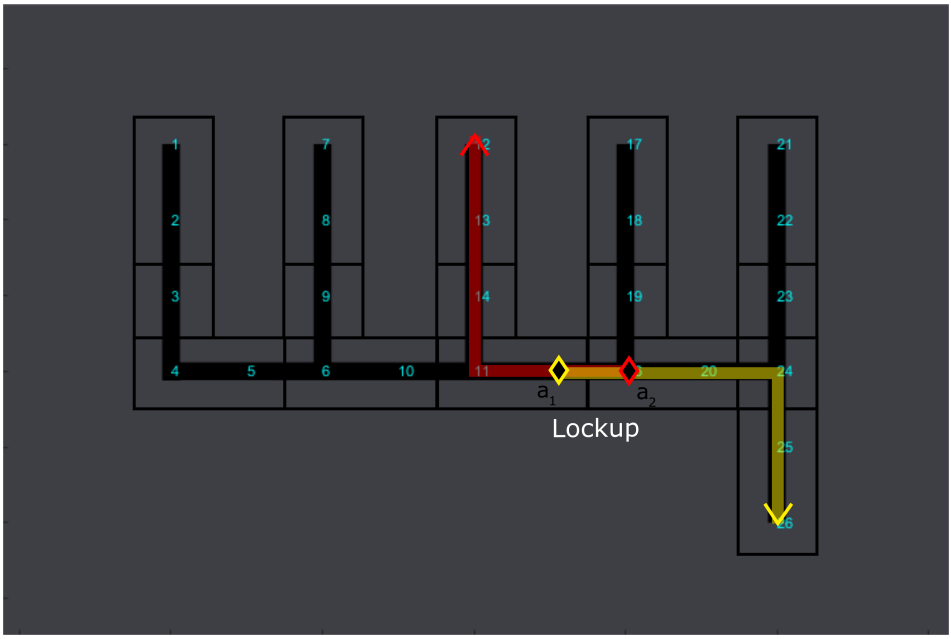

Figure 2.6: *Two agents trying to pass each other, hence are stuck in a deadlock.*

A functional application of a method which allows a higher level of coordination between the vehicles heavily increase method complexity. Finding paths which both deviate from the shortest route to save time and can vary between different dump sites requires a complete path finding algorithm. The fleet would have to work together to coordinate a joint plan for all vehicles. Problems such as these are covered in the field of motion planning and have been studied for decades. Due to the industrial development it has become increasingly relevant in recent years.

3

Motion Planning Theory

This chapter presents the theory behind motion planning along with some popular solutions to solve the problems.

3.1 General about Motion Planning

Motion planning is a field where the goal is to find a plan consisting of an action sequence which allows one or several agents to move from their origin states to their respective destination. Motion planning algorithms are often evaluated in terms of completeness level (whether a solution is guaranteed if one exists), optimality and complexity [5]. A valid solution is sought which optimises an objective function. This is usually related to either makespan: time spent for the agents to reach their goal vertices, or sum of action costs for the agents.

Multi-agent path planning has applications in fields spanning from warehouse management, airport towing, autonomous vehicles, robotics, digital entertainment, traffic control, and aviation [3] [9] [2].

Multi-agent collision-free path planning has been researched for decades, and is easy to describe yet difficult to solve. The difficulty originates from path coupling between agents, which results in a vast state space and large branching factors. Complete and optimal algorithms do not perform well for highly coupled problems except very small ones. The opposite can be seen for faster performing algorithms which can not guarantee solution optimality. This is all due to the state space enormity meaning evaluating all options is unreasonable [10].

Methods developed for motion planning problems generally require environment discretization. One approach is to in detail define impassable terrain and use a

model of vehicle dynamics to allow the agents to maneuver freely around the valid area. The method of terrain definition uses a quantitative approach to discretize the environment by assembling potential action sequences by random. This is called a rapidly-exploring random tree (RRT) [4] which for each agent iteratively evaluates the environment and constructs a graph tree to identify a path from the agent's start to its goal, see Figure 3.1. Agents are in this application free to choose their velocity and steering angle between at every state. The solution to such problems includes vehicle control inputs.

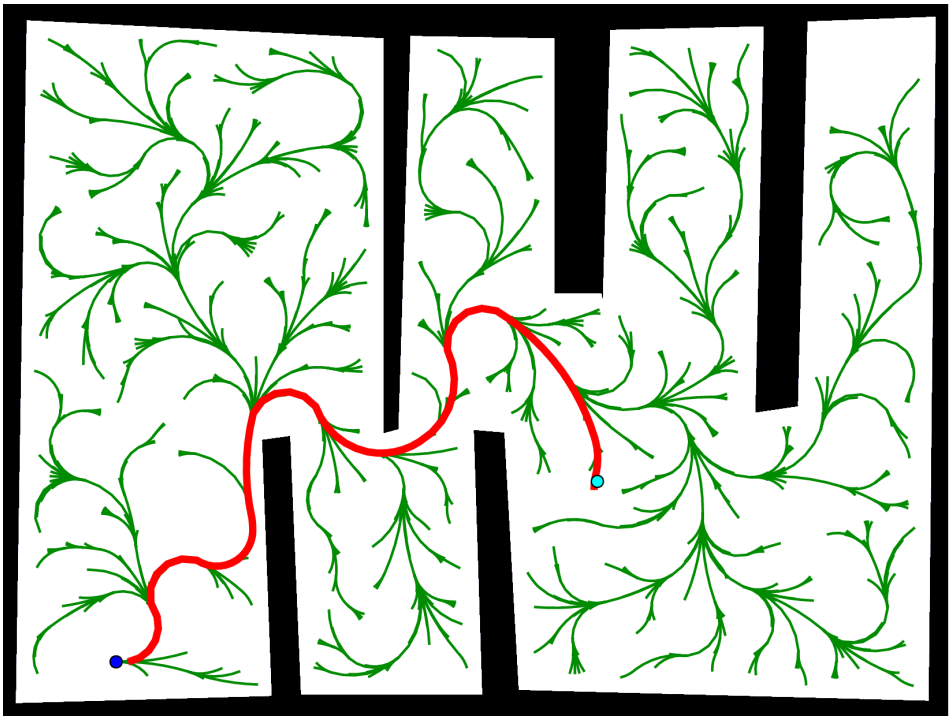

Figure 3.1: *Example of an RRT for a kinematically constrained agent. Spending more time generating a higher density tree would allow the agent to evaluate more options. This should lead to a shorter path being found.*

Instead of using the RRT with random exploration, another approach is qualitatively and systematically setting up a permanent graph as environment representation, see Figure 3.2. This method defines what states and actions are possible, instead of defining which are not. By controlling the graph structure, the model can be smaller compared to a randomly generated graph. The vertex resolution can be chosen to adequately represent the environment, increasing density for tricky sections whilst being more sparse during long straights.

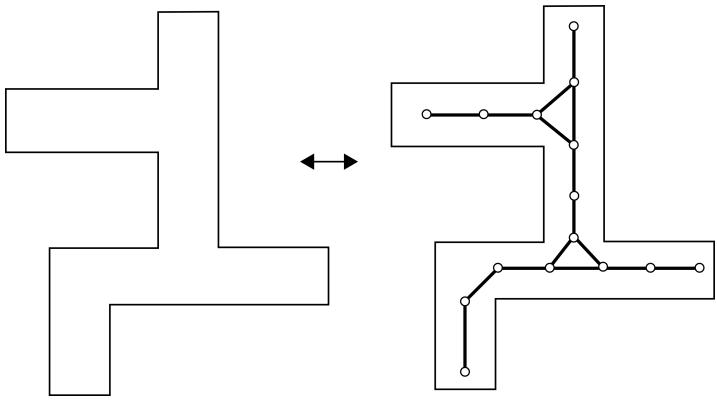

Figure 3.2: *An example on how an environment can be modeled into a graph consisting of vertices and edges.*

3.1.1 Differentially Constrained Agents

Capturing the dynamics of a vehicle is often done by considering the vehicle's state and kinematics when generating the RRT. This produces a tree consisting of states all reachable from the starting position, which produces smooth branching as seen in Figure 3.1.

To produce a general graph for the entire map not based on random generation (as in the RRT) is often done by assembling what is called a state lattice. A state lattice consists of a complete and uniform grid of nodes interconnected with edges fulfilling the kinematic constraints of the agents. A lattice example with four viable state orientations can be seen in Figure 3.3.

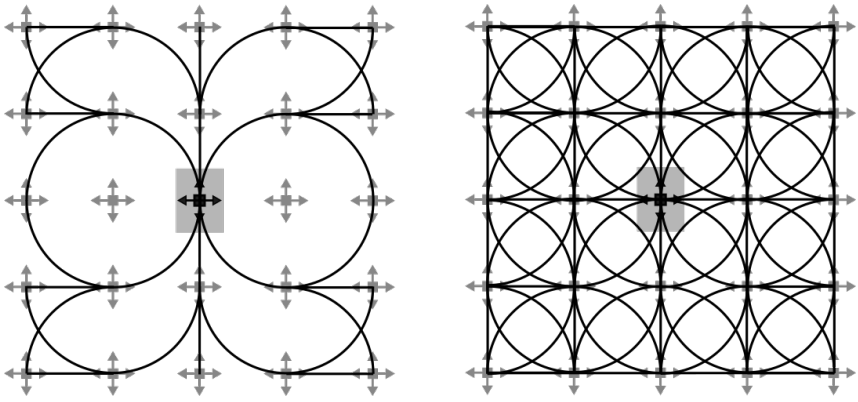

Figure 3.3: *An example of a state lattice where each state is limited to $\pi/2$ angle intervals.*

3.1.2 Graph Setup Decision

In this thesis the systematic graph method has been chosen for two main reasons: limiting environment and existing autonomy system. Firstly, due to the narrow environment the agents behave more like short trams on railway than free-moving vehicles. Producing an RRT would most likely be time wasted on re-identifying a path on a graph which could be closely predefined anyway. One interesting aspect with this case is that the mine is more or less designed to work as a network for the vehicles to move through.

The real vehicles already use carefully produced action sequences to follow determined paths within the mine. To then create a solver which defines its own plan of actions would heavily overlap with existing methods. However, if the situation was different and no current method for autonomous operation was implemented, I would not be so quick to rule out an RRT-based method to produce both paths and controlling inputs. This could be valuable to examine closer for new projects with other circumstances.

3.2 MAPF Definition and Terminology

Deciding how multiple agents are to traverse a graph is known as a multi-agent path finding (MAPF) problem and will here be defined.

The MAPF problem is specified by a graph $G = \langle V, E \rangle$ in which $V = \{v_i\}$ is the set of vertices and $E = \{v_i, v_j\}$ is the set of edges, see Figure 3.4. Located on the graph are a set of k agents $a_1 \dots a_k$ where each agent a_i has a start position $s_i \in V$ and a goal position $g_i \in V$. Time is here discretized into time steps. In every time step, each agent will perform a single action. The action space for each agent includes either movement to a neighboring node, or waiting at its current location. A sequence of actions for one agent is denoted a path, whereas the full set of paths is denoted as a plan. A valid solution to a MAPF problem is a plan which allows each agent to move from its start to its goal state without colliding with the other agents.

