Using Machine Learning Techniques for Autonomous Planning and Navigation with Groups of Unmanned Vehicles

GERBEN BERGWERFF

JULY 19, 2016

MASTER'S THESIS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE,
UNIVERSITY OF GRONINGEN,
THE NETHERLANDS

Internal supervisor:
DR. MARCO WIERING
Artificial Intelligence,
University of Groningen,
The Netherlands

External supervisor:
DAVID MOBACH
D-CIS Lab,
Thales Research & Technology,
The Netherlands

university of groningen

faculty of mathematics and natural sciences

Contents

List of Figures

List of Tables

Abstract

Planning trajectories of multiple unmanned vehicles is a complicated task with a large amount of possible solutions. In recent research different types of algorithms are used to solve these planning issues with mixed results. In this research we use two different types of machine learning algorithms and compare these against a baseline greedy method. The first method is based on reinforcement learning (RL) with features, the second method is based on multi ant colony systems (MACS). To measure the performance of the algorithms we created a grid world environment with a task where a number of UAVs need to visit a number of areas. When testing both the RL and MACS algorithm on this problem, we found that the MACS algorithm gives the best solution but is computationally intensive when the problem is scaled. The RL algorithm scales better but is outperformed by the greedy method, making the MACS algorithm the best performing among the tested algorithms.

Chapter 1

Introduction

Unmanned vehicles (UxV) are becoming increasingly popular for a wide variety of tasks. One of the reasons UxVs became popular lately is due to UxVs being a cheap alternative to human operated vehicles. UxVs can be used for tasks that are impossible for human operated vehicles (e.g. inspection of nuclear reactors) [3], or tasks that are simply too expensive when carried out by human operated vehicles (e.g. making an aerial recording of amateur sport events).

Interesting commercial and military applications for UxVs are for example surveillance or reconnaissance of a terrain. UxVs can monitor crops [4], forest fires [5, 6] or a battle field [7]. The above tasks can be carried out by using a single UxV, but the size of the area to be monitored is restricted by the range and speed of the UxV. Using a team of multiple UxVs can overcome this problem, but introduces new challenges. The main challenge for applying teams of UxVs in these surveillance and reconnaissance missions lies in planning of the trajectory each UxV travels.

Recently the field of multi-UxV planning became popular, resulting in extensive research of the subject. A lot of multi-UxV planning research focusses on planning a trajectory to a single target destination, instead of planning the order of the targets set for the UxVs [8]. Another large research area focusses on interface design for letting operators interact with a group of UAVs [9]. Some interesting research subjects are found in the area of swarm robotics, where a large number of robots with only a simple set of rules can result in complex behaviour. An example of this is the use of formation flights with multiple UAVs [10, 11]. UxVs have technical and practical limits for communicating with each other, the issue of communication in multi-UxV systems is also addressed in recent research [12, 11].

Centralised off-line planning algorithms are used widely to solve planning tasks concerning a group of agents that need to execute a unified goal [13]. However, centralised off-line planning has the disadvantage that it is not very robust to changes in the environment or partial observability of the environment. As an alternative, centralised on-line planning can be used. This has the advantage of being able to handle changes in the environment, since the planning is constantly adjusted by the detected changes in the environment [13]. Both types of centralised planning have the disadvantage of needing a dedicated central planner and constant communication between the agents and this planner. Efforts to overcome the problems of centralised planning have been made by decentralised planning, where every agent participates in the planning process and the agents together find the best trajectory based on all their observations and goals. While this reduces the need for a dedicated central planner, communication between the agents still proposes a problem. For each agent to make the best decision, information about goals and observations of other agents is needed. This information has to be obtained by some form of communication. Furthermore there can be technical or practical limits to communication because of limits in range or because of a threat of detection.

Planning of multiple UxV trajectories can be seen as a Markov decision process (MDP), where the UxVs are in a state and have several actions they can choose which will lead to a new state. Planning then becomes a matter of sequentially choosing the best possible action to reach a goal. Reinforcement learning is an area of machine learning that focuses on exactly this problem: What actions should agents choose in order to maximise the reward [14].

Another way to look at planning is in the sense of a graph representation, where the nodes are the states the UxV can be in and the edges are the actions that a UxV can take. Planning in a graph is choosing edges to visit nodes in an optimal order. Ant system algorithms [15] are a type of algorithms inspired on the foraging behaviour of ants. These algorithms are proven to be efficient and give high quality solutions for planning of optimal routes in a graph [16].

1.1 Research Question

In this thesis we will continue on research for the UAV surveillance task, by researching algorithms already proven to be effective in comparable problems. We will implement MACS and RL algorithms for autonomous multi-UxV trajectory planning in surveillance and reconnaissance missions.

The main research question is:

- What is the best machine learning technique for autonomous planning in a group of UxVs?

Sub questions that will be answered, to help answering the main research question are:

- How will a proposed solution deal with a different number of unmanned vehicles?
- How will a proposed solution deal with scaling of the problem?
- Is the proposed solution usable as a general solution for the autonomous planning problem as a whole?

1.2 Outline

In this thesis, we will first review research about different machine learning techniques that are relevant for answering our research questions. This is done in the Theoretical Framework, Chapter 2.
Then in Chapter 3, we will introduce the UAV planning task we use to compare the different machine learning techniques. We also introduce three different algorithms based on different machine learning techniques to solve this planning task.
The experiments and their results are given in Chapter 4, which explains the experiments that are used to determine how the algorithms deal with a different number of unmanned vehicles and how they deal with scaling of the UAV planning task.
The final chapter of this thesis, Chapter 5, discusses the results of the experiments and answers our research questions.

Chapter 2

Theoretical Framework

In this thesis, different forms of machine learning will be compared to one another. We chose to research branches of machine learning which are very different in the way they solve the problem.
As a first machine learning technique we look at reinforcement learning [17]. This learning technique shows great results for generating universal solutions for a problem, furthermore it has the ability to be used as a decentralised planner.
As a second machine learning technique we look at ant colony inspired machine learning [15]. These algorithms are known for their excellent solutions on difficult planning problems. However, as opposed to reinforcement learning, ant inspired algorithms are a form of off-line centralised planning that generate problem-specific solutions.
In this chapter we will summarise those techniques and emphasise relevant research used to construct our methods.

2.1 Reinforcement Learning

A very broad definition of reinforcement learning is found in [1]: Dynamic programming (DP) and reinforcement learning (RL) are algorithmic methods for solving problems in which actions (decisions) are applied to a system over an extended period of time, in order to achieve a desired goal.
The basic principles of RL can be found in the psychology on behavioural conditioning. An early example of the psychological basis of RL is Thorndike's Law of effect: "responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation." [18]. Thorndike explains the general idea of RL as it is used today. The RL agent receives negative rewards for unwanted behaviour and positive rewards for correct behaviour. With the rewards it learns to maximise its performance and thus maximise its rewards.

One of the first researchers that researched this law of effect as a way to create artificial intelligence was Turing. He developed his idea of a pleasure-pain system [19], which is basically an RL agent as we know it today. In his research he defines how a pleasure-pain system should be designed and under what conditions it should perform.
RL differs from supervised learning methods as neural networks and unsupervised learning methods as clustering. While an RL agent receives feedback of its performance, as in supervised learning methods, the agent does not receive examples of optimal actions in a situation. All information about the performance is given in a single reward, which is a measure of how good or bad an RL agent performs.
The type of problems suitable for unsupervised learning and RL differ. In for example a clustering task, the outcome should be a number of clusters and the data points inside each cluster. In RL, the agent has a continuous decision task, where at every time step there are several actions to be chosen from.

2.1.1 Markov Decision Process

A Markov decision process (MDP) is a theoretical framework that can be used for decision making in environments that are fully observable [14]. MDPs are at the basis of Reinforcement Learning. An MDP contains all states a world can be in and all the transitions between these states.
An MDP is written as a quintuple of $\langle S, A_s, T_a(s,s'), R_a(s,s'), \gamma \rangle$. Where S is a set of all states ($s \in S$) the MDP can be in. A_s is the set of actions the agent can choose when in state s ($a \in A_s$). The transition $T_a(s,s')$ is the probability of getting to state s' when the agent is in state s and chooses action a. The reward function $R_a(s,s')$ is the corresponding reward when action a in state s leads to state s'. γ is the discount factor, if $\gamma = 0$ only the immediate reward is deemed to be of any importance, if $\gamma = 1$ the future reward (i.e. the sum of all rewards) is of most importance.
An example of an MDP would be a game of tic-tac-toe. All the possible states the game can be in would form the states of the MDP. Transitions between those worlds would be the addition of a cross or a circle, resulting in another state of the MDP. These transitions would be the possible transitions between the states of the MDP.
For the notion of this project, it is easier to look at RL as an agent interacting with an environment through actions, and receiving the state of the environment through sensory input, as seen in figure 2.1. The dashed arrows in the schematic show the reward function. The agent receives a reward for the interaction with the environment. This reward is based on the action of the agent and the state of the environment.

2.1.2 Reward Function

RL is learning to take the correct action by trial and error, based on the received reward over time. This type of learning has similarities with human learning, where a negative

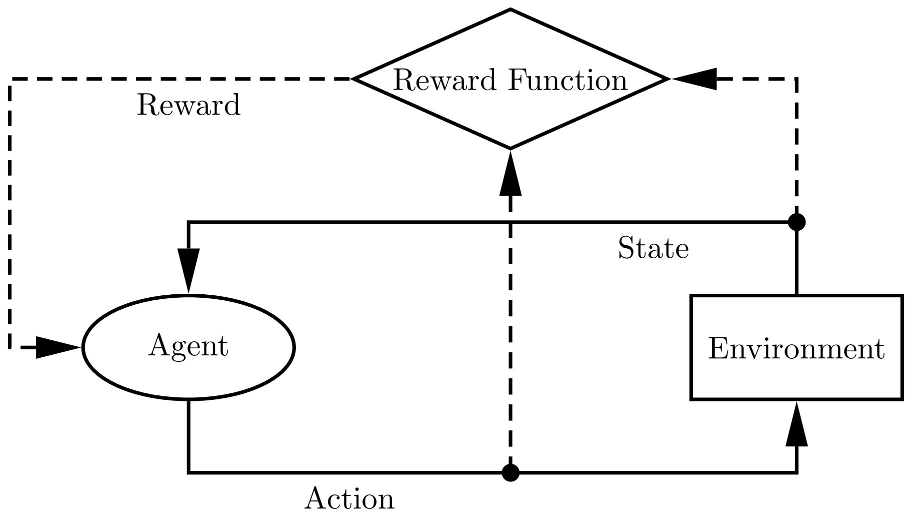


Figure 2.1: Control loop of Reinforcement Learning [1]. The reward function is depicted with dashed arrows.

reward is pain and a positive reward is pleasure. The goal of the RL problem is expressed to the agent in the form of a reward. This reward is mostly given at once if a certain goal is reached, although other types of reward function are possible.

Defining the rewards an agent receives is for a large part depending on the problem the agent needs to solve. When the agent is trained to solve a game, most rewards are already defined by the rules of the game. But when transforming an arbitrary problem to an MDP for solving by RL, the reward function is not quite as straightforward and it may require some trial and error to find a reward function that satisfies the goal.

The agent can receive rewards at any time step. If the agent receives a reward for winning a game of chess, there is a whole sequence of actions that led to this reward. In the game of chess, the first move is of particular importance for the rest of the game. However, how should an agent without any particular knowledge of chess give credit for the good moves within a sequence and blame the bad moves within a sequence? This problem is known as the credit assignment problem: what action or actions within a sequence of actions led to the received reward. This problem was introduced in a paper by Minsky [20]. RL agents are constantly trying to solve this problem in order to maximise their rewards.

2.1.3 Value Functions and Function Approximation

RL tries to find a value function to predict how good or bad a state is. A value function predicts the return of a state by estimating the sum of future rewards resulting from the state. With this value function, it is easy to estimate the best action amongst all possible

actions at any time. At any given time, the agent knows its own state and the possible actions. For our small tic-tac-toe example, the number of states is manageable. When an agent is learning the game, the easiest and most understandable way to store the value of each state, is to simply create a table with the same size as the number of states. Each row in this table represents the value of the corresponding state. This is a perfectly fine solution for small state-space problems. But imagine a game of Go, the state-space of Go is huge [21]. A look-up table for Go would be equally large. These huge look-up tables are unmanageably large, since each state needs to be visited at least once to calculate a value. For an accurate value, a state needs to be visited multiple times, hence look-up tables are a bad idea for large state-space problems.
An alternative to look-up tables are function approximators. Instead of constructing a table of the values, the agent tries to find a function that predicts the value of a state. This reduces the need to visit every state several times, but introduces new complexity in the form of choosing the correct function approximator for the problem. Examples of function approximation will be given in Section 2.1.8.

2.1.4 Exploration-Exploitation Dilemma

Other than in supervised learning or unsupervised learning, in RL there is a constant dilemma between exploring better solutions and exploiting the knowledge the agent already has. This is known as the exploration-exploitation dilemma.
Agents are faced with different actions resulting in different states if chosen. Through experience, agents know which action has the highest value. But the action with the current highest value is not necessarily the best action. An agent can explore other actions that may not seem attractive given their current values, but are possibly better than the best known action. However, this may also have an impact on the overall performance of the agent. The key to the solution is finding an equilibrium between exploring enough actions to increase overall performance by overcoming local optima and exploiting the current knowledge of the agent. Although attempts have been made to set boundaries for finding an optimal ratio, some of these methods make assumptions about the problem being solved [17]. These exploration strategies are further explained in Section 2.1.7.

