Malte F. Speidel

# Navigating Path Influenced Environments: MCTS and Two-Phase A* for Multi-Objective Roundtrip Problems



FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

# Navigating Path Influenced Environments: MCTS and Two-Phase A* for Multi-Objective Roundtrip Problems

Master Thesis

Malte F. Speidel

March 31, 2026

Supervisor: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Advisor: M.Sc. Carlo Nübel

**Malte F. Speidel:** *Navigating Path Influenced Environments: MCTS and Two-Phase A* for Multi-Objective Roundtrip Problems*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2026.

# Abstract

Path Influenced Environments (PIEs) are a novel class of pathfinding problems in which an agent alters the environment by moving obstacles, making its structure dependent on the taken path. In this thesis, we want to address a roundtrip problem in PIEs, where an agent must travel from a designated starting point to a waypoint and back. This extension of pathfinding in these environments was not yet studied and poses an interesting challenge. Here two objectives will be minimized at the same time, which are path length and the total weight shifted along the path, which makes this a multi-objective problem. A Monte Carlo Tree Search approach is proposed by us, featuring multiple tree traversal strategies (UCB1, hypervolume based, crowding distance based and a novel adaptive epsilon archiving based approach) as well as three rollout methods (light, heavy distance and weight based, heavy square sampling based). As a baseline we introduce a novel two-phase A* pipeline, which is able to approximate a Pareto front of paths for the formulated problem. Experiments were conducted across four different environments which each were generated as a $35 \times 35$ and $50 \times 50$ two-dimensional grid. Results show that the two-phase A* consistently outperformed MCTS across all possible configurations, producing Pareto fronts with multiple solutions and of higher quality. Among the different configurations that were tested for the MCTS approach, crowding distance and hypervolume based tree selection performed best and UCB selection performed worst in all environments. The analysis of the results also revealed that the used mechanisms to simulate child nodes do not have a significant impact on the quality of the results, making the tree selection mechanisms the driving factor.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Pathfinding has seen a rise in scientific interest in recent times, with a new algorithm finally beating the runtime complexity of Dijkstra [12]. The authors of this paper did not give a name to this algorithm but describe it as a combination of Dijkstra [11] and Bellman-Ford [5]. While this new algorithm is only applicable under specific circumstances, the interest that it invoked in the field shows that pathfinding in general is still a very important and influential topic in the world of computer science [37][1]. Most approaches to pathfinding are used in environments with mostly static and immovable obstacles. There are approaches for dynamic obstacles, i.e. in robotics, but they focus more on obstacle avoidance [2][25][27]. This is intuitive, since the primary objective in most scenarios is to reach the goal via the most direct path while avoiding any obstacles that can hinder progress. To illustrate this, consider a scenario where the direct route between two cities is obstructed by an accident. Here, the objective is not to move or remove the damaged vehicles, but to rather avoid this path and take a less direct but free route around it. While these situations are more common, there are also scenarios where we wish to move the obstacles, instead of avoiding them. The first thing that comes to mind for these kinds of scenarios is a snowplow. As a primary objective snowplows need to clear snow from the road, making it easier and, most importantly, safer for other vehicles to travel. Another example is terrain leveling, where large amounts of material are moved to achieve a flat surface, enabling the construction of i.e. a road that more directly connects two locations. What these tasks have in common are that dynamic obstacles are moved around and also in some form combined with each other. In the case of the snowplow, snow is stacked onto other snow, which creates a new and heavier obstacle, that in turn would need more energy to be moved again. The same applies for terrain leveling since the material that is moved can also be placed on top of each other, creating a similar situation. Furthermore, the path that is taken while moving the obstacles has a great impact on the overall energy and effort that needs to be spent to achieve the desired outcome. When explor-

ing this problem, it becomes apparent that the length of the chosen path and also the amount of energy spent to move obstacles while taking this path are both objectives to be considered, which makes this a multi-objective problem. The environments in which these pathfinding problems take place are known as so-called "Path Influenced Environments". While a less coarse description is provided in the related work chapter, these environments can be thought of as containing mostly dynamic obstacles that an agent can move to create a new path and in doing so, stacking or merging obstacles, which results in heavier or denser ones. These new obstacles then take more energy to move again. The crux of these environments is, that each move has the possibility to create a new optimal path through the environment, that minimizes both the length of the path and the energy it takes to create it. Of course there can also be multiple paths with different tradeoffs between the objectives, which is in the nature of multi-objective problems. What this thesis aims for is to create a new approach to solving these kids of pathfinding problems efficiently by applying Monte Carlo Tree Search [28] to them. For the purpose of this thesis, we can say that MCTS consists of four main components, which are the tree traversal or tree selection, root selection, simulation strategies and backpropagation. In this context, tree traversal/selection determines which nodes are visited when traversing the tree from the current root to a new leaf node, root selection determines which child of the current root becomes the next root, and the simulation strategy defines how a newly created leaf node is simulated. Backpropagation updates the values of nodes whenever a new child node is generated in a subtree, ensuring that each node reflects the quality of its subtrees. While this is only a superficial depiction of a Monte Carlo Tree Search algorithm, it suffices to describe which different parts we want to experiment with in this thesis. We want to figure out how different tree traversal strategies like UCB1, a crowding distance based and a hypervolume based strategy would perform when applied to pathfinding in PIEs. Furthermore, we want to gain insight on how these traversal strategies work in combination with rollout methods that use random moves, as well as those that use heuristics to decide what their next move should be. For the root selection, we will only use a hypervolume based technique, since more combinations that need evaluation would exceed the scope of this work. As a baseline we propose a two-phase A* approach, since vanilla A* [16] is a well established path finding algorithm, that can be easily adapted to the problem at hand. Our research questions and hypothesis for this thesis are as follows:

1. **RQ1:** Can MCTS be used for pathfinding in path influenced environments?
   *$H_0$:* MCTS cannot solve pathfinding problems in PIEs, even when adapted to the specific demands of the problem.
   *$H_1$:* MCTS can solve pathfinding problems in PIEs, provided it is adequately adapted to the specific demands of the problem.

2. **RQ2:** Which algorithm performs better on these problems, two phase A* or MCTS?
   *$H_0$:* There is no significant difference in performance between MCTS and the two phase A* approach on PIE pathfinding problems.
   *$H_1$:* MCTS outperforms the two phase A* approach on PIE pathfinding problems due to its simulation capabilities, which allow it to more accurately predict good paths.
   *$H_2$:* The two phase A* approach outperforms MCTS on PIE pathfinding problems, as the deterministic nature of its components allows it to more reliably find optimal paths.

3. **RQ3:** Which combination of MCTS mechanisms produces the best results?
   *$H_0$:* There is no significant difference in performance among the various combinations of MCTS mechanisms.
   *$H_1$:* There is a significant difference in performance among the various combinations of MCTS mechanisms.

Having formulated the research questions and hypotheses, the thesis proceeds with a related work chapter covering concepts relevant to this thesis, followed by a methodology chapter presenting a problem description, the adapted Monte Carlo Tree Search algorithm, the two-phase A* approach, and the experimental setup. The results chapter then evaluates the outputs of both approaches independently, compares the different MCTS configurations, and concludes with a direct comparison of both algorithms in terms of their found optimal solutions. Afterwards we are going to summarize the results of the experiments and answer the presented research questions, and also present an outlook on possible future work.

# 2 Related Work

In this chapter, we will discuss all the concepts that are necessary to understand what we want to do in this thesis. This chapter covers the concept itself, existing works on path influenced environments, as well as Monte Carlo Tree Search and multi-objective A*. Furthermore, we will cover the concepts of Pareto dominance, crowding distance, hypervolume, and epsilon grid archiving, as all of these will be applied later on.

## 2.1 Path Influenced Environments

This first section of this chapter will focus on the available published research on Path Influenced Environments. Here we will describe what PIEs are and what the main approaches on solving pathfinding in these environments were. In the introduction we already stated that Path Influenced Environments (PIEs) are a relatively novel field of research, which is still evolving. At the time of writing, the literature on PIEs remains limited, with only three published works being currently available. As the two papers and bachelor thesis discussed afterward describe PIEs in a general sense, we want to start by also giving such a description ourselves. Path Influenced Environments are environments in which an agent alters its surroundings through movement, typically by moving obstacles out of its path, such that the state of the environment depends on the path taken. The first published paper regarding this topic was authored by Heise et al. [17]. In this paper, PIEs were first mentioned as environments that change in correspondence to the taken path, and this in turn affects the objective values. It is also notable that the authors differentiated between pathfinding and path paving. As an example, Heise et al. mention railroad construction for path paving and needle insertion into the human body for pathfinding. Both share the core objectives of traditional pathfinding, namely finding the shortest path with minimal energy

consumption. However, they differ from traditional pathfinding in that the latter disregards the agent's effect on the surrounding environment. The main contribution of this paper was to introduce Path Influenced Environments as a new benchmarking framework for pathfinding algorithms, in which the goal is to be as minimally invasive to an environment as possible [17]. It has to be noted here, that the topic of pathfinding in PIEs was only introduced here, but no optimization approaches were actually employed. The second work that can be found regarding this topic is Speidel's bachelor thesis [33]. In his thesis, he tested the pathfinding capabilities of an adapted A* algorithm as a baseline against a mutation only evolutionary algorithm in these environments. For a definition of evolutionary algorithms, we can refer to Bartz-Beielstein et al. who stated that evolutionary algorithm is used as an umbrella term that describes population based algorithms with stochastic components that mimic natural evolution [4]. This high-level characterization suffices here, as evolutionary algorithms are not the central focus of this thesis. Speidel found in his thesis that an adapted A* algorithm and the evolutionary approach were able to solve the pathfinding problem in PIEs. Furthermore, he found that the A* approach produced better results than the evolutionary algorithm in three out of four scenarios with one tie. This finding is valuable since we will later also explain A* in detail, as well as propose a new algorithmic A* pipeline that should be able to solve the problem at hand reliably and with a high solution quality. The most recent work on this topic by Nübel et al. [30] compares several state-of-the-art multi-objective evolutionary algorithms, namely NSGA-II, NSGA-III, R-NSGA-II, DNSGA2, AGE-MOEA, SPEA-II and SMS-EMOA, in terms of their ability to produce solutions for PIE pathfinding problems. Here the concept of PIEs is described as "... every time an agent takes a step in the environment, and there is an obstacle blocking its path, it changes this environment by moving the obstacle out of its path. This way, the environment dynamically changes while the agent is moving." [30]. In the performed experiments, all algorithms were tested in three different environments, the radial-gradient-, sinusoidal-, and meandering-river-environment. This is significant for this thesis since this paper is the only one, besides Speidel's bachelor thesis, that reported empirical experiments in path influenced environments, and thus two of these three environments will be reused for the experiments that are performed in this thesis. Furthermore, the authors tested various different shifting methods for obstacles that were encountered along the path through each environment. The objectives that were optimized during the experiments

were the path length, as well as the weight that was shifted along the path. Results indicated that all algorithms were able to generate a non dominated set, but the quality of the obtained solutions varied based on the chosen shifting method. This leads to the conclusion that the way in which obstacles are shifted has a large influence on the overall quality of the path. As a summary for all three works, we can state that Heise et al. first introduced the concept of Path Influenced Environments, Speidel created a baseline with the adapted A*, while Nübel et al. focused on evolutionary algorithms as a way to solve this newly proposed class of problems. This highlights the novelty and the ongoing methodological progress in the field. It is notable that the only approaches that were taken are EA's and A* which leaves many non-evolutionary methods like Monte Carlo Tree Search unexplored. This raises the question on how these non-evolutionary approaches would perform in PIEs.

## 2.2 Problem Classification

Now that we know what Path Influenced Environments are, we can discuss where to place this novel problem among the existing pathfinding literature. For this purpose, we are going to look at the related work section of Nübel et al., since the authors already included a classification of pathfinding problems [30]. According to the authors, these problems can be classified using different categories with the first one mentioned being the representation of the environment. Here they distinguish between grid based problems and graph based problems. In Peter Yap's work "Grid-Based Path-Finding", which was published in 2002, he describes grid based pathfinding problems as overlaying a discrete grid on a continuous space [39]. Furthermore, Yap states that this overlaying enables the usage of graph search algorithms like A* [16] to solve these kinds of problems. Graph based problems on the other hand were defined by Laparra as pathfinding on graphs that consist of vertices, which represent different positions, and weighted edges, where the weight acts as movement cost [22]. It could be argued that grid based problems are a sub-category of graph based problems since every grid based problem could be represented as a graph, where cells are nodes in the graph which are connected to the neighboring nodes/cells by vertices. Nübel et al. continue by using the type of the environment as a category for classification [30]. Here the authors differentiate between static and dynamic environments. In static environments, obstacles

are fixed and do not change [26], while in dynamic environments include static and moving obstacles [19]. The next category named by Nübel et al. is the environment's behavior which can be either deterministic or stochastic [30]. Skrynnik et al. describe stochastic environments as those where obstacles appear and disappear at random moments in time [32] and thus deterministic environments can be described by the absence of such random events. For the last two categories, Nübel et al. [30] refer to the number of agents and the number of objectives. They differentiate between single and multiagent as well as between single, multi and many objective problems. Single objective problems are those, where only one objective needs to be optimized [10], while in multi objective problems, we need to optimize multiple objective functions at the same time [14]. The same as for multi objective problems can be said for many objective problems, since in both one optimizes multiple objectives, but many objective problems specifically describe problems with four or more objectives according to Li et al. [24]. With these categories established, the roundtrip problem in Path Influenced Environments, which we want to solve in this thesis, can now be classified. The environment is grid-based with dynamic obstacles, as these can be moved by the agent, and its behavior is deterministic since no random changes occur. In terms of agents and objectives, the problem is a single-agent multi-objective one, as the goal is to simultaneously minimize both step count and weight shifted along the path. While we are able to classify the problem using these categories, as stated by Nübel et al. [30], none of the known pathfinding problems known besides the one at hand considers how the environment is influenced by the taken path. This emphasizes the novelty of the problem itself, especially since no research exists for the roundtrip extension of the base pathfinding problem.

## 2.3 Monte Carlo Tree Search

In this section, we will describe the concept of Monte Carlo Tree Search. We will start with the Monte Carlo Method which is the conceptual basis for the algorithm itself. Furthermore, the algorithm and its origins will be discussed, as well as UCB1, which is a commonly applied mechanism in MCTS.

### 2.3.1 Monte Carlo Method

We begin with the Monte Carlo Method, which, as the name suggests, forms the foundation of the Monte Carlo Tree Search. The first mention of this method was made by Metropolis et al. in 1994 [28]. In their paper, the authors described that certain types of equations, i.e. differential equations of a certain complexity, are impractical when employing analytical methods to solve them. As one practical example, the authors mention the study of cosmic rays. In their example a particle with great energy enters the atmosphere, in turn producing a chain reaction of smaller nuclear events in which new particles are produced that then also produce new particles through further reactions until there is not enough energy to set of new reactions. This problem is very complicated to model mathematically since we have many interacting parts that can potentially influence each other, i.e. the concrete amount of energy or the direction of the particle movement. To solve that, the authors propose that instead of completely modeling the example scenario or analytically compute the solutions for very complex equations, this could be solved by statistical sampling. For this approach, the system is modeled as a stochastic process, where then samples of events are generated from random numbers in a uniform distribution. It is emphasized by the authors that by using statistical sampling instead of analytically solving the problem, they avoid for example multiple integrations and instead sample a single chain of events. This is then repeated iteratively until numerous of these event chains are generated and from here statistical analysis yields i.e. expectations, probabilities of events or distributions.

### 2.3.2 Monte Carlo Tree Search Algorithm

After having explained the core concept, we can now proceed to describing Monte Carlo Tree Search. This algorithm was created by Rémi Coulom and introduced at the 5th international Computer and Games Conference in 2006 [8]. The author opens by noting that the traditional approach of alpha-beta search with a heuristic position evaluator produced strong results in two-player zero-sum games with perfect information, but proved ineffective for the game of Go. He attributes this, among other difficulties, to the challenge of creating an accurate static position evaluator, due to the game of Go having an incredibly large search space. Coulom continues with a mention of Monte Carlo

evaluation, that fits the dynamic nature of Go positions, and that the accuracy of this method can be improved by combining it with tree search. The author then describes, that Monte Carlo Tree Search works by iteratively running random simulations from the root of the tree. Each node therefore represents a game state and each edge a possible move to perform in that game state. The value of a node in his example is tied to the score of the game that is being played. Nodes in the tree also count how often they were visited during the random game simulations. If a node has already been visited, a new action is sampled from that position, provided it does not already exist as an edge to a child node. The values of newly created nodes are then propagated back up the tree, such that each ancestor node reflects the quality of its corresponding subtree. While Coulom also provides mathematical formulas which describe how values are propagated through the tree and how nodes are selected until a new one is created, these will not be discussed in detail here, as the focus later in this section will be on the state-of-the-art methods used today. A more formalized and concise description of Monte Carlo Tree search was given by Chaslot et al. during the Computers and Games Conference 2008 [7]. In this paper, the authors presented a graphic (Fig. 1. Scheme of Monte-Carlo Tree Search [7]) in which they described the algorithm as follows. At first, a selection function is applied recursively until a node with no children is found. After that, the expansion takes place where one or more children of this node are created. This is then followed by a simulation of the game from the newly created node as the starting point using a specific simulation strategy. The result of this game is then propagated back up the tree and the whole process is repeated X times, where X represents a budget. This budget here is described as a predefined amount of time. After this budget is used up, the most promising child node of the root is chosen as the next move to be played and in turn, this child becomes the new root node of the tree. This is then repeated until the game concludes. For the purpose of this thesis, this more accurately reflects the MCTS version, that we plan on using.