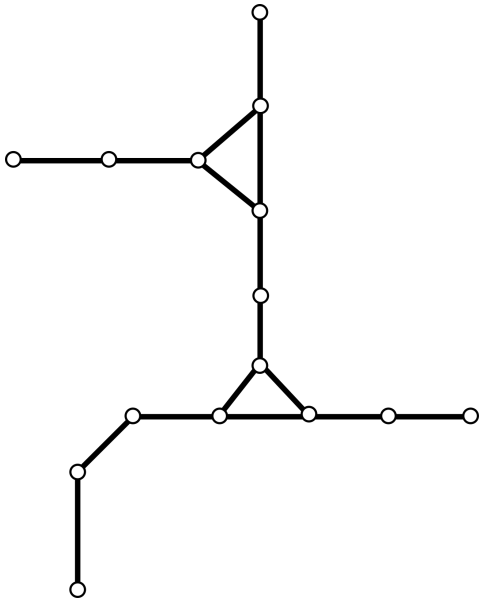

Figure 3.4: *An illustration of a graph. Vertices can be seen as white circles connected with the black edges.*

To achieve a collision-free solution, MAPF solvers use the notion of conflicts between the agents. The definition of a conflict varies from case to case, but we list the predominant seen in MAPF research [9]:

- **Vertex conflict** means more than one agents enters a single vertex at the same time step.
- **Swap conflict** means two agents located at neighboring nodes take each others spots during a single time step.
- **Following conflict** occurs when an agent plans to occupy a vertex at time step t+1 which was occupied by another agent at time step t.
- **Cyclic conflict** occurs by closing a loop of a number of following conflicts meaning three or more agents can cycle into each others positions through synchronous movement.

The last two: following and cyclic conflicts, can for single vertex agents be carried out without an actual collision occurring. To cycle back to what was mentioned as a valid solution: for the solution to be valid, no vertex or swap conflicts are allowed in the joint plan.

3.2.1 Centralized vs Distributed

Most MAPF algorithms can be separated into two categories: centralized and distributed (decoupled).

In a distributed setting, agents are controlled and planned through their own individual computational power where communication is managed through e.g. message passing. Agents' paths are here planned individually and combined to avoid collisions. These methods usually have lower complexity and greater scalability than a coupled planner [5].

The centralized category is handled through a single computational unit which coordinates a plan and delivers instructions to the agents. Here the robots paths are planned simultaneously. Coupled algorithms can achieve completeness and optimality if paired with a complete search method [5]. They are however limited due to a quickly expanding configuration space with additional agents adding new dimensions.

This thesis will only consider the distributed approach due to optimality not being of high priority, meaning an approach of lower complexity can be taken.

3.2.2 Example of a MAPF Problem

In Figure 3.5, a 2-agent MAPF problem is depicted. Each agent must find a path through the network to their respective goal. Agent a_1 is to move from s_1 to g_1 whilst agent a_2 is to move from s_2 to g_2. Trivially, the shortest path for a_1 is $\{v_1, v_3, v_5, v_6\}$ and similarly for a_2: $\{v_2, v_4, v_5, v_7\}$. As both paths coincide at v_5 at time step t_2, one agent will be required to await the other to pass through first. Giving all actions a cost of 1 results in an optimal solution with cost 7.

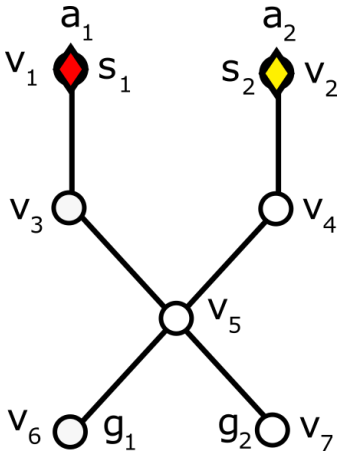

Figure 3.5: *An example of a 2-agent MAPF problem. The optimal solution requires either a_1 or a_2 to wait during one time step to let the other pass v_5 first.*

3.3 Research Overview

Since a large part of this thesis has an investigative nature, a study of applicable methods has been made to help determine which directions the solution can be taken. What can be said about MAPF solvers is that the best choice varies from case to case. All have their strengths but also their weaknesses. These methods can also be tweaked to better fit the problem they are applied to solve.

Much of the research use the same simplifications where time is discretized into time steps, the duration of every action is one time step, and in every time step each agent occupies exactly one single location. These limitations enables focus on the core issue of coordination.

3.3.1 A*

A* (pronounced A-star) is a single agent graph traversal algorithm and has been widely used due to it's optimality and completeness [6]. Here, by exploring vertex by vertex, the algorithm searches for the optimal plan for the agent to reach the goal. To shorten the process it uses a prioritized search list where a heuristic function helps to determine a directed guess as to which vertices should be explored next. By using a heuristic function, parts of the graph unlikely to be part of the final plan can be avoided exploring which makes the solver considerably faster. If the algorithm finds alternative ways to reach the same point it will choose the cheapest option, ensuring the solution being optimal once found. A* has been a useful tool for many applications and works well as a basis for applying additions required for more complex problems.

3.3.2 Cooperative A*

One alternation has been to allow multiple agents moving through the same graph and avoiding collisions whilst doing so. This is an algorithm named Cooperative A* (CA*) [7]. The introduction of multiple agents complicates the problem substantially. The approach is deconstructed into a series of standard single search A* searches. Each individual search is performed and stored in a reservation table representing all spatiotemporal (space and time) dimensions. As each agent finds its way through the environment, it reserves the planned vertices in the table, so that subsequent agents plan their paths with account taken to the proceeding agents' movements. This method has however proven itself inadequate at times. Since the first agents plans are done without consideration to the others, it is not uncommon that deadlocks and bottlenecks occurring. The problems occur increasingly frequent with increasing agent-per-vertex ratio [7].

3.3.3 Local Repair A*

Similar to cooperative A*, the local repair A* (LCA*) algorithm is another attempt at making a multi-agent A* adaptation. Here, instead of giving all agents a picking order at the graph vertexes, they each plan their individual paths without

consideration to each other. By then identifying occurring conflicts, the partaking agents' paths are from that point rerouted to locally resolve the conflict. This algorithm has its uses but the results are very similar to the shortcomings of CA*, with recurring deadlocks, bottlenecks and cyclic repetition [7].

3.3.4 Hierarchical Cooperative A*

Hierarchical Cooperative A* (HCA*) is an offline method where individual paths are calculated for each agent according to a predefined order [7]. Each calculated path is stored in a reservation table to which upcoming agent searches must heed to/abide by. Any previously reserved vertices or edges must not be used which assures no conflicts occur. It has been noticed that the success of the HCA* method depends greatly on the planning order, see Figure 3.6.

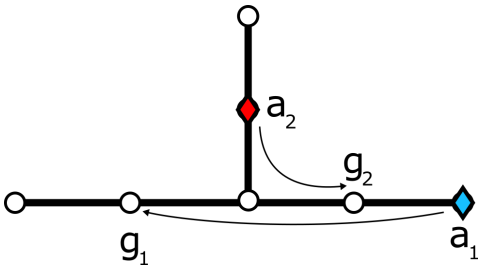

Figure 3.6: *An example where planning order between two agents are crucial to the solution of the problem. Allowing a_2 to plan first results in a_1 being unable to reach its goal g_1. The opposite allows both to find their respective goals.*

This issue with planning order was approached in the extension called Windowed Hierarchical Cooperative A*.

3.3.5 Windowed Hierarchical Cooperative A*

Windowed Hierarchical Cooperative A* (WHCA*) is an effort to mitigate some of the issues of the HCA* method. The search is now turned online by introducing an operation cycle between planning and moving phases. WHCA* adds two parameters to the search method: W for the planning phase and K for the move phase. W depicts the coordination window length meaning how far ahead the paths are to be coordinated to avoid conflict. K controls how far the move phase steps into the produced plan. What researchers [7] have found is that moving through the entire plan can result in unfavorable positioning as the next planning phase commences, [7]. What instead has shown to be effective is stepping only partway through the plan meaning giving K < W. After the move phase has completed the K steps of the plan, the reservation table is cleared, agent order is shuffled, plans are reset and the cycle starts over. The relation between K and

W could be determined arbitrarily, but Silver in [7] only used one value for K: $K = W/2$.

Shorter planning phases bridged by move phases allows the planning order to be rotated to resolve order issues. Consider the situation depicted in Figure 3.6, but the agents after reaching the goal are to return to their initial positions. Reaching the goal requires a_1 having priority, but returning can only be completed by transferring priority to a_2. This type of fluent prioritization can be achieved with WHCA*.

This version of HCA* has shown clear improvements compared to the original algorithm but there are still some issues. With W and K as fixed parameters there can occur situations where agents position themselves inefficiently if conflicts lies just beyond the parameter values.

3.3.6 Conflict-Oriented Windowed Hierarchical Cooperative A*

Efforts have been made to address the drawbacks of WHCA* by placing flexible planning windows around conflicts. This method is called Conflict-Oriented Windowed Hierarchical Cooperative A* (CO-WHCA*) [2]. This method has implemented a strategy where past plans denoted leftovers are saved through the next cycle, so that re-planning can be done more efficiently. According to [2], the CO-WHCA* algorithm outperforms WHCA* on all tests carried out.

3.3.7 Complete and Scalable Multi-Robot Planning in Tunnel Environments

Peasgood et al. [5] has taken an approach which distances itself from the mentioned approaches. The method specializes on narrow tunnel graphs where graph theory is used to execute a phase sequence of operations to control the agents. The method analyzes the graph to identify what in graph theory is called leaves, basically meaning vertices with only a single neighbor such as that of an end of a tunnel. By starting out with storing all agents in these leaf-nodes, the map opens up to find simple single-agent searches. Agents are then prioritized according to goal depth from a set graph root, and sequentially moved there. The method has full completeness for up to $L - 1$ agents where L is the number of leaves in the graph, which is significant. The downside however is that it for all fleet sizes delivers inferior optimality compared to the popular decoupled planners, [5].

3.4 Research Analysis

The mine's coordination problem differs some from most encountered studies. The agents' missions are cycles where they usually eventually return to the same area after visiting a crusher. The map structure is also similar to a tree graph. A tree is in graph theory defined as a structure where there is exactly one solution to traverse between any two points without revisiting the same node more than

once. This produces choke points throughout the graph which is also an issue when dealing with multi agent problems.

One of the studied papers covered a very similar map structure [5]. They approached the problem by utilizing graph end points, called leaves, to store the agents to clear up the choke points. As long as there were more leaves than agents, any agent could find a path regardless of planning order. This is an interesting approach to solve problems with a high agent-to-vertex ratio, and can be used to resolve difficult deadlocks. However issues occur when including a different agent dynamics compared to the research. As agents in the mine are on constant missions back and forth from loading and dumping, the agents goal points change constantly. Each new goal would most likely require a restart of the global process of first moving agents to the leaves to enable the new routes to be found. This would therefore be highly inefficient for normal operation.

Considering the decoupled A* algorithms instead. They are all based on sequential planning with the differences between them being how planning order and reservation is carried out. The research suggests that the best performing method is CO-WHCA*, but it is possibly unnecessarily intricate for this problem.