2.1.5 Partial Observability

In the real world, very few problems are fully observable and thus fulfil the prerequisites of an MDP. In order to still be able to solve these problems through RL, Partial Observable MDPs (POMDPs) have been introduced. The difference between an MDP and a POMDP lie in how the state-space is defined. A POMDP works with belief states, this can be seen as an estimation of the real state. As in any MDP, the agent makes transitions from one state to another. The belief state is chosen as a probability of the agent being in each possible state. These probabilities are calculated by observation of the real-world state through sensory input.

2.1.6 Temporal Difference Learning

Temporal Difference (TD) learning uses the idea of bootstrapping to estimate the value of a state: using estimates of other values to estimate a value [14]. The set of algorithms that use TD-learning are specific to RL.

Q-Learning

One of the TD-based algorithms that is often used is Q-learning [22]. Q-learning is an off-policy, or policy independent, algorithm. This means it tries to find an optimal policy for the problem, which is different from the policy the agent is using.
In equation 2.1 we see the update step of Q-learning. $Q(s_t, a_t)$ is the value of acting by performing action a_t in state s_t at time t. ω is the learning rate. γ is the discount factor as explained in the MDP paragraph. r_t is the reward at time t.
Q-learning updates the value using the value of the greedy action a in state s_{t+1} and the obtained reward r_t.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \omega[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.1}$$

SARSA

SARSA stands for State-Action-Reward-State-Action [14]. SARSA is an on-policy TD algorithm [23, 17]. In equation 2.2 we see the update step of SARSA. This update step is almost the same as the Q-learning update step in equation 2.1, but differs in omitting the greedy action selection and using an on-policy approach.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \omega[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.2}$$

SARSA updates the value using the reward r_t and the value of the next state s_{t+1} and the policy action a_{t+1}. One of the benefits of SARSA is that it is useful when function approximation is used, because of the on-policy approach [14]. SARSA is also proven to converge to the optimal solution, under the assumption that all actions are chosen infinitely in every state [14].

2.1.7 Action Selection Methods

At any time step, the agent finds itself in a state and must select an action to perform. The agent can choose to either exploit or explore. There are several methods to determine how to divide exploration and exploitation.

$\mathcal{E}$-Greedy

The ε-*greedy* method is one of the easiest and simple to understand methods for dividing between exploration and exploitation. By default the greedy action is chosen, as defined in equation 2.3.

$$a_t = \arg\max_a Q_t(s_t, a) \tag{2.3}$$

However, to accommodate exploration, a random action is chosen with a probability of ε.

Boltzmann Exploration

Taking random actions for exploration is not always desired behaviour, because this means that it is equally likely that the worst possible action is chosen as that the second best possible action is chosen. Especially if the worst possible action is very bad for the agent, it would be better to use a probability an action is chosen instead.
One example of an action selection method that uses the probability that an action is chosen is the Boltzmann exploration method [17, 1]. This method can be seen in Equation 2.4, which determines the probability an action a is selected in state s_t.

$$P\left(a|s_t\right) = \frac{e^{Q_t(s_t,a)/\tau_t}}{\sum_b e^{Q_t(s_t,b)/\tau_t}} \tag{2.4}$$

Boltzmann exploration uses a temperature τ_t to determine the randomness of the exploration, where $\tau_t \to 0$ equals greedy action selection and $\tau_t \to \infty$ equals random action selection. The temperature is usually diminished over the number of steps, so a lot of random exploration takes place in the first steps, but as knowledge about the world is gathered the exploration becomes more guided by the Q-function.

2.1.8 Function Approximation

One of the largest challenges in RL is what is called the curse of dimensionality. This means that when the number of dimensions of a problem increases, so does the number of possible solutions for the problem. In RL this is particularly problematical for the number of the state-action pairs. For algorithms as Q-learning and SARSA, all state-action pairs should be tried infinitely many times in order for the algorithms to converge. In practice this is of course impossible, but the larger the number of state-action pairs becomes, the harder it becomes to make an accurate estimate of the state-action value function.
One way to handle this problem is by trying to reduce the number of trainable parameters by generalising in some way. In RL it is common practise to make use of what is called function approximation (FA) [1]. In FA, the number of trainable parameters is reduced in order to reduce the number of possible solutions. This can be done by converting the state into a number of features, thus resulting in fewer dimensions and fewer possible solutions.

Tiling

A popular and easy to understand method is *tiling*. In tiling, features are created by dividing the state-space in small areas, called tiles. These tiles do not need to be symmetrically shaped or square, but can have any arbitrary form and amount. The result is a collection of features that are binary: high or "1" if the current state is inside the tile, low or "0" when the current state is not inside the tile. Multiple features can be active at the same time, tiles may also overlap each other. The coarser the tiling, the more generalisation takes place. The finer the tiling, the more accurate the solution, but the more computational intensive the process.

Features

One straightforward approach for FA is using a linear combination of features to estimate the value of a state [24]. This can be seen in equation 2.5, where the features of a state ϕ_s are multiplied by the weights of the features θ_t at time t.

$$V_t(s) = \vec{\theta}_t^{\intercal} \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{2.5}$$

This method has successfully been used to create a reinforcement learning agent using features for playing the Tetris game [24, 25]. Examples of the features that are used in these agents are the maximum and minimum height of the stacked blocks, the sum of the depth of wells between the stacked blocks and the number of these wells.

These approaches are successful for creating an agent for playing the Tetris game, while using relatively few resources.

2.2 Ant Colony Algorithms

Ant Colony algorithms [15, 16] are based upon the behaviour of real world ants. Goss et al. [2] ran experiments with living ants, using an experiment setup where the ants had two ways of reaching their food. The ant colony was situated on one side of the setup and the food was situated on the other side of the setup. There were two links between the colony and the food, two bridges that differ in total length of the path from the food to the colony, as seen in Figure 2.2. The experiment showed that the ants prefer the shortest path (i.e.: the short bridge) above the longer path (i.e.: the long bridge). It was also shown that the larger the difference in total length between the two paths, the more ants chose the shorter path over time. This research showed that ants converge to the shortest path to food over time. The larger the difference in path length, the faster the ants converge.

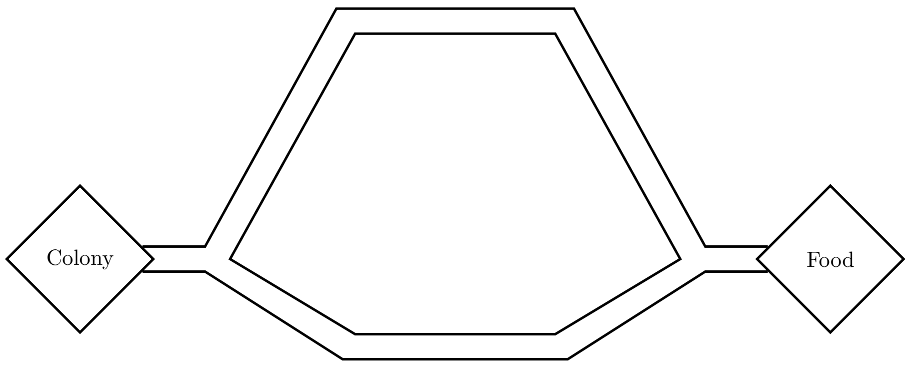


Figure 2.2: Ant colony experiment setup with two bridges to reach food that differ by total path length. Over time ants preferred the shorter path over the longer path. [2]

2.2.1 Ant System

The first ant colony optimisation (ACO) algorithm was inspired on the results of the research of Goss et al. [2]. This ACO algorithm is the Ant System (AS) [15]. The idea of AS is to represent problems as a weighted graph with nodes that are interconnected by edges. As with every weighted graph, each edge has a cost $\delta(r, s)$. On top of this, AS introduces a new variable for each edge: desirability $\tau(r, s)$. This desirability represents *pheromone* in the ant analogy. Pheromone levels are constantly updated by all *artificial ants* (ants) in the AS.
In the travelling salesman problem (TSP), there are i cities that are all connected to one or more cities, with the distance between cities as cost of travelling. This forms a weighted graph. The goal is to visit all cities in such an order that the total cost of the tour through all the cities is minimal. The paper by Dorigo [15] solves a TSP by use of AS, so all formulae are to be interpreted in this context.

Random-Proportional Rule

In equation 2.6 we see the state transition rule used by AS. In this rule, the probability of an Ant k moving from node r to node s is given.

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)] \cdot [\eta(r,s)]^\beta}{\sum\limits_{u \in j_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^\beta} & \text{if } s \in J_k(r) \\ 0 & \text{else} \end{cases} \tag{2.6}$$

The ant uses the pheromone between two nodes τ and the inverse of the distance $\eta(r, s) = 1/\delta(r, s)$ to determine how attractive an edge is to visit. Only edges that are not visited yet are considered, these are contained in $J_k(r)$. The parameter β is used to create a bias for either distance or pheromone in the decision.

Pheromone Update Rule

Equation 2.7 gives the pheromone update rule. The pheromone levels $\tau(r,s)$ of all edges are updated at the end of a tour. Parameter α controls pheromone decay, this means that the pheromone intensity of an edge is decreased according to α every end of a tour. If $\alpha = 0$ there is no decay, if $0 < \alpha \leq 1$ pheromone decay takes place. The next part of the rule sums $\Delta\tau_k(r,s)$ for all m ants.

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \sum_{k=1}^{m}\Delta\tau_k(r,s) \tag{2.7}$$

The definition for $\Delta\tau_k(r,s)$ is given in equation 2.8, where L_k is the length of the tour of ant k. This means that ants that found a short tour leave relatively more pheromone on an edge, thus the colony as a whole will converge to a minimal cost solution over time. This method has similarities with RL as explained in Section 2.1. The pheromone update step reinforces the best solutions and by pheromone decay the other edges are made less attractive for the ants.

$$\Delta\tau_k(r,s) = \begin{cases} \frac{1}{L_k} & \text{if } (r,s) \in \text{tour done by ant } k \\ 0 & \text{else} \end{cases} \tag{2.8}$$

The pheromone in an AS serves the function of memory and indirect communication, ants deposit pheromone to memorise the best tours. Ants also are attracted to high pheromone edges when deciding to choose an edge at a node, this form of communication is known as *stigmergy* [26, 27].

2.2.2 Ant Colony System

The Ant Colony System (ACS) is an improved and optimised version of AS, created by Dorigo et al. [16]. The goals of ACS were to improve AS on three aspects [16]:

- Improve the state transition rule by incorporating exploitation and guided exploration.
- Only apply global pheromone updates for the best tour instead of all tours.
- Use a local pheromone update when ants are constructing their tour.

State-Transition Rule

ACS uses a pseudo-random-proportional rule. This is a combination of Equation 2.9 and Equation 2.6. Equation 2.9 is used to balance between exploitation of the knowledge within the system and guided exploration. q is a random variable ($0 \leq q \leq 1$), q_0 is a parameter that controls the exploration-exploitation ratio, S is a city selected by Equation 2.6 [16].

$$s = \begin{cases} \arg \max\limits_{u \in J_k(r)} \{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}\} & \text{if } q \leq q_0 \\ S & \text{else} \end{cases} \tag{2.9}$$

In all cases where $q \leq q_0$, the knowledge of the system is exploited. The most attractive known edge is chosen in this case.
In all other cases, guided exploration takes place according to the random-proportional rule (Equation 2.6). This means an edge is chosen by using the distribution of the random-proportional rule.

Global Update Rule

The second improvement over AS is the new global update rule as seen in Equation 2.10. The global update rule only uses information of the best tour to update the pheromone deposits on the edges.

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s) \tag{2.10}$$

As opposed to AS, in ACS only the best ant deposits pheromone on edges. The amount of pheromone deposit $\Delta\tau(r,s)$ is determined in Equation 2.11. The decay of pheromones over time is regulated by α.

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & \text{if } (r,s) \in \text{global best tour} \\ 0 & \text{else} \end{cases} \tag{2.11}$$

Local Update Rule

ACS also implements a local update rule, given in Equation 2.12. In this rule $\Delta\tau(r,s) = \tau_0$. τ_0 is a parameter that contains the default level of pheromone used at initialisation time $t = 0$. The parameter ρ is used to control the relative importance of the local update.

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{2.12}$$

The goal of local updating is to encourage diversification while constructing a tour. If there is no local updating, all ants are likely to construct a tour near the then optimal tour. This diversification is realised by reducing pheromone levels on an edge if an ant has passed that edge, this will encourage the next ant passing the same node constructing its tour using another edge, thus encouraging exploration.

2.2.3 Multiple Ant Colony Systems

Multiple ant colony systems (MACS) are a combination of 2 or more ACSs. MACS have the advantage that they can cope with more than one goal, as opposed to ACS.
The TSP solved by AS and ACS, is a problem with a single agent making a choice based on a single cost measure. However, a lot of planning problems do not consist of a single agent or a single cost measure. Take for example the bus stop allocation problem (BAP) of de Jong and Wiering, as described in [28]. In the BAP there are n bus lines and m bus stops that need to be visited by at least one bus line. Bus stops have a supply of passengers that need to be transferred from one bus stop to another bus stop. In order to solve this problem with an ACS, the authors use MACS, where each ACS constructs a bus line. All ACSs cooperate to solve the problem of constructing bus lines to find a set of bus lines that has the lowest total costs.
This idea of using MACS to deal with problems where there are either multiple agents or multiple goals is also used in other types of research. Notable usage is in the vehicle routing problem with time windows (VRPTW) [29] or the vehicle routing problem with back hauls (VRPB) [30].
Vermeulen [31] used MACS as a way to solve issues in the planning of train drivers and conductors that occur after a disruption. In his research, all the train drivers and conductors were represented by an ACS. The MACS algorithm gives solutions in the form of employee shifts between trains.