### Upper Confidence Bound 1

A commonly used technique for the selection of nodes while traversing the search tree is the Upper Confidence Bound 1. Such a selection function/policy is used to traverse the game tree from the root to a so-called leaf node, meaning a node that itself has no children. There are many functions that can be

employed to select the nodes throughout the tree, but the Upper Confidence Bound function is one of the more popular choices and was first introduced by Auer et al. in 2002 [3]. Auer et al. proposed this function as a solution to the exploitation vs. exploration dilemma in reinforcement learning. This dilemma consists of balancing exploration as a way to find new profitable actions and exploitation of the best currently known action, i.e. the action that is currently known to yield the highest reward. The authors refer to the multi-armed bandit problem as an instance of this problem. Specifically, the K-armed bandit problem is mentioned where we have K gambling machines. These machines each yield a reward which depends on the order in which they are played. The goal is to maximize the cumulative reward by optimizing the playing order. According to the authors, the different rewards are identically distributed and independent of one another. Auer et al. describe that a policy in this case is an algorithm that chooses the next machine to be played. This algorithm chooses based on the plays made and rewards obtained in the past. Another crucial concept proposed here is regret. Regret, as described by the authors, is the expected loss due to the fact the policy does not always play the best machine to facilitate exploration. In an attempt to minimize regret, UCB1 is introduced and sketched as follows. For the initialization each machine is played once, and then we enter a loop in which we play a machine $j$ that maximizes

$$\mathrm{UCB}_{\mathrm{value}} = \bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

where $\bar{x}_j$ is the average reward of machine $j$, $n_j$ is the number of times $j$ has been played so far and $n$ is the overall number of plays that occurred up to this point [3]. Furthermore, the authors have shown that this strategy achieved logarithmic regret uniformly over $n$ without prior knowledge about the distribution of rewards. The concept of UCB1 can be applied to Monte Carlo Tree Search as, in this context, a child node is chosen using the highest UCB1 value (assuming you want to maximize the score) among the available child nodes. For the formula $\bar{x}_j$ is the averaged score of child node $j$ that is either its raw value when $j$ has no child nodes itself or the averaged value of $j$'s subtrees. Furthermore, $n_j$ is the number of times node $j$ has been visited and $n$ refers to the total visits of $j$'s parent node. Since it is a commonly used concept, we will employ an adapted version of UCB1 in this thesis as a tree selection strategy.

## 2.4 A*

Since we have finished explaining all necessary concepts for the MCTS algorithm, we will continue with A*, since it will be a part of the second algorithmic pipeline presented in this thesis. The base concept for this algorithm was introduced in 1986 by Hart et al. [16]. In this paper, the authors focus on finding the minimal cost path through a given graph. As an example, a set of cities with different paths connecting the cities to each other is given. The objective is to find a path through a network of connected cities from a start to a goal city. This is similar to the commonly discussed Traveling Salesman Problem, in which a salesman wants to visit each town in a given set exactly once, starting at and returning to his hometown, where the objective is to find the shortest path for this trip [20]. Furthermore, in the paper of Hart et al. [16] it is mentioned that this algorithm makes use of special knowledge. In their example, this would mean that the shortest distance between the start and the goal can not be shorter than, i.e. the airline distance, assuming this metric is known. The authors differentiate the approaches for solving these kinds of problems in two categories, either being mathematical or heuristic in nature. It is further specified that mathematical approaches are primarily concerned with finding solutions within the computational limits of the developed algorithm, whereas heuristic approaches leverage problem-specific knowledge to improve computational efficiency. Hart et al. want to combine these two different approaches to problem-solving by incorporating the special knowledge into a formal mathematical approach. To explain the algorithm in greater detail, we need to explain what expanding a node means in this context. The authors mention that expanding a node in a graph is done by using a succession operator, which generates a part of the sub-graph. In a grid world, this would for example mean that we look at the neighboring cells of the cell we want to expand. Hart et al. describe that we keep track of the minimum cost path from the starting point to each expanded node while also keeping a reference of the predecessor to that node. This is done so eventually, when we reach the goal node, the path can be reconstructed by reiterating through the predecessor nodes. Furthermore, for A* the authors remark that to expand as few nodes as possible the algorithm must make informed decisions on what node to expand next, since if nodes that can not be the optimal path are expanded, it is a waste of effort. But it is also possible that this can fail to find a path, which has to be taken a consideration. Hart et al. propose the usage

of an evaluation function $f(n)$ that can be calculated for every node $n$ and use that function in a way that the node with the smallest value for $f$ should be expanded next. This $f$ value is generally a combination of the actual objective costs, i.e. how much did it cost us to get up to this point and the heuristic costs that tries to estimate how much it will cost us to get from the current state to the goal.

**Require:** start node $s$, goal set $T$, successor operator $\Gamma$, evaluation function $f$
**Ensure:** optimal path from $s$ to some $n \in T$
1: Mark $s$ as *open*
2: Compute $f(s)$
3: **while true do**
4: Select the *open* node $n$ with smallest $f(n)$ ▷ ties resolved arbitrarily, but in favor of $n \in T$
5: **if** $n \in T$ **then**
6: Mark $n$ as *closed*
7: **return** optimal path from $s$ to $n$
8: **end if**
9: Mark $n$ as *closed*
10: Apply $\Gamma$ to $n$ to obtain successors $n_1, \ldots, n_k$
11: **for all** successors $n_i$ of $n$ **do**
12: Compute $f(n_i)$
13: **if** $n_i$ is not marked *closed* **then**
14: Mark $n_i$ as *open*
15: **else if** $f(n_i)$ is smaller than when $n_i$ was marked *closed* **then**
16: Mark $n_i$ as *open* ▷ re-open $n_i$ (optional: update parent / costs here)
17: **end if**
18: **end for**
19: **end while**

Algorithm 1: Search Algorithm A*

Algorithm 19 shows how the authors describe the algorithm in their paper. What this algorithm does not show is how the proposed evaluation function $f(n)$ works. Hartman et al. defined $f(n) = g(n) + h(n)$, where, $g(n)$ describes the distance from the start node $s$ to the current node $n$ and $h(n)$ as the heuristic cost from current node $n$ to the goal node $a$. This is an example

of what was already mentioned earlier with $f$ being a combination of actual objective costs and heuristic costs. Furthermore, the authors have proven the optimality of A* in this paper mathematically, showing that this algorithm always finds the optimal path from start to goal and under the assumption that tie breaks, where nodes have the same $f$ value as each other, are always resolved in the same deterministic way. Since the research field of pathfinding in Path Influenced Environments currently lacks a deterministic method for finding optimal paths, this algorithmic concept will later be utilized to propose a pipeline that may serve as such an approach. Nevertheless, it has to be stated that we do not proof or disprove if the algorithmic pipeline that we are going to propose is deterministic.

## 2.5 Manhattan Distance

After discussing both concepts that we want to focus on in this thesis, we now will start to explain concepts, metrics and techniques that are utilized in this thesis. For all distance measures from here on out, we employ the so-called Manhattan distance. According to Thompson et al. the Manhattan distance or "taxicab metric" is defined as

$$d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

where $x_1, y_1$ and $x_2, y_2$ the coordinates of the two points for which the distance is to be calculated [34]. The authors describe that "taxicab geometry" results from the same concept as the Manhattan distance. For this geometry, it is assumed that we have a perfect city with all roads being either horizontal or parallel. As a consequence of this, we can not compute an accurate distance using the Euclidean distance, since it would ignore the layout of the roads and act more as a distance via air travel. This is solved by the Manhattan/Taxicab distance, since the layout of the roads is taken into consideration in the formula. Furthermore, this example of a road network can be abstracted to a graph with only horizontal and vertical edges or simply a grid, which is what we use in this thesis to simulate the environments.

## 2.6 Multi-Objective Optimization Concepts

In this section, we want to discuss multiple concepts that are related to the topic of multi-objective problems. Multi-objective problems, as defined by Miettinen [29], involve the simultaneous optimization of multiple objective functions, in contrast to single-objective problems which optimize only one. A solid understanding of the concepts of Pareto dominance, crowding distance, hypervolume, epsilon domination, and adaptive epsilon archiving is essential for following the contributions of this thesis.

### 2.6.1 Pareto Dominance

The first and most important concept is Pareto dominance. In single objective optimization problems, we only have one objective value that needs to be optimized, as was already mentioned. This makes it a fairly simple endeavor to determine whether one solution is better than another, which can be accomplished by applying the proper operator to the two values. For maximization problems this would be '$>$' since larger values are preferred. Analogous to that for minimization problems '$<$' is used. This is not as simple for multi-objective problems. Here we have more than one objective which needs to be optimized simultaneously, and this makes it difficult to determine which solutions are better in comparison to others. For these situations the concept of Pareto dominance is applied. Mark Voorneveld described the concept of Pareto dominance as such that a solution $x$ dominates another solution $y$ if $x_i \leqq y_i$ for all $i$ with one $i$ being unequal between $x$ and $y$ [36]. Applied to minimization, $x$ dominates $y$ when all objective values of $x$ are less than or equal to those of $y$, with at least one being strictly less. Solutions that are not Pareto dominated by others are so-called Pareto optimal solutions. In most multi-objective optimization problems, there will be more than one of these non-dominated solutions, and the set of those solutions is called a Pareto front. Furthermore, if there is more than one Pareto optimal solution, each solution represents a different objective tradeoff.

### 2.6.2 Crowding Distance

Now that we have discussed what Pareto optimal solutions and Pareto fronts are, we can come to mechanisms that operate on these fronts. In some cases for example, a mechanism is needed to select one or multiple solutions from the Pareto front itself. This task can prove to be difficult since, per definition and as already mentioned, all members of the Pareto front are non-dominated by other solutions and thus each represent a desirable tradeoff between the objectives that have to be optimized. One approach to accomplish such a selection is the so-called Crowding Distance (CD). Deb et al. used this operator to estimate how many solutions are in the neighborhood of another solution [9]. In general, this concept is defined as the average distance between one solution and its two nearest solutions along each objective. The authors used this operator in an evolutionary algorithm to select solutions for the next generation from the Pareto front. Crowding Distance was used in this context since for an evolutionary algorithm, it is important to keep diversity in its solutions to not prematurely converge towards a local optimum. In other words, Crowding Distance is used as a diversity preservation method. When we are faced with a Pareto front of some sort, members can be clustered in certain areas with outliers further away. It would not be meaningful to retain all clustered solutions when selecting from this Pareto front, as they are assumed to offer little variation between them. A better approach would be to calculate the Crowding Distance for each member and then take those solutions that have the largest CD, since this ensures, that the selected solutions are not too similar to each other, preserving diversity. As mentioned, the CD is defined as the average distance between one solution and its two nearest solutions along each objective, which leads to the question, what happens with solutions that do not have two neighbors. These solutions are given the maximum crowding distance in this case, since these are known as boundary solutions and when ensuring that these solutions are selected every time, we prevent the Pareto front from "shrinking" inward.

### 2.6.3 Hypervolume

Another concept that is often used when speaking about Pareto fronts is the so-called hypervolume or hypervolume indicator. Shang et al. describe the hypervolume (HV) as a performance indicator for a set of solutions obtained

by an evolutionary multi-objective algorithm [31]. The concept revolves around setting a reference point that is typically not part of the Pareto front, after which the area between the Pareto front members and this reference point is measured. According to Shang et al. the bigger the hypervolume of a given set of solutions in comparison with others using the same reference point the better.

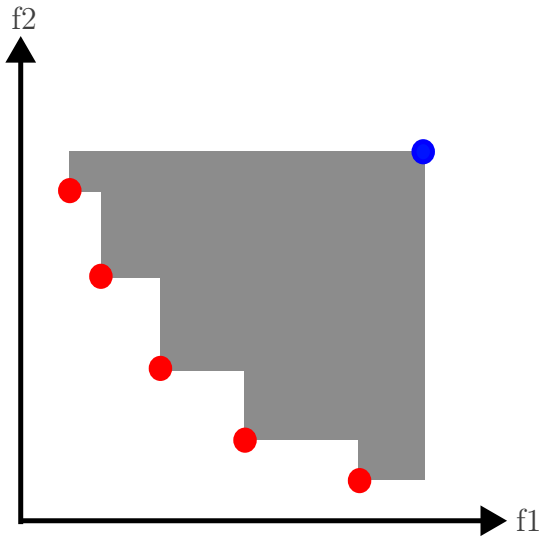


Figure 2.1: Graphic Representation of the Hypervolume Indicator

In figure 2.1 you can see a graphic illustration of the HV. The red dots in this graphic are members of the Pareto front for a minimization problem with objectives $f1$ and $f2$ with the blue dot being the reference point. These together then form the gray area, which indicates the hypervolume of this point configuration. Furthermore, the authors define the HV as

$$HV(A, r) = \mathcal{L}(\bigcup_{a \in A}(b | a \succeq b \succeq r))$$

where $A$ represents a set of points, $r$ is the reference point and $\mathcal{L}$ describes the Lebesgue measure. Note that in the paper of Shang et al., this concept is applied to a maximization problem, hence the $\succeq$. For a minimization problem, this has to be changed to $\preceq$, since it denotes the domination relationship between the reference point and the Pareto front. Keeping that in mind, the authors also describe another important concept, being the hypervolume contribution. As the name suggests, this is a measure for each individual point

of the Pareto front that indicates, how much each point contributed to the HV. Shang et al. denote that this contribution is calculated as

$$HVC(p, A, r) = HV(A \cup \{p\}, r) - HV(A \setminus \{p\}, r)$$

This can be explained as subtracting the HV of the point set $A$ without the point $p$ (the point for which we want to compute the HV contribution) from the hypervolume of the whole point set, including $p$. To summarize, given two Pareto sets that are evaluated with the same reference point, the set with the larger hypervolume is considered superior. The HV contribution of a single solution determines its individual importance by measuring the decrease in HV when that solution is removed from the set.

### 2.6.4 Epsilon Domination

We have now discussed two concepts that are applied to a Pareto front that is found using Pareto dominance. This is only one possible domination criteria. Since dominance itself can be defined differently, it is only natural, that different dominance criteria and methods exist. One of these methods is the so called $\epsilon$ domination. This concept was introduced by Laumanns et al. in 2002 [23]. In this paper, the authors use $\epsilon$ domination in the context of multi-objective evolutionary algorithms, to overcome the problem of those algorithms not having a proof of convergence to the real Pareto set. Here the "real" Pareto set refers to the set of definitive optimal solutions to a problem. Normally, evolutionary algorithms only approximate the Pareto set and those approximated sets are not always equal to the true Pareto set. In their work, Laumanns et al. define $\epsilon$ dominance as $f$ $\epsilon$-dominating $g$ for some $\epsilon > 0$ if for all objective values $i$ $(1 + \epsilon)f_i \geq g_i$. Rather than comparing raw objective values between candidates using the $\geq$ or $\leq$ operators as in Pareto dominance, the objective values of one solution vector are multiplied by a factor $\geq 1$ prior to comparison. This allows nearly identical solutions to dominate one another even when standard Pareto dominance would not, resulting in smaller yet more diverse Pareto fronts. The fronts are more diverse in this case, since, with this concept, very similar solutions are basically treated as equal, which leads to one them dominating one another so that only one of the similar solutions is kept. That keeps the front smaller and stable.

### 2.6.5 Epsilon Grid Archiving

In the same paper that we used as a reference for epsilon domination, Laumanns et al. describe an adjacent concept that works on two levels [23]. On the first level, their approach discretizes the search space by dividing it into "boxes". The Idea here according to the authors is that each member of the set of points uniquely belongs to one box, so no point can belong to two of these boxes at the same time. In their paper the authors did not start out with a Pareto set of points, so they apply a generalized dominance relation to these boxes to keep only non-dominated ones, which lets them inherently form a Pareto front. On the second level they make sure that only one element of the solution set per box is kept as a representative. This is enforced using a dominance relationship between the points in each box, where only the non dominated representative is kept. Laumanns et al. point out that this is a way to keep the diversity of solutions while converging to the Pareto set. They also describe an addition to this strategy where an archive of solutions can maintain a fixed size by dynamically adjusting the $\epsilon$ value used for determining the box size. This concept is known as Adaptive Epsilon Grid Archiving and will be explained in the methodology section in more detail.

## 2.7 Z-Score Normalization

Since we have now discussed everything revolving around Pareto fronts that we will need for this thesis, we now need two more elementary concepts that are normally used in multi-objective optimization problems. The first of these concepts is the so-called z-score normalization. A common issue when using objective values for decision-making is that of mismatching scales between objectives, meaning that some objective values may be of a large magnitude while others are very small. In terms of Pareto dominance or dominance criteria in general, if we have an objective $A$ that has this large scale and objective $B$ with a very small scale, "small" fluctuations in objective $A$'s scale can have a larger impact on the dominance relationships, than small fluctuations on the already small scale of objective $B$. This can lead to mainly objective $A$ being optimized while objective $B$ is neglected. Furthermore, this can also influence metrics like the hypervolume since the larger the scale of an objective is, the larger the resulting HV of the Pareto front will be. To prevent this, normaliza-

tion techniques are used and one of these techniques is aforementioned z-score normalization. This is a widely used technique so the paper from Henderi et al. from which the following formula is taken is a mere reference. Other references can be found in many textbooks. Henderi et al. describe that z-score normalization is based on the mean and standard deviation of the data itself [18]. The formula for this normalization according to the authors is

$$X_{new} = \frac{X - Mean(X)}{StdDev(X)}$$

, where $X_{new}$ is the new normalized and $X$ the raw objective value. In other words, z-score normalization computes how far the current objective value is from the mean of its distribution, with standard deviations as the unit of measurement.

## 2.8 Min-Max Normalization

Another and also more elementary normalization technique is the min-max normalization. Since this is widely known, the concept itself can be found in many books such as the one from Han et al. [15]. The formula used for this normalization approach is

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where $x'$ is the normalized value, $x$ the raw value and $\max(x), \min(x)$ the respective maximum/minimum of that value that can be found among all datapoints of the set. This approach basically scales the raw value by using the value ranges to a range between zero and one.

# 3 Methodology

To begin this chapter, we will give an in depth description of the problem at hand, to get a grasp on what we are actually trying to solve. Afterwards, we will explain the A* pipeline as one approach to solving the problem, followed by a description of our Monte Carlo Tree Search approach. This includes a description of the algorithm itself as well as different node selection and simulation/rollout approaches for MCTS. To finish this chapter, we will go over the experimental setup that is used to conducted the different experiments.