By allowing an agent to be the sole operator within a specific drift, and letting the mission include the entire dumping cycle, the problem itself becomes much easier to solve. This way, no agents' end points are conflicting, as the dump site is just a waypoint in the mission. For this specific type of operation, a simple CA* would be sufficient for most cases, even for any agent planning order. However, an agent positioned far from its personal drift zone at the algorithm initiation could be problematic as it runs out of room when other agents collapse in on it. Due to this, extending the CA* algorithm with additional modules able to handle more complex situations is the plan.

4

System and Environment Model

Applying the path planning algorithms requires modeling both the environment and the vehicles operating within. By putting priority on calculation times and algorithm simplicity the models have been developed to be simple.

4.1 Map Model

As described in Chapter 3, the approach chosen for environment discretisation is to manually setup a permanent graph. It will consist of vertices representing possible agent states and edges defining possible vertex transitions. Choosing how many vertices are to represent the map is a balance act between two aspects: accuracy and complexity.

In theory, the graph can be very sparse but issues could occur for vehicle space reservation which is done by marking vertices occupied by the agent in a space-time table. This will be further covered in later sections but to ensure the entire vehicle is protected, the reservation extends an extra vertex neighbour in each direction. For a sparse graph, the method will reserve a much larger space than necessary. Increasing the node resolution allows the graph reservation to more accurately represent the vehicle. However, as the path planner explores node by node, having a higher graph density implies longer calculation times.

4.2 Vehicle Model

Since the problem includes nonholonomic agents, consideration has to be taken to their size and orientation to make sure the methods work as intended. The vehicle's bucket will be considered to be placed at the front of the vehicle.

4.2.1 Vehicle Size

To enable a collision-free operation, the vehicles' sizes must be accounted for. A way to do this is using a model which approximates the vehicle shape and then examine overlap between the graph and the vehicle model [3]. In an effort to keep the required calculations to a minimum, a simple model is the goal.

In the current coordination method developed at Combitech, the vehicles are already modeled to the extent that their positions, acceleration, and articulation is captured. The position is however only modeled as a single point since additional information was redundant up until now. The vehicle point, which describes its position, lies in the centre of the front axle denoted p, see Figure 4.1.

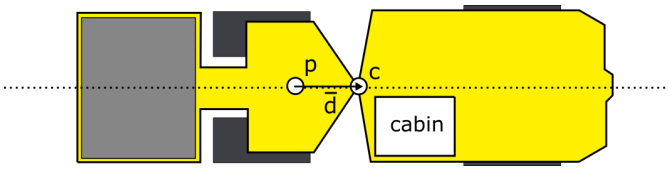

Figure 4.1: *Illustration of the graphical vehicle model.*

The geometrical centre of the vehicle lies behind p, approximately in the point denoted c in Figure 4.1. Point c can be found combining the known point p and translating a set distance $norm(\overline{pc})$ backwards.

Allow c to be the centre of a rectangle with a set width and height, fully enclosing the vehicle. Consider this rectangle the vehicle's bounding box, or footprint denoted FP. Due to articulated steering, the vehicle's shape is not fixed. To capture the variable vehicle shape could either be handled by expanding the fixed footprint for a worst case scenario, or update the footprint as a function of the pivot angle α. The simpler approach would be to assign fixed footprint and then as a safety precaution expand for extra margin, see Figure 4.2. The requirement for safe operation is that the actual vehicle area A_v as viewed from above is modeled using the footprint approximation such that

$$A_v \subseteq FP \tag{4.1}$$

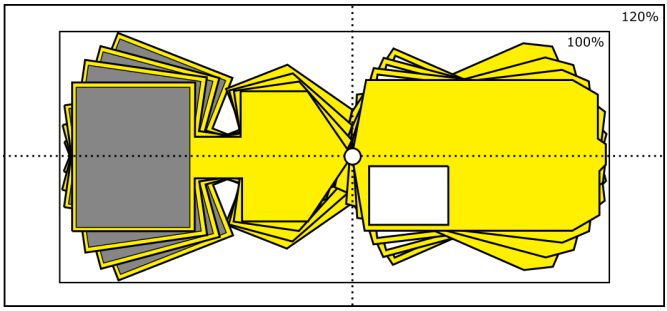

Figure 4.2: *Illustration of the agent footprint.*

Even with vehicles having expanded footprints for extra safety, there can still occur collisions if only the vertices within the footprint are reserved, see Figure 4.3. As two vehicles extend towards each other past the outer most vertex they contact, it is possible that a collision will occur. Just as the footprint must extend past the vehicle itself, the space reservation (SR) must extend past the entire footprint. This is required to ensure generated plans are collision free .

$$FP \subseteq SR \tag{4.2}$$

A method to avoid this problem is to not only reserve the nodes within the footprint, but also their neighboring nodes. The reservation will now enclose the entire footprint, see Figure 4.4.

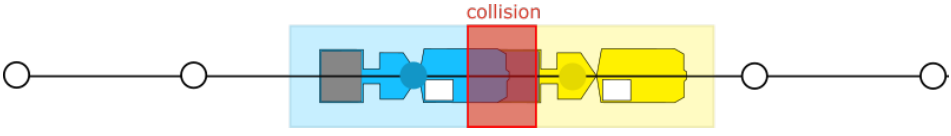

Figure 4.3: *Collisions can occur if the graph reservation is not done according to 4.2.*

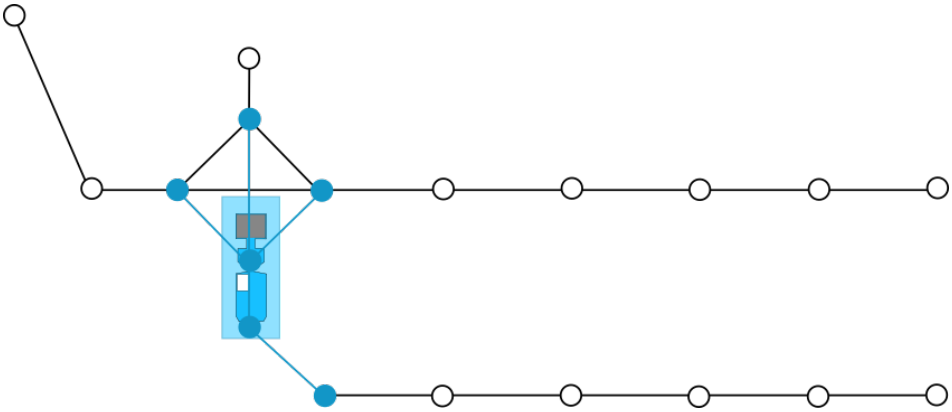

Figure 4.4: *Example of a set of reserved nodes for a vehicle entering an intersection effectively prohibiting other vehicles entering contact distance.*

Using this method, agents will now leave at least one edge gap between themselves, seen in Figure 4.5. Tuning how close the vehicles get to each other can be done by either reducing the footprint size margin, or increasing the vertex density in the graph.

It is however worth noting the method is not flawless, graphs could be defined where this method fails such as in Figure 4.6. The difference between Figure 4.4 and 4.6 is the level of vertex connectivity at the intersection. Since the respective vertices where the two agents are positioned are not connected through any common neighbors, nothing allows the model to identify their incoming collision even though their plans does not share nodes. This case is very specific but highlights a weakness in the method worth noting.

Figure 4.5: *Illustration of the reserved vertices by considering edges touching the footprint.*

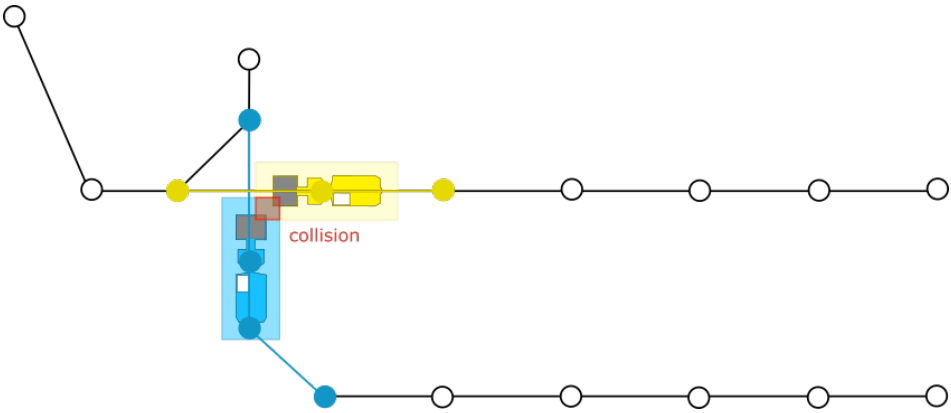

Figure 4.6: *Illustration of a case where the reservation method fails to protect the vehicles from collision. Requires an improbable graph definition but still worth noting if such environments were to be modeled.*

4.2.2 Vehicle Orientation

Each goal state in the operation is, other than a position, also defined by an orientation. Some goal states require arrival in reverse and others facing forwards, see Figure 4.7. The planner must therefore find a way which ensures the vehicle reaches the goal state orientated correctly. To find a solution which fulfills this constraint, the vehicle orientation must be included in the graph exploration.

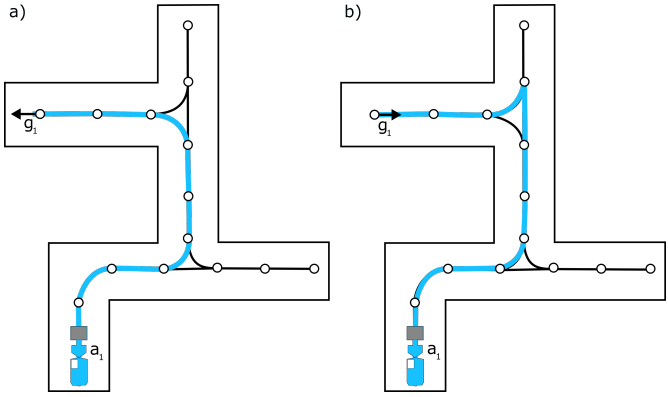

Figure 4.7: *Illustration of goal orientation requiring different solutions. a) Arriving to g_1 facing left can be achieved by maintaining forward drive throughout the graph. b) Arriving to g_1 facing right requires performing a two-point turn at one of two viable intersections before finishing the remaining route in reverse.*

Let a new set of graph parameters to be introduced. Each vehicle state is limited

to two valid and opposite directions of orientation, both along one line. Allow one of the two directions (arbitrarily chosen) to be considered the direction of each vertex in the entire set. Let a vector k be placed with its origin at each node, directed along the nodes' direction at each node with angle θ.

Let each agent a_i carry a binary parameter denoted $d(a_i)$, which keeps track of the agent's orientation in relation to the current vertex's direction, see Figure 4.8. By combining $d(a_i)$ with θ the agent's orientation at any vertex can be determined, but how is a change of orientation identified during edge traversals?