Chapter 3

Methods

Our main research question is: “what is the best machine learning technique for autonomous planning in a group of UxVs?”. From this section on, we will discuss the issue of UAV planning, making the assumption that promising high level UAV planning algorithms also gain good results for UxV planning.
To answer the research question we first made an artificial framework for testing the performance of a planning algorithm. This is explained in Section 3.1.
We then created three algorithms that solve the planning problem in a different way. The first algorithm being a baseline approach where we use a greedy planner based on A* search for the shortest path, this is further explained in Section 3.2. The second algorithm being a reinforcement learning approach, where we use a feature-based linear function approximation in combination with Q-learning, explained in Section 3.3. The last algorithm uses a multi ant colony system in combination with A* for path planning, explained in Section 3.4. The performance of the algorithms is then compared by using research techniques explained in Section 3.5.

3.1 UAV Grid World

To compare different algorithms on a task, we first created a framework and a task where a fleet of UAVs needs to visit a number of areas. The UAV framework is based on a grid world, where each square in a grid resembles an area in the real world. This can be seen in the same way as a grid on a map, where each square in the grid represents an area of the world.
The grid world consists of a grid with X and Y coordinates, each X and Y coordinate corresponding to a square in the grid. The grid also has layers, which can be seen as a Z coordinate, but do not represent height. The layers are implemented to be able to let a grid contain different agents at the same X and Y coordinates.
In this research we focus on the high level planning. This means we write algorithms that

plan paths for the group of UAVs. Paths are to be interpreted as a list of neighbouring squares, where $\Delta x \in \{-1,\ 0,\ 1\}$, $\Delta y \in \{-1,\ 0,\ 1\}$ and $|\Delta x| + |\Delta y| \leq 1$ for each next neighbour. The low level planning, e.g. obstacle avoidance when travelling between two neighbouring squares, is left as further research.

3.1.1 Agents

Each $\{X, Y, Z\}$ may contain one agent. We implemented different types of agents, being:

- Area of interest: a static agent representing an area that needs to be visited once by a UAV. The areas carry a one-time reward of 1.
- No-fly area: a static agent representing an area that needs to be avoided. The areas carry a negative reward of -0.1, given to a UAV each time the area is passed.
- UAV: A dynamic agent that represents a UAV and can move in three directions, forward, left-forward and right-forward.

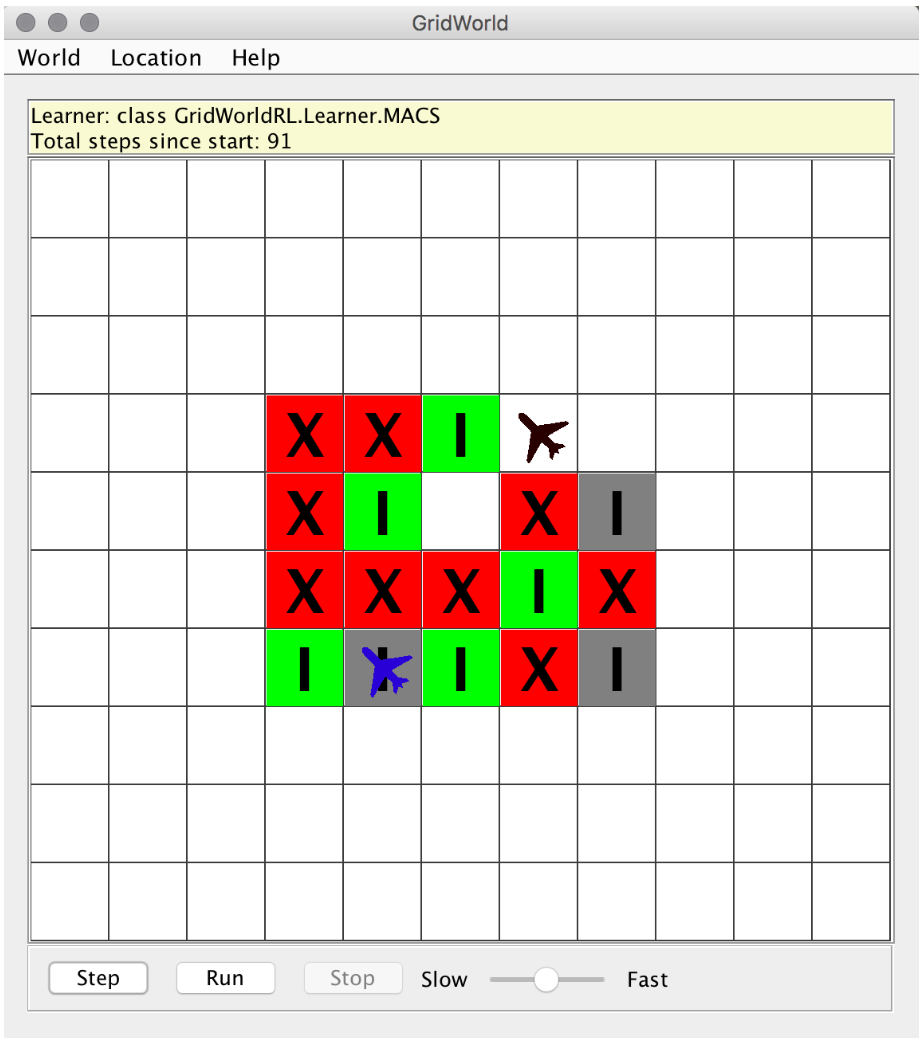
3.1.2 Path and Costs

Each area of interest and no-fly zone carries a reward, but to account for the cost of movement we also implemented a cost function. This resembles the total cost of travelling along a path. A path consists of a sequence of neighbouring squares, where each transition to a neighbouring square is limited to the square in the forward direction of the UAV, the square in the left-forward direction and the square in the right-forward direction. This restriction is implemented to make the simulation more natural for non-helicopter UAVs. These restrictions can be left out for simulating helicopter type UAVs. The cost function is implemented as the sum of euclidean distances between a sequence of neighbouring squares, deducted by a possible reward, as can be seen in Equation 3.1.

$$Cost = \sum_{i=0}^{i=l_{path}-1} \left(\sqrt{(X_i - X_{i+1})^2 + (Y_i - Y_{i+1})^2} - R_{i+1}\right) \tag{3.1}$$

3.2 Greedy Algorithm

An easy to implement and sometimes very effective method for solving a planning problem is a greedy algorithm. The Greedy algorithm is trivial in the sense that it chooses the shortest possible route to an unvisited area of interest at every time step. It does not consider the total reward at the end to be of any interest, but only focusses on the highest reward it can get immediately at every step. This makes a greedy algorithm a very good baseline method, to compare against our other algorithms.



Learner: class GridWorldRL.Learner.MACS
Total steps since start: 91

Figure 3.1: The grid world framework as created for visualisation and testing of the different algorithms. Green squares represent the areas of interest that are unvisited and thus contain a reward, grey areas represent the areas of interest that have already been visited, the coloured planes represent the UAVs, the red squares represent the no-fly areas which contain a negative reward.

For the greedy algorithm, we assume full observability of the environment, except for the location of other agents. Furthermore, we assume that there is communication at the point an agent visits an area of interest.

3.2.1 Grid to Graph

To be able to find the highest reward using the Greedy algorithm, we need to have some kind of measure between two points of interest that need to be visited. We do this by creating a weighted directed graph from our grid world environment. The nodes of the graph are all areas of interest and the edges are routes between the nodes that are planned using A* [32]. Since movement actions of UAVs are restricted for more realistic planning, we also included the direction of the UAV in a graph node. This means that a path between two areas of interest is translated to eight different edges in the graph, corresponding to the eight possible directions of a UAV (i.e. $d \in \{0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°\}$). By including information about the direction we can find an accurate cost of a transition between two nodes. It is notable to point out that when a UAV is on a node, only three edges are visible for that UAV. Namely, the edge corresponding to the direction of the UAV and the edges corresponding to the UAV direction minus 45° and plus 45°. The other edges are not relevant for the planning with that particular UAV at that particular time step, since they are not reachable given the movement restrictions of the UAV.

3.2.2 Greedy Algorithm

The Greedy algorithm can be seen in Algorithm 1. At any point in time, it checks the A* distance from the current position of the UAV to all areas of interest that are not visited. These distances are retrieved from the edges of the graph we discussed in the previous paragraph.

3.3 Feature Reinforcement Learning

Because of the size of our grid world, a simple tabular reinforcement learning approach with Q-learning or SARSA would take an incredibly large amount of time to converge. A relatively simple non-tabular reinforcement learning approach is to make an abstraction of the world by a number of features and then use reinforcement learning to learn from those features. We used this technique to create our Feature-Q algorithm.

For the Feature-Q algorithm we assume full observability of the environment, except for the location of other agents. Furthermore, we assume that there is communication at the point an agent visits an area of interest.

Algorithm 1 Greedy algorithm

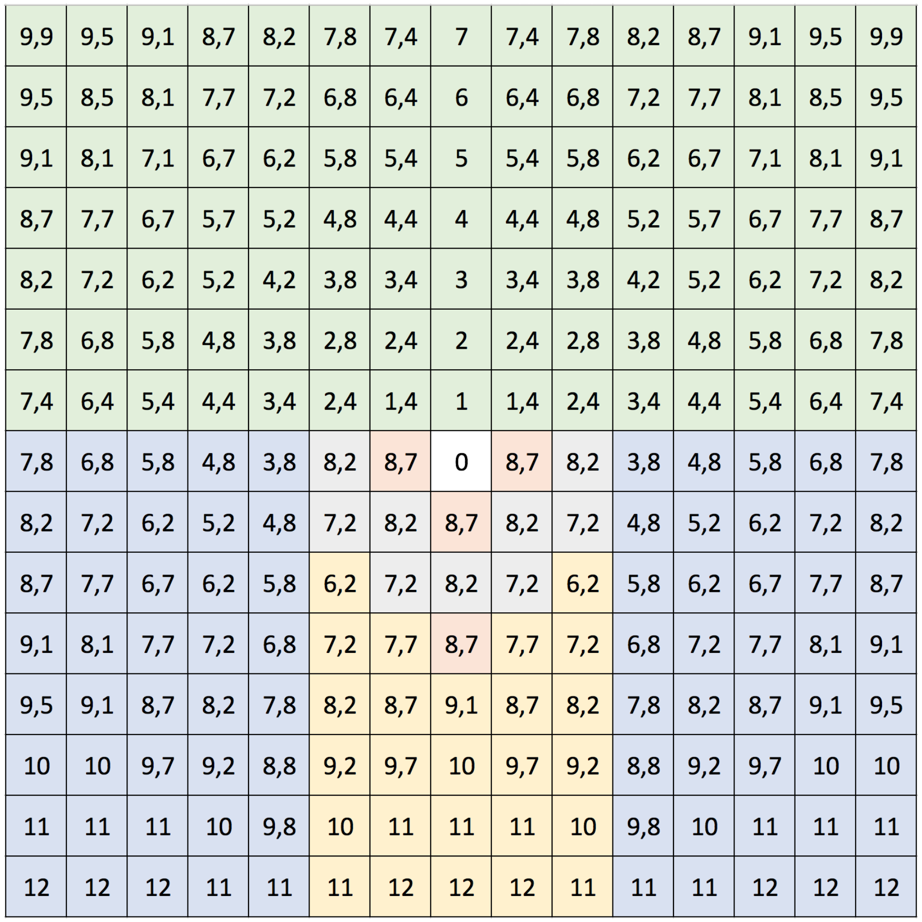
```
1: while not all areas of interest visited do
2:     $distance = \infty$
3:     $next$ = null
4:     for areas of interest do
5:         if not visited then
6:             $AStar$ = A-star distance to area of interest
7:             if $AStar < distance$ then
8:                 $distance = AStar$
9:                 $next$ = area of interest
10:            end if
11:        end if
12:    end for
13:    if $next = null$ then
14:        break while
15:    else
16:        Move to $next$
17:    end if
18: end while
```

3.3.1 Grid to Features

A very important part of the Feature-Q algorithm are the features that are extracted from the state of the grid world at every time step. To make the path planning of UAVs as realistic as possible, we restrict the actions a UAV can take at every time step to: forward, left-forward and right-forward. This imposes an issue when we want to measure the distance between two points. A simple Euclidean or Manhattan distance measure will not give an accurate measure, because of these restrictions in actions. To overcome the issues imposed by action restrictions, we first used the same A* planner we use for the Greedy algorithm to calculate the distance between the agent and all unvisited areas of interest. While this was a valid solution to the problem of determining distance, speed and scalability remained a concern.

When analysing the distances given the restrictions of actions, we found that there were patterns which could be translated to a simple set of formulae, thus reducing the need for a searching method like A*. The patterns we found are denoted as coloured areas in Figure 3.2.