## 3.1 Problem Definition

As stated in the introduction chapter, in this master thesis we want to navigate Path Influenced Environments. Drawing from the works of Nübel et al. [30] and Heise et al. [17], Path Influenced Environments can be characterized as environments which an agent alters through its movement, typically by moving obstacles out of its path, so that the state of the environment depends on the path taken. The environment itself is designed as a grid, in which each cell has a predefined weight that is based on the map generation principle and will be a number $0 \leq w \leq 1$. When $w = 0$ the cell will be treated as empty and no shifting action needs to be performed when the agent moves into this cell. Otherwise, the agent performs a shift by moving to the next cell and deciding the direction in which the weight in that cell must be moved. More formally, a target cell is selected and the weight $w'$ of the target cell will get increased by the $w$ leading to

$$w'_{new} = w'_{old} + w$$

In this case the weight $w$ at the new position of the agent is then set to 0 since the weight was completely shifted into the other cell. There is no upper limit for how dense/heavy cells can get. For both the two-phase A* pipeline and Monte Carlo Tree Search, we will consider a von Neumann neighborhood.

This concept was introduced by John von Neumann in 1966 in his work on self-reproducing automata, where he defined the neighborhood of a grid cell as consisting of the central cell and its four orthogonally adjacent neighbors: above, right, below, and left [35]. Furthermore, agents are not always allowed to shift the obstacle into all the von Neumann Neighborhood cells. We restrict all movement to the boundaries of the map so the new position of the agent and the obstacle must satisfy the following condition

$$0 \leq x_{new} \leq dim_{environment} \wedge 0 \leq y_{new} \leq dim_{environment}$$

In this thesis, we want to navigate these kinds of environments in the form of a round-trip. This means that the path that the agent takes starts at a predefined starting position, moves towards a waypoint until it reaches this position, and then returns to the original starting point. While creating the path, we want to balance two objectives, with them being the length of the path itself (heron after this is used synonymously with "number of steps") and the amount of weight that has to be shifted along the path. Both of these objectives should be minimized. Being a roundtrip problem influences the formula for the Manhattan distance, which will later be used for the MCTS approach, since the full trip is now not only defined the distance from agent to goal. Instead, the distance is defined as:

$$d = \begin{cases} d_{agent \to waypoint} + d_{waypoint \to start}, & \text{if waypoint not visited} \\ d_{agent \to start}, & \text{otherwise} \end{cases}$$

Here we define the Manhattan distance as $d$ and see that, as long as we have not visited the waypoint or "turn-around-point", we add the distance between the agent and this point to the total distance between the starting position and the waypoint. This task itself differs from the experiments that Nübel et al. have performed, since they only traveled to the goal and not back to the starting position [30]. The roundtrip is what makes this problem more complex since now the shifts should not be performed arbitrary and rather free a path that can be used as the preferred way back to the start, since this would avoid shifting more weight. Later on in this thesis, specifically in the experimental setup section, we will discuss how the environments that we use are designed and what challenges they pose.

## 3.2 Monte Carlo Tree Search for Path Influenced Environments

After gaining an understanding of the problem we want to solve in this thesis, we will continue with explaining our chosen Monte Carlo Tree Search approach. We will start by giving a basic explanation of the MCTS algorithm itself, followed by an explanation of the Pareto archive that is used in this thesis. The section will continue with discussing the node selection as well as the simulation methods for new nodes that are used in this thesis.

### 3.2.1 Basic Algorithm Explanation

To explain the algorithm and how we implemented it, we want to first provide the pseudocode shown in algorithm 2.

**Require:** $root\_node$, $total\_budget$, $per\_sim\_budget$, $simulations\_per\_child$,$rollout\_function$, $root\_selection\_function$, $tree\_selection\_function$
**Ensure:** $solution$
1: $current\_root \leftarrow root\_node$
2: $solution \leftarrow$ `None`
3: **while not** $current\_root.is\_terminal\_state()$ **and** $current\_root.\_depth < 1500$ **do**
4: $used\_simulation\_counter \leftarrow 0$
5: **while** $used\_simulation\_counter \leq total\_budget$ **do**
6: $current\_node \leftarrow tree\_selection\_function(current\_root)$
7: **if** $current\_node \neq$ `None` **then**
8: $child \leftarrow current\_node.expand()$
9: **if** $child \neq$ `None` **and not** $child.is\_terminal\_state()$ **then**
10: $used\_simulation\_counter \leftarrow used\_simulation\_counter + rollout\_function(child, simulations\_per\_child, per\_sim\_budget)$
11: **end if**
12: $backpropagate(child, current\_root)$
13: **else**
14: $used\_simulation\_counter \leftarrow used\_simulation\_counter + per\_sim\_budget$
15: **end if**

```
    end while
    current_root ← root_selection_function(current_root)
    prune_siblings(current_root)
    if current_root.is_terminal_state() then
        solution ← current_root
    end if
end while
return solution
```

Algorithm 2: Implemented Monte Carlo Tree Search

### Tree Structure

Before we start going over the different phases of the algorithm, we want to describe what exactly the nodes and edges of the tree stand for. Each node symbolizes a state of the environment, which includes the positions and weights of the obstacles at this specific point in time, as well as the agent and its collected metrics. An edge symbolizes a movement and shift combination, so it gives us the direction in which the agent's movement occurs and the direction in which the obstacle at the new position would be shifted.

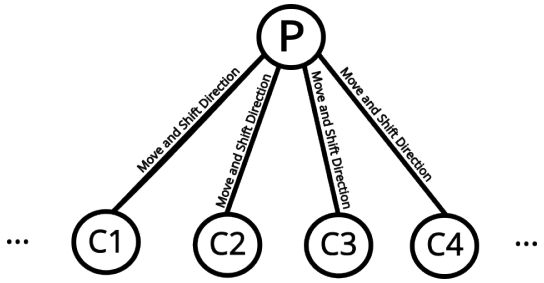


Figure 3.1: Parent and Children Nodes in the MCTS Tree

In figure 3.1 you can see the resulting tree structure. We have one parent, which is given by the node with a **P** inside, that has several connections or edges to children, which are marked with **C**s. Each edge represents a combination of a movement and a shifting action. In a von Neumann Neighborhood [35], this would give us $4 \cdot 4 = 16$ possible children of a node overall, with four possible movement directions and four directions in which to shift the obstacles. For now, it is sufficient to state that the depth of a node equals the number of steps

and shift operations performed. Knowing that, we impose a depth limit of 400 for the root node, meaning that if 400 steps and shifts have been performed, and the root is not in a terminal state, the algorithm stops. In the case of a $50 \times 50$ environment, where the corresponding start and goal position are on the opposite sites sharing at least one axis, this would be approx. four times the roundtrip distance of 98. Therefore, it is reasonable to assume that 400 steps is a big enough budget to facilitate all optimal paths. When the current root reaches a terminal state, the final path is reconstructed by tracing back from that terminal node to the root of the search tree via parent pointers. Now that we know the structure of the tree, we can describe the different phases which dictate how the tree is constructed.

#### Initialization

We start by setting the $current\_root$ to the new $root\_node$ and set the found solution $solution = None$.

#### Selection and Expansion

While the current root is not in a terminal state or until we reach a terminal depth of 400 we repeat the steps of MCTS in a loop. A terminal state is defined as a state where the agent started at the specified starting position, paved its path to the goal, collected it and then returned to the start. Inside this loop, we set the $used\_simulation\_counter$ to zero with each iteration. This variable is used to count how many total steps in simulations were done during the rollouts. As long as this counter is smaller than the $total\_budget$ we select the $current\_node$ using the specified $tree\_selection\_function$. The different functions that are used in our experiments are later explained in the selection approaches section. These functions perform a heuristics guided traversal of the tree from the $current\_root$, selecting either a leaf node (with no children) or a partially expanded node. Continuing in the algorithm, when we have found such a node with either none or not all possible children, we save it as $current\_node$ and expand it. The expansion process consists of multiple steps. We start by determining if there are any legal shifting and movement combinations missing as children of $current\_node$. Such a combination is legal, if the movement as well as the shifting results in a valid position for agent and obstacle within the boundaries of the map. It can be the agents old

position, since we allow the agent to shift the obstacle into all directions of the obstacles von Neumann neighborhood. If the agent shifts the obstacle to its old position, it can be imagined as the agent lifting the obstacle over itself. When we find such a combination, we create a new node and save it as *child*.

### Simulation

If this *child* is not in a terminal state (from which we do not want to simulate further) and is also not *None*, we perform $simulations\_per\_child$ simulations from the state of the *child* node and each simulation gets a budget of $per\_sim\_budget$. We do multiple simulations since most heuristics used for these rollouts rely on randomness and with our approach we allow multiple random rollouts to increase the probability of a good rollout being performed. Across all simulations, we then build a Pareto front of the best results and choose in most cases semi randomly, which will be discussed further when we explain the different rollout functions. In the case that we found a valid new *child* and performed the simulations, the metrics of the best simulation become the stored metrics of the *child* node and these values are then propagated from the new child to the root.

### Backpropagation

During the backpropagation, we do an incremental average update [38], which means that each metric is updated using the following formula

$$value = \frac{current\_value \cdot (current\_visits - 1) + new\_value}{current\_visits}$$

These updated metrics are the key to traversing the tree, since all nodes reflect how good their respective subtrees are by giving an average of the metrics of their subtrees. Another process that happens during backpropagation is the maintenance of what we call the Pareto path archive. Each node in the tree stores an archive of the best paths that were found in its subtrees. When a new node is simulated, its values are given to the parent node which uses Pareto dominance to determine if other nodes in the archive are now dominated by the new node. If this is the case, the dominated nodes are removed. Overall, this allows us to employ different archive based selection techniques during tree traversal, which will also be described later in this

chapter. When the simulations were executed, we add the number of steps actually used in the simulations to the *used_simulation_counter*. A step here refers to the combination of a movement and a shifting action. Simulations can conclude earlier and not use their whole budget when they, i.e. complete the roundtrip. If we either did not find a valid child or the new child already starts in the terminal state, we add the complete *per_sim_budget* to the *used_simulation_counter*. This is done to progress the algorithm, even if we do not find new solutions or nodes to expand. When the budget is used up, we use a specified $root\_selection\_function$ to select the new root for the tree among the children of the current root. After setting the new root, we prune its siblings from the parent to limit the total size of the tree, which is done purely for performance optimization purposes. It is possible to do this since after selecting the new root, we will never traverse the siblings of the new root again to find solutions using this algorithm. This is then repeated until we find a root node that is in a terminal state or the depth limit of 400 is reached. When the algorithm is concluded we can have two possible values for *solution*. These can either be a node, meaning that we found a root node in a terminal state, or it can be $None$. In the case that we found the node, we reconstruct its path (including the shifts) from the node up to the original root and in the case that it is $None$ we take it as the approach not finding a path in the specified boundaries/parameters of the experiment. Even though we described the algorithm in greater detail here, we did not mention the different selection and rollout functions. These, as well as the mentioned Pareto archive, will now be described in the next subsections, to provide a structured understanding of how everything works and what the differences between the approaches are.

### 3.2.2 Archive

The concept of the Pareto archive was already mentioned earlier, and now we want to describe it in more detail. As we plan to employ selection methods based on hypervolume and crowding distance, which will be explained in the next subsection, we need to discuss how we plan to integrate those into the MCTS approach. Both HV and CD are methods that are used on Pareto fronts which leads us to a problem when trying to apply them to tree traversal where we pick from the children of a designated parent node. This problem arises due to the children often not forming a Pareto "front" since it can happen

that i.e. only one child dominates all others or different solutions having the same objective values. To remedy this issue, we made an addition to each node, which we call a Pareto archive. When we traversed the tree, found a new node and did the necessary rollouts, we start backpropagation. During this backpropagation, we build the taken path from the newly added leaf to the root of the tree. This path is at each new parent node associated with its objective values and checked against this parent node's Pareto archive. If it Pareto dominates other paths that are already in the archive and is not dominated by others, it is accepted into the archive and the other dominated paths are removed. Otherwise, if it is dominated by any path in the archive it is not accepted. This process enables us to have an archive of non dominated paths at each node and in turn allows us to use hypervolume and crowding distance selection approaches using this archive. It becomes clear that, if we were to use raw or averaged values for the paths, early nodes near the root would always dominate nodes that are deeper in the tree. To fix this, we do not compare raw objective values and instead introduce a so called to go metric at each node. The to go value is calculated as

$$t = l - n$$

where $t$ symbolizes the to go value, $l$ the raw objective values of the newly created leaf and $n$ the raw objective values of the current node. This gives us the ability to compare nodes across different depths of the tree and fixes the issue of nodes near the root always dominating deeper nodes. To keep the size of this archive below a maximum of 20 solutions, we employ a technique called adaptive epsilon archiving, which is a specific form of the epsilon grid archiving presented in the related work section. Instead of a fixed $\epsilon$ value, we start with an $\epsilon$ value of $10^{-4}$ and increase it by this value with every iteration until the desired number of representative children remains. Overall this can be seen as an archive maintenance step.

### 3.2.3 Selection Approaches

Now that the archive has been explained, we can continue to discuss which selection approaches were used for tree traversal and choosing the next root node. In this subsection we will discuss all used mechanisms starting with UCB1 and hypervolume based selection and ending with a crowding distance

based selection as well as a hybrid approach based on adaptive epsilon archiving.

### UCB Selection

The first selection mechanism we want to go into further is "Upper Confidence Bound 1" (UCB1). This concept was already explained in the related work section, so here we will concentrate on the changes made to apply it to the problem at hand. Instead of maximizing a score, like in the K-armed bandit problem, we want to now minimize multiple objectives. We aim to minimize the length of the chosen path and the amount of weight shifted along the way, together with the remaining roundtrip distance as we have explained previously. Keep in mind that the remaining distance is more of a heuristic that should lead the algorithm towards a possible solution. Using these three metrics for all children which we want to select from, we calculate a vector of three UCB values for each child. Before the calculation of these values, we normalize the objective values using z-score normalization, which was already explained in the related work chapter. We need to normalize these values, since otherwise different value ranges for these metrics might lead to one of these metrics dictating the selection process. When we discussed UCB1 in the related work section of this thesis, we saw the formula

$$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

where $\bar{x}_j$ represents the current value of the node and $\sqrt{\frac{2 \ln n}{n_j}}$ represents the new exploration value. We can not use the formula as it is, since we have a minimization problem and multiple dimensions instead of one. Because of that, we adapted the formula to

$$\bar{\mathrm{UCB}}_j = \bar{v}_j - \bar{\alpha} \cdot \bar{e}$$

where $\bar{v}$ represents the vector of normalized metric values, $\bar{\alpha}$ a weighting vector for each dimension and $\bar{e}$ the vector of exploration values that is calculated as

$$e_{metric} = \sqrt{\frac{2 \cdot \ln n}{m}}$$

Here $m$ represents the number of times the child and $n$ the number of times the parent was visited and $\bar{\alpha}$ is used as a weighting factor to have a way to

balance the importance of each dimension. In the experiments we fixed it to $\alpha_{distance} = 0.3$ and $\alpha_{weight\_shifted} = \alpha_{steps\_taken} = 1$. This, together with the subtraction of the exploratory part $\bar{e}$ from $\bar{v}$, $\bar{\alpha}$ places an emphasis on the minimization of steps taken and weight shifted. The subtraction is done since we minimize, so we subtract the exploration value from the normalized value to lower it. After we have calculated all necessary values for all children, we use Pareto dominance to identify the non dominated individuals. From those non dominated children we choose one at random to return as the selected node.

### Hypervolume Selection

While UCB selection works already with the vanilla version of Monte Carlo Tree Search, the next selection method we want to look at does not. The concept of hypervolume was described in the related work section of this thesis as the measurement of volume between a Pareto set and a specified reference point. Additionally, the HV contribution of a single Pareto front member can be determined by computing the difference between the hypervolume of the front with and without that member. To select a child node during tree traversal, we make use of the previously described Pareto path archive of the parent node. As explained, this archive stores the to-go values at every point of the path and to make use of them we normalize these values. We need to normalize them since when looking at step count and weight shifted, it is possible, that one of these values becomes much bigger than the other, which can either happen due to many cells in the path not containing an obstacle, because it was moved already or there was never one to begin with. Furthermore, it can also occur that an obstacle is pushed multiple times, becoming increasingly heavy, which could make the weight shifted value much larger. This can heavily influence the hypervolume and to avoid this issue we use min-max normalization on the to go objective values. We then compute a reference point, which, in general, this point needs to be worse than all the points for which we wish to calculate the HV. The reference point is defined as

$$ref = w + 0.1 \cdot (r + 10^{-12}),$$

where $w$ is the vector of worst normalized objective values per dimension and $r$ denotes the corresponding value ranges. These value ranges are computed by taking the difference between the highest and lowest value in each objective.

Using that formula we are able to consistently create a reference point that fits the described criteria. Now that we have everything that is needed for the hypervolume calculation, we compute the hypervolume contribution for each point. The contribution values are then used for a roulette wheel selection, where greater contributions to the HV lead to a better chance to get selected. Using this method, one of the children is chosen at random. In this setting, the hypervolume contribution can be utilized, since the dominance check during backpropagation, which determines whether a newly discovered path is included in the archive, guarantees the maintenance of a proper Pareto front in the Pareto archive.

### Crowding Distance Selection

Hypervolume selection is not the only technique in this thesis that needs a Pareto front to be employed, because the next selection technique we want to discuss is the crowding distance selection. We have discussed the general concept of crowding distance as a diversity preservation method in multi-objective evolutionary algorithms in the related work section and want to now show how it was used and implemented in our Monte Carlo Tree Search approach. At this point, we again use the Pareto archive to apply this concept because it allows us to calculate the crowding distances for each point of the front, which we then plan to use as weights for another roulette wheel selection. For the calculation of the CDs, we use min-max as a normalization technique for the to go objective values of each dimension. There is an important issue that we have to discuss with this, which comes in the form of the extreme points of the front, which, in practice, have an infinite crowding distance, since they by definition only have one neighbor. To handle this problem, we need a way to assign finite weights to these extreme points. We do this by summing up the crowding distances of all points in the front, that are not extreme points. The extreme points then both receive 10% of this sum, while the remaining 80% are distributed among the remaining points according to their contribution towards the total sum. To compute the exact weight of the exact weight of the non-extreme solutions, we are using the formula

$$w_i = \frac{\mathrm{CD}_i}{s} \cdot 0.8$$

where $w_i$ is the weight for point $i$, $\mathrm{CD}_i$ the original crowding distance for point $i$ and $s$ the sum of crowding distances for all non-extreme points. This fixes

the issue with infinite CDs and makes us able to use the proposed roulette wheel selection.