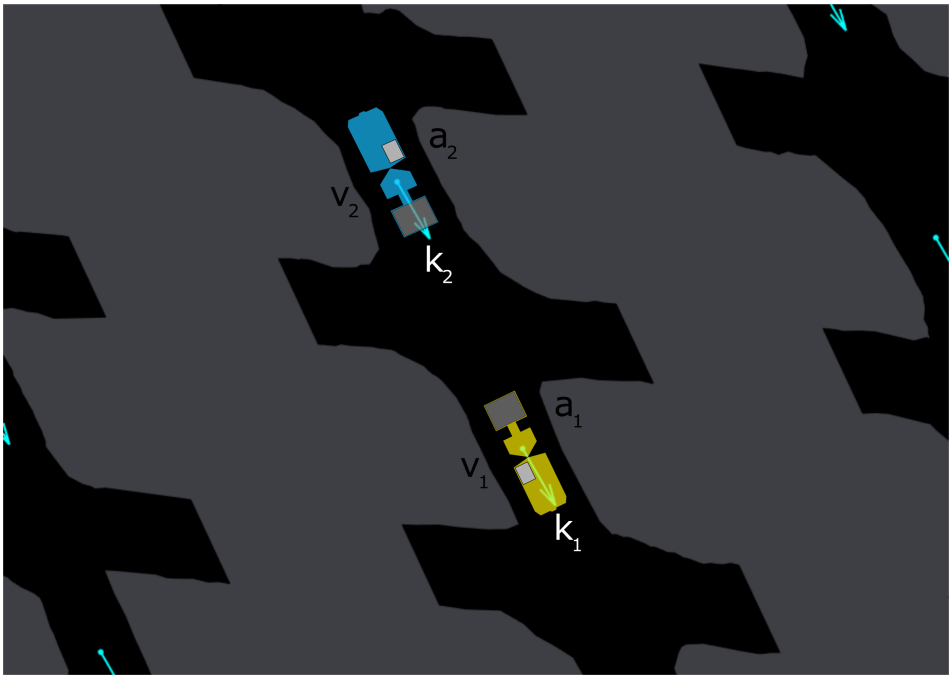

Figure 4.8: *An illustration of vehicle orientation. Vehicle v_2 is oriented along its vertex v_2's vector k_2 meaning its orientation state $d(a_2) = 0$. Vehicle v_1 has the opposite meaning $d(a_1) = 1$.*

Capturing changes in orientation can be done by assigning each edge a binary property describing the relative directional relationship between the nodes it connects, denoted f, see Figure 4.9. Consider an agent a at vertex v_1 with $d(a) = 0$, implying the vehicle is oriented along k_1. It will traverse edge e_1 to vertex v_2 which has the directional vector k_2 with opposite angle that of k_1. The traversal results in the agent going from $d(a) = 0$ at v_1 to $d(a) = 1$ at v_2. This is permanent property of the edge, meaning each time it is traversed, d of the traversing agent is changed. This property is described by setting the edge property $f_1 = 1$.

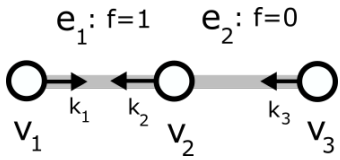

Figure 4.9: *A graph depiction of vertices including directions and the edge property describing the transfer.*

Since f is a fixed property of the graph, identifying the values can be done offline. A method to automatically identify the values of all edge properties is by looking at the angle differences between the connected nodes. If $\cos(\alpha)$ is positive, in most cases, edge property is $f = 0$, see Equation (4.3). However this requires that there are sufficient nodes representing the map structure. If a corner can be taken, but the graph depicts it as $\pi/2$ or sharper, it will incorrectly be interpreted as a change of orientation.

$$\cos(\alpha) = \frac{k_1 \cdot k_2}{|k_1| \cdot |k_2|} \tag{4.3}$$

Looking at Figure 4.10, one can see a situation where an agent traverses two edges in sequence which results in the vehicle going from $d = 1$ at the start of the sequence to $d = 0$ at the end. Traversal along the edge between v_1 and v_2 flips d, whereas d is maintained over the second edge.

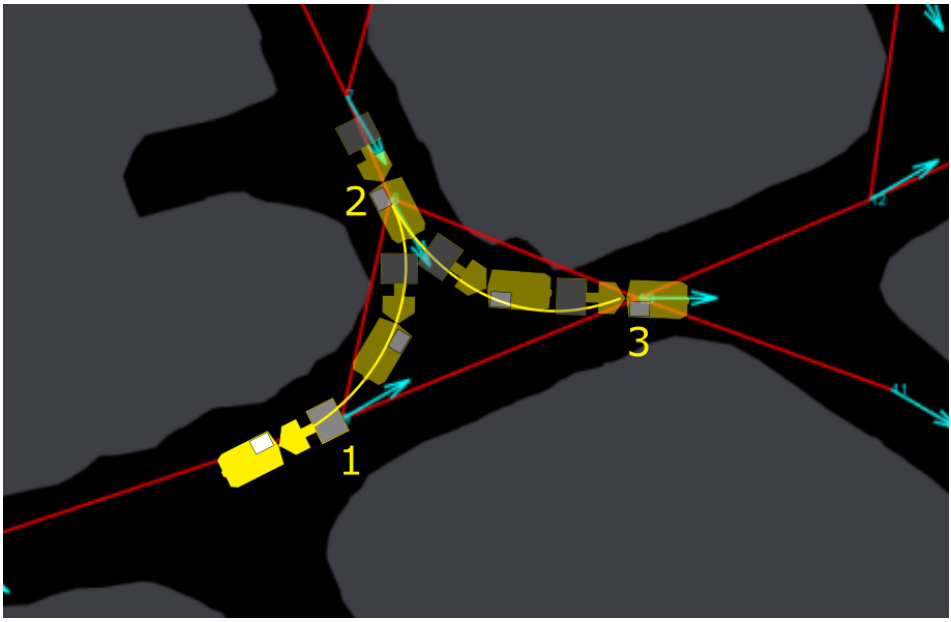

Figure 4.10: *An action sequence resulting in a vehicle switching direction of travel.*

This method ignores agents' direction of travel, whether movement is forwards or backwards. It only considers in which direction the vehicles are pointing which allows constraint-fulfilling paths to be found consistently.

5
Path Planner

The path planner part in this thesis involves the graph exploration to identify the best path to complete the agent's mission. As searches are done individually for each agent, this section describes the work that goes into finding a path for a single agent in a multi-agent setting. Here, both method and solutions are presented. Coordination through a reservation table is covered last in the chapter.

5.1 Graph Exploration Including Time Dimension

This section handles graph exploration and is an expansion on the MAPF-definition in Chapter 3. The method is based on the common A* algorithm.

The standard-A* graph search initiates at a specific vertex of a graph, where it aims to find the best path to the given goal vertex. Alternative actions and paths are evaluated by cost which is defined to represent optimality of the objective function. Evaluating a visited vertex's actions to transition to neighboring vertices is called exploring. The newly discovered neighboring vertices (*children* vertices) are placed in order into a queue list for upcoming explorations, while the vertex itself (*parent* vertex) is removed from the queue and placed in a list of visited nodes. Once the goal is found or the queue becomes empty, the search is stopped.

For the search to capture a dynamically changing environment, a static graph representation is non-sufficient. To overcome this problem, one can consider the time-dimension as an additional space dimension to the graph. The original graph can be considered a space-graph whereas the expanded graph can be considered the space-time graph. Vertices in the space-time graph will be referred to as st-points.

The standard method to expand the search into the space-time dimension for 2D grid maps is stacking layers of the space-graph and connecting actions from one time step to the next [8]. All actions from a vertex v at time step t will connect to the neighbors of v at time step $t + 1$, see Figure 5.1. The waiting action is simply represented by connecting the same vertex between two time steps. To make the agents conform to the laws of time, the graph is directed such that traversal backwards in time is prohibited. Reaching an st-point already found through another neighbor, the path with the lowest cost will be the one chosen and stored.

By including the time dimension, the state space grows by the factor of time steps within the set planning horizon. This can for some cases risk making computation times very long. With a larger state space to evaluate, the heuristic function estimating remaining cost is of increased importance.

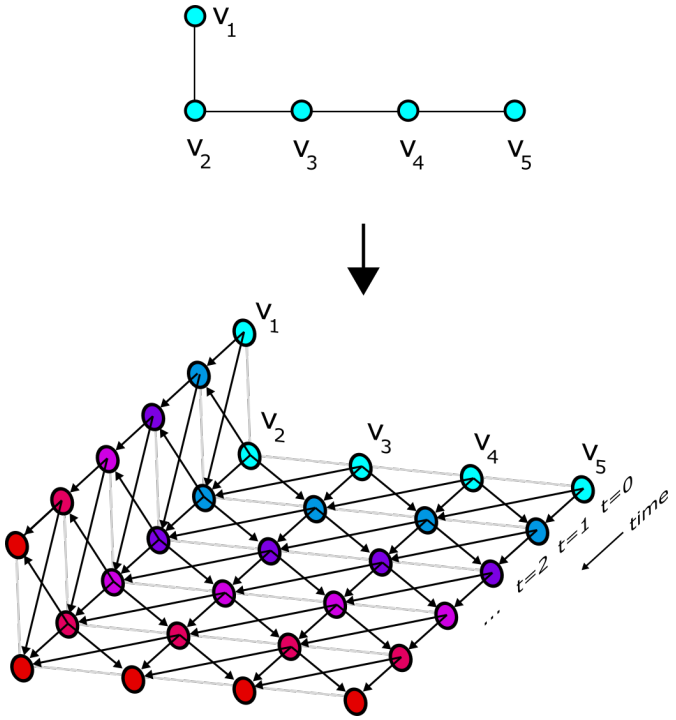

Figure 5.1: *An illustration of how a graph can be expanded to include time. Since movement backwards in time is unavailable, all edges are required to be directional as seen in the illustration.*

5.2 Waypoint Planner

The conventional A* path planner aims to identify paths for one or several agents to reach their respective goals. This project's operation mostly consists of loading-

dumping cycles meaning visiting two goal vertices in sequence. First, the vehicle travels to the dump site and once reached, it starts looking for its way back to a new loading site, see Figure 5.2.

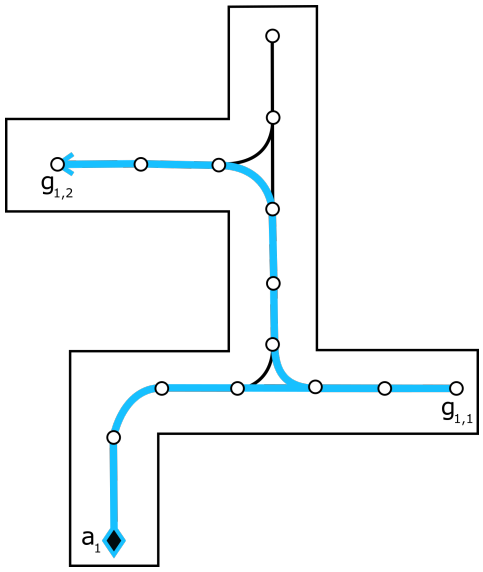

Figure 5.2: *Example of a multiple goal plan where agent a_1 identifies a path connecting its goals $g_{1,1}$ and $g_{1,2}$ in order.*

The initial implementation in testing consisted of first running a planner to reach the goal, and afterwards generating the return plan from a fresh search. This resulted in some issues with agents going blindly into the busy dump sites without a plan on how to exit, producing lockups as other agents approached. What instead appeared to be more efficient was to plan the entire cycle at once, by using a waypoint planner.