The set of formulae all have a set of criteria that define the area in which they are valid. Please note that the formulae are only valid for UAVs that have a north (or 0°) direction. The distances in the green area can be calculated by Equation 3.2. In the blue area by equation 3.3. In the yellow area by Equation 3.4. In the red area by equation 3.5 and in



9,9	9,5	9,1	8,7	8,2	7,8	7,4	7	7,4	7,8	8,2	8,7	9,1	9,5	9,9
9,5	8,5	8,1	7,7	7,2	6,8	6,4	6	6,4	6,8	7,2	7,7	8,1	8,5	9,5
9,1	8,1	7,1	6,7	6,2	5,8	5,4	5	5,4	5,8	6,2	6,7	7,1	8,1	9,1
8,7	7,7	6,7	5,7	5,2	4,8	4,4	4	4,4	4,8	5,2	5,7	6,7	7,7	8,7
8,2	7,2	6,2	5,2	4,2	3,8	3,4	3	3,4	3,8	4,2	5,2	6,2	7,2	8,2
7,8	6,8	5,8	4,8	3,8	2,8	2,4	2	2,4	2,8	3,8	4,8	5,8	6,8	7,8
7,4	6,4	5,4	4,4	3,4	2,4	1,4	1	1,4	2,4	3,4	4,4	5,4	6,4	7,4
7,8	6,8	5,8	4,8	3,8	8,2	8,7	0	8,7	8,2	3,8	4,8	5,8	6,8	7,8
8,2	7,2	6,2	5,2	4,8	7,2	8,2	8,7	8,2	7,2	4,8	5,2	6,2	7,2	8,2
8,7	7,7	6,7	6,2	5,8	6,2	7,2	8,2	7,2	6,2	5,8	6,2	6,7	7,7	8,7
9,1	8,1	7,7	7,2	6,8	7,2	7,7	8,7	7,7	7,2	6,8	7,2	7,7	8,1	9,1
9,5	9,1	8,7	8,2	7,8	8,2	8,7	9,1	8,7	8,2	7,8	8,2	8,7	9,1	9,5
10	10	9,7	9,2	8,8	9,2	9,7	10	9,7	9,2	8,8	9,2	9,7	10	10
11	11	11	10	9,8	10	11	11	11	10	9,8	10	11	11	11
12	12	12	11	11	11	12	12	12	11	11	11	12	12	12

Figure 3.2: The patterns found when analysing the lowest cost to reach an area. The UAV is located at the white square in the middle, with cost 0. The direction of the UAV is north (0°), facing the green coloured area. Formulae corresponding to the coloured areas are found in Section 3.3.

the grey area by Equation 3.6.

$$\begin{aligned} &Criteria : \Delta y > 0 \\ &\qquad distance = \min\left(|\Delta y|, |\Delta x|\right) * \sqrt{2} \\ &\qquad\quad + \max\left(|\Delta y|, |\Delta x|\right) - \min\left(|\Delta y|, |\Delta x|\right) \end{aligned} \tag{3.2}$$

$$\begin{aligned} &Criteria : \Delta y \leq 0,\ |\Delta x| > 2 \\ &\qquad distance = \left(\min\left(|\Delta y|, |\Delta x| - 3\right) + 2\right) * \sqrt{2} \\ &\qquad\quad + \max\left(|\Delta y|, |\Delta x| - 3\right) - \min\left(|\Delta y|, |\Delta x| - 3\right) + 1 \end{aligned} \tag{3.3}$$

$$\begin{aligned} &Criteria : \Delta y \leq 0,\ |\Delta x| \leq -2,\ \left(|\Delta y| + |\Delta x|\right) \geq 4 \\ &\qquad distance = \left(\min\left(|\Delta y|, |\Delta x| - 3\right) + 2\right) * \sqrt{2} \\ &\qquad\quad + \max\left(|\Delta y|, |\Delta x| - 3\right) - \min\left(|\Delta y|, |\Delta x| - 3\right) + 1 \end{aligned} \tag{3.4}$$

$$\begin{aligned} &Criteria : \Delta y \leq 0,\ \left(|\Delta y| + |\Delta x|\right) = 1 \quad or \quad \Delta y = \text{-3},\ \Delta x = 0 \\ &\qquad distance = 4 * \sqrt{2} \\ &\qquad\quad + 3 \end{aligned} \tag{3.5}$$

$$\begin{aligned} &Criteria : \Delta y < 0,\ 2 \leq \left(|\Delta y| + |\Delta x|\right) \leq 3 \\ &\qquad distance = 4 * \sqrt{2} \\ &\qquad\quad + 6 - \left(|\Delta y| + |\Delta x|\right) \end{aligned} \tag{3.6}$$

A UAV can have 8 directions in our grid world. Three of these directions (90°, 180°, 270°), are effectively the same as the solution for 0°, but rotated. To not have to go through the whole process of finding the patterns and formulae again, we came up with a solution that effectively rotates both the UAV and area to which the distance needs to be calculated. This way we can reuse the same set of formulae as used for the 0° solution.

Our grid world uses rows and columns with an origin in the upper left corner of the grid, while the formulae for determining distances are designed for an origin at the point of the UAV. To overcome this matter of convention we came with a set of formulae for converting from rows and columns to a Δx and Δy. These formulae are shown in Equation 3.7.

$$\Delta x = \begin{cases} col_{goal} - col_{uav} & \text{when } 0^\circ \\ row_{goal} - row_{uav} & \text{when } 90^\circ \\ col_{goal} - col_{uav} & \text{when } 180^\circ \\ row_{goal} - row_{uav} & \text{when } 270^\circ \end{cases}$$
$$\Delta y = \begin{cases} (row_{goal} - row_{uav}) * -1 & \text{when } 0^\circ \\ col_{goal} - col_{uav} & \text{when } 90^\circ \\ (row_{goal} - row_{uav}) * -1 & \text{when } 180^\circ \\ col_{goal} - col_{uav} & \text{when } 270^\circ \end{cases} \tag{3.7}$$

The formulae for calculating the distances for the other 4 directions (45°, 135°, 225°, 315°) were found using the same procedure. We first found patterns for distances in the 315° directions and created formulae the same way as we did for the 0° solution. Next, we convert the rows and columns to a Δx and Δy as by Equation 3.8, then we simply use the formulae for the 315° direction which we determined and can be found in Equations 3.9, 3.10, 3.11, 3.12 and 3.13.

$$\Delta x = \begin{cases} row_{goal} - row_{uav} & \text{when } 45^\circ \\ (col_{goal} - col_{uav}) * -1 & \text{when } 135^\circ \\ (row_{goal} - row_{uav}) * -1 & \text{when } 225^\circ \\ col_{goal} - col_{uav} & \text{when } 315^\circ \end{cases}$$
$$\Delta y = \begin{cases} col_{goal} - col_{uav} & \text{when } 45^\circ \\ row_{goal} - row_{uav} & \text{when } 135^\circ \\ (col_{goal} - col_{uav}) * -1 & \text{when } 225^\circ \\ (row_{goal} - row_{uav}) * -1 & \text{when } 315^\circ \end{cases} \tag{3.8}$$

$$\begin{aligned} &Criteria : \Delta y \geq 0,\ \Delta x \leq 0 \\ &\quad distance = \min(|\Delta y|, |\Delta x|) * \sqrt{2} + \\ &\qquad + \max(|\Delta y|, |\Delta x|) - \min(|\Delta y|, |\Delta x|) \end{aligned} \tag{3.9}$$

$$\begin{aligned} &Criteria : \Delta y \leq -1,\ \Delta x \leq -2 \\ &\quad distance = (\min(|\Delta y| - 1, |\Delta x| - 2) + 1) * \sqrt{2} \\ &\qquad + \max(|\Delta y| - 1, |\Delta x| - 2) - \min(|\Delta y| - 1, |\Delta x| - 2) + 1 \end{aligned} \tag{3.10}$$

$$\begin{aligned} &Criteria : \Delta y = -2,\ \Delta x = 2 \\ &\quad distance = 3 * \sqrt{2} + 3 \end{aligned} \tag{3.11}$$

$$\begin{aligned} &Criteria : -2 \leq \Delta y \leq -1,\ -1 \leq \Delta x \leq 2 \\ &\quad distance = 3 * \sqrt{2} \\ &\qquad + 6 - (|\Delta y| + |\Delta x|) \end{aligned} \tag{3.12}$$

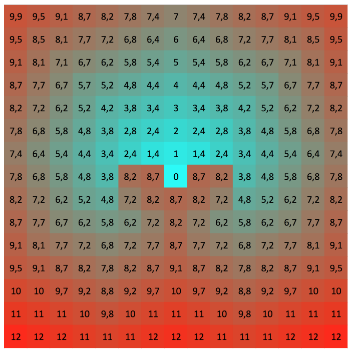
$$\begin{aligned} &Criteria : \Delta y + \Delta x \leq 0 \\ &\quad distance = (\min(|\Delta y| - 3, |\Delta x| + 1) + 2) * \sqrt{2} \\ &\qquad + \max(|\Delta y| - 3, |\Delta x| + 1) - \min(|\Delta y| - 3, |\Delta x| + 1) + 1 \end{aligned} \tag{3.13}$$

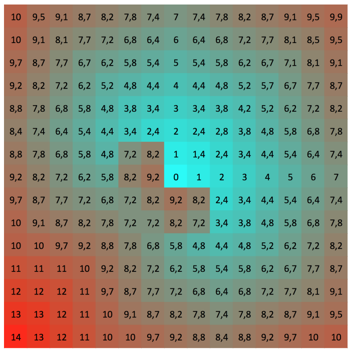
When combining the distance formulae and Equations 3.7 & 3.8, we created a heat map to visualise the costs of reaching a square in all eight directions a UAV can be in. These heat maps can be found in Figure 3.3. When observing the heat maps, we see that the formulae are a correct interpretation of the distance with restricted actions, we found by using A*. This can be seen if we compare Figure 3.2 and Figure 3.3. Furthermore, we see that the rotation function is also behaving as expected.
The restricted distance is calculated by Algorithm 2. Please note that this function is only valid for the assumption we made about the movement restriction of a UAV. If a UAV has no movement restrictions (e.g. helicopter-type UAVs), a simple euclidean distance would be equal to our restricted distance.
Apart from the complex restricted distance function, we also implemented a simple binary feature that indicates if an area is of a certain sort. We use both the restricted distance and the area indication to construct a vector of a total of 4 features. The features are calculated given a state, which contains all areas and the location of the UAV. These features being:

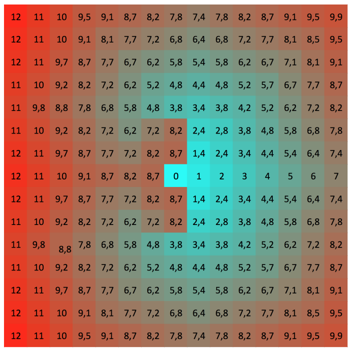
- Restricted distance from the UAV location to the nearest area of interest.
- Restricted distance from the UAV location to the nearest no-fly area.
- Is the UAV location an area of interest.
- Is the UAV location a no-fly zone.

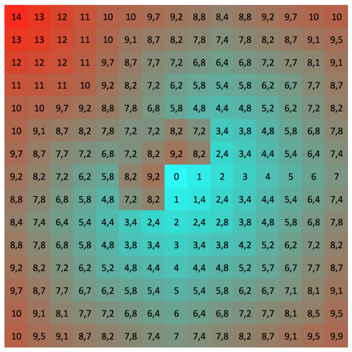
3.3.2 Feature-Q Algorithm

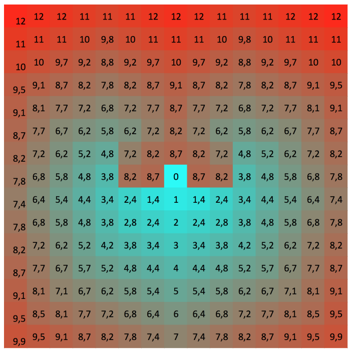
The features of the previous paragraph are used in combination with an adapted Q-learning algorithm, called Feature-Q. This can be seen in Algorithm 3. This algorithm can be seen as non-tabular Q-learning, where a set of weights are updated instead of a value. The features and weight vectors are multiplied and summed, resulting in a value for a state. Based on the reward received, weights are updated according to how active the features are.

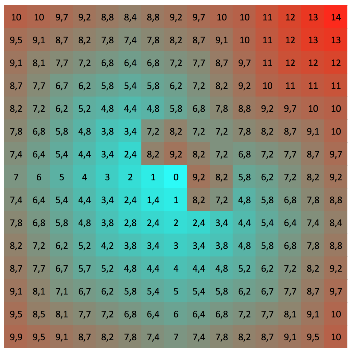

(a) Cost at 0°

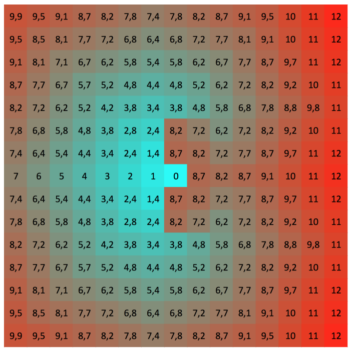

(b) Cost at 45°


(c) Cost at 90°


(d) Cost at 135°


(e) Cost at 180°


(f) Cost at 225°


(g) Cost at 270°

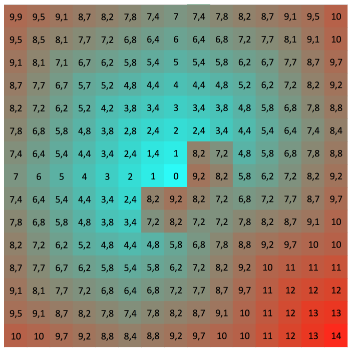

(h) Cost at 315°

Figure 3.3: Heat map of the restricted cost function for different angles. The position of the agent is in the middle of these heat maps. This is used as a feature in the Feature-Q learner.