### Adaptive Epsilon Archiving Selection

The last selection method that we propose is a new approach that is based on adaptive epsilon archiving and to our knowledge it was never used prior to this thesis. For this method, we again make use of the Pareto path archive, by first employing adaptive epsilon archiving to retrieve a specified number of representative paths from the archive. This number is not fixed but instead increases with the number of visits received by the node from which the selection is performed. We use the formula

$$\#paths = \max\left(2, \lfloor\sqrt{visits_n}\rfloor\right)$$

to achieve this behavior, where $visits_n$ stands for the number of visits of the current node $n$. With this formula, we receive at least two representative paths of which the values are then normalized via min-max normalization. These normalized values are then used to calculate a quality term for this solution. This term is the inverse L2 norm, which is calculated using

$$quality = \frac{1}{||\vec{v}||_2 + \theta}$$

where $\theta$ is set to $10^{-9}$ to circumvent accidental division by zero. Besides the quality term, we also calculate an exploration term that is calculated using the formula

$$\text{exploration} = 0.5 \cdot \sqrt{\frac{\log\left(\text{parent-visits}\right)}{\text{child-visits}}}$$

, which is similar to UCB. In this context, the parent visits correspond to the number of visits to the current node from which a child is selected, while the child visits refer to the child associated with the representative path. This formula helps with exploration since it will produce higher values if a child node was not visited often. Together, these two terms are added up to a weight for the corresponding representative path, which is then used for roulette wheel selection. This method combines the diversity preserving idea that comes with the adaptive epsilon archiving approach and the exploration exploitation balance, which is beneficial to MCTS.

### 3.2.4 Rollout Approaches

In this subsection, we are going to discuss the rollout approaches that were used in this thesis. Rollout in this case refers to the simulations that are done when a new node is created after tree traversal. There is a difference in how you can approach these rollouts which is indicated by the use of the words "light" or "heavy". Heavy rollouts make use of heuristics while light rollouts do not, which is why some methods are prefixed with "heavy" in front of their name.

#### Light Rollout

The light rollout is the simplest concept that can be used. Here we start in the state that the node is currently in and consecutively sample and execute random moves and shifts. These moves and shifts have to be valid, which means that the new position of the shifted obstacle, as well as the new position of the agent have to be inside the boundaries of the map. This is done until either a terminal state is reached, which would be the completion of the roundtrip that the agent should ultimately perform, or the budget for the rollout, which indicates how many movement-shifting combinations can be executed in this simulation, is used up.

#### Heavy Distance Weight Rollout

For this rollout method we use the distance for the remaining roundtrip and the weight as heuristics to decide which move we want to use during the rollout. To do this we again start from the current state of the agent. From this state we start with all possible valid moves and out of those, we pick the ones that decrease the distance to the goal. After determining these moves, we compare the weight that would need to be shifted and pick the one with the lower weight. If we have, i.e. two moves with the same weight shifted, we would just pick one of them at random. There is a problem with the distance selection in here, which stems from us using the Manhattan distance in combination with the von Neumann neighborhood. When at least one axis of the agent is aligned with an axis of the goal, i.e. they have the same x-coordinate, there would only be one move in the von Neumann neighborhood that decreases the distance to the goal, which would lead to a straight path from the current

position to the goal, completely disregarding the weight aspect. As a remedy, we also consider moves that increase the distance to the goal as long as the agent is aligned with an axis of the next goal point, thereby breaking the axis alignment.

### Heavy Square Sampling Rollout

The heavy square sampling rollout method uses mainly the remaining distance to the goal as the deciding heuristic. From the current position, we sample in a square around the agent. The edges of the square are all equally far away from the agent's position, which in our case is a fixed radius of five cells. This makes each edge of the square 10 cells long, covering 100 cells in total. If we were to be at a position where not all 100 cells would be valid due to being at the edge of the environment, we automatically adjust the square as such that we only include valid cells and basically cut off the rest of the non-viable ones. Inside this square, we then sample five different cells and use the Manhattan distance to determine which of them is closest to the goal. If there is a tie, we break it by random selection. After the sampled point is selected, we then use a greedy strategy to move to this sampled point, which is similar to the heavy distance weight rollout method. At every step we get the valid movement directions and pick the one that is closest to the sampled point and break ties with random selections. Here we do not use specific shifts and instead select one at random. When the current waypoint is in the sampling radius, which is either the goal or the starting point, depending on what phase of the roundtrip we are in, we instead go directly to that point using the described strategy.

## 3.3 Two Phase A*

Having examined our first proposed method in depth, we now turn to the A* pipeline as a second approach to solving the problem. Path influenced environments are a novel field of research with few tested methods. The only two approaches applied to these problems so far have been evolutionary algorithms and an adapted version of A* [30][33]. Since we are trying to solve a multi-objective problem by minimizing the number of steps and the weight shifted on the path, we first thought to use multi-objective A*. The problem with multi-objective A* becomes apparent, when we look at the challenge at hand.

With every node, we would have to enter 16 new nodes into the open list, since we can move in four directions within the von Neumann neighborhood and at each of these new positions we can shift the obstacle again in four directions within their neighborhood. Each move into a non-empty cell and the occurring shift changes the board state, which then can not be compared with other states that have different weight configurations on the board. This gives us a very large amount of possible nodes in the open list and since we want the whole Pareto front of solutions, computing them would take an infeasible amount of time. Instead, we opted into using a novel two-phase A* approach, which will be described in the following subsections.

### 3.3.1 Phase One: Shortest Path

The first phase of this approach focuses on determining the minimum number of steps required to complete the roundtrip within a given environment. The goal of this phase is to calculate the shortest amount of steps that a path could take. In our scenario, this could typically be achieved using only the Manhattan distance, as the map contains only movable obstacles. However, since we aim to propose a framework applicable to arbitrary maps, we employ A* for this phase.

1: $start\_state \leftarrow (start, false)$
2: $OPEN \leftarrow$ priority queue
3: $counter \leftarrow 0$
4: push $(0, counter, start\_state)$ into $OPEN$
5: $CLOSED \leftarrow$ empty map
6: **while** $OPEN$ not empty **do**
7: $(f, \_, pos, goal\_collected, steps) \leftarrow$ pop smallest element from $OPEN$
8: $state \leftarrow (pos, goal\_collected)$
9: **if** $state \in CLOSED$ **and** $CLOSED[state] \leq steps$ **then**
10: **continue**
11: **end if**
12: $CLOSED[state] \leftarrow steps$
13: **if** $goal\_collected$ **and** $pos = start$ **then**
14: **return** $steps$
15: **end if**
16: **for all** $next\_pos \in validMoves(pos)$ **do**
17: $next\_goal\_collected \leftarrow goal\_collected$

```
18:         if next_pos = goal then
19:             next_goal_collected ← true
20:         end if
21:         next_steps ← steps + 1
22:         if not next_goal_collected then
23:             h ← Manhattan(next_pos, goal) + Manhattan(goal, start)
24:         else
25:             h ← Manhattan(next_pos, start)
26:         end if
27:         counter ← counter + 1
28:         push (next_steps+h, counter, (next_pos, next_goal_collected, next_steps))
    into OPEN
29:     end for
30: end while
31: return ∅
```

The pseudocode shown above accurately portrays how this first phase works. We start by creating a *start_state* which is composed of the current state and a boolean marker that indicates whether we have collected the as "goal" designated waypoint or not. This boolean indicator is needed, since it represents in which phase of the roundtrip we are currently. $OPEN$ is implemented as a priority queue, i.e., a queue in which elements are ordered according to a key that assigns a priority to each entry. Our primary key that we use for sorting is our $f$ value which is the sum of the current objective value $g$ and the heuristic value $h$. This heuristic value measures the remaining roundtrip distance from the current position using the Manhattan distance. The secondary key that is used here is a counter. It is set to zero before the loop and then incremented on each loop. This counter serves as a tie-breaker in case that the $f$ value of two nodes in the queue is identical. We push the starting entry $(0, counter, start\_state)$ into this queue. At this point it is not necessary to calculate the real $f$ value for the start node, since when we start the loop, we instantly pop it out of the queue. After popping a node from the queue we inspect its state. If there is a node already in closed, that is in the same state but has smaller or equal steps to the popped node, we simply continue with the next iteration of the loop. In the case of the new node being better, the one that is already in closed gets replaced by it. Otherwise, we insert the node into the closed list with its state as the key and the steps taken as the value. Furthermore, if this new node visited the waypoint and returned to the

start, it completed the roundtrip and the amount of steps is returned. If this is not the case, we calculate all possible and valid next positions. A position is seen as valid, if both coordinates of the position are within the bounds of the environment. For each of these valid positions, the waypoint collection status is evaluated, and both $h$ and the number of steps taken are computed. After that, these children are pushed into the open queue. This loop either ends when we hit the return statement when the first solution was found or when $OPEN$ is empty, which would mean that there is no solution.

### 3.3.2 Phase Two: Weight Minimization

Having determined the minimum number of steps required to solve the map, we can now optimize the shifted weight based on this step limit, as illustrated in the pseudocode for the next part of our A* pipeline.

**Require:** step limit $L$
1: $OPEN \leftarrow$ priority queue ordered by accumulated weight
2: $counter \leftarrow 0$
3: $startNode \leftarrow \text{Node}(start, false, 0, 0, null)$
4: push $(0, counter, startNode)$ into $OPEN$
5: $CLOSED \leftarrow$ empty map
6: **while** $OPEN$ not empty **do**
7: $(\_, \_, current) \leftarrow$ pop smallest element from $OPEN$
8: **if** $current.steps > L$ **then**
9: **continue**
10: **end if**
11: $remaining \leftarrow L - current.steps$
12: **if not** $current.goalCollected$ **then**
13: $minNeeded \leftarrow Manhattan(current.pos, goal)$
14: $+ Manhattan(goal, start)$
15: **else**
16: $minNeeded \leftarrow Manhattan(current.pos, start)$
17: **end if**
18: **if** $minNeeded > remaining$ **then**
19: **continue**
20: **end if**
21: $state \leftarrow (current.pos, current.goalCollected, current.steps)$
22: **if** $state \in CLOSED$ **and** $CLOSED[state] \leq current.weight$ **then**

```
23:         continue
24:     end if
25:     CLOSED[state] ← current.weight
26:     if current.goalCollected and current.pos = start then
27:         return current
28:     end if
29:     for all nextPos ∈ validMoves(current.pos) do
30:         nextGoalCollected ← current.goalCollected
31:         if nextPos = goal then
32:             nextGoalCollected ← true
33:         end if
34:         nextSteps ← current.steps + 1
35:         cellWeight ← weight(nextPos)
36:         nextWeight ← current.weight + cellWeight
37:         nextNode ← Node(nextPos, nextGoalCollected,
38:                         nextSteps, nextWeight, current)
39:         counter ← counter + 1
40:         push (nextWeight, counter, nextNode) into OPEN
41:     end for
42: end while
43: return ∅
```

The structure of this A* is relatively similar to the first one. We again use a priority queue for $OPEN$ but this time we use the accumulated weight up to this position as the first key and a counter again as the tie-breaker. Inside the loop, we pop the node that is first in the queue and check how many steps it has taken till now. If it is more than the given step limit we continue with the next iteration of the loop since it can not be a solution for the current step limit. We then calculate the remaining step budget and if this is smaller than the minimum amount of steps needed to finish the roundtrip, we also skip to the next iteration of the loop. Otherwise, we get the state of the current node, which consists of its current position, the current waypoint collected status and the amount of steps taken. If there is already a node with this state in the $CLOSED$ dictionary with a smaller weight, we continue with the next loop iteration. Otherwise, if the weight of the node in $CLOSED$ is higher than the one of the new one, the node is replaced with the newer one. If not we insert the current node with the state as the key and its weight as the value in the $CLOSED$ dict. We next check if the current node has completed the roundtrip

and if so, we return it. When the roundtrip is not completed, we again insert all possible valid next positions and its values into the $OPEN$ queue. These nodes then have a pointer to their parent node, so we can reconstruct the path in the end. With that, we can get the path with the smallest amount of weight shifted for a given step length.

### 3.3.3 Shifting Optimization

It is very important to notice that we did not actually optimize for the shifting direction yet. In the second phase, we calculated the shifted weight as the sum of weights of visited cells. We did not consider that unfortunate shifting may lead to us pushing one weight multiple times, but this makes this admissible as a heuristic since we can guarantee, that this produces the lower bound of weight shifted on the taken path. Having identified a path through the environment with the specified number of steps that minimizes the transported weight, we simulate its execution using a function that, at each step, selects a shifting action that moves the weight to a position outside the path. If there is no weight in one of the cells already, the shifting direction is chosen at random, since it does not influence the path this way. This works since the path that is created by the second A* function should always result in a corridor, which makes such shifting possible. We also keep the admissible shifting cost since the agent never shifts a weight twice when applying this shifting strategy. If such a shifting is not possible, we disregard this path as not optimal.

### 3.3.4 Epsilon Constraint Search

With that we come to the last piece that connects the described mechanisms, which is the epsilon constraint search. This piece connects the two separate A* algorithms. Here we specify a maximum $\epsilon$ value which is set to max_epsilon $= 3 \cdot$ environment_dimension. This value was found to be sufficient for all environments used in the experiments of this thesis. We then compute the length of the shortest possible path using the first A* function. After that we enter a loop in which the step limit is increased incrementally until we arrive at $step\_limit = shortest\_path\_length + max\_epsilon$. For each step, we use the second A* function to calculate the path with the least amount of weight on it, for the given step limit. All of these paths then go

through the shifting optimization where the function tries to find shifts that fulfill the described criteria. Furthermore, after the shifting directions have been assigned for all paths, the solutions go through a Pareto filter, which ensures that only Pareto optimal solutions persist and that we only have paths that are dissimilar from each other. This means, that if we would have two paths that are the same in all aspects but differ in shifting directions, we would only keep one of them.

## 3.4 Experimental Setup

Now that we know the problem at hand, as well as the approaches we want to take, we are continuing with the description of our experimental setup. We perform our experiments in four different environments in which both the MCTS and the two phase A* approach will be tested. The first of these environments is named 'easy map' and serves primarily as a proof of concept. In this environment, weight is randomly distributed across cells, with one clear path containing no weight. Furthermore, this free path is the shortest between the start and the goal waypoint, which means that it minimizes both steps taken and weight shifted. This makes it the optimal path through the environment that each algorithm should follow, because it is not possible to find another path that either shifts less weight or takes fewer steps during the roundtrip.

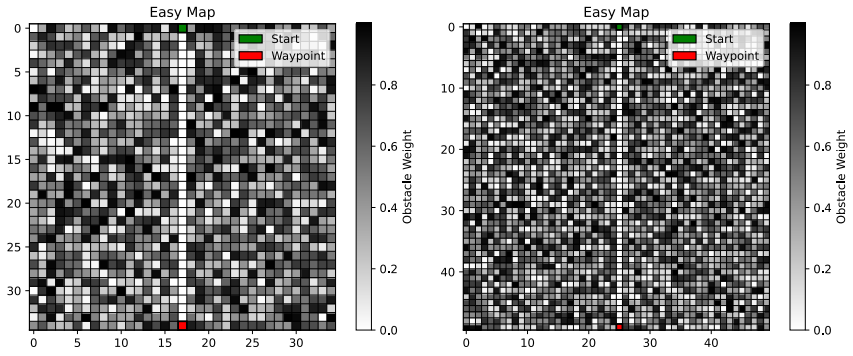


Figure 3.2: Easy Map with Dimensions 35x35 (left) and 50x50 (right)

Figure 3.2 illustrates both environments that will be used in the experiments. As explained we can see an obstacle free path down the middle with the start

being marked as green and the waypoint the agent needs to visit as red. The second environment in which we want to test our algorithms is the random environment. As the name suggests, we have a random distribution of weights across all cells, where all weights are in the interval between zero and one. The random environment is one of the most interesting cases since it poses a near unsolvable challenge for humans. There is neither an obvious path that goes through it, nor a hint on where the best paths are.

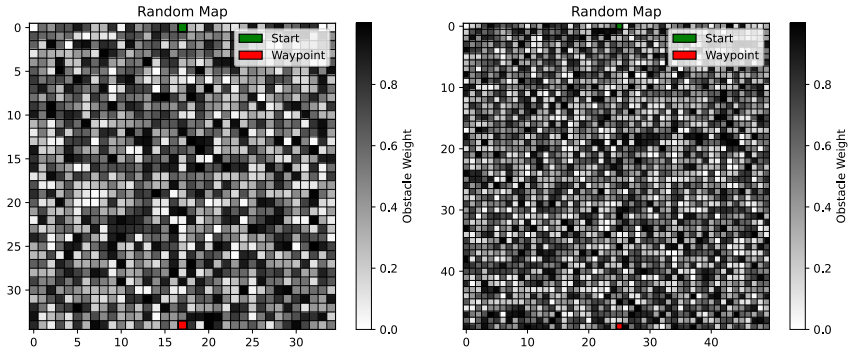


Figure 3.3: Random Map with Dimensions 35x35 (left) and 50x50 (right)

Figure 3.3 shows both versions of the random environment. As described, there is no obvious path like in the easy environment, which makes it harder to solve.
Beyond the standard test problems, two additional environments sourced from Nübel et al. [30] are considered. One of these environments is called meandering river. As the name suggests, we have a structure that resembles a river, where the obstacles in and along the river bed are very light, while the rest is very heavy. This would suggest taking a path through the river bed if shifting less weight is an objective, and the direct path between start and waypoint if we were to minimize the steps taken. According to Nübel et al. the river or the sinusoidal curve that represents the river is generated using the formula

$$u(y) = \frac{D}{2} + \frac{D}{3} \cdot \sin\left(2\pi \frac{y}{D}\right)$$

and after that a Gaussian filter is applied to the edges of the river to smooth the transition. $D$ here stands for the environment dimension.