One can also imagine that reaching a specific dump site as fast as possible might not be part of the best solution when looking over the entire cycle. If it means traffic on the return trip it might have been better to choose a different dump site to begin with. By running the entire cycle at once, the planner can take decisions with a better planning horizon which hopefully is advantageous in terms of optimality.

The waypoint planner works in such a way that each agent is assigned a list of goal nodes to visit in order. For a normal loading-dumping cycle it would be a dumping site followed by a new loading site. The list of st-points in the exploration queue individually carries their own copy of the goal lists. During exploration, the children inherits the goal list from the parent. If the planner steps into the goal node, that goal gets removed from that st-point's goal list, making it aim for the next goal in the mission. Any explorations based on the branch that

touched the first goal now inherits the updated goal list. This continues until a branch finds all goals in the mission.

5.3 Action Cost Model

The action cost model depicts how different agent behaviour is prioritized, making it the module for algorithm decision making. By altering the values of these parameters, the agents will behave differently. Action costs are evaluated throughout the graph search process. Consider an agent starting at x_s. The goal is to find the lowest cost solution to reach x_g. At each step of the search, the accumulative cost to reach a vertex x' from the x_s is denoted $g(x_s, x')$, where an estimation of the remaining cost to complete the path uses the heuristic function $h(x', x_g)$. The sum of g and h is the sequence cost $f(x_s, x_g)$:

$$f(x_s, x_g) = g(x_s, x') + h(x', x_g). \tag{5.1}$$

As the search finds the goal: $x' = x_g$, $h = 0$, and $f = g$.

Different actions have different costs C:

- Cost of waiting at a vertex x: $C(x)$
- Cost of traversing between x_1 and x_2: $C(x_1, x_2)$

In the current implementation, the cost values were balanced to induce the preferred behaviour. Each moving action were given the same cost of 1. For situations where progress is inhibited by a blocking agent, the trapped agent will need to decide how to spend its time. During testing, agents were observed pacing back and forth while waiting since they had no preference to remain stationary compared to moving. To give them an incentive to save fuel, the waiting cost was given a slightly lower value than moving (0.5). This makes the agents move as close to the goal as possible and there await the blockage to clear.

This cost model can be further expanded by including weight parameters to include other properties of interest. Action weights w could even be a product of a number of vehicle properties so that individual action costs would vary. Suggestion to this for an agent a performing a moving action:

$$g(x_2) = g(x_1) + w(a) \cdot C(x_1, x_2), \tag{5.2}$$

where $w(a)$ can be a function between several properties of agent a such as:

- Bucket status affecting action costs. A full bucket has a noticeable impact on vehicle mass, increasing wear and fuel usage during acceleration.
- A cost on acceleration would result in an agent's speed could enable smoother

movements if the path ahead is occupied. This could make the mining operation to appear more fluent and at the same time reduce fuel and wear.

These additional weights to prioritise between vehicles has not been evaluated in the thesis and will be left for future work.

5.4 Heuristic Function

In A* searches, the use of a priority list gives the ability to prioritize exploration of states likely to be part of the final solution. This is done by estimating each states remaining cost to finish the path. The approximation, even if rough can help significantly reduce computational times. One aspect required for the solution to be optimal is for the function to be admissible. This means the heuristic must not overestimate the remaining cost to finish. The heuristic function can be adapted for each case to better approximate remaining cost. However, there are a few reoccurring methods worth evaluating.

Euclidean distance and *Manhattan distance* functions are mathematically similar, computationally simple to use and can be highly effective. The central aspect of them is that they approximate the remaining distance without consideration to obstacles. The success is dependant on the level of obstruction the map has. The agent will gravitate towards nodes closest to the goal, which is great for open environments but poor for the opposite. An simple example of a situation poorly suited for an euclidean distance model can be seen in Figure 5.3 where the solution requires movement away from the goal. However, methods better approximating distances requires knowledge of the environment. For problems with known environment a true-distance heuristic becomes possible where a table with precalculated distances between graph points can be consulted. A solver using this will always know which node to prioritize, finding the optimal route in the minimum amount of possible time steps. One aspect the function will not include is other agents obstructing the path.

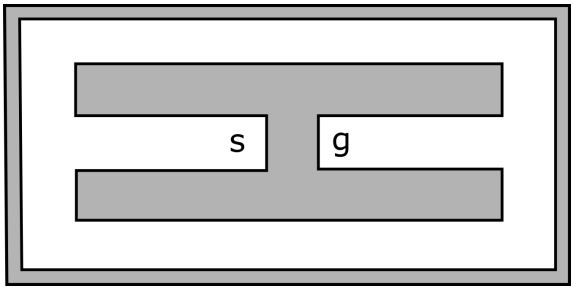

Figure 5.3: *Example of an environment poorly suited for a euclidean distance heuristic.*

5.5 Multiple Goal State Decision Making

A valuable behaviour discussed in Chapter 2 was allowing agents to make decisions as to which dumping site they visit. This is expected to improve the efficiency of the vehicles. To implement this, each agent's path finding search would be completed for multiple dump sites. Once path search for each dump site has been completed, the local cost of each operation sequence would be compared allowing the lowest cost option to be chosen. For simplicity's sake in this algorithm, each agent will evaluate every goal site. It could, however, be limited to only evaluating the closest few to reduce calculation times with suspected low negative impact on optimality due to the others being increasingly unlikely to be chosen. This is something that can be further evaluated if calculation times are a problem. Any searches for goal alternatives that fail are ignored, and the remaining successful plans are ordered in ascending cost order. If none of the searches were successful, the agent is stuck which requires other agents to transfer their priority for the situation to resolve.

5.6 Reservation Table

As briefly mentioned in Chapter 3 as it is used in several popular path planning algorithms, the reservation table is a popular method to coordinate agents. It consists of a table representing all vertices in the graph over the entire planning horizon. Once an agent's path has been generated, the path will be placed into the reservation table. All cells are initiated containing a value representing "cell available", here a "0". As a plan is entered, the cells are given a value corresponding to the planning agent's id. During node exploration in the planning phase, a function compares all available actions to the reservation table entries for the relevant time steps. If an st-point is flagged 'reserved' meaning > 0, any actions leading other agents to that specific vertex are made unavailable. To enable smarter conflict resolving, the identification values of any encountered agents during the search are stored and exported upon search failure. If the agent fails, it will be because of the agents encountered during search are obstructing the way. To clear the obstructing agent's plan and rotating their planning order often resolves such issues according to testing.

5.7 Rotating Agents

The planner must find a solution which fulfills the constraint that the arrival orientation of the agent must be in accordance to the goal state. If not, the solution is invalid. Due to the project's time limit, such a constraint has not been implemented into the algorithm but it has been given some thought worth mentioning.

The easiest solution is to simply have an extra constraint for goal arrival meaning the goal is only reached if in the correct orientation. This is a quantitative method where the planner attempts to reach the goal in any orientation. If the orientation

state does not match the goal, the search will continue until the vehicle has found a path which turns the vehicle around.

What I expect from the vehicle behaviour is that it will find the turning point nearest to the goal state. This could, however, be inefficient since intuitively, the best option would be to perform the two-point turn in a place where it would not inhibit traffic flow which is highest close to the crushers. This would minimize the risk of path coupling with other agents executing their own paths which is expected to be advantageous to global cost minimization. This is an additional reason to include the orientation state into the planning heuristic to allow the search to purposely find a path with the necessary turn performed far from the goal.

5.8 Planning Dynamics as the Fleet Changes

The vehicles operation cycles are naturally separated by the manual loading process. It would be inaccurate to make individual movement plans stretching past the manual loading which according to company data can vary enough to make attempts at predictions pointless. Another alternative is to perform a vehicle checkout into manual mode as the vehicle finishes its plan. Once the loading process is complete, the vehicle would then be reentered into the autonomous fleet.

Reentering the newly loaded agent into the fleet would change the system circumstances meaning that an agent requires planning. What should happen here with the rest of the planned fleet's paths? Allowing all agents with plans to continue moving gives the newly added agent the lowest priority to plan. This would make any attempts at agent prioritization redundant and in practice work as a FIFO (first-in-first-out) queue, with the first agent to finish reenters in the end of the queue.

The optimisation perspective to approach this issue would be to send all agents into planning mode if any agent needs a new path. This allows proper prioritization between agents, but will lead to many completed plans being discarded. Another problem is to decide what should happen with the vehicles as the fleet goes into planning mode which could last between a few milliseconds up to half a second for worst case scenario. During this time, a loader at cruise speed can move a distance significant to the planning.

Stopping all agents to complete the planning would be highly inefficient. An interesting alternative idea would be to use a transition point according to the following: As an operator enters an agent into the autonomous fleet, there is already a plan being executed for the rest of the fleet. The algorithm keeps executing the current plan as it produces a new plan starting at the transition point a moment ahead in time. Once the plan is found, the executing plan is cut off at the transition point, where the new plan can take over. This would allow a proper agent prioritization and at the same time avoid inefficient planning stops.

5.9 Considering Unplanned Agents

With inspiration from the hierarchical-part of WHCA*, agents are listed in a set planning order. During planning phase, agents take turns planning paths and making reservations. The hierarchical nature allows the high-priority agents to make plans disregarding the rest of the fleet, shifting conflict avoidance responsibility to the lower priority agents. The method considers agents semi-selfish where they all act according to their own cost function, all working to lower their own action sequence cost.

Since yet unplanned agents are ignored as the first agents enter planning phase, this can result in them planning paths straight through the current position of others. This transfers the conflict avoidance responsibility to the agent who now has the sole responsibility to avoid an incoming collision as well as find its own way to the goal. For most situations, this should not impose a problem yet special cases could require multi-step maneuvers to navigate around. Without a smarter algorithm, eventual situations like these could possibly be resolved using human intervention where the vehicles could be manually directed out of the conflict. Additional testing is required to determine more closely how common these situations are. This lands outside the scope of this thesis however.

5.10 Prioritization Order

Bnaya & Felner [2] suggest using a prioritized round-robin/FIFO to capture both deliberate prioritization together with a model for iteratively shuffling within the order to resolve search failures. To see what model can be used to prioritize the order, we take a closer look at the vehicle cycles where the following actions can be identified:

1. Loading: Manual operation. Filling the bucket
2. Transportation: Automatic operation. Agent visits the dumping site and returns
3. Dumping: Autonomous operation. Agent empties it's bucket into a crusher
4. Conflict avoidance: Autonomous operation. Representing any time spent which increases the cycle time includes waiting and rerouting

The only operation which this thesis examines is the time spent avoiding conflicts, which for this reason can be assumed variable in contrast to the other operations which can be considered fixed.

$$\begin{aligned} T_{conflictAvoidance} &= T_{variable} \\ T_{load} + T_{transport} + T_{dump} &= T_{fixed} \end{aligned} \tag{5.3}$$

To calculate the production rate (R_P) for a single agent, the following equation

can be used

$$R_p = a_{cap} \cdot \frac{1}{T_{fixed} + T_{variable}}, \tag{5.4}$$

where the agent's bucket capacity is denoted a_{cap}.