Algorithm 2 Restricted distance

```
 1: $direction \leftarrow$ UAV direction
 2: $\Delta x \leftarrow$ by Equation 3.7 & 3.8
 3: $\Delta y \leftarrow$ by Equation 3.7 & 3.8
 4: if ($direction$ MOD 45 $= 0$) then
 5:     if ($\Delta y \geq 0$ AND $\Delta x \leq 0$) then
 6:         return $\leftarrow$ by Equation 3.9
 7:     else if ($\Delta y \leq -1$ AND $\Delta x \leq -2$) then
 8:         return $\leftarrow$ by Equation 3.10
 9:     else if ($\Delta y = -2$ AND $\Delta x = 2$) then
10:         return $\leftarrow$ by Equation 3.11
11:     else if $-2 \leq \Delta y \leq -1$ AND $-1 \leq \Delta x \leq 2$ then
12:         return $\leftarrow$ by Equation 3.12
13:     else if ($\Delta y + \Delta x \leq 0$) then
14:         return $\leftarrow$ by Equation 3.13
15:     end if
16: else
17:     if ($\Delta y > 0$) then
18:         return $\leftarrow$ by Equation 3.2
19:     else if ($\Delta y \leq 0$ AND $|\Delta x| > 2$) then
20:         return $\leftarrow$ by Equation 3.3
21:     else if ($\Delta y \leq 0$ AND $|\Delta x| \leq -2$ AND $(|\Delta y| + |\Delta x|) \geq 4$) then
22:         return $\leftarrow$ by Equation 3.4
23:     else if ($\Delta y \leq 0$ AND $(|\Delta y| + |\Delta x|) = 1$) OR ($\Delta y =$ -3 AND $\Delta x = 0$) then
24:         return $\leftarrow$ by Equation 3.5
25:     else if ($\Delta y < 0$ AND $2 \leq (|\Delta y| + |\Delta x|) \leq 3$) then
26:         return $\leftarrow$ by Equation 3.6
27:     end if
28: end if
```

Algorithm 3 Feature-Q algorithm

```
1: function MAIN
2:     $weights \leftarrow \{0, 0, 0, 0\}$
3:     while (not all areas of interest visited) do
4:         $bestAction \leftarrow null$
5:         $value \leftarrow 0$
6:         for $(action \in actions)$ do
7:             $s$ = current state
8:             $features \leftarrow$ GETFEATURES($s$, $action$)
9:             if $value < (weights * features)$ then
10:                $value = (weights * features)$
11:                $bestAction = action$
12:            end if
13:        end for
14:        if RANDOMDOUBLE() $< \alpha$ then
15:            Act randomAction
16:        else
17:            Act $bestAction$
18:        end if
19:        $s'$ = current state
20:        UPDATEQ($s$, $bestAction$, reward, $s'$)
21:    end while
22: end function
23:
24: function GETFEATURES(State s, Action $a$)
25:     $features \leftarrow \{0, 0, 0, 0\}$
26:     $features(1)$ = nearest unvisited area of interest after $a$ by Algorithm 2
27:     $features(2)$ = nearest no-fly area after $a$ by Algorithm 2
28:     $features(3)$ = UAV is on area of interest after $a$.
29:     $features(4)$ = UAV is on a no-fly area after $a$.
30:     return $features$
31: end function
32:
33: function UPDATEQ(State $s$, Action $a$, Reward $r$, State $s_{t+1}$)
34:     $delta \leftarrow \omega \cdot (r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$
35:     $weights \leftarrow weights + delta * features$
36: end function
```

3.4 Multi Ant Colony System

For the third approach we implemented a Multi Ant Colony System (MACS) [16, 28]. This method has been proven to be an effective search algorithm for multiple vehicle routing, as can be read in the Theoretical Framework of this thesis. When we implement a way to transform the grid to a graph representation, the UAV routing problem we are solving can be seen as a multi vehicle routing problem.
For the MACS algorithm, we assume full observability of the environment. Furthermore, we assume that there is full communication about the location of the agents.

3.4.1 Grid to Graph

We use the same conversion from a grid world to a graph representation as we use with the Greedy algorithm that is explained in Section 3.2. The conversion algorithm calculates all distances between all areas. The distance is calculated for every of the eight possible directions. This ensures that for every direction of the UAV, there is a correct distance measure between the area the UAV is on and the direction of the UAV, to all other areas. The algorithm to create the graph can be found in Algorithm 4. Every distance is calculated using A*, where the path formed during the search is limited by the possible actions at every time step. The UAVs are only allowed to move forward (current direction), left-forward (current direction − 45°), right-forward (current direction + 45°).

Algorithm 4 Grid to graph algorithm

```
1: function MAIN
2:     $distances[\,][\,][\,]$
3:     for $areaFrom \in areas$ do
4:         for $areaTo \in (areas \setminus areaFrom)$ do
5:             for $direction \in directions$ do
6:                 $distances[areaFrom][areaTo][direction]$
7:                         $= \textsc{AStar}(areaFrom, areaTo, direction)$
8:             end for
9:         end for
10:    end for return $distances[\,][\,][\,]$
11: end function
```

3.4.2 MACS Algorithm

The MACS algorithm is implemented as n ACSs, with m ants. The number of UAVs equals n. We also included a heuristic to determine the number of ants per colony, this is called the ant factor and it is implemented as a factor m multiplied by the number of areas of interest that need to be visited, as can be seen in Equation 3.14

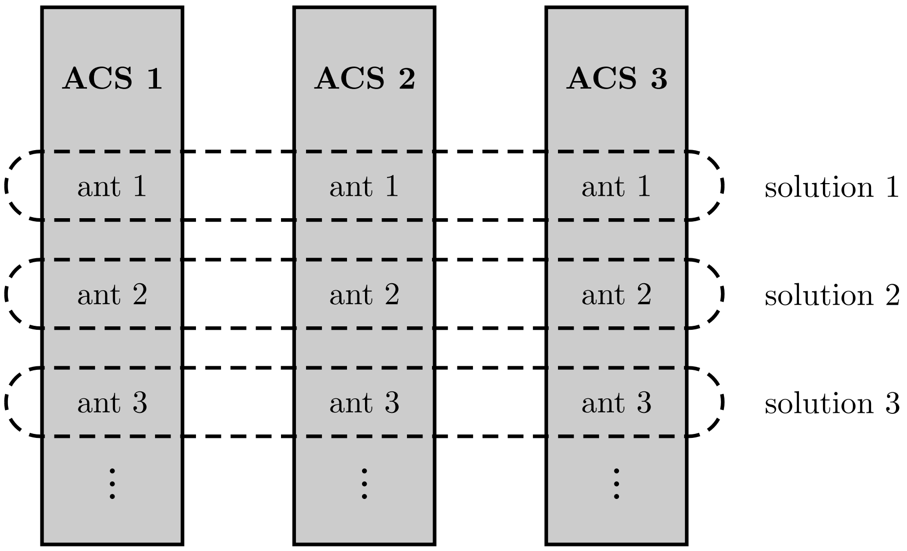


Figure 3.4: Ants at the same index of an ACS form a solution to the problem together.

$$m = antFactor * \textsc{length}(areasOfInterest) \tag{3.14}$$

The MACS algorithm is shown in Algorithm 5. Each ACS in the MACS is initialised with the same number of ants. Ants with the same index within an ACS, form groups of size equal to the number of UAVs. Ants with the same index form a solution for the problem, where each ant represents a UAV. A schematic overview of this can be seen in Figure 3.4.

3.5 Research

The goal of this thesis is to determine which of the proposed algorithms is the best for routing multiple UAVs. To determine this, we created a research method that is explained in this section.

3.5.1 Parameter Influence

Both the MACS and Feature-Q algorithms have a number of parameters that can be adjusted. For example, the learning rate, or the chance of choosing a random area of interest as a next destination. We will first run a series of experiments to determine the influence of the most important parameters on the performance of the algorithms. To measure this we will create a grid world of 30x30 and initialise a number of areas of interest and no-fly areas at random positions. We will run the same algorithm on the same grid world, while adjusting a parameter to see its influence on the performance.

Algorithm 5 MACS algorithm

```
1: function MAIN
2:     $n \leftarrow$ LENGTH(UAVs)
3:     $m \leftarrow$ by Equation 3.14
4:     $bestTour[\,][\,] \leftarrow null$
5:     $bestTourLength \leftarrow \infty$
6:     for ($iterations$) do
7:         initialise all ants at the UAV location area
8:         for ($antIdx = 0 : m$) do
9:             while (!$\forall$ areas visited) do
10:                for ($acsIdx : n$) do
11:                    $nextArea \leftarrow$ SELECTNEXTAREA($acsIdx$,$antIdx$)
12:                    VISITTOWN($nextArea$)
13:                    ADDTOTOUR($acsIdx$, $antIdx$, $nextArea$)
14:                    LOCALUPDATE(currentArea, $nextArea$) by Equation 2.12
15:                end for
16:            end while
17:            $tourLength \leftarrow$ TOURLENGTH($antIdx$)
18:            $currentTour \leftarrow$ GETTOUR($antIdx$)
19:            if ($tourLength < bestTourLength$) then
20:                $bestTour \leftarrow currentTour$
21:                $bestTourLength \leftarrow tourLength$
22:            end if
23:        end for
24:        GLOBALUPDATE($bestTour$) by Equation 2.10
25:    end for
26: end function
```

3.5.2 Baseline Performance

As a baseline, we create a grid world where we initialise areas of interest and no-fly areas at random squares in the grid world. After the random initialisation we run all three our proposed methods on that same grid world. The total cost of the solutions is the sum of costs of all individual UAVs. The cost of a single UAV is calculated by Equation 3.1.

3.5.3 Grid World Influence

To determine the influence of scaling up the problem, we will use the same method as described in the baseline section and run that for grid worlds of different sizes.

3.5.4 UAV Influence

We will also investigate the influence of scaling the number of UAVs on the performance by again using the same method as described in the baseline section and running that for different numbers of UAVs.

Chapter 4

Experiments and Results

For all research, we used a 30 by 30 grid world, except when stated otherwise. The grid world contains 5 UAVs, 50 no fly areas, 26 areas of interest. All of these UAVs, no fly areas and areas of interest are randomised at the end of a run. The randomisation process makes sure our results are valid for the problem as a whole and not only for a specific world. We first did a parameter sweep for both the Feature-Q algorithm and the MACS algorithm. Using the results of the parameter sweep, we will compare all three algorithms.

4.1 Feature-Q Parameters

Our Feature-Q algorithm has three main parameters that can be changed and have a possible influence on the quality of the solution. All parameters are tested using a fixed number of 50.000 learning iterations. Each UAV has its own learner, so with 5 UAVs there are 5 Feature-Q algorithms active in parallel. After the learning phase, a testing phase of 50.000 iterations takes place. During this testing phase, both the learning rate and exploration are set to zero. This is done to be able to see the quality of the solution after 50.000 learning iterations, without changing the solution while testing it.
The parameters we tested are:

- Learning rate (ω): the influence of a single input on the adjustment of the weights, default value: 0.05
  The measurements can be seen in Figure 4.1a.
- Greedy preference (α): the chance of choosing a totally random action as opposed to the best action, default value: 0.05
  The measurements can be seen in Figure 4.1b.
- Discount factor (γ): the preference for a short term reward as opposed to possible future rewards, default value: 0.01
  The measurements can be seen in Figure 4.1c.

In all figures we use the cost of the solution as measure of quality. The cost is defined in Equation 3.1 as the sum of Euclidean distances of the path of the UAV deducted by the sum of all collected rewards, the possible rewards being: 1 for an area of interest and -0.1 for a no-fly area.
The plots contain box plots, where the bold line in the middle is the median, the lower line of the box is the lower quartile (i.e. 1st quartile or 25th percentile) and the upper line is the upper quartile (i.e. 3rd quartile or 75th percentile). From the plots we see that the algorithm performs best at a learning rate of $\omega = 0.1$ and a greedy preference of $\alpha = 0.01$. The discount factor seems to be of little influence, but best performance is seen at $\gamma = 0.1$.

4.2 MACS Parameters

Our MACS algorithm has six main parameters that can be adjusted and have a possible influence on the quality of the solution. The results are gathered by running the MACS algorithm at a randomised grid world. For every random grid world, the MACS algorithm is run with the different values of the parameter we test. This is done to minimise the influence of the randomisation on the parameter we want to investigate.
The parameters we tested are:

- Number of iterations: The number of iterations of the MACS algorithm before giving a solution, default value: 500
  The measurements can be seen in Figure 4.2a.

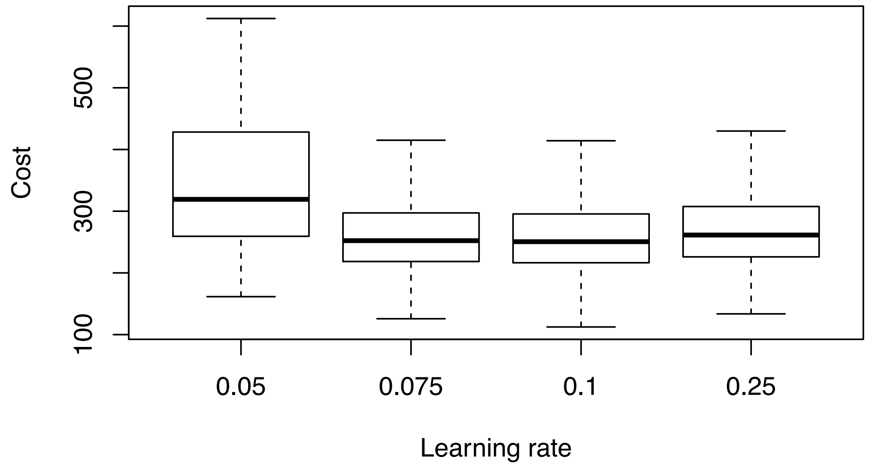
- Local update strength (ρ): The strength of the local update of an ant, default value: 0.05
  The measurements can be seen in Figure 4.2b

- Greedy preference (β): the preference of ants for a low cost edge as opposed to a high pheromone edge, default value: 5
  The measurements can be seen in Figure 4.2c.