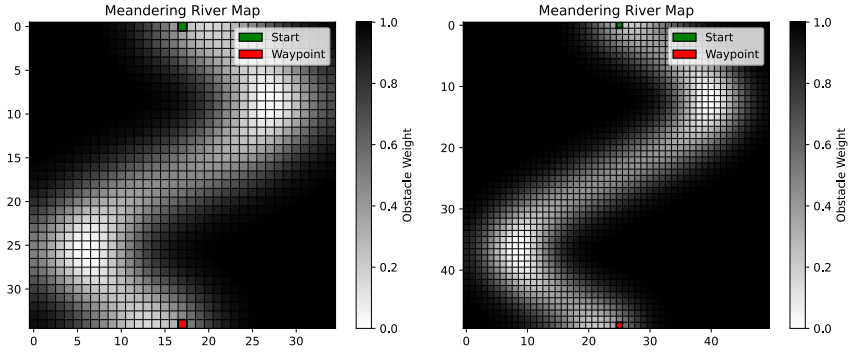


Figure 3.4: Meandering River Map with Dimensions 35x35 (left) and 50x50 (right)

In figure 3.4, we can see the described environment structure and how the "river" crosses through the environment. While the authors originally used the width and height of the environment in these formulas, these can be substituted with a single dimension, as the environments used here are square and thus have equal width and height.

The last map that is used for the experiments also comes from Nübel et al. and is named the sinusoidal environment[30]. According to the authors it can be generated using

$$s(x, y) = \sin\left(\frac{5\pi x}{D}\right) \cdot \cos\left(\frac{5\pi y}{D}\right)$$

where $D$ again stands for the map dimension. This generates a map consisting of peaks and valleys, which resembles a checkerboard pattern.

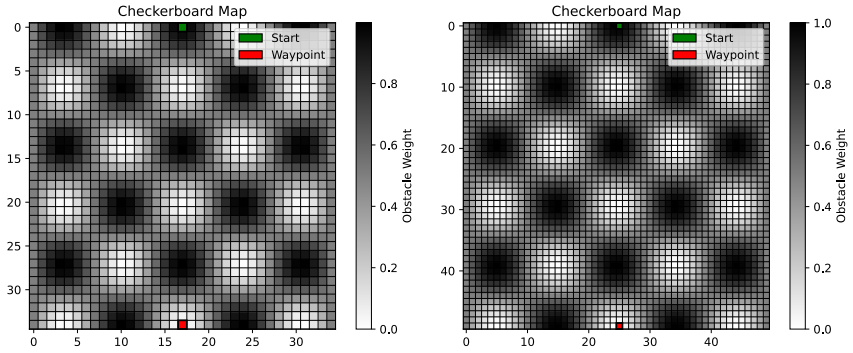


Figure 3.5: Sinusoidal Map with Dimensions 35x35 (left) and 50x50 (right)

Figure 3.5 illustrates the structure of the sinusoidal environment. It would be natural to assume that the shortest path once again runs directly between the start and the waypoint, while the path shifting the least weight navigates between the peaks.
As seen in all presented figures, we will test our approaches on all of these maps with dimensions of $35 \times 35$ and $50 \times 50$. Furthermore, we are employing the previously discussed tree traversal mechanisms, which will dictate how we select nodes on the way from the root to a node that not already has all its children. Only one root selection method will be used in this thesis and this is a hypervolume based selection. This selection method passes the children first through a Pareto filter that removes dominated children and afterwards computes the hypervolume contribution of each remaining child. The child with the highest hypervolume is then selected, with the method relying on averaged objective values that are updated with each backpropagation.
For the hyperparameters that we want to use for the experiments, we have the total budget, the budget per simulation and the number simulations that we do per child. Because the concept of MCTS was never used on path influenced environments, we ran preliminary experiments where we used a number of different values for the hyperparameters and tested them on the $35 \times 35$ random map with a single seed. The preliminary experiments were limited in scope, as parameter tuning is not the focus of this thesis and the aim was merely to identify which parameters reliably produce results. It turned out that for the total budget 200000, 5000000 and 1000000, for the per simulation budget 75, 100 and 150 and for the number of simulations per child 25, 50 and 100 produced good results. In turn, these will be the hyperparameters that we intend to use in the experiments. Overall, we will use all possible combinations of the different budgets, selection and rollout methods. These will be tested on 31 different seeds to achieve statistical significance for the experiment results. For the A* pipeline we will only do one run, since it always produces the same results in the tested environments, but it still remains to be proven that this approach is deterministic in nature.

## 3.5 Evaluation Metrics

To evaluate the results of both algorithms, we focus on the hypervolume of the obtained Pareto fronts. Before computing the hypervolume, all objective

values are normalized using min-max normalization, with a fixed reference point of $(1.1, 1.1)$ on the normalized scale. A higher hypervolume indicates a better set of solutions. For the easy environment, where a single known optimal solution exists, we additionally use the hit count, which is the number of runs in which a configuration successfully found this optimal point, as a tiebreaker between configurations of equal hypervolume. Furthermore, we track the number of runs that failed to produce any valid solution within the given step budget, referred to later as the None count, to assess robustness. We will also employ the Kurskal-Wallis test [21] in combination with the Dunn test [13] and the Bonferroni correction [6] on the obtained hypervolume values, to test for significant differences in performance between the different MCTS configurations. For the final comparison between MCTS and the two-phase A*, we overlay the Pareto fronts of both approaches and evaluate dominance relationships alongside the hypervolume.

# 4 Results

After finishing the methodology chapter, we will now discuss the results of our experiments. We will start by analyzing our baseline two phase A* approach and continue with the evaluation of the MCTS results. Afterwards we will compare the results of different Monte Carlo Tree Search configurations to determine which combinations produced the best results. At the end of this chapter we will then compare both approaches to each other to determine which approach found solutions of higher quality.

## 4.1 Baseline - Two Phase A*

We will start by evaluating the results of the two phase A*. Beginning with the easy environment, we can see that the algorithm found the optimal path through the map in both dimensions. To reiterate, what was described in the methodology chapter, this map has randomly distributed weight in all cells of the grid besides the ones on the direct path between start and waypoint.

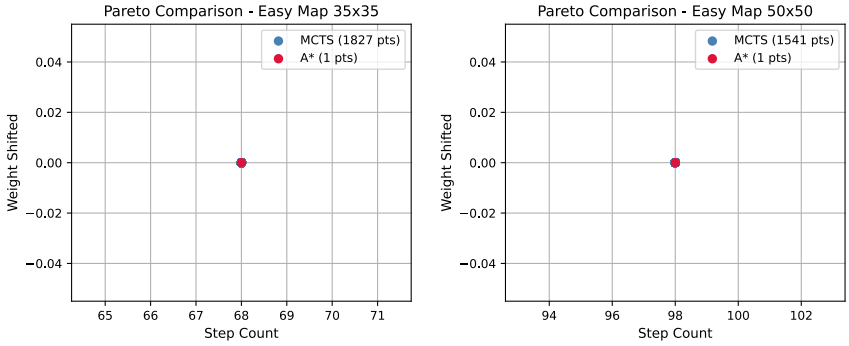


Figure 4.1: Pareto Optimal Solution by A* for Easy Map with Dimensions 35 (left) and 50 (right)

In figure 4.1 we can see the pareto optimal solution resulting from the A* approach. One can see that both results are at 68 and 98 steps respectively and that no weight was shifted along the path. That indicates that the optimal weight free path through the environment was found both times.

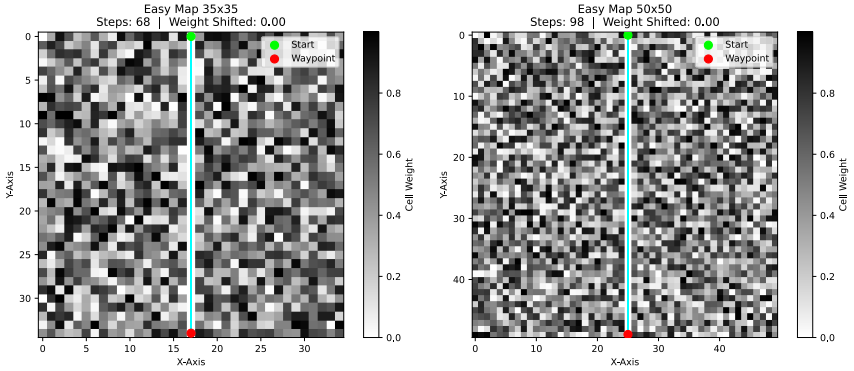


Figure 4.2: Pareto Optimal A* Paths for Easy Environment

This is further supported by figure 4.2 which shows both optimal paths on their respective map. It further illustrates that only one optimal path through the environment exists, which is why this environment was used as a proof of concept for both approaches. A more interesting result was achieved in the sinusoidal/checkerboard environment. This environment consisted of alternating peaks and valleys, where valleys would have very light obstacles and peaks very heavy ones. It was one of the two environments presented by Nübel et al. [30] and here the algorithmic pipeline again found only a single path in both dimensions.

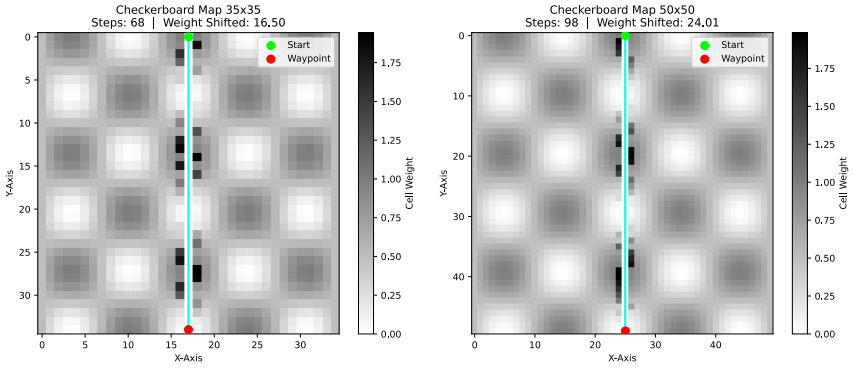


Figure 4.3: Pareto Optimal A* Paths for Sinusoidal Environment

The path that was found both times is the shortest possible through the environment as can be seen in figure 4.3. In the smaller environment, the path required 68 steps to be taken and shifted 16.5 units of weight, while in the larger environment it required 98 steps and shifted 24.01 units of weight. This suggests that no meaningful tradeoff between objectives exists for this environment, and that structural improvements would be needed, either by increasing obstacle weights in denser areas or decreasing them in lighter ones. If these changes were applied, there could be more than one optimal Path but for our specific setup, there was only one. The third environment we want to examine is the random one. This environment had randomly distributed weights in every cell and poses one of the biggest challenges since there is no obvious high quality solution. Here the A* pipeline found more than one Pareto optimal path, which stands in contrast to the two previously discussed environments.

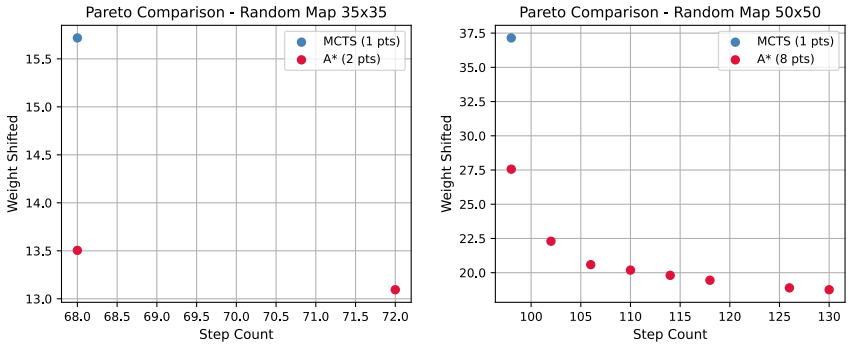


Figure 4.4: Pareto Fronts A* for Random Environment with Dimensions 35 (left) and 50 (right)

As we can see in figure 4.4, two Pareto optimal paths were found for the smaller environment and eight for the larger one. This indicates, that there are indeed multiple optimal paths in this random environment, that have differing objective tradeoffs. The optimal paths for the smaller environment had a step count of 68 and 72, as well as a shifted weight of 13.504 and 13.095 respectively.

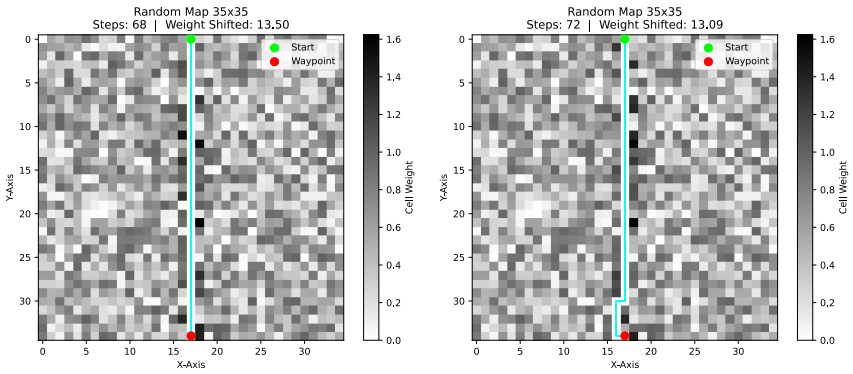


Figure 4.5: Pareto Paths Random Environment $35 \times 35$

Both paths are shown in figure 4.5 and follow a near-straight route down the middle, with the exception of the 72-step path, which takes a small detour before reaching the waypoint to avoid the weight on the direct route. The results become more interesting when examining the $50 \times 50$ variant of the random environment. As mentioned, figure 4.4 shows, that our A* pipeline produced eight different Pareto optimal paths for this scenario.

| Path# | Step Count | Weight Shifted |
|---|---|---|
| 1 | 98 | 27.562 |
| 2 | 102 | 22.300 |
| 3 | 106 | 20.589 |
| 4 | 110 | 20.189 |
| 5 | 114 | 19.808 |
| 6 | 118 | 19.450 |
| 7 | 126 | 18.892 |
| 8 | 130 | 18.757 |

Table 4.1: Pareto Path Values Random Environment $50 \times 50$

Table 4.1 shows the objective values of all these Pareto optimal solutions. While the first and second paths differ substantially in the amount of weight shifted, the later paths show only incremental changes in shifted weight, accompanied by a progressively increasing number of steps. This is precisely what the A* approach is designed to achieve, as once the optimal step length

is found, it is relaxed incrementally to identify paths that are worse in terms of step count than the shortest path, but better in terms of weight shifted.

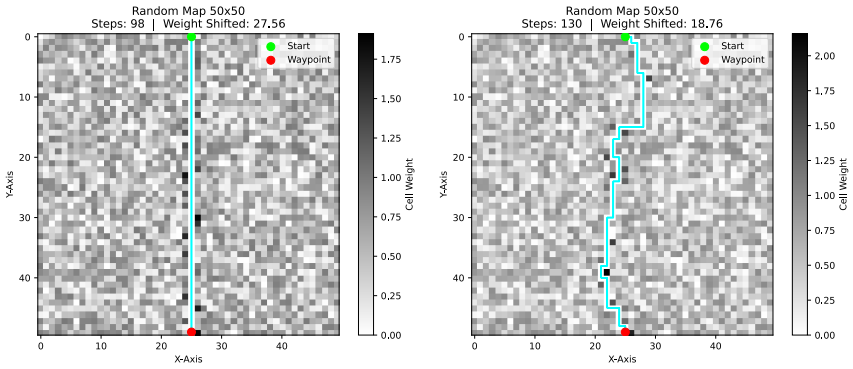


Figure 4.6: Extreme Points of Pareto Optimal A* Paths for Random Environment with Dimension 50 - Lowest Step Count (Left) and Lowest Weight Shifted (Right)

In figure 4.6 we can see that the path with the lowest steps on the left, while the path with the lowest weight shifted is on the right. While the path with the least amount of steps is again a straight line as it always is in our environments, we can see that the path with the lowest weight shifted takes additional detours to avoid high density obstacles. This behavior can again be seen in the results for the meandering river environment, the second environment that was taken from the works of Nübel et al. [30].

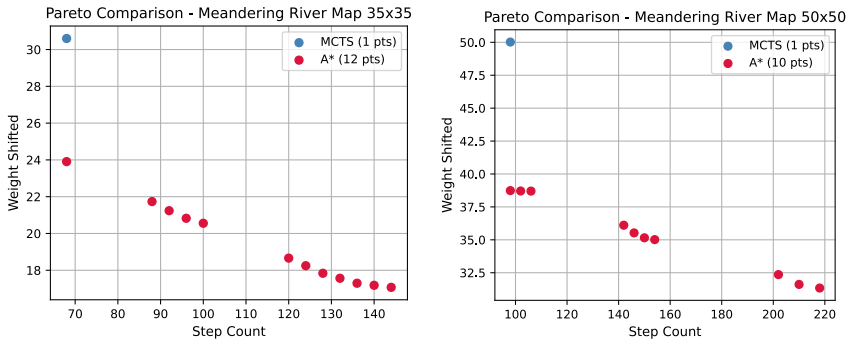


Figure 4.7: Pareto Fronts A* for Meandering River Map with Dimensions 35 (left) and 50 (right)

Figure 4.7 shows the Pareto fronts that were found for both tested map dimensions. What is interesting here is that the Pareto fronts seem to be disconnected in certain areas. In the front for the environment with dimension 35, we can see that there is no Pareto point between 68 and 88, as well as between 100 and 120 steps. The same can be spotted for the larger dimension of 50, where we have no candidate between 106 and 142, and 154 to 202 steps. This behavior suggests that the meandering river environment is structured in such a way that no Pareto optimal path exists for certain step counts, as no path with those step counts is able to adapt to the features of the environment. Furthermore, it becomes even more evident when we look at the paths for each extreme point in the Pareto front. In Figure 4.8 we can see these paths for the environment with dimension 50. We again see the straight path that dominates in terms of step count as was expected. For the other extreme the path has perfectly adapted to the structure of the river bed, which guarantees that it needs more steps but also needs to shift less weight. The results in between these extremes show different levels of adaptation to the river structure so i.e. shorter paths may take shortcuts through regions with heavier obstacles while, with increasing step size, more detours through less dense areas are taken. In the variation with the smaller dimension of 35, the results are similar but with less possible "levels" of adaptation since the river itself is smaller in size.