The function for the production rate for the entire fleet can be seen in Equation (5.5), where all variables, except the variable time, are considered constants. Each of the n_{agents} agents have unique bucket capacities $a_{n,cap}$, and fixed times $T_{n,fixed}$. Finding a solution which maximizes the production rate is the objective function. Doing so the agents must fulfill the collision-constraint placing the conflict avoidance time at the agent best suited.

$$\text{Maximize } R_p(T_{n,variable}) = \sum_{n=1}^{n_{agents}} a_{n,cap} \frac{1}{T_{n,fixed} + T_{n,variable}} \tag{5.5}$$

To stay within scope in this examination, all agent's buckets are considered equal which is the current case in application. I will, however, not eliminate the bucket capacity from the equation completely.

Looking at the function, one can see that increasing the variable time will decrease the production rate.

Even though we have no control over the $T_{n,fixed}$, it will in reality vary since path lengths are different. To take this into account using a quantitative approach, recorded data has been used which together should cover the window of usual operation, see Table 5.1. With these different cases with different fixed times, the variable time is then varied between 0 seconds representing an unobstructed path, and 300 seconds representing a heavily obstructed path, see Figure 5.4.

Table 5.1: *Data calculated from all 131 currently recorded cycles. The values include the manual loading process which represents around 60s.*

Unit	Average	Median	Min	Max
Time [s]	364.6	360.6	257.4	526.2

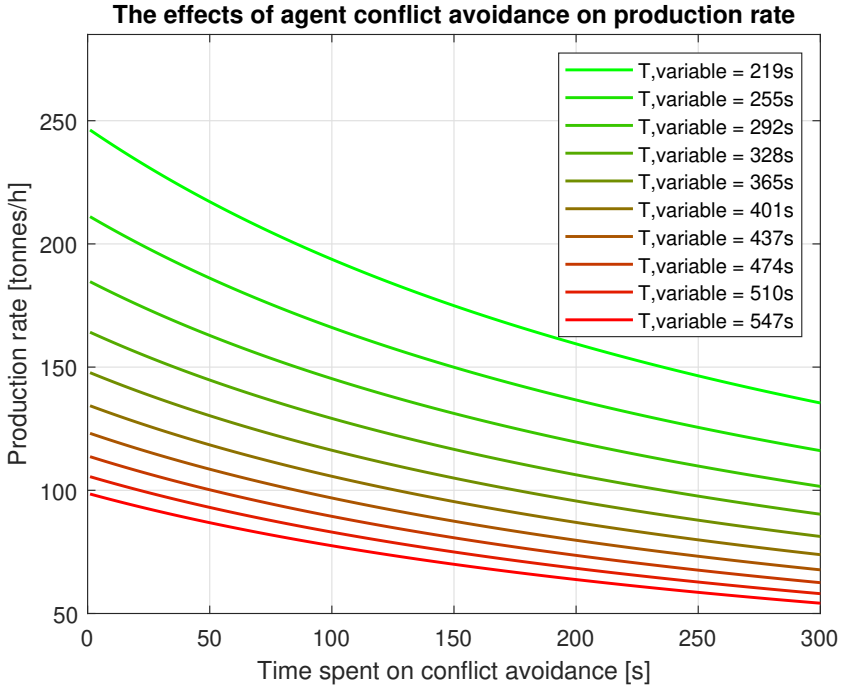

Figure 5.4: *A quantitative comparison where a single agents production rate is examined as a function of its time spend avoiding conflicts for different cycle lengths.*

Looking at the results, a few things can be observed:

- As one can expect, having shorter fixed cycle times results in a higher production rate for a single agent. The length of the fixed cycles is not something that can be controlled over prolonging operation.
- The derivative of the production rate over conflict avoidance time is of higher absolute value for shorter cycles. As an example, consider the interval between 0 and 50 seconds of conflict avoidance. The decrease of production rate for the shortest cycle is 410-330 = 80 tonnes/h, whilst the same value for the longest cycle is 175-145 = 30 tonnes/h. Extend this dynamic over multiple hours of operation and the effect would be apparent. This can be seen between all scenarios, meaning prioritizing agents with short routes is better from a simple production rate standpoint.

The results from this speak about how the agents can be prioritized in conflict avoidance. However, there are several aspects to the operations which are not included in this model. A full bucket puts extra strain on the machine which in this state requires more fuel to drive and experiences increased wear. In this aspect, it would make sense that agents with empty buckets in first hand do the

conflict avoidance maneuvers. In a perfect method, this would be captured if included in the cost model, but since agents are individualistic in this method it would not affect prioritization by adding weights to specific agents. It is, however, an interesting expansion to consider.

5.11 Algorithm

The algorithm cycle, see Algorithm 1, is parted into two main parts; plan phase and move phase. As long as there are agents requested to move, the planning phase will iteratively attempt assembling plans for all agents. If one agent fails it is due to others obstructing, a reset of the involved is required. After the plan reset, the order is changed in an attempt to solve a lockup. Once all agents have plans moving forward, the move phase will initialize where each agent sequentially performs their actions until one agent's plan ends. The logic then loops back to plan phase again.

Algorithm 1: High Level Operations Algorithm

```
while some agents are not at their goal do
    reset 𝒯;
    while some agents are missing plans do
        planPath(a_i,𝒯);
        if search fail agent a_i then
            a_c = identifyBlockage();
            resetAgent(a_c, 𝒯);
            shufflePlanningOrder();
    while movePhase do
        for each agent a_i in parallel do
            Move a_i according to plan;
            if planEnd(a_i) then
                movePhase ← false;
```

5.11.1 planPath-Algorithm

In Algorithm 1, the function planPath(a_i, $\mathcal{T}$) is used to produce a movement plan for agent a_i using the current reservation table $\mathcal{T}$. Here follows a brief summary

of the process.

Algorithm 2: planPath(a_i,$\mathcal{T}$)

```
agentSuccess ← false ;
for alternatives to evaluate do
    while !agentSuccess and iterations < maxIterations and searchFail do
        if currentNode equals currentGoal then
            removeCurrentGoal();
            if isEmpty(currentGoal) then
                agentSuccess ← true;
                results.plan ← plan;
                break;
            end
        end
        exploreNode(currentNode, T);
        if searchFail then
            results.plan ← empty;
            break;
        end
        iterations ← iterations + 1;
    end
end
```

The agent carries its mission consisting of a list of goal nodes to visit as the algorithm commences. The list of goals can either be a single goal such as in a request to move an agent, or several such as when performing a load-dump cycle. The following for-loop encloses the entire function and rotates any alternative dump sites into the list of goals to evaluate. The following while-loop continuously performs a *search and check* where it checks whether or not the best prospect st-point in the state space matches the goal id. If it matches, the current goal is removed from the search list as the vertex is added to the sequence. If the list of goals is empty or not controls if the search is continued or stopped. As long as the agent has goal vertices left to visit, it performs an exploration on the st-point to evaluate the next level actions from this point. If either the st-point list runs empty, or the while-loop has reached the max number of iterations, the search is abandoned and an empty plan is returned.

If the algorithm passes through the for-loop without ever clearing the entire list of goals in the mission, the parameter *agentSuccess* stays false. This means as the name suggests that the agent failed to find a solution to its mission. This flag signals the main algorithm to attempt to resolve the conflict on a higher level by changing the planning order and trying again.

The following while-loop will iteratively perform checks whether the current best prospect node in the list is the first goal vertex, perform a call to the node explorer function which explores the next action steps of the best vertex prospect.

6
Results

To evaluate the new method's performance a number of tests were performed in the simulator. A graphical illustrator built into the software played out the agents' movements. The illustration is however something very difficult to illustrate without the option of moving picture, thus requiring a more 2D-suited evaluation method. The performance results are mostly interesting in comparison to the known current state. Therefore a replica of the original planner was setup in the same simulator to enable clear comparison between the old and new method. The results from one test will here be presented.

6.1 Numerical Results Comparison

A test case similar to the discussed scenario in the problem analysis was setup to test the new algorithm compared to the old, both for Alpha and for Beta. As mentioned in the introduction, Alpha is well suited for the old coordination method as the shared static zones between two agents are small; meaning low path coupling. This means that the new method is suspected to have a lower impact on Alpha's efficiency since it has less to work with. Beta is a different scenario since it contains over 6 times more loading drifts per dump site compared to Alpha, see Table 6.1. Fewer dump sites leads to higher path coupling which requires more complex coordination.

Table 6.1: *Comparison between the two maps indicating the increased difficulty of coordinating traffic on the Beta map.*

Map	Loading drifts	Dumping sites
Alpha	7	4
Beta	24	2

The tests consists of two agents completing a total of 10 loading-dumping cycles. The loading sites were chosen specifically to represent a scenario where the original planner produced inefficient plans. The results from a couple of test cases are presented in Table 6.2.

Table 6.2: *A comparison for average production rate between the new and the old method. Production rate values is based on the mine simulator and therefore differ from real world operation numbers.*

Map	Old Method	New Method
Alpha	0.88 tons/s	0.92 tons/s
Beta	0.39 tons/s	0.58 tons/s

Table 6.3 sums up the improvement seen from the method update for both maps.

Table 6.3: *Production rate improvements of the new method seen in the simulator. The data is from a two-vehicle operation over 10 loading-dumping cycles.*

Map	Production Rate Increase
Alpha	+5%
Beta	+49%

An additional interesting aspect of the results is determining the algorithms scalability of fleet size. Two sets of data are here presented: production rate and planning time over fleet size. All data is from simulated operations of map Beta. The total production rate over fleet size in Figure (6.1) shows a linear behaviour meaning agents have low impact on each others ability to perform their missions. By looking at the data derivative, one can also see the value stays somewhat constant.

By dividing the production rate by fleet size, we get the derivative which is a constant value between 0.3 and 0.4 matching the where each agent has similar contribution, see Figure (6.2).

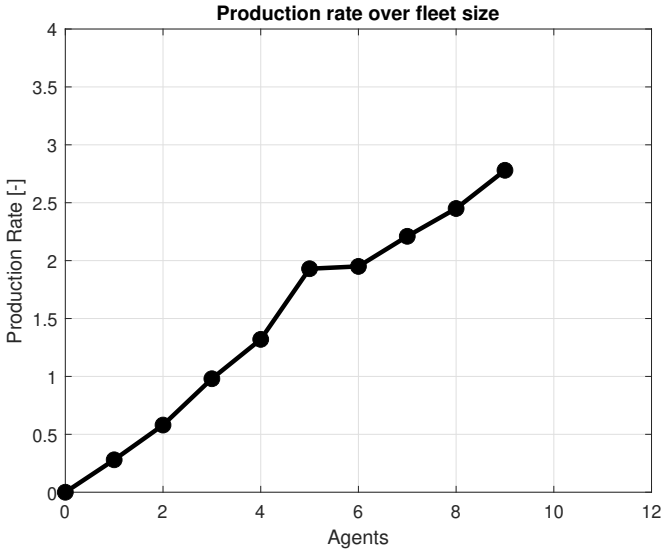

Figure 6.1: *Total fleet production rate as a function of fleet size of map Beta.*

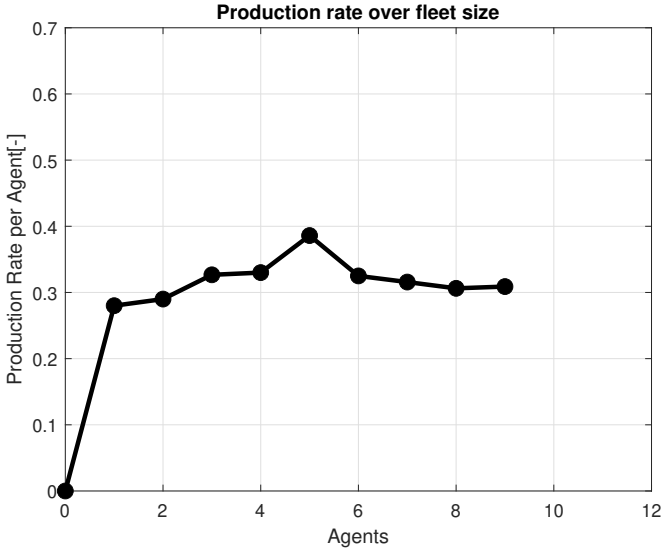

Figure 6.2: *Derivative of the total production rate data from Figure (6.1).*

The planning time data for one planning phase for the same operation is shown in Figure (6.3).