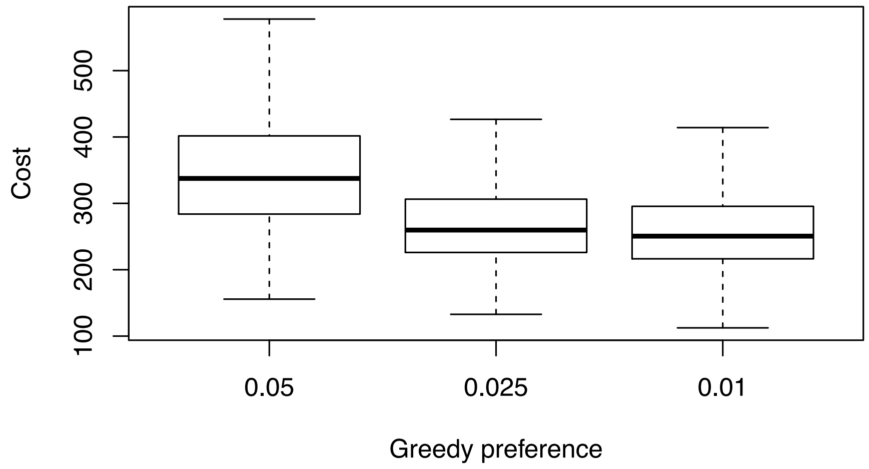
- Ant factor: The number of ants as opposed to the number of areas of interest, default value: 5
  The measurements can be seen in Figure 4.2d.

- Evaporation rate (α): The rate at which the pheromones decay, default value: 0.1
  The measurements can be seen in Figure 4.2e.

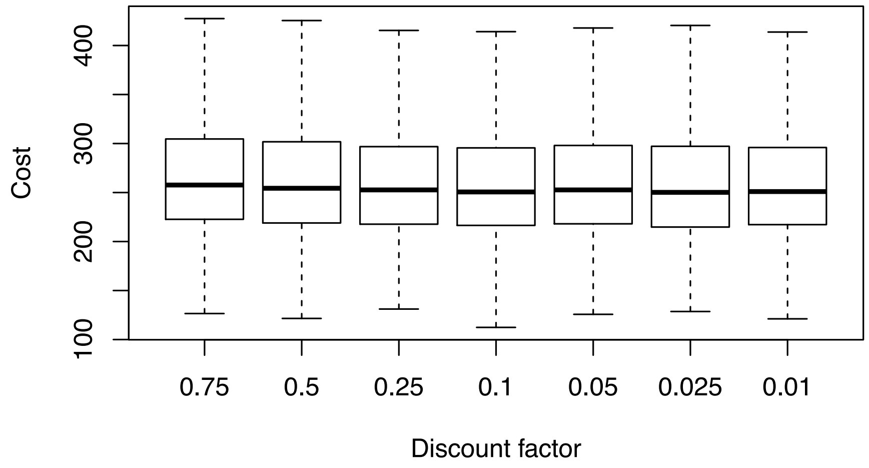
- Random probability (q_0): the chance of choosing a totally random action as opposed to the best action, default value: 0.01
  The measurements can be seen in Figure 4.2f.



(a) Learning rate influence on cost

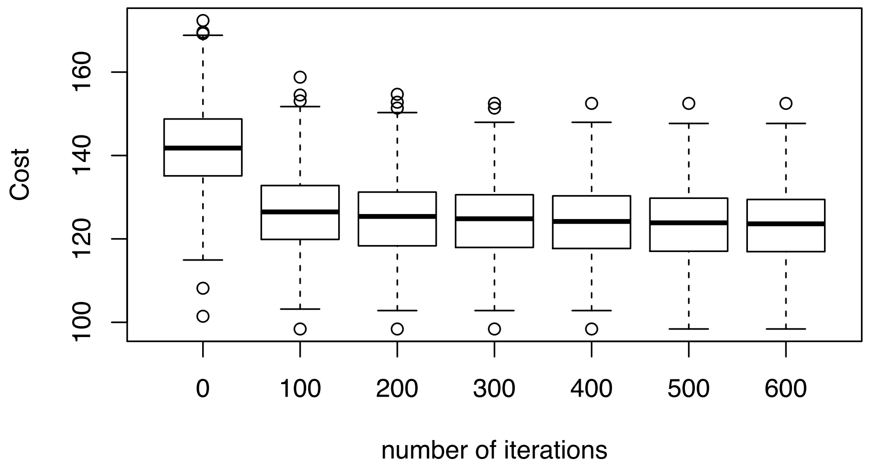


(b) Greedy preference influence on cost

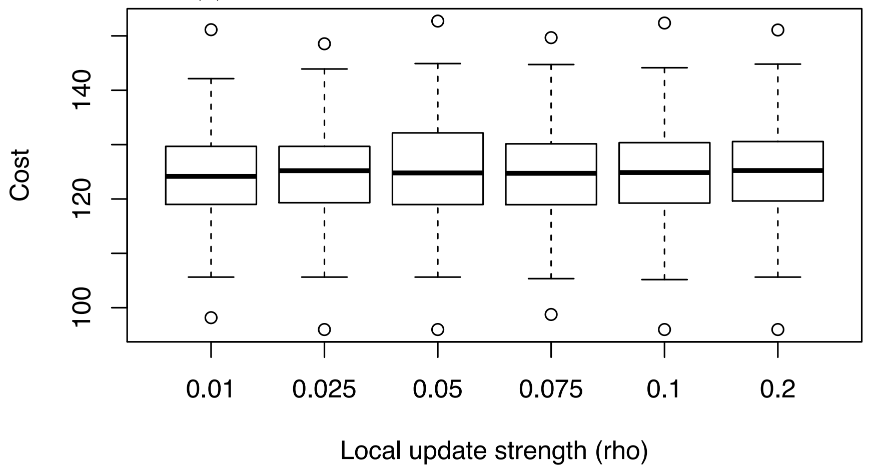


(c) Discount factor influence on cost

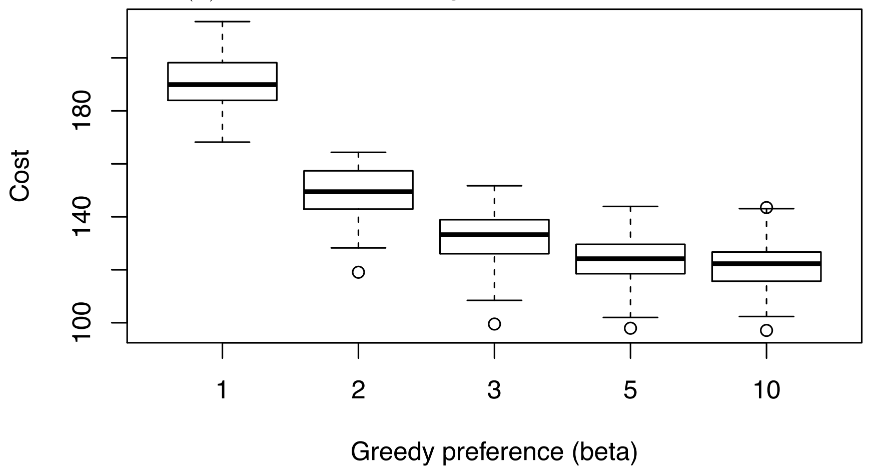
Figure 4.1: Parameter sweep of the Feature-Q algorithm



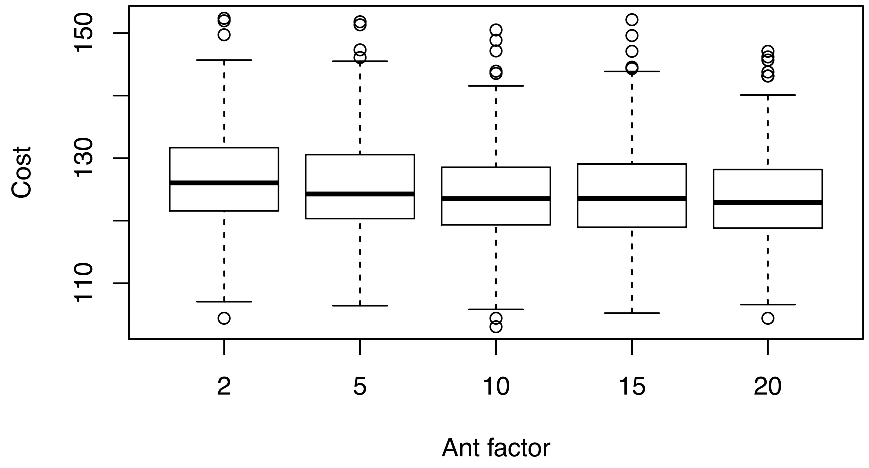
(a) Number of iterations influence on cost



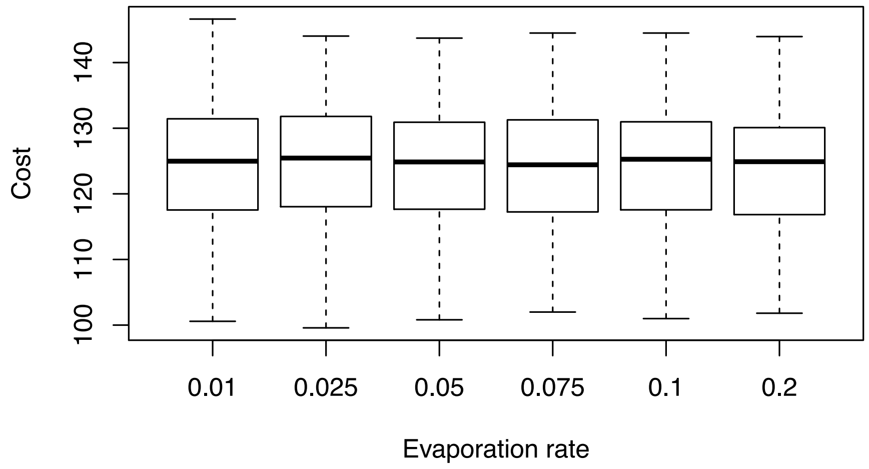
(b) Local update strength influence on cost



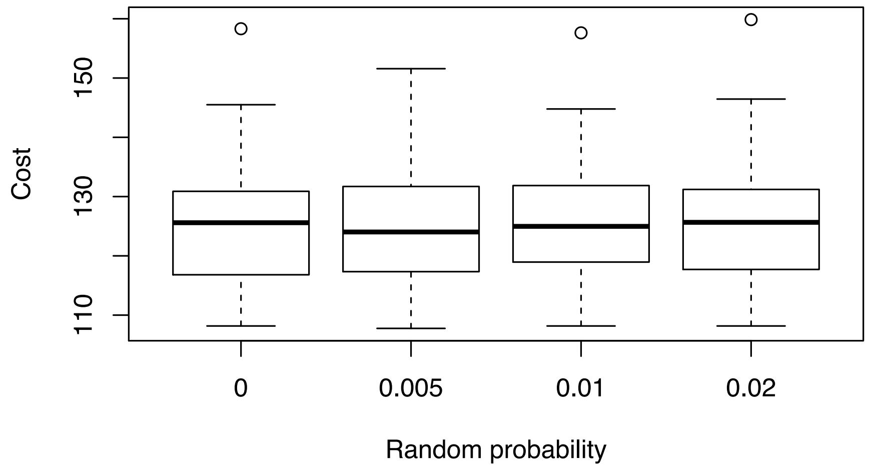
(c) Greedy preference influence on cost



(d) Ant factor influence on cost



(e) Evaporation rate influence on cost



(f) Random probability influence on cost

Figure 4.2: Parameter sweep of the MACS algorithm

	Greedy	Feature-Q	MACS
Greedy	n/a	0.157	**0.000**
Feature-Q	0.157	n/a	**0.000**
MACS	**0.000**	**0.000**	n/a

Table 4.1: P-values of the algorithmic performances being different from each other. Solutions were created for a randomised 30x30 grid world containing 5 UAVs, 26 areas of interest, 50 no-fly areas.

Algorithm	Cost
Greedy	225.3
Feature-Q	247.9
MACS	124.1

Table 4.2: Mean cost of 100 solutions of the algorithms. Solutions were created for a randomised 30x30 grid world containing 5 UAVs, 26 areas of interest, 50 no-fly areas.

From the plots we see that the number of iterations settles down at 500 iterations and the ant factor seems of little influence at values higher than 10. The algorithm performs best at a local update strength of $\rho = 0.01$, a greedy preference of $\beta = 10$ and the evaporation rate and random probability seem to be of little influence, but best performance is seen at an evaporation rate of 0.05 and random probability of 0.005.

4.3 Baseline Performance

Using the values for the parameters found by the parameter sweep, we compared the performance of the different algorithms on the same randomised grid world. The Feature-Q algorithm is trained for 50.000 iterations on multiple randomised grid worlds. The MACS algorithm is trained and then tested on a single randomised grid world. The trained Feature-Q algorithm is used to solve the same grid world as the MACS algorithm. As a baseline, we also solve that same grid world with the Greedy algorithm. This procedure is repeated 100 times, to compare the performance of the algorithms on the UAV planning problem.

The cost of all solutions are compared using Friedman's test [33, 34] and p-values are corrected using Finner's procedure [35, 34]. As we see in Table 4.1 there is no significant difference between the performance of the Greedy algorithm and the Feature-Q algorithm. There is however a significant difference between the MACS algorithm and the other algorithms, thus showing the MACS algorithm performs best. In Table 4.2 we see the mean cost of 100 solutions, showing us that the MACS algorithm has the lowest mean cost for a solution, followed by the Greedy algorithm.