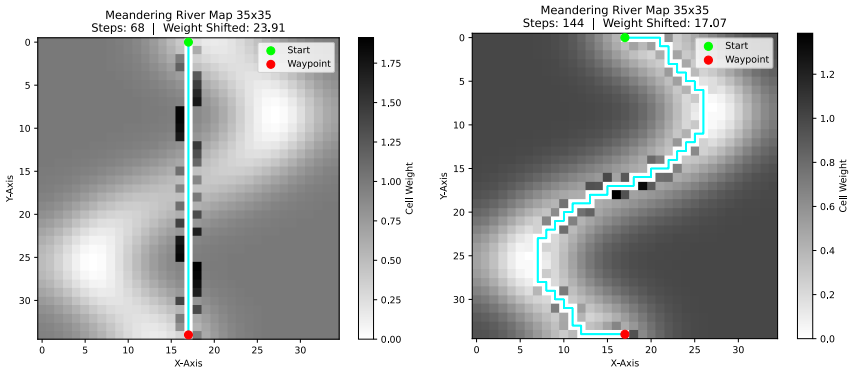


Figure 4.8: Extreme Points of Pareto Optimal A* Paths for Meandering River Environment with Dimension 50 - Lowest Step Count (Left) and Lowest Weight Shifted (Right)

## 4.2 Monte Carlo Tree Search Evaluation

Now that we have seen the results produced by the A* pipeline, we will discuss the different results of the Monte Carlo Tree Search approach. We will firstly examine the combined results of all configurations per environment, where we will determine which paths were dominating and amongst the results. A "configuration" in this context refers to the tree selection, the root selection and the simulation method. This will also answer research question one which addresses if MCTS can be used for pathfinding in PIEs. It is important to notice that we only used hypervolume based root selection in our experiments, so this is a given for all configurations. At the end of this section, we will evaluate which specific configuration achieved the best results in each of the tested environments which will answer research question three. To achieve this, we will compute the Pareto front per environment per configuration and then measure the hypervolume to a shared reference point.

### 4.2.1 Environment Evaluation

We will start the per environment evaluation of MCTS with the easy environment, as we have for A*. Figure 4.9 displays all found solutions by the different configurations of the algorithm. It is important to notice that these results are only grouped by their tree selection and simulations mechanisms. This combines all possible hyperparameters for the different settings. While figure 4.9 suggests that the hypervolume based tree selection in combination with the heavy square sampling rollout produced many good results, generating the actual Pareto front from these solutions yields only a single remaining solution. This dominant solution is the shortest path through the environment which corresponds to what was found by A*. Figure 1, which due to its length is located in the appendix, shows which parameter combinations found this optimal path and how frequently it was found across their 31 runs. We observe that the configurations which reached this point most often were both using the hypervolume tree selection method. Both of these MCTS configurations found the perfect path 30 out of 31 times. In terms of tree selection, the hypervolume and crowding distance based mechanisms produced the best results in this environment, while adaptive epsilon archiving based selection also found the optimal path, albeit less frequently than the other methods. It has to be noted that our UCB tree selection method did not find this path

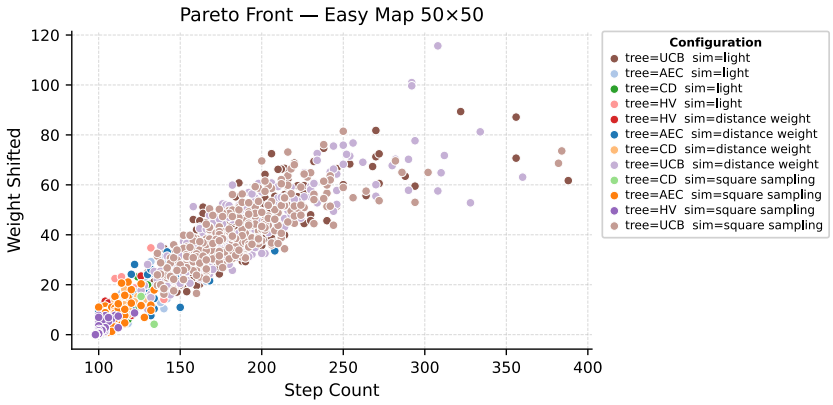


Figure 4.9: All MCTS Results for the Easy Environment with Dimension 50

even once, which means that it fails in the proof of concept environment. This is again confirmed by figure 4.9 where the points for UCB selection are ranging from around 130 to about 380 steps, which is far from the optimum of 98 steps on this map. Figure 4.5 shows all results for the easy environment with a dimension of 35. The results for this map can also be found again in the

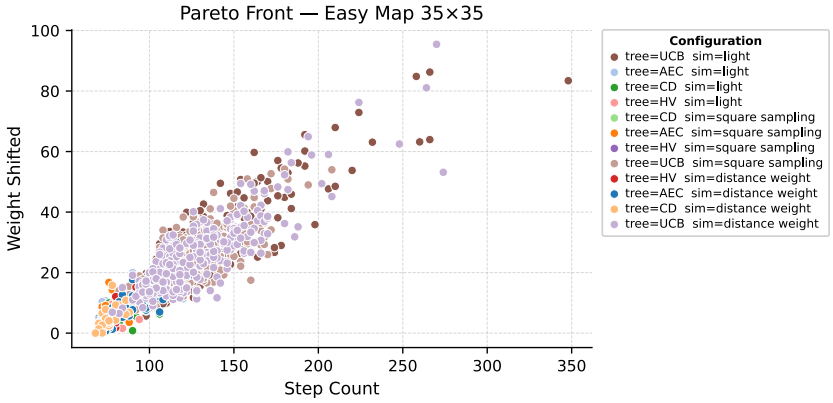


Figure 4.10: All MCTS Results for the Easy Environment with Dimension 35

appendix as table 1. We can see here, that nine different combinations found the optimal path in all 31 runs. Five of those nine used the crowding distance tree selection and the rest used the hypervolume tree selection. In terms of simulation strategies we can see that all of them were used in three different combinations amongst the top nine results. Adaptive epsilon archiving selection performed similarly to the results from the same map with dimension 50. Again UCB selection did not find the perfect path once. Next we want to

look at the MCTS performance in the sinusoidal/checkerboard environment. Figure 4.11 shows the performance of all configurations of the Monte Carlo

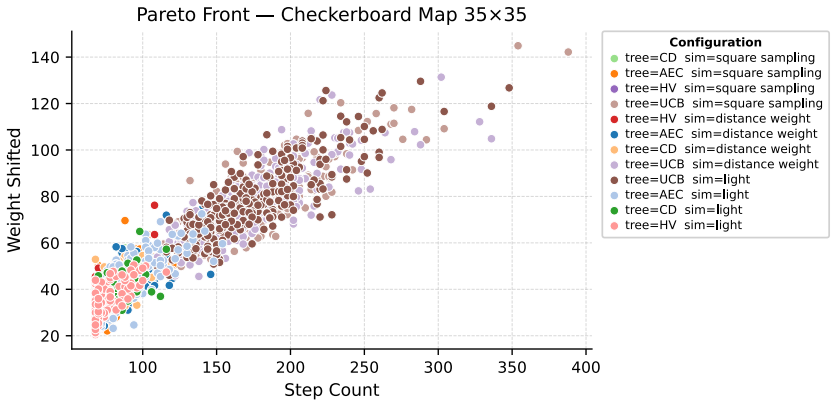


Figure 4.11: All MCTS Results for the Sinusoidal Environment with Dimension 35

Tree Search that were tested. We can see that all configurations that used the UCB tree selection performed the worst overall, which could also be observed for both environments that we have discussed already. It can be theorized that this could be due to a bad parameter tuning inside the UCB method, but we can not confirm this here since we did not test different parameter configurations inside UCB. This would exceed the scope of this work and will be touched on again in the future work section. For both sizes of the environment, we got only one solution that was found to be optimal by MCTS. In the smaller checkerboard environment the combination of hypervolume based tree selection and light simulations found the optimal path with a step length of 68 and a shifted weight of 20.5 units. Furthermore, this configuration used a total evaluation budget of 200000, performed 100 actions per simulation and simulated each child 50 times. This path is the dominating one amongst all results of all configurations and no other configuration was able to find it. Almost the same can be said for the larger version of the environment. Figure 4.12 shows the combined results of all configurations in the $50 \times 50$ checkerboard environment. It becomes very apparent that the configurations that produced the worst results again were those that used UCB selection as the tree selection method. The configuration consisting of adaptive epsilon archiving based tree selection and heavy square sampling rollout found the dominant path also exactly once. This dominant path had a step count of 98 and a shifted weight

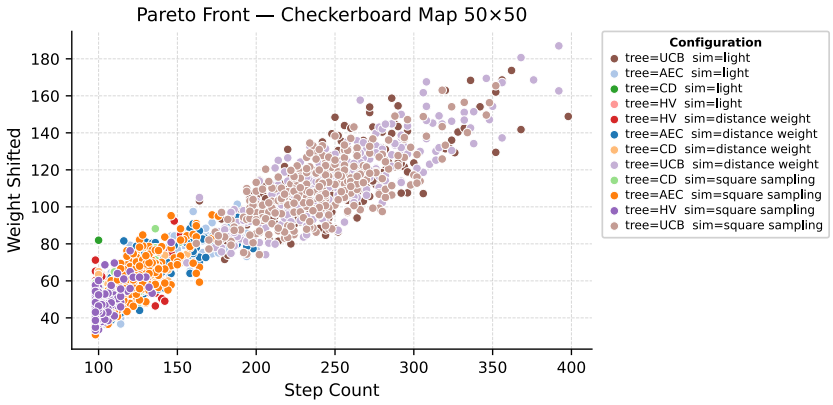


Figure 4.12: All MCTS Results for the Sinusoidal Environment with Dimension 50

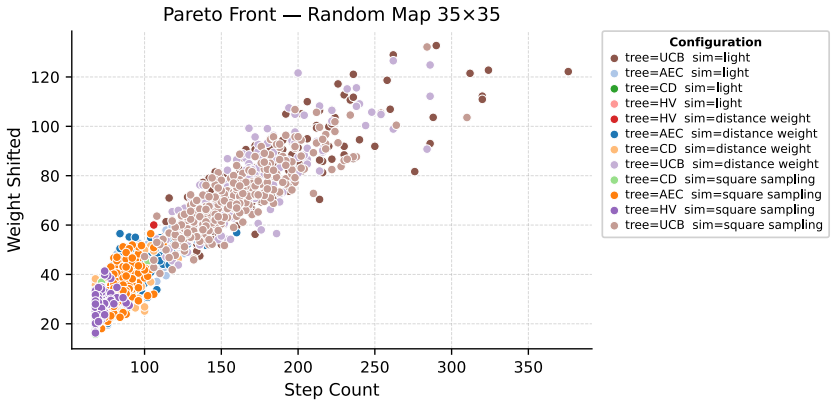


Figure 4.13: All MCTS Results for the Random Environment with Dimension 35

of 30.92 units. That result can even be seen in the lower left corner of figure 4.12. Specifically, the configuration that found this path used a total evaluation budget of 200000, a per simulation budget of 100 actions and did 25 simulations per child. Similarly to the smaller environment, this path was only found once amongst all configurations. Continuing with the evaluation, we want to evaluate the per configuration results in the random environment. Figure 4.13 shows the combined results of all configurations for the smaller version of the environment. As we have already seen for the sinusoidal and the easy environment, configuration using the UCB tree selection method produced the worst results. We again only have one dominating solution, which was found by

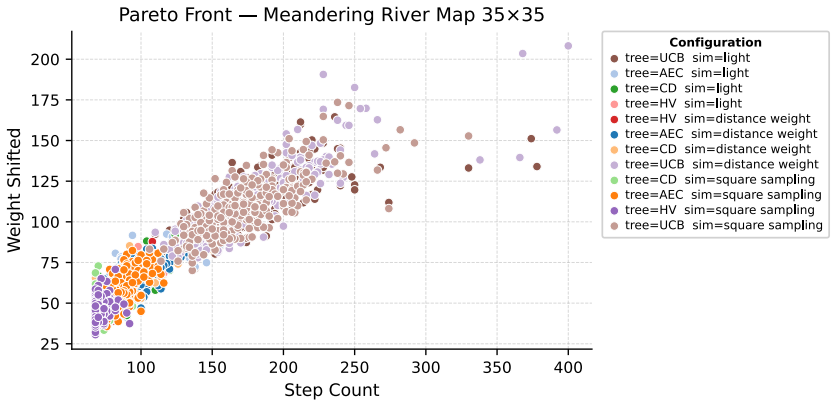


Figure 4.14: All MCTS Results for the Meandering River Environment with Dimension 35

the configuration of crowding distance based tree selection and heavy square sampling simulations. To be more specific, this solution was found once by the configuration that used 200000 total evaluations, 100 actions per simulation and 50 simulations per child. The found path has a step count of 68 and a shifted weight of 15.72 units. For the larger environment, the results are similar. Not only did the configurations that were using UCB tree selection produce the worst results but furthermore, the dominating solution was only found once by a single configuration. This configuration consisted of crowding distance based tree selection and light simulations. A total evaluation budget of 1000000 as well as 75 actions per simulation and 100 simulations per child were used. The final path had a step count of 98 and 37.149 units of weight were shifted along it.

The last configuration comparison that we want to perform is for the meandering river environment. Figure 4.14 shows that the trend of UCB tree selection configurations producing the worst results continues. Furthermore, we again only have a single path that dominates all others which was found by the configuration consisting of hypervolume based tree selection and heavy square sampling simulations. A total evaluation budget of 500000 was used for this as well as 100 steps per simulation and 100 simulations per child. The path that the named configuration found had 68 steps and a total of 30.602 units of weight were moved. Its higher dimensional counterpart again produced similar results. UCB configurations again produced the worst results and the one dominating path was found by using crowding distance based tree selection,

hypervolume based selection and the heavy distance weight simulations. To achieve this the algorithm had a total budget of 500.000 with 100 steps per simulation and 100 simulations per child. The final path took 98 steps and 50.02 units of weight were shifted along it.

### 4.2.2 Configuration Comparison

As we have finished with evaluating the combined results of all configurations in each environment, we now want to continue by comparing the different configurations with each other. This comparison will answer research question three, which asks which of the proposed approaches performed better overall. Starting with the large easy environment we have a nine way tie for the best algorithm configuration. Across all results, steps ranged from 98 to 388 in this environment, with shifted weight ranging from 0 to 115.66 units. These ranges are computed with dominated points taken into account, but the hypervolume computation will by definition only use the members of the pareto front. As explained at the end of the previous chapter, their values are normalized using min-max normalization and the reference point will always have the values $(1.1, 1.1)$ on the normalized scale.

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.210 | Adaptive Epsilon Archiving | Light |
| 2 | 1.210 | Crowding Distance | Light |
| 3 | 1.210 | Hypervolume | Light |
| 4 | 1.210 | Hypervolume | Distance Weight |
| 5 | 1.210 | Adaptive Epsilon Archiving | Distance Weight |
| 6 | 1.210 | Crowding Distance | Distance Weight |
| 7 | 1.210 | Crowding Distance | Square Sampling |
| 8 | 1.210 | Adaptive Epsilon Archiving | Square Sampling |
| 9 | 1.210 | Hypervolume | Square Sampling |

Table 4.2: MCTS Configuration Rankings by Hypervolume for Easy Environment $50 \times 50$

Table 4.2 shows that all nine listed methods share a hypervolume of 1.210. This is the case since in the easy environment, all these configurations found

the optimal path through the middle that shifts no weight. To break the tie, one can refer to table 1, which shows how often each configuration was able to find this path. In doing so, we find that hypervolume based tree selection combined with heavy square sampling simulations produced the best overall results. This configuration has found this path a total of 238 times across all hyperparameter settings, while the second-best approach, crowding distance based tree selection in combination with the light rollout, found the path a total of 231 times. Furthermore, the third-best approach for this environment seems to be the hypervolume based tree selection and the light rollouts with this configuration finding the optimal path a total of 216 times. The results for the smaller $35 \times 35$ version of the easy environment are very similar to the ones seen in the larger version. Here the step count was in a range from 68 to 388 and the weight shifted between 20.51 and 144.89 units.

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.210 | Adaptive Epsilon Archiving | Light |
| 2 | 1.210 | Crowding Distance | Light |
| 3 | 1.210 | Hypervolume | Light |
| 4 | 1.210 | Crowding Distance | Square Sampling |
| 5 | 1.210 | Adaptive Epsilon Archiving | Square Sampling |
| 6 | 1.210 | Hypervolume | Square Sampling |
| 7 | 1.210 | Hypervolume | Distance Weight |
| 8 | 1.210 | Adaptive Epsilon Archiving | Distance Weight |
| 9 | 1.210 | Crowding Distance | Distance Weight |

Table 4.3: MCTS Configuration Rankings by Hypervolume for Easy Environment $35 \times 35$

Table 4.3 shows that we again have a nine way tie for the first place, with all configurations again having a hypervolume of 1.21. With the same tie breaking methods that we used for the larger environment, consulting table 2, we can determine that the configuration using crowding distance based tree selection and square sampling takes first place, finding the optimal path 270 times. Hypervolume based tree selection with square sampling ranks second, having found the path 258 times, followed in third place by crowding distance based tree selection with distance weight sampling at 239 times. Now that we have seen the results for the easy environment, we will be continuing with the

sinusoidal one. Table 4.4 shows the top three configurations for the $35 \times 35$ version of this environment.

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Hypervolume | Light |
| 2 | 1.2026 | Crowding Distance | Distance Weight |
| 3 | 1.2020 | Crowding Distance | Square Sampling |

Table 4.4: MCTS Configuration Rankings by Hypervolume for Sinusoidal Environment $35 \times 35$

Hypervolume tree selection in combination with light rollouts achieved the optimal hypervolume of 1.2100, which is the case because this configuration was the only one to find the path that dominates all others. The second-best configuration with a hypervolume of 1.2026 goes to crowding distance tree selection paired with distance weight rollouts. The third place configuration, combining crowding distance based tree selection with square sampling simulations, closely trails second place with a hypervolume of 1.202. At this point it seems like crowding distance and hypervolume being the superior tree selection methods while the simulation approaches are evenly matched, which is solidified by the results for the $50 \times 50$ sinusoidal environment.