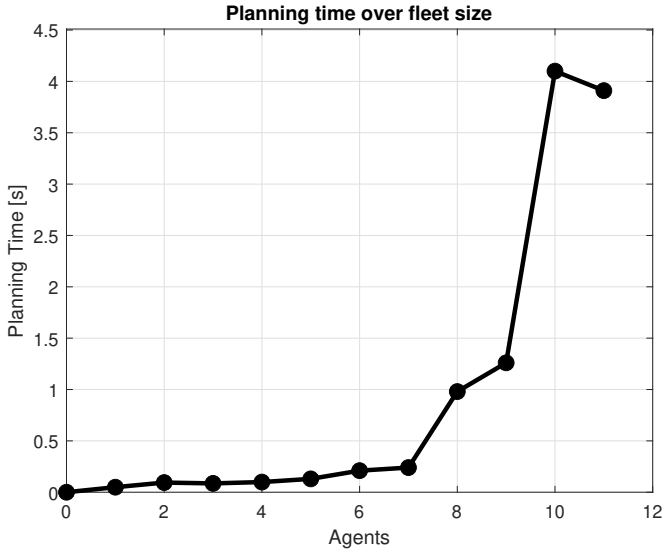

Figure 6.3: *Planning time of one planning phase for map Beta. As fleet size increases the problems quickly becomes more complex which is shown in the planning time.*

The planning time seems to accelerate with an increasing fleet size which is a known property of MAPF problems. At fleet size 12 for map Beta, the algorithm failed to find a solution. At this point, one of the two crushers is at full capacity, and the traffic is possibly enough to inhibit agents to pass through to the one with yet free.

7
Discussion

This chapter contains general discussion about both results and future work.

7.1 Discussion Regarding Results

What agent behaviour can be seen with the new algorithm?

By more accurately defining the no-collision constraints, the algorithm is given the responsibility to ensure safe operation instead of using manually set paths as before. This opens the possibility to allow vehicles to move more freely in the environment without risking accidents. By giving the algorithm this set of planning tools combined with a clear objective function it can maximize the time efficiency any way it sees fit. Vehicle collisions are now avoided using the minimum effort possible according to the cost function. By giving the planner time perception it now sees action possibilities where it else would not.

By looking further into the results from testing the source of the improvement could be better understood. Consider a theoretical scenario where two agents have a complete path coupling using the old method, meaning once one is moving the other need to remain stationary. This results in roughly 50% efficiency since two machines do the work that a single one could do. By utilizing smarter conflict avoidance, much of this time is avoided, significantly increasing efficiency. This is expected to be the source of the efficiency improvement from the test for the highly coupled Beta. This could also explain the little improvement seen in Alpha, since less path coupling means less room for improvement.

7.2 Planning Order

Looking closer at the specific objective function which is to be minimized, the individual agents are of no interest. Due to the path coupling, most agent's decision will impact the others. To find an optimal solution requires making decision on a fleet-scale which enables capturing the chain reaction one path has on the others. Making decisions on this level is, however, very demanding due to the size of the state space.

As described in Chapter 1, optimality is not the main interest if it compromises a fast decision making. This allows a shift where the method, instead of looking at fleet-cost, rather allows agents to make individual cost evaluations, disregarding the implications this has on others. This will make planning considerably faster since it reduces the problem, however it can produce highly inefficient plans for specific scenarios, see Figure 7.1.

In this 2-agent scenario, a_1 has a goal which would obstruct one of two ways for a_2 to reach the goal. From a global optimization perspective, it is easy to realize that a_2 should proceed through first, followed by a_1 such that they both traverse the shortest routes. Instead, using the individual agent's perspective, a_1 with planning priority would advance and stop at its goal, making the remaining option for a_2 to approach its goal using a long detour. With a_1 having priority, the problem can be completed in an 11 action sequence, instead of 4 actions for the opposite order.

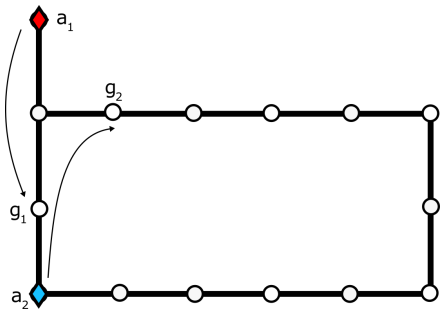

Figure 7.1: *A 2-agent scenario where a_1 planning priority leads to a heavily suboptimal solution being found.*

A variation of this which plays out much differently is if the detour does not exist. If a_1 plans first, a_2 will fail its search due to the obstruction by a_1. By identifying this obstruction, resetting both agents and flipping the planning order, the globally optimal solution would be found which is interesting to note.

A possible method to systematically avoid heavily suboptimal decisions such as this is by trying to detect solutions several times longer than the shortest distance and looking at shuffling the involved agents' planning order.

7.3 Scalability

Changing the graph size or the number of agents will affect calculation times and even performance. I will here share the noted properties regarding these aspects.

The population limit aspect of MAPF problems was discussed in [5] where the author, as mentioned in Chapter 3, showed the system population limit was closely related to the number of graph branches. Exceeding that limit quickly impacted the success rate for the worse where agents failed. The similar properties were here observable in the testing phase, where stacking several agents in a single loading drift caused frequent failures of the planner. Sticking to one agent per drift seemed to be the highest population able to perform sustained operation. The aspect of allocating each agent a personal loading drift allows them to have a safe space to await openings in the traffic flow. This also isolates the vehicles in manual loading process from traffic. However, this limit seems to also be a function of the level of path coupling. Application of the algorithm on Alpha supported as many agents as available drifts, however, the same did not stand for Beta. The increase in both path coupling and drift/crusher ratio, see Table 6.1, meant that a full drift population produced gridlocks. There was simply not enough room for the entire fleet to find successful paths. Pushing operation past this limit would possibly require a completely different approach with a solver where all agents are planned collectively as a fleet instead of individual selfish agents.

Another scaling limit was seen when the population was increased up to a point where dump sites approached 100% occupancy rate. Since agents entire paths are planned from start to finish, their success requires them finding an unoccupied dump site to visit. If none are found during its entire planning horizon, the planner will fail. However this is expected since dump site occupancy is a bottle neck for production and this simply is a property of the mine operation rather than a failure in path planning.

7.4 Limitations

Despite efforts in making the algorithm functional in all thinkable operation cases, there are still blind spots it will fail to solve. One of these scenarios is very specific but showcases an important aspect. Important to remember, the agents will not commit to moving unless they find a complete path in this method. They can, however, be interrupted mid-route and replan from this point with no apparent problems.

The problem case is depicted in Figure 7.2. Here, two agents are at opposite sides of a narrow tunnel. In the exact centre is an extra slot able to fit no more than one agent. Each agent's goal is past the other agent, and the only way to solve is if one enters the side-pocket to let the other pass. The issue with this case is that once the first agent, disregarding the second, plans its route straight ahead, the second agent has nowhere to go nor can it reach the pocket in time until the first

blocks it off. Since the problem is symmetrical the same happens if the planning order is switched.

An interesting aspect to mention is that if one agent is closer to the pocket, the problem will be solved by the conflict resolving agent having enough time to avoid the collision before it is too late.

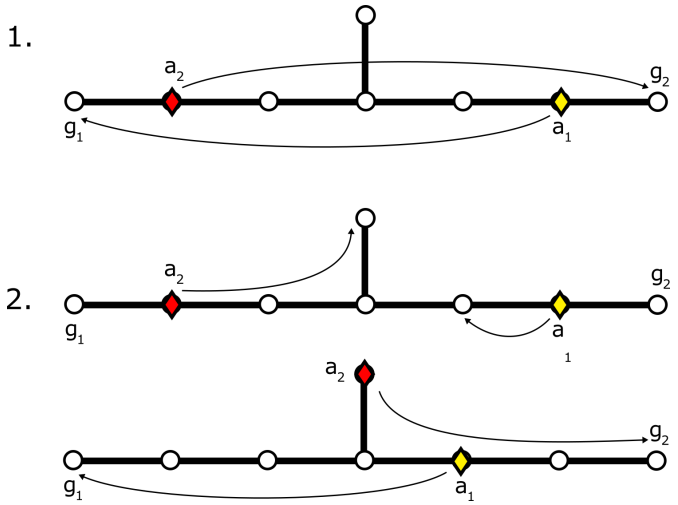

Figure 7.2: *1) A case the algorithm can not solve without adding further functionality. 2) The solution requires the agents to act selflessly since the top priority agent must give way to the next for any of them to succeed.*

To solve scenarios like this, a module is needed which could do something similar to the following:

1. Identify cycles of lockup between two agents there none of them can finish their missions.
2. Locate a spot one of the agents can reach which is outside of the conflict.
3. Set this goto-point as the new mission for that agent.

7.5 Future Work

The approach to solve this problem using MAPF was expected to be a challenge but proved to require even more time than anticipated. This section covers developed but yet untested ideas on how the remaining problems could be approached. A list of simplifications are used to enable application of theory on the real operation. Since the method is planned to be used in practice, this section will present method outlines to close the gap between the two. How can the model be evolved to better represent reality to the extent that it is practical and can be used?

7.5.1 Equidistant Vertices

Aligned with most research on the subject, this thesis has mostly considered all edges equally long. This has heavily simplified application for a low resolution discrete time planner such as this. In reality, the graph will not have equidistant vertices which is a hurdle to overcome. As mentioned earlier, each edge is suggested to be given a property describing its length which could be used as input to make travel time predictions for planning.

7.5.2 Direction of Travel

An aspect of interest in operations such as this is having a preferred direction of travel for certain situations. An example of which is preferring vehicles with empty buckets reversing. The orientation method introduced does not include direction of travel, merely orientation at each vertex. The first step to include direction of travel in the path planning is to identify direction of travel from the graph search itself. A suggestion to what that might look like is the following:

Consider a vehicle at vertex v_n. Allow a line l_n to intersect v_n such that it is perpendicular to the vertex's direction vector k_n. As the vehicle is lengthwise aligned with k_n, l_n is also perpendicular to the vehicle's orientation, see Figure 7.3 Knowing the vehicle orientation, the direction of travel to reach the neighboring nodes can be determined by looking at which side of l_n the nodes are positioned. Using this method, the move-action costs can be altered between forward/reverse drive to receive the preferred vehicle behaviour. An illustration of this can be seen in Figure 7.4.