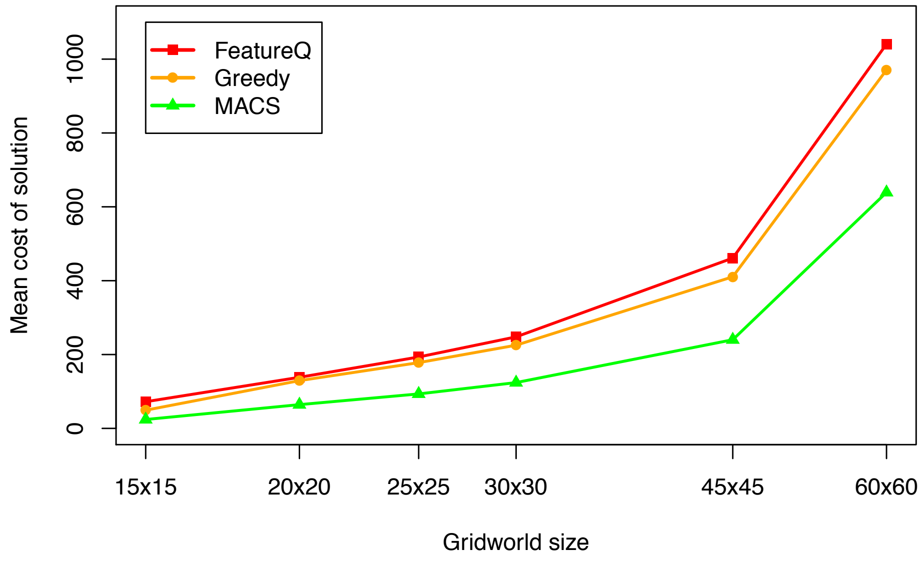


Figure 4.3: Influence of the size of the grid world on the total cost of the solution for the Greedy, Feature-Q and MACS algorithms.

4.4 Grid World Influence

We repeated the same procedure of the baseline performance for six different sizes of the grid world ∈ {15x15, 20x20, 25x25, 30x30, 45x45, 60x60}. The density of the no-fly zones and the areas of interest is kept the same throughout the different grid world sizes (i.e. the ratio between the number of no-fly zones and number of areas of interest is not changed, but their total number increases with a larger grid to keep their density the same throughout the experiment).

We used Friedman's test and corrected the p-values using Finner's procedure. The results per grid world size can be seen in Table 4.3.

The mean cost of 100 runs of all algorithms for different sizes of the grid world can be seen in Figure 4.3. From Table 4.3 we see that the Feature-Q and Greedy algorithms are significantly different at grid worlds of 15x15. For the other sizes there is no significant difference. We can also see that for all sizes of the grid world the difference in performance between the MACS algorithm and the other two algorithms is significant. This means the MACS algorithm performs significantly better than the other two algorithms, regardless of the grid world size.

	Greedy	Feature-Q	MACS
Greedy	n/a	**3.954e-09**	**3.331e-14**
Feature-Q	**3.954e-09**	n/a	**0.000e+00**
MACS	**3.331e-14**	**0.000e+00**	n/a

(a) 15x15

	Greedy	Feature-Q	MACS
Greedy	n/a	5.716e-01	**0.000e+00**
Feature-Q	5.716e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(b) 20x20

	Greedy	Feature-Q	MACS
Greedy	n/a	1.513e-01	**0.000e+00**
Feature-Q	1.513e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(c) 25x25

	Greedy	Feature-Q	MACS
Greedy	n/a	2.254e-01	**0.000e+00**
Feature-Q	2.254e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(d) 45x45

	Greedy	Feature-Q	MACS
Greedy	n/a	2.757e-01	**0.000e+00**
Feature-Q	2.757e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(e) 60x60

Table 4.3: P-values of the algorithmic performances being different from each other for different sizes of the grid world. Solutions were created for a randomised grid world.

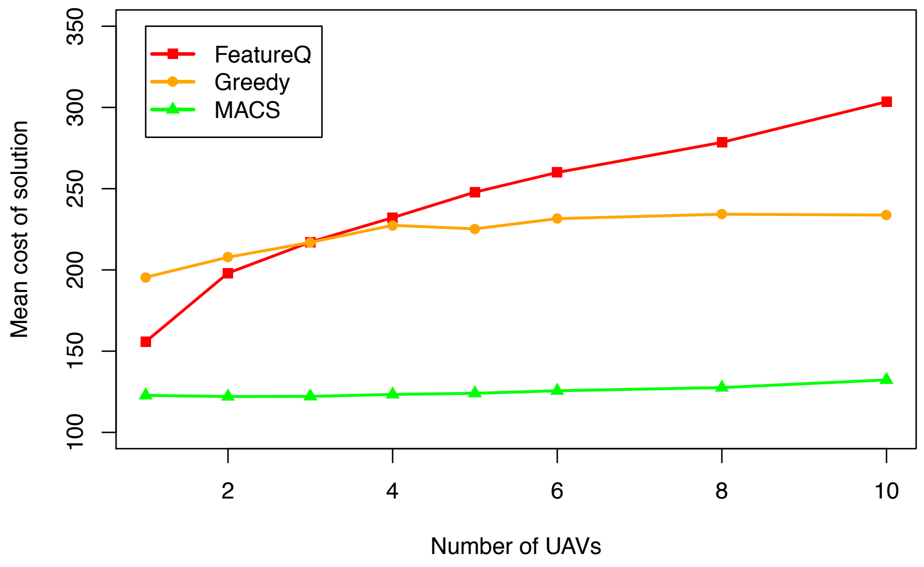


Figure 4.4: Influence of the number of UAVs on the total cost of the solution for the Greedy, Feature-Q and MACS algorithms.

4.5 UAV Influence

As a last factor we tested whether the number of UAVs was of influence on the quality of the solution. We used our standard 30x30 randomised grid world, containing 26 areas of interest and 50 no-fly areas. For this research we investigated the solutions using all three algorithms on this grid world containing a different number of UAVs $\in \{1, 2, 3, 4, 5, 6, 8, 10\}$.

We again used Friedman's test and corrected the p-values using Finner's procedure. The results per number of UAVs can be seen in Table 4.4.

The mean cost of 100 runs of all algorithms for a different number of UAVs can be seen in Figure 4.4. From Table 4.4 we can see that the difference between the Feature-Q and Greedy algorithms is not significant in the case of 3 and 4 UAVs. With all other number of UAVs there is a significant difference between the algorithms. The MACS algorithm has a significant difference with the other two algorithms at any number of UAVs. This means that again the MACS algorithm performs better than the other two algorithms at any number of UAVs. The second best algorithm depends on the number of UAVs used, if there is only 1 or 2 UAVs, the second best algorithm would be the Feature-Q algorithm, and if there are 5 or more UAVs the Greedy algorithm is second best. In the case of 3 or 4 UAVs there is no significant difference between the Feature-Q and Greedy algorithms.

	Greedy	Feature-Q	MACS
Greedy	n/a	**1.603e-11**	**0.000e+00**
Feature-Q	**1.603e-11**	n/a	**8.240e-13**
MACS	**0.000e+00**	**8.240e-13**	n/a

(a) 1 UAV

	Greedy	Feature-Q	MACS
Greedy	n/a	**2.831e-02**	**0.000e+00**
Feature-Q	**2.831e-02**	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(b) 2 UAVs

	Greedy	Feature-Q	MACS
Greedy	n/a	1.299e-01	**0.000e+00**
Feature-Q	1.299e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(c) 3 UAVs

	Greedy	Feature-Q	MACS
Greedy	n/a	4.795e-01	**0.000e+00**
Feature-Q	4.795e-01	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(d) 4 UAVs

	Greedy	Feature-Q	MACS
Greedy	n/a	**3.864e-02**	**0.000e+00**
Feature-Q	**3.864e-02**	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(e) 6 UAVs

	Greedy	Feature-Q	MACS
Greedy	n/a	**1.697e-04**	**0.000e+00**
Feature-Q	**1.697e-04**	n/a	**0.000e+00**
MACS	**0.000e+00**	**0.000e+00**	n/a

(f) 8 UAVs

	Greedy	Feature-Q	MACS
Greedy	n/a	**9.908e-07**	**6.661e-16**
Feature-Q	**9.908e-07**	n/a	**0.000e+00**
MACS	**6.661e-16**	**0.000e+00**	n/a

(g) 10 UAVs

Table 4.4: P-values of the algorithmic performances being different from each other for different numbers of UAVs. Solutions were created for a randomised 30x30 grid world containing 26 areas of interest and 50 no-fly areas.

4.6 Case Studies

To give the reader an idea of the differences in solutions between the three algorithms, we made two case studies that emphasise these differences. The results are not to be confused with the empirical research done on the performance of the algorithms, as conducted in the previous part of this section, but merely as an addition to this research.
The first case study includes three isles of areas of interest, which need to be visited. We also included a variant of the isles with a wall of no-fly zones.
The second case study includes three areas of interest, separated by two large no-fly zones.

4.6.1 Isles Case Study

For the isles case study, we created two variants. A normal variant in which there are three isles of areas of interest, this can be seen in Figure 4.5. We also created a variant with a wall in between the isles, this van be seen in Figure 4.6.
There are some notable outcomes of this particular case:

- Both the Greedy and Feature-Q algorithm do not divide over the isles. In both algorithms, all the UAVs head to the nearest isle and visit all areas. If the visit isle is fully visited, they head to the next nearest. The MACS algorithm divides the UAVs between the isles, this makes for a faster solution.
- When we add a wall in between the isles (Figure 4.6), we find that when using the Greedy or Feature-Q algorithm the UAVs have a tendency to visit the right-most isle first. The MACS algorithm still divides between the isles.
- Another interesting finding is that the Feature-Q algorithm sometimes decides to cross the no-fly wall, to reach an area of interest.

4.6.2 No-fly Zone Case Study

To further illustrate the differences, we set up a second case where there are three areas of interest that are divided by two large no-fly zones, see Figure 4.7. We will again give some notable outcomes of this case:

- When using the Greedy an Feature-Q algorithm, all three UAV from a train to head to the middle area of interest. After this area is visited, they head to the right-most area of interest to end at the left-most area of interest.
- The MACS algorithm divided the UAVs between all three areas of interest, significantly lowering the time needed to visit all three areas of interest.
- In a few observations we saw the Feature-Q algorithm chose to cross a no-fly zone to reach an area of interest.