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Adaptive Epsilon Archiving | Square Sampling |
| 2 | 1.2034 | Hypervolume | Distance Weight |
| 3 | 1.1970 | Crowding Distance | Distance Weight |

Table 4.5: MCTS Configuration Rankings by Hypervolume for Sinusoidal Environment $50 \times 50$

Table 4.5 shows the three configurations that produced the best results on the larger version of the sinusoidal environment. Interestingly on this map the adaptive epsilon archiving based tree selection in combination with square sampling simulations achieved the best hypervolume, since it also produced the non dominated solution across all configurations. The second and third-best combination are the hypervolume and the crowding distance based tree

selection, both using the distance weight rollout methods. In this environment the found solutions of all approaches had a step range of 98 to 398 and a range of shifted weight between 30.92 and 187. Something that is interesting about this is that 400 steps are the maximum length that a path was allowed to be. With a maximum step count of 398, it seems plausible that some runs did not produce results since they reached the step limit. While in other environments this would happen one to two times during all runs of all configurations combined, in the sinusoidal $50 \times 50$ environment, this happened a total of 13 times. In all cases where no solution was found due to the maximum step limit being exhausted, UCB tree selection was used. Table 4.6 shows us all misses for each configuration in each environment, including the ones we have already evaluated. We can see that in the current environment distance weight rollouts didn't find a solution to the environment 6, light rollouts 4 and square sampling rollouts 3 times. Next we want to look at the results for the random

| **Environment** | **Total** | **Tree Selection** | **Rollout** | **Count** |
|---|---|---|---|---|
| Sinusoidal 50x50 | 13 | UCB | Distance Weight | 6 |
| | | UCB | Light | 4 |
| | | UCB | Square Sampling | 3 |
| Random 50x50 | 2 | UCB | Light | 1 |
| | | UCB | Square Sampling | 1 |
| Meandering River 50x50 | 1 | UCB | Light | 1 |
| Easy 50x50 | 1 | UCB | Distance Weight | 1 |
| Sinusoidal 35x35 | 1 | UCB | Square Sampling | 1 |
| Random 35x35 | 1 | UCB | Distance Weight | 1 |

Table 4.6: None File Occurrences per Environment and Configuration

environment. For the smaller dimension the step count ranged from 68 to 376 and the weight shifted from 15.72 to 132.72.

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Crowding Distance | Square Sampling |
| 2 | 1.2046 | Hypervolume | Square Sampling |
| 3 | 1.2001 | Hypervolume | Light |

Table 4.7: MCTS Configuration Rankings by Hypervolume for Random Environment $35 \times 35$

Table 4.7 shows that the configuration consisting of crowding distance based tree selection and square sampling rollouts achieved the perfect solution. The second biggest hypervolume of 1.2046 was achieved by the hypervolume tree selection and again the square sampling simulations while the third place with a hypervolume of 1.2001 goes to again hypervolume tree selection and light rollouts. For the larger random environment the results can be seen in table 4.8. In this environment we had a step range of 98 to 398 and a weight shifted range from 37.15 to 172.51 units. The results are again similar to those of the random environment with a dimension of 35.

Table 4.8: MCTS Configuration Rankings by Hypervolume for Random Environment $50 \times 50$

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Crowding Distance | Light |
| 2 | 1.2099 | Hypervolume | Light |
| 3 | 1.2095 | Crowding Distance | Square Sampling |

The last environment that we want to compare configurations for is the meandering river. Starting again with the $35 \times 35$ version of the environment, the step range here was between 68 and 400. This again indicates that some runs might have not found a solution due to exceeding the step limit. Returning to table 4.6, only the configuration combining UCB tree selection with distance weight rollouts failed to find a solution on one occasion. Here the best solution was found by the configuration of hypervolume based tree selection and square sampling simulations. Places two and three achieved a hypervolume of 1.2082 and 1.2001 respectively, with both using crowding distance based tree selec-

tion, while the second-best configuration used light rollouts and the third-best square sampling simulations. This can be seen again in table 4.9.

Table 4.9: MCTS Configuration Rankings by Hypervolume for Meandering River Environment $35 \times 35$

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Hypervolume | Square Sampling |
| 2 | 1.2082 | Crowding Distance | Light |
| 3 | 1.2001 | Crowding Distance | Square Sampling |

These results reinforce the emerging trend of hypervolume and crowding distance based selection consistently finding the best paths through the environment. Furthermore, the results for the larger meandering river environment are similar to those we have seen till now. The steps ranged from 98 to 398 and the weight shifted from 50 to 309.07 amongst all solutions. Referring to table 4.6 even though we have a maximum number of steps that is very close to our 400 cutoff, we always found a solution for this map. The best result was achieved by crowding distance based tree selection and distance weight rollouts, while the second and third-best result utilize crowding distance and hypervolume based tree selection with a hypervolume of 1.2048 and 1.2025 respectively. Both used the square sampling simulation method.

Table 4.10: MCTS Configuration Rankings by Hypervolume for Meandering River Environment $50 \times 50$

| Rank | HV | Tree Selection | Simulation |
|---|---|---|---|
| 1 | 1.2100 | Crowding Distance | Distance Weight |
| 2 | 1.2048 | Crowding Distance | Square Sampling |
| 3 | 1.2025 | Hypervolume | Square Sampling |

To statistically validate the observed performance differences between configurations, a Kruskal-Wallis [21] test was conducted per environment using the normalized hypervolume of each individual run as the test metric. It was significant across all eight environment-dimension combinations (all $H > 2284$, all $p < 0.001$), confirming that the configurations do not perform equally. A

subsequent Dunn post-hoc test [13] with Bonferroni correction [6] was applied to identify which specific configurations differed significantly. It revealed a consistent three tier structure across all environments. The first and thus best performing tier encompassed all configurations using crowding distance or hypervolume based tree selection, regardless of the used simulation method. Furthermore, the second tier consisted of all adaptive epsilon archiving based configurations, and the third and worst-performing tier contained all UCB based configurations. All cross-tier differences were significant at $p < 0.001$ after correction, while no significant differences were found within any tier. Taking the random map with dimension $35 \times 35$ as a representative example, tier 1 achieved a mean normalized hypervolume of $1.107 \pm 0.065$ compared to $0.980 \pm 0.100$ for tier 2 and $0.498 \pm 0.156$ for Tier 3. Critically, within each tier the simulation method never produced a statistically significant difference. This confirms that the tree selection method is the sole driver of performance differences between configurations. Table 4.11 summarizes how often each specific configuration combination produced the best result across all environments. Crowding distance and hypervolume based tree selection each achieved the best results four times, while adaptive epsilon archiving based selection placed first once on the $50 \times 50$ sinusoidal environment. Square sampling appears most frequently amongst the winning combinations, though as established by the statistical analysis above, this advantage was not statistically significant.

| Tree Selection | Simulation | Number of Best Results |
|---|---|---|
| Crowding Distance | Square Sampling | 2 |
| Hypervolume | Square Sampling | 2 |
| Hypervolume | Light | 1 |
| Crowding Distance | Light | 1 |
| Adaptive Epsilon Archiving | Square Sampling | 1 |
| Crowding Distance | Distance Weight | 1 |

Table 4.11: Number of Best Results per MCTS Configuration Combination

## 4.3 Algorithmic Comparison

Now that we have evaluated the results for both the A* baseline and the MCTS approach, we are ready to compare them to each other. We will again go through environment by environment and compare the best results of both approaches.

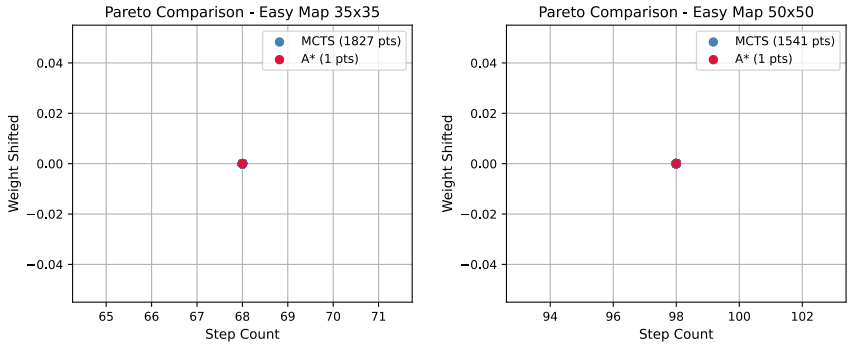


Figure 4.15: Pareto Front Comparison of MCTS and Two-Phase A* in the Easy Environment

Starting with the easy environment figure 4.15 shows the Pareto fronts for both approaches for each size of the environment. In both cases the A* pipeline only produced one result, since there is only one optimal path through the environment. Over all its configurations and runs, for the smaller environment MCTS found this path a total of 1827 times. Since each environment would have 3348 runs per dimension, this means that the optimal path was found in 54.6% of runs. For the larger environment MCTS found it 1541 which makes a percentage of 46% of all runs. This makes the two phase A* approach the better option for this environment, since it finds the best path deterministically all the time. The same trend continues when we proceed to the sinusoidal/checkerboard environment. Figure 4.16 shows the Pareto fronts of both approaches.

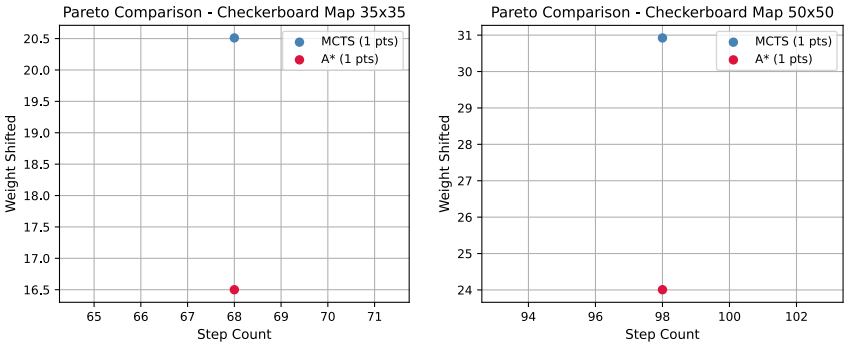


Figure 4.16: Pareto Front Comparison of MCTS and Two-Phase A* in the Sinusoidal Environment

For both dimensions, the A* approach found a better path through the environment than MCTS. Furthermore, while both algorithms found the optimal step length for each environment with 68 and 98 respectively, MCTS seems to have shifted the weight in a worse way than A*. What also needs to be mentioned is that in both instances, the dominating path for MCTS was only found once. This means that MCTS has a 0.03% chance of finding this path for both environment sizes, while the path it produces is dominated by the deterministic two-phase A* solution, making the A* approach superior for this environment. Turning to the random environment further highlights another issue with the lackluster performance of the MCTS observed so far. Figure 4.17 shows the comparison of both Pareto fronts where it can be observed that MCTS again only found its respective dominating path once which again gives us a 0.03% chance to find this path amongst all 3348 runs with different hyperparameter settings. Meanwhile, the A* pipeline found a front for both sizes of the environment, and in both instances a member of the A* front dominates the MCTS path. It has to be noted that even if the weight shifting was not optimal, MCTS found the minimum amount of steps required for both environments. What is problematic is that MCTS only found one path that dominated all others and, according to the solutions that A* produced, the found MCTS path is not even part of the actual Pareto front. This makes A* the better approach for this case.

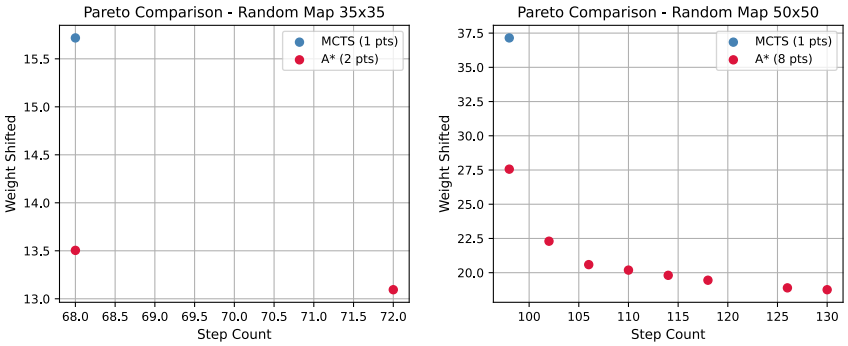


Figure 4.17: Pareto Front Comparison of MCTS and Two-Phase A* in the Random Environment

Lastly we want to look at the results for the meandering river environment. Figure 4.18 shows the comparison of the Pareto fronts for both approaches. In essence these results are the same as the ones for the random environment. While the A* pipeline found an actual front of solutions MCTS only found one dominating path once. Furthermore, the found MCTS solution is worse than the A* solution with the same step length, but it can be seen as positive that MCTS figured out the shortest path for this environment, even if the weight shifting is off again. Overall the two phase A* approach seems to be the better option for this environment also.

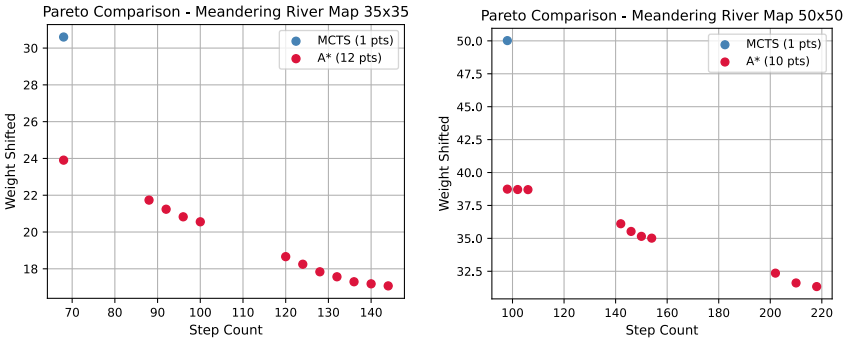


Figure 4.18: Pareto Front Comparison of MCTS and Two-Phase A* in the Meandering River Environment

# 5 Conclusion and Future Work

After thoroughly evaluating the obtained results, in this chapter we want to summarize our findings and answer the research questions that we posed in the beginning of this thesis. The first research question we proposed was if the MCTS approach is able to solve the roundtrip pathfinding problem in PIE's. We hypothesized in that it should be able to due to its overall success in other pathfinding problems, given that it is adequately adapted to the problem. In both the map evaluation and configuration comparison subsections of the evaluation chapter, we could see that MCTS was consistently able to find paths through the environment. Referring to table 4.6, we saw that only 19 out of 26784 total runs did not produce a result and all of these used UCB tree selection. This might be the case because the parameters for UCB tree selection were not tuned well enough. We did not focus on this in this work, since we wanted to propose general concepts to solve roundtrip problems in path influenced environments. With that being said, our hypothesis that our proposed MCTS would be able to solve the problem at all was proven right. The second question we wanted to answer in this thesis was which algorithm would perform better on the proposed problem. We hypothesized that the two phase A* approach would be able to outperform MCTS due to the deterministic nature of its components that allows it to find paths more reliably. Across almost all environments, the A* pipeline produced superior results to MCTS, with the easy environment being the sole case where both approaches found the optimal path. Even in the easy environment, A* could be considered superior, having found the optimal path in every run compared to MCTS, which succeeded only 54.6% and 46.0% of the time for the smaller and larger environment sizes respectively. The A* approach most definitively outperformed MCTS in the meandering river and random environments, where it produced a full Pareto front of solutions, while MCTS found only a single optimal point across all its runs, which was subsequently dominated by members of the A* Pareto front. For this thesis, we can say that the two phase A* pipeline works much better than Monte Carlo Tree Search, at least for the specific MCTS parameter

configuration that we chose. After answering this, we still want to answer the question which configuration of MCTS achieved the best results overall, which refers to RQ3. In the introduction, we hypothesized that there will be a significant difference in performance among the combinations of MCTS mechanisms. This was confirmed by the Kurskal-Wallis test and was supported by the Dunn post-hoc test with Bonferroni correction which revealed a three tier structure. In this structure, crowding distance and hypervolume based tree selection formed the best performing, adaptive epsilon clustering1 the second best and UCB the worst tier. There were no statistically significant differences between the simulation methods, which implies that they do not meaningfully affect the solution quality. The poor performance of UCB based tree selection can be attributed to insufficient parameter tuning. To close this chapter we want to name the scientific contributions of this thesis to the field of pathfinding in path influenced environments. The first novelty is the problem itself. A roundtrip problem in PIE's was never considered in the papers that are published at this point in time. While Nübel et al. [30] and Speidel [33] tried to solve the problem of pathfinding in these environments, they only considered problems that had a definitive start and goal point, which differs from a roundtrip. One could argue that the roundtrip problem is more challenging, as it requires more structured shifts to carve a corridor, ensuring that no weight needs to be moved during the second half of the path. Another novelty proposed in this thesis is the two phase A* pipeline that we aimed to use as a baseline. Speidel [33] used a single A* algorithm with scalarized objective values to navigate the environment, which is a valid approach for the problem but lacks flexibility and produces only one result per run. Our approach finds a complete Pareto front in a single run and is applicable to both PIEs and traditional pathfinding environments with a mix of static and dynamic obstacles. Lastly, the final novelty of this thesis is the application of MCTS to PIEs. While the concept of MCTS is widely known and finds application in traditional pathfinding and pathplanning, it was thus far never used in this kind of new environment. Again referring to Nübel et al. [30] and Speidel [33], both of them used mainly evolutionary algorithms to solve the problem. Besides the mentioned A* version in Speidel's bachelor thesis, no other approach was ever tested. Thus, we adapted this algorithm to the problem at hand and also proposed a new tree selection method in adaptive epsilon archive based tree selection, which was to our knowledge not known before. While it did not produce the results we hoped for, we again want to reiterate that a parameter

tuning study would be needed to accurately determine how powerful MCTS could be in the shown configurations.