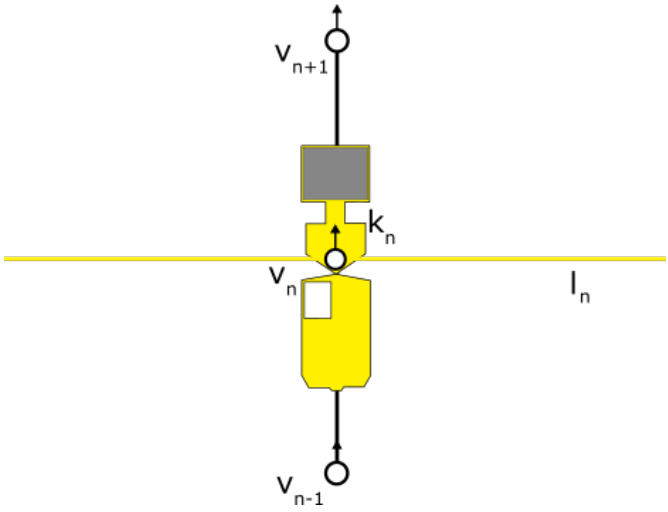

Figure 7.3: *Looking at an orthogonal vector to the direction of the vehicle allows identifying that travel to v_{n+1} is done by forward drive whilst reaching n_{n-1} is done in reverse.*

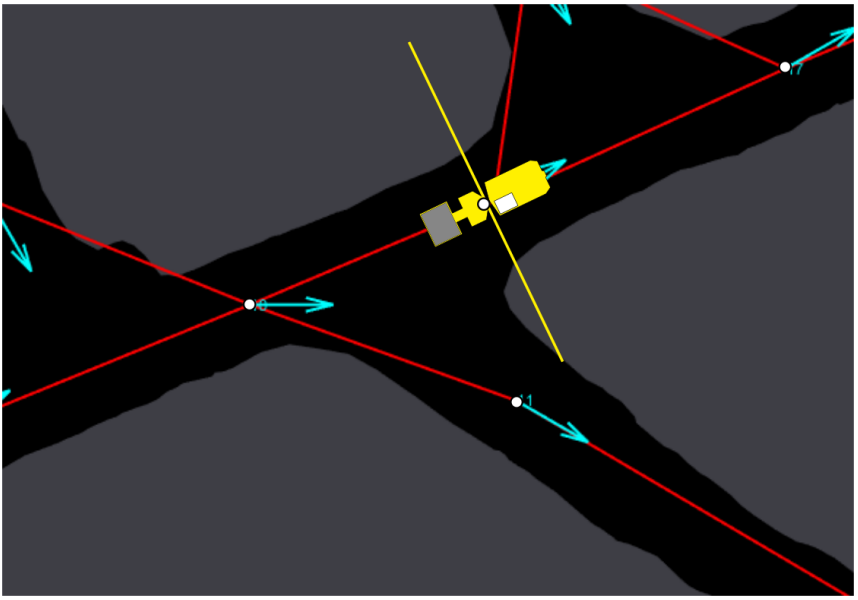

Figure 7.4: *By looking at which side of the vehicle (front/rear) the neighboring node is located, it is possible to tell travel direction to reach them.*

For this method to work, the graph needs a sufficient amount of vertices to accurately model the vehicles' movements in the environment. One particular trap to avoid is modeling sharp turns with too few points along the trajectory. A suggestion is to always model each corner with a minimum of two spaced out vertices such as depicted in Figure 7.5. If not, it is possible that the model incorrectly interprets a corner to imply a change of direction.

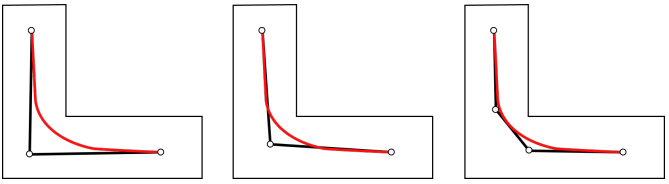

Figure 7.5: *An illustration of an environment corner. The real corner trajectory can be seen in red. Modeling a corner with too few vertices can result in the failing of the method of detecting vehicle direction. The left model with a graph corner sharper than 90 °, would assume the corner requires turning around when in reality it does not. The center case works as the angle is slightly obtuse, thus being correctly interpreted.*

7.5.3 Safety Aspect

To guarantee a safe operation for the customer, the algorithm needs to be robust towards any loss of communications, prediction errors or similar. The size and weight of the vehicles impose great forces if collision occurs which will damage the expensive machinery. The algorithm presented in this thesis works by placing full trust in planning accuracy. In the developed simulator, plans are executed perfectly which disregards the safety risk of imperfect operation. Before any similar solution could be implemented in actual application, some observing system would need to verify the output movements against the references.

In operations where agent collisions impose a significant risk of property damage or safety such as in air traffic control, a human supervisor can be asked to verify a solution before it is executed [1]. This could be an interesting aspect to investigate, however, it presents a new issue where multi-dimensional plans such as these are notoriously difficult to present in a way that allows human evaluation.

7.5.4 Synchronization Issues

Making movement plans for the vehicles requires predicting positions through existing plans. If one vehicle makes its plan in relation to an estimation of another vehicle's plan and the estimation turns out to be inaccurate, what happens to the coordination? How can the synchronization be handled to ensure safe operation?

Using time based synchronization blindly from predictions seems too fragile against prediction errors. One idea is to continuously update the prediction as the plan is executed to communicate updates to other vehicles. If one in the fleet is lagging behind plan as it is executed, the velocity profile of the other could be matched such that they collectively stay synced although with somewhat slower movements.

7.5.5 Agents of Different Movement Speed

One of the central simplifications made to ease the development of the algorithm was assuming each action takes one time step to complete. This is far from how the vehicles behave in operation. This has simplified how planning can be performed since plans can fit neatly into matrices. To alter the planner to handle agents of varying velocities could be desired for a future implementation.

8
Conclusions

The developed algorithm is based on a variety of multi-agent A* planners and can coordinate a fleet of agents with significantly improved movement plans compared to the current solution. The no-collision constraint was more accurately defined compared to before and the agents were freed from predetermined paths. This gave the planner better tools to create plans where the agents aim to complete their goals in the shortest possible time. Agents now better utilize their environment to avoid conflicts.

The improvements seen for operations in Alpha were modest as expected due to there being low room for an improvement. However, the goal to improve the operation at map Beta has been showing more interesting results. A test for two vehicles put in a previously shown inefficient operation showed a sustained joint improvement of 49% production rate. The algorithm's performance also scales well into larger fleets where the total production rate over fleet size remained close to linear. This means the implied increased problem complexity due to additional agents does not appear to negatively affect production rates. However, Alpha is limited by the number of loading drifts (7), and frequent planning failures occur when surpassing 11 agents for Beta. The operations in the latter is believed to be limited due to saturation of the graph where gridlocks are increasingly harder to avoid as reservation of the crusher area approaches 100%.

By finding a way to utilize the strengths of the MAPF approach, the simulations suggest that there are some significant improvements for the industry to further explore.

Appendix

A

Code & Object Properties

For anyone interested, here follows an additional look at the code structure and its implementation.

A.1 Object Properties

Graph consists of:

- Vertex set
- Edge set
- Dump site node IDs
- Load site node IDs
- True-distance heuristic matrix (offline generated)
- Look-up table to couple vertex to edge

A vertex is described by:

- **id:** identity value
- **x:** x-coordinate
- **y:** y-coordinate
- **k:** direction vector
- Neighboring nodes

An edge is described by:

- **id:** identity value
- Start and end node IDs
- **f:** orientation change parameter (binary)
- Edge length

Agents consists of

- **pos:** current node position
- **d:** orientation in relation to current node direction
- Max bucket capacity
- Bucket status (current content $\{0 : max\_cap\}$)
- Current goal node sequence
- Completed rounds
- Total delivered
- Illustration color

Reservation table:

- Matrix with size $[planning\_horizon \text{ x } num\_nodes]$

A.2 Functions

A star: Single agent path planning algorithm. The function is described in greater detail in Chapter 5, but the algorithm overview is here repeated. The function completes a series of events to find the best solution for one agent to complete its mission.

Algorithm 3: planPath(a_i,$\mathcal{T}$)

```
agentSuccess ← false for alternatives to evaluate do
    while !agentSuccess and iterations < maxIterations and searchFail do
        if currentNode equals currentGoal then
            removeCurrentGoal();
            if isEmpty(currentGoal) then
                agentSuccess ← true;
                results.plan ← plan;
                break;
            end
        end
        exploreNode(currentNode, T);
        if searchFail then
            results.plan ← empty;
            break;
        end
        iterations ← iterations + 1;
    end
end
```

check next: API: $exploreNode(currentNode, \mathcal{T})$
A function which outputs the neighboring nodes to the input node $currentNode$, together with the edge distances to each one. Any actions conflicting with reservations made in $\mathcal{T}$ are considered unavailable.

sort agents: API: $sort\_agents(agent)$
In an attempt to produce efficient operation, the agents are primarily sorted in planning order with increasing distance to goal as discussed in Chapter 5.

shuffle agents: API: $shuffle\_agents(agent)$
Function that randomizes the order of agents in the prioritization list if the order is insufficient in finding a solution.

Bibliography

[1] Shaull Almagor and Morteza Lahijanian. Explainable multi agent path finding. *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, (19):34–42, 2020. URL https://shaull.github.io/pub/AL20.pdf.

[2] Zahy Bnaya and Ariel Felner. Conflict-oriented windowed hierarchical cooperative A*. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. URL https://ieeexplore.ieee.org/document/6907401.

[3] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T.K. Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 2019. URL https://ojs.aaai.org//index.php/AAAI/article/view/4756.

[4] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. URL http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf.

[5] Mike Peasgood and John McPhee. Complete and scalable multi-robot planning in tunnel environments. *IFAC Proceedings Volumes*, 39:26–31, 2006. URL https://www.sciencedirect.com/science/article/pii/S1474667015311824.

[6] Stuart J. Russell. *Artificial Intelligence*. Alan Apt, 1996.

[7] David Silver. Cooperative pathfinding. 2005. URL https://www.aaai.org/Papers/AIIDE/2005/AIIDE05-020.pdf.

[8] David Silver. Cooperative pathfinding. 2020. URL https://www.davidsilver.uk/wp-content/uploads/2020/03/coop-path-AIWisdom.pdf.

[9] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T.K. Satish Kumar, Eli Toyarski, and Barták Roman. Multi-agent pathfinding: Definitions,

variants, and benchmarks. *Symposium on Combinatorial Search*, (12):151–159, 2019. URL https://arxiv.org/abs/1906.08291.

[10] Jingjin Yu and Steven M. LaValle. Planning optimal paths for multiple robots on graphs. *IEEE International Conference on Robotics and Automation*, 2013. URL http://lavalle.pl/papers/YuLav13.pdf.