What we now want to talk about to finalize this thesis is the future research that in our opinion might be interested in this field. A parameter study would be a crucial step toward effectively applying Monte Carlo Tree Search to pathfinding problems in Path Influenced Environments. If we look at our setup we can see that we have many tunable hyperparameters, which range from the total evaluation budget, the number of actions per simulation and the number of simulations per child to the hyperparameters of UCB1 or the different sampling methods. Similarly, a correlation between different map sizes and parameter settings could also be evaluated. Another important research question regarding the proposed A* pipeline would be whether it is capable of deterministically finding the true Pareto front for PIEs, which remains to be formally proven or disproven. The last and perhaps most significant area of research specific to Monte Carlo Tree Search in this field would be the exploration of alternative tree and root selection methods, as well as novel simulation techniques. While we found that there were no significant differences among the simulation methods that we used, there might be approaches that can effectively exploit the characteristics of the problem, which may lead to better results. In our opinion these different mechanisms on their own can have a large impact on the quality of results that are achieved and while we tried to adapt to the problem using i.e. the Pareto archive, there might be other and better approaches to make this algorithm competitive.

# Bibliography


[1] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015(1):736138, 2015.

[2] Anusha Alexander, Kalaichelvi Venkatesan, Jinane Mounsef, and Karthikeyan Ramanujam. A comprehensive survey of path planning algorithms for autonomous systems and mobile robots: Traditional and modern approaches. *IEEE Access*, 2025.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[4] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195, 2014.

[5] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[6] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R istituto superiore di scienze economiche e commericiali di firenze*, 8:3–62, 1936.

[7] Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, pages 60–71. Springer, 2008.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[10] Kaikai Diao, Xiaodong Sun, Gerd Bramerdorfer, Yingfeng Cai, Gang Lei, and Long Chen. Design optimization of switched reluctance machines for performance and reliability enhancements: A review. *Renewable and Sustainable Energy Reviews*, 168:112785, 2022.

[11] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: his life, work, and legacy*, pages 287–290. 2022.

[12] Ran Duan, Jiayi Mao, Xiao Mao, Xinkai Shu, and Longhui Yin. Breaking the sorting barrier for directed single-source shortest paths. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 36–44, 2025.

[13] Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

[14] David Griffel. Multi-objective optimization using evolutionary algorithms, by kalyanmoy deb, pp. 487.£ 60. 2001. isbn 0 471 87339 x (wiley). *The Mathematical Gazette*, 87(509):409–410, 2003.

[15] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, Waltham, MA, USA, 3 edition, 2012.

[16] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[17] Julia Heise, Jens Weise, and Sanaz Mostaghim. Towards benchmarking of pathfinding algorithms in path-influenced environments. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 69–70, 2023.

[18] Henderi Henderi, Tri Wahyuningsih, and Efana Rahwanto. Comparison of min-max normalization and z-score normalization in the k-nearest neighbor (knn) algorithm to test the accuracy of types of breast cancer. *International Journal of Informatics and Information Systems*, 4(1):13–20, 2021.

[19] Satoshi Hoshino and Kenichiro Uchida. Interactive motion planning for mobile robot navigation in dynamic environments. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 21(4):667–674, 2017.

[20] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[21] William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[22] D Monzonís Laparra. Pathfinding algorithms in graphs and applications. *Universitat de Barcelona*, 2019.

[23] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002.

[24] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 48(1):1–35, 2015.

[25] Ke Liu, Honglin Wang, Yao Fu, Guanzheng Wen, and Binyu Wang. A dynamic path-planning method for obstacle avoidance based on the driving safety field. *Sensors*, 23(22):9180, 2023.

[26] Yaping Lu and Chen Da. Global and local path planning of robots combining aco and dynamic window algorithm. *Scientific Reports*, 15(1):9452, 2025.

[27] Sha Luo, Mingyue Zhang, Yongbo Zhuang, Cheng Ma, and Qingdang Li. A survey of path planning of industrial robots based on rapidly exploring random trees. *Frontiers in Neurorobotics*, 17:1268447, 2023.

[28] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[29] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 1999.

[30] Carlo Nübel, Malte Speidel, and Sanaz Mostaghim. Navigating path-influenced environments using evolutionary multi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1462–1470, 2025.

[31] Ke Shang, Hisao Ishibuchi, Linjun He, and Lie Meng Pang. A survey on the hypervolume indicator in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 25(1):1–20, 2020.

[32] Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. Pathfinding in stochastic environments: learning vs planning. *PeerJ Computer Science*, 8:e1056, 2022.

[33] Malte F Speidel. Path influenced environments and navigation approaches within. `https://ci.ovgu.de/is_media/Master+und+Bachelor_Arbeiten+/BachelorThesis_MalteSpeidel-p-7718.pdf`, 2023.

[34] Kevin Thompson and Tevian Dray. Taxicab angles and trigonometry. *Pi Mu Epsilon Journal*, pages 87–96, 2000.

[35] John Von Neumann, Arthur Walter Burks, et al. Theory of self-reproducing automata. 1966.

[36] Mark Voorneveld. Characterization of pareto dominance. *Operations Research Letters*, 31(1):7–11, 2003.

[37] Shiyue Wang, Haozheng Xu, Yuhan Zhang, Jingran Lin, Changhong Lu, Xiangfeng Wang, and Wenhao Li. Where paths collide: A comprehensive survey of classic and learning-based multi-agent pathfinding. *arXiv preprint arXiv:2505.19219*, 2025.

[38] Barry Payne Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[39] Peter Yap. Grid-based path-finding. In *Conference of the canadian society for computational studies of intelligence*, pages 44–55. Springer, 2002.

# Appendix

Table 1: Configurations reaching the Pareto-optimal point (`steps=98`, `weight=0.0`) on `easy_map` $50 \times 50$. *Hits* = number of runs achieving the optimum out of 30. Tree selection: `hv` = hypervolume, `cd` = crowding distance, `aega` = adaptive epsilon grid archiving. Simulation: `light` = light rollout, `heavy_dist` = heavy distance weight, `heavy_sq` = heavy square sampling.

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| hv | light | 1,000,000 | 150 | 50 | 30 |
| hv | heavy_dist | 500,000 | 150 | 25 | 30 |
| hv | light | 1,000,000 | 100 | 25 | 29 |
| hv | heavy_dist | 200,000 | 100 | 50 | 29 |
| cd | heavy_sq | 500,000 | 75 | 50 | 29 |
| cd | light | 200,000 | 75 | 25 | 28 |
| cd | light | 500,000 | 75 | 50 | 28 |
| hv | light | 1,000,000 | 75 | 100 | 28 |
| hv | light | 500,000 | 100 | 100 | 28 |
| cd | heavy_dist | 1,000,000 | 75 | 100 | 28 |
| hv | heavy_dist | 1,000,000 | 75 | 100 | 28 |
| hv | heavy_dist | 1,000,000 | 100 | 25 | 28 |
| hv | heavy_sq | 1,000,000 | 75 | 100 | 28 |
| hv | heavy_sq | 500,000 | 75 | 50 | 28 |
| hv | heavy_sq | 500,000 | 100 | 100 | 28 |
| cd | heavy_sq | 1,000,000 | 100 | 25 | 28 |
| hv | light | 500,000 | 75 | 50 | 27 |
| cd | heavy_dist | 1,000,000 | 150 | 50 | 27 |
| cd | heavy_dist | 500,000 | 150 | 25 | 27 |
| hv | heavy_sq | 1,000,000 | 100 | 25 | 27 |

continued on next page

Table 1 – continued

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| hv | heavy_sq | 1,000,000 | 150 | 50 | 27 |
| hv | light | 200,000 | 100 | 50 | 26 |
| cd | light | 500,000 | 150 | 25 | 26 |
| cd | light | 1,000,000 | 150 | 50 | 26 |
| cd | heavy_dist | 200,000 | 100 | 50 | 26 |
| cd | heavy_dist | 200,000 | 75 | 25 | 26 |
| hv | heavy_dist | 500,000 | 75 | 50 | 26 |
| cd | heavy_dist | 500,000 | 75 | 50 | 26 |
| cd | heavy_sq | 200,000 | 75 | 25 | 26 |
| cd | light | 500,000 | 100 | 100 | 25 |
| hv | light | 200,000 | 75 | 25 | 25 |
| cd | light | 1,000,000 | 75 | 100 | 25 |
| cd | heavy_sq | 200,000 | 100 | 50 | 25 |
| cd | heavy_sq | 500,000 | 100 | 100 | 25 |
| hv | heavy_sq | 500,000 | 150 | 25 | 25 |
| hv | heavy_dist | 200,000 | 75 | 25 | 24 |
| hv | heavy_dist | 500,000 | 100 | 100 | 24 |
| hv | heavy_dist | 1,000,000 | 150 | 50 | 24 |
| cd | heavy_sq | 1,000,000 | 75 | 100 | 24 |
| hv | heavy_sq | 200,000 | 100 | 50 | 24 |
| hv | heavy_sq | 200,000 | 75 | 25 | 24 |
| cd | heavy_sq | 500,000 | 150 | 25 | 24 |
| hv | light | 500,000 | 150 | 25 | 23 |
| cd | heavy_dist | 500,000 | 100 | 100 | 23 |
| cd | light | 200,000 | 100 | 50 | 22 |
| cd | heavy_dist | 1,000,000 | 100 | 25 | 22 |
| cd | heavy_sq | 1,000,000 | 150 | 50 | 22 |
| cd | light | 1,000,000 | 100 | 25 | 21 |
| aega | heavy_dist | 200,000 | 75 | 25 | 18 |
| aega | light | 500,000 | 75 | 50 | 17 |
| aega | light | 1,000,000 | 150 | 50 | 17 |
| aega | heavy_dist | 1,000,000 | 150 | 50 | 17 |
| aega | heavy_sq | 200,000 | 75 | 25 | 16 |
| aega | heavy_dist | 1,000,000 | 75 | 100 | 14 |

continued on next page

Table 1 – continued

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| aega | heavy_dist | 500,000 | 150 | 25 | 14 |
| aega | heavy_dist | 500,000 | 75 | 50 | 14 |
| aega | heavy_sq | 500,000 | 75 | 50 | 14 |
| aega | heavy_sq | 1,000,000 | 100 | 25 | 14 |
| aega | light | 1,000,000 | 100 | 25 | 13 |
| aega | heavy_dist | 1,000,000 | 100 | 25 | 13 |
| aega | heavy_sq | 1,000,000 | 150 | 50 | 13 |
| aega | heavy_sq | 500,000 | 150 | 25 | 13 |
| aega | heavy_sq | 1,000,000 | 75 | 100 | 12 |
| aega | light | 1,000,000 | 75 | 100 | 11 |
| aega | light | 500,000 | 150 | 25 | 10 |
| aega | light | 200,000 | 75 | 25 | 9 |
| aega | heavy_dist | 500,000 | 100 | 100 | 8 |
| aega | light | 500,000 | 100 | 100 | 7 |
| aega | light | 200,000 | 100 | 50 | 7 |
| aega | heavy_dist | 200,000 | 100 | 50 | 6 |
| aega | heavy_sq | 200,000 | 100 | 50 | 5 |
| aega | heavy_sq | 500,000 | 100 | 100 | 5 |
| hv | heavy_sq | 200,000 | 150 | 100 | 2 |
| cd | light | 200,000 | 150 | 100 | 1 |
| cd | heavy_dist | 200,000 | 150 | 100 | 1 |
| hv | heavy_dist | 200,000 | 150 | 100 | 1 |

Table 2: Configurations reaching the Pareto-optimal point on `easy_map` 35 × 35. *Hits* = number of runs achieving the optimum out of 31. Tree selection: `hv` = hypervolume, `cd` = crowding distance, `aega` = adaptive epsilon grid archiving. Simulation: `light` = light rollout, `heavy_dist` = heavy distance weight, `heavy_sq` = heavy square sampling.

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| cd | light | 200,000 | 75 | 25 | 31 |
| hv | light | 500,000 | 75 | 50 | 31 |

continued on next page

Table 2 – continued

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| hv | light | 500,000 | 100 | 100 | 31 |
| cd | heavy_sq | 500,000 | 75 | 50 | 31 |
| hv | heavy_sq | 500,000 | 75 | 50 | 31 |
| hv | heavy_sq | 500,000 | 150 | 25 | 31 |
| hv | heavy_dist | 200,000 | 75 | 25 | 31 |
| cd | heavy_dist | 1,000,000 | 75 | 100 | 31 |
| cd | heavy_dist | 1,000,000 | 150 | 50 | 31 |
| hv | light | 500,000 | 150 | 25 | 30 |
| hv | light | 200,000 | 75 | 25 | 30 |
| hv | light | 1,000,000 | 150 | 50 | 30 |
| cd | heavy_sq | 1,000,000 | 75 | 100 | 30 |
| hv | heavy_sq | 200,000 | 100 | 50 | 30 |
| hv | heavy_sq | 200,000 | 75 | 25 | 30 |
| hv | heavy_sq | 1,000,000 | 100 | 25 | 30 |
| hv | heavy_dist | 1,000,000 | 75 | 100 | 30 |
| cd | heavy_dist | 1,000,000 | 100 | 25 | 30 |
| cd | heavy_dist | 500,000 | 75 | 50 | 30 |
| hv | light | 200,000 | 100 | 50 | 29 |
| hv | light | 1,000,000 | 100 | 25 | 29 |
| cd | light | 500,000 | 75 | 50 | 29 |
| hv | light | 1,000,000 | 75 | 100 | 29 |
| cd | light | 1,000,000 | 75 | 100 | 29 |
| cd | light | 1,000,000 | 150 | 50 | 29 |
| hv | heavy_sq | 1,000,000 | 75 | 100 | 29 |
| cd | heavy_sq | 500,000 | 150 | 25 | 29 |
| cd | heavy_sq | 1,000,000 | 100 | 25 | 29 |
| hv | heavy_dist | 500,000 | 100 | 100 | 29 |
| cd | heavy_dist | 200,000 | 100 | 50 | 29 |
| cd | heavy_dist | 500,000 | 100 | 100 | 29 |
| hv | heavy_dist | 1,000,000 | 150 | 50 | 29 |
| hv | heavy_dist | 200,000 | 100 | 50 | 29 |
| hv | heavy_dist | 1,000,000 | 100 | 25 | 29 |
| cd | heavy_dist | 500,000 | 150 | 25 | 29 |
| hv | heavy_dist | 500,000 | 150 | 25 | 29 |

continued on next page

Table 2 – continued

| Tree | Simulation | Budget | Per-Sim | Sims | Hits |
|---|---|---|---|---|---|
| cd | light | 500,000 | 100 | 100 | 28 |
| cd | light | 500,000 | 150 | 25 | 28 |
| hv | heavy_sq | 1,000,000 | 150 | 50 | 28 |
| hv | heavy_sq | 500,000 | 100 | 100 | 28 |
| cd | heavy_sq | 1,000,000 | 150 | 50 | 28 |
| cd | heavy_dist | 200,000 | 75 | 25 | 28 |
| hv | heavy_dist | 500,000 | 75 | 50 | 28 |
| cd | light | 1,000,000 | 100 | 25 | 27 |
| cd | light | 200,000 | 100 | 50 | 27 |
| cd | heavy_sq | 200,000 | 75 | 25 | 27 |
| cd | heavy_sq | 500,000 | 100 | 100 | 27 |
| cd | heavy_sq | 200,000 | 100 | 50 | 24 |
| aega | heavy_sq | 200,000 | 75 | 25 | 21 |
| hv | heavy_sq | 200,000 | 150 | 100 | 21 |
| aega | heavy_sq | 500,000 | 150 | 25 | 21 |
| aega | heavy_dist | 500,000 | 75 | 50 | 20 |
| aega | light | 1,000,000 | 100 | 25 | 19 |
| aega | light | 200,000 | 75 | 25 | 19 |
| aega | heavy_sq | 500,000 | 75 | 50 | 19 |
| aega | light | 1,000,000 | 75 | 100 | 18 |
| aega | heavy_dist | 1,000,000 | 75 | 100 | 18 |
| aega | heavy_dist | 1,000,000 | 100 | 25 | 18 |
| aega | light | 500,000 | 150 | 25 | 17 |
| aega | heavy_sq | 500,000 | 100 | 100 | 17 |
| aega | heavy_sq | 1,000,000 | 100 | 25 | 17 |
| cd | heavy_sq | 200,000 | 150 | 100 | 17 |
| aega | heavy_dist | 200,000 | 75 | 25 | 17 |
| aega | heavy_dist | 500,000 | 100 | 100 | 17 |
| aega | heavy_dist | 1,000,000 | 150 | 50 | 16 |
| aega | heavy_sq | 1,000,000 | 75 | 100 | 15 |
| aega | heavy_sq | 1,000,000 | 150 | 50 | 15 |
| aega | heavy_dist | 500,000 | 150 | 25 | 15 |
| aega | light | 500,000 | 75 | 50 | 14 |
| aega | light | 500,000 | 100 | 100 | 12 |

continued on next page

Table 2 – continued

| **Tree** | **Simulation** | **Budget** | **Per-Sim** | **Sims** | **Hits** |
|---|---|---|---|---|---|
| `aega` | `heavy_sq` | 200,000 | 100 | 50 | 12 |
| `aega` | `light` | 1,000,000 | 150 | 50 | 11 |
| `aega` | `light` | 200,000 | 100 | 50 | 8 |
| `aega` | `heavy_dist` | 200,000 | 100 | 50 | 6 |
| `aega` | `heavy_sq` | 200,000 | 150 | 100 | 5 |
| `cd` | `heavy_dist` | 200,000 | 150 | 100 | 2 |

Appendix: STATEMENT OF AUTHORSHIP of the Student

Thesis: Navigating Path Influenced Environments: MCTS and Two-Phase A* for Multi-Objective Roundtrip Problems

Name: Malte Florim Surname: Speidel

Date of birth: 12.06.2001 Matriculation no.: 227252

I herewith assure that I have written the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or according to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Furthermore, I confirm that I am aware that the use of content (including but not limited to text, figures, images and code) generated by artificial intelligence (AI) in the thesis must be disclosed. In those cases I have specified the AI system used, I have marked the specific sections of the thesis where AI-generated content was used, and I have provided a brief explanation of the level of detail at which the AI system was used to generate the content. I also stated the reason for using the tools.

I assure that even if a generative AI system has been used, the scientific contribution has been made entirely by myself.

Magdeburg, 31.03.2026

*Signature*