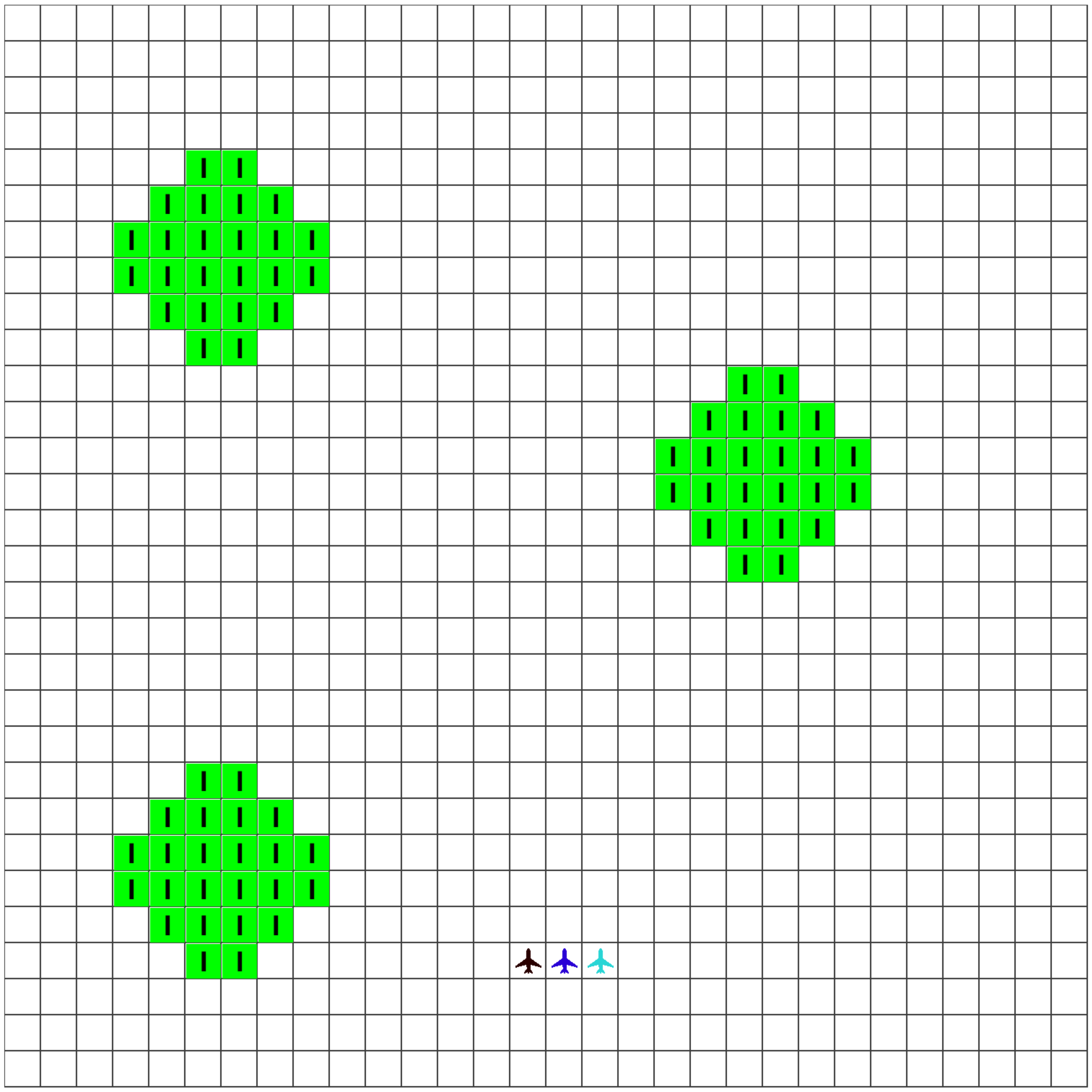


Figure 4.5: Isles grid world

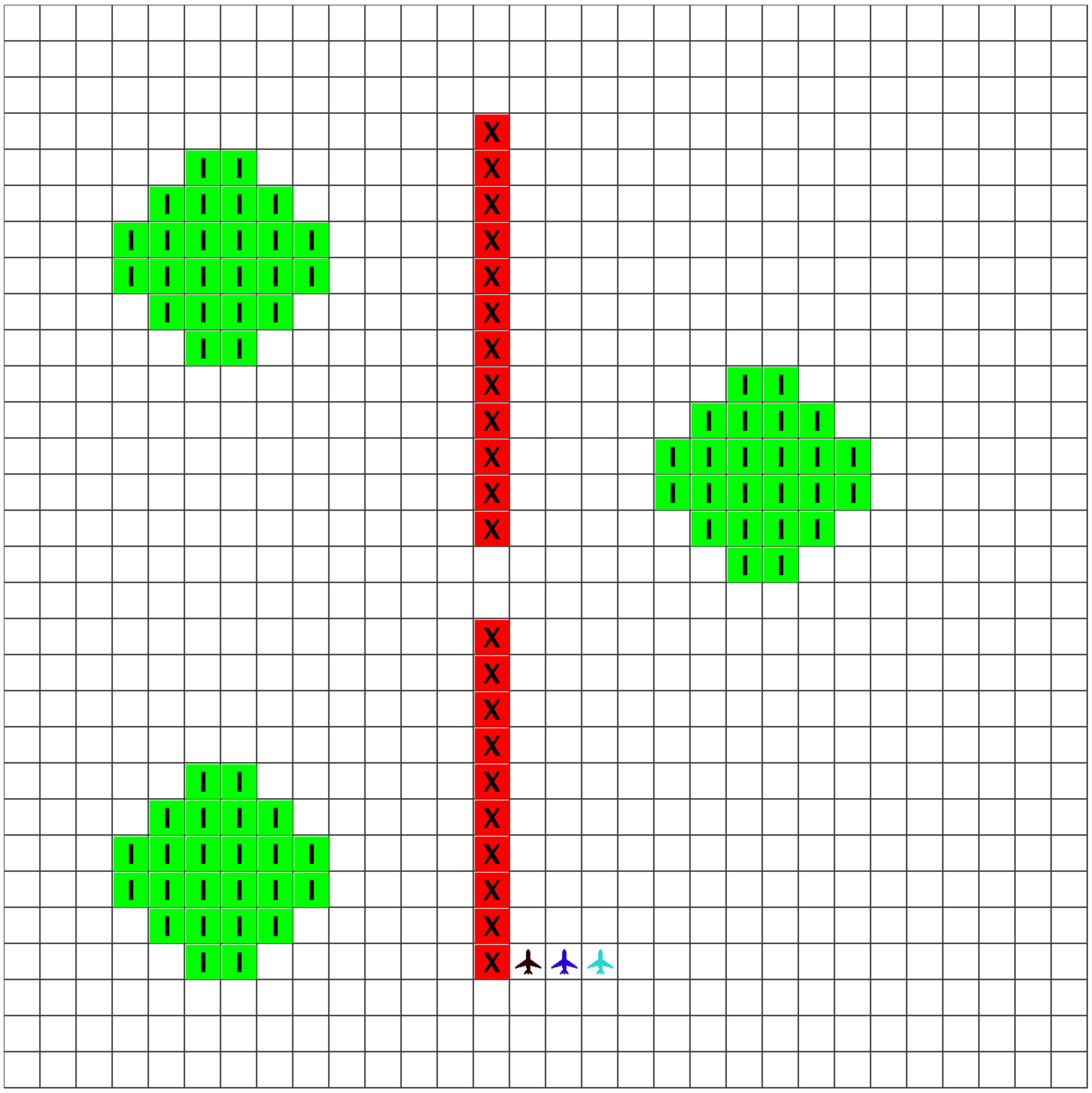


Figure 4.6: Isles with wall grid world

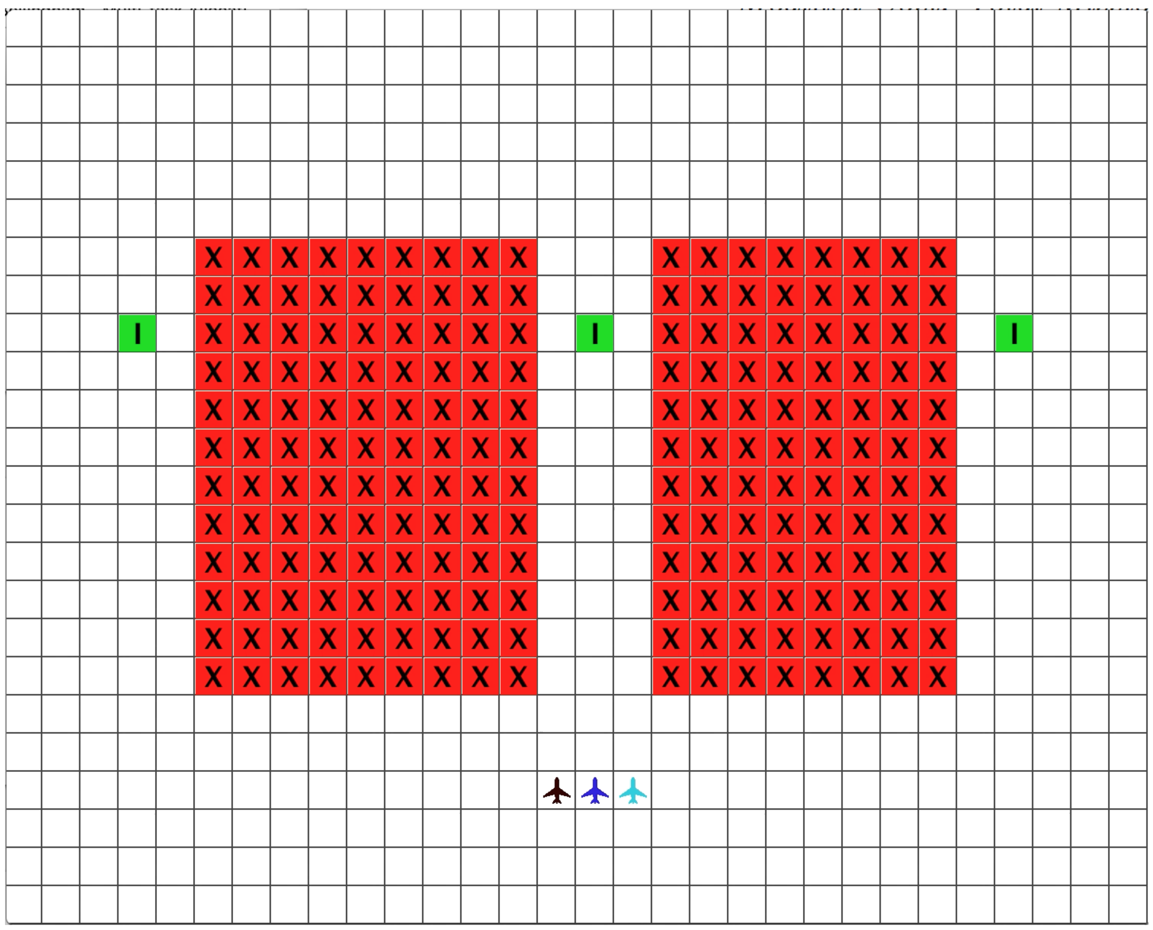


Figure 4.7: No-fly zone grid world

Chapter 5

Discussion and Conclusion

We compared the Greedy, MACS and Feature-Q algorithms on performance and scalability. The results from this research gave us insight in the usability of all algorithms for the UAV routing task. We will answer our main research question and sub questions for all three algorithms in the three next paragraphs.

5.1 Greedy

The Greedy algorithm was very easy to implement and gave good results when a small number of UAVs was used. However, the algorithm has no advantages over the MACS algorithm except compute time.
One of the reasons the performance of the Greedy algorithm is disappointing at a higher number of UAVs, is because of the fact that there is no communication between the UAVs except for a list of visited areas. Therefore we see that UAVs have a tendency to follow each other, resulting in a train of UAVs all heading to the same best destination given the Greedy algorithm.
Despite the fact that the Greedy algorithm is easy to understand and is not computational heavy, the process of creating the graph representation from the grid world is. Creating the graph initiates approximately $N_{interest}^2$ A-star searches to determine all edges on the graph, making it an algorithm that does not scale very well.

5.2 Feature-Q

The parameter sweep showed us that the discount factor was of little influence on the performance. If we look at the problem we are trying to solve, this makes sense. When a UAV reaches a high value state, there is no guarantee that the next state will also be of high value.

The performance of the Feature-Q algorithm is somewhat disappointing, but given the fact that this level of performance is reached with only four features leaves room for improvement. The Feature-Q algorithm suffers from the same lack of communication as the Greedy algorithm. The agents only get a list of areas that still need to be visited. Current destinations are not considered. This makes for the same potential behaviour of multiple UAVs following each other as is the case with the Greedy algorithm.

The learning of the value function takes a fair amount of CPU-time. However, when the algorithm has finished learning it has learned to solve the problem as a whole and the solution it has found is independent of the grid world. This is a big advantage the Feature-Q algorithm has above the other two algorithms, who both need to recalculate the optimal solution if something in the world changes.

An argument against the Feature-Q algorithm would be that the knowledge of this algorithm is in the features, which is partially true. While the creation of informative and high quality features is done by using human knowledge, reinforcement learning is used to find the weights of those features that give the highest reward. A human with extensive knowledge of the problem would probably be able to guess these weights by trial and error, but this would consume a reasonable amount of time and would not necessarily render a better solution.

The features are both the greatest part of this algorithm as they are its Achilles heel. They make that this algorithm can be used in a dynamic on-line environment, without the need to relearn for every new world or change of the environment. But since the performance of the algorithm is as good as its features, more features should be added and tested to use the full potential of feature-based RL on UxV routing.

5.3 MACS

The parameter sweep of the MACS algorithm showed us that three parameters were of little influence: the local update strength (ρ), the random probability (q_0) and the evaporation rate (α). A possible explanation for this can be found in the number of iterations of the MACS algorithm. All these parameters promote exploration to find a better solution, but due to the high number of iterations we chose it is very possible the algorithm converges to a low cost solution with minimal exploration. Therefore more exploration would not result in a higher quality solution.

All the results point to the MACS algorithm being the best performing amongst all algorithms used in this research. So we cannot conclude otherwise than that the MACS approach is the best learning technique in the sense of quality of the solution.

But as we already pointed out in the previous paragraph, a large disadvantage of the MACS algorithm is the increase in computational complexity with the size of the problem. This makes the MACS algorithm a very good choice for small and static planning problems, but a less ideal choice for larger or more dynamic problems. In our research we noticed that a

grid of 100x100 is about the maximum that can be calculated with the MACS algorithm in a reasonable amount of time (i.e. < 5 minutes on standard hardware). But a large part of this is due to creating the graph from the grid world. Actually running the MACS algorithm on the graph takes about the same time as creating the graph.
Furthermore, the MACS algorithm is a centralised form of planning, where all planning is done on a single server and a route is given to the UAVs. As opposed to the MACS algorithm, the Feature-Q algorithm can be run locally on every UAV.

5.4 Research Question

Our main research question was: "What is the best machine learning technique for autonomous planning in a group of UxVs?".
We can answer this question in two ways. The best algorithm in terms of quality of the solution is the MACS algorithm. Its solutions have the lowest cost and they also have the lowest increase in cost if we scale the problem.
If we take in account the computational complexity, the Feature-Q algorithm is not as computational intensive when the learning is finished and the knowledge of the algorithm is exploited. Furthermore the Feature-Q algorithm scales better in terms of computational complexity than the MACS algorithm. When we look at the computational power of a standard UAV, we find that this is very limited. Therefore running any learning algorithm on-board of a UAV makes little sense. In case of the MACS algorithm, this is not a problem, since all planning is done at a central server. In case of the Feature-Q algorithm, the limited computational power might be a problem, but only when the UAV is learning to solve the problem. The UAV could also learn the task by simulation, thus removing the need to learn the task on the on-board hardware of the UAV. If the Feature-Q algorithm is trained, execution is as simple as multiplying detected features with learned weights, resulting in an action the UAV should take.
All our algorithms depend on a certain amount of communication. Communication on-board standard UAVs however, has limited range and reliability. This imposes an issue for all our algorithms, but in particular for the MACS algorithm as it assumes full communication.
The Feature-Q algorithm is the only algorithm of the three that generates a universal solution to the problem, as opposed to the problem specific solutions of the MACS and Greedy algorithm. While a universal solution is more useful than a problem specific solution that needs to be recalculated at every change, the quality of the solutions of the Feature-Q algorithm are significantly worse than those of the other two algorithms.
To answer the research question, we make the choice that the quality of the solution weights heavier for a planning problem than scalability or a universal solution. Therefore we will conclude that the MACS algorithm is among the tested techniques the best machine learning technique for autonomous planning in a group of UxVs. The Feature-Q algorithm

has a great potential given its scalability and universal solution, but needs more research and better features to be of practical use for solving planning problems.

5.5 Future Work

As a last part of this thesis, we would like to mention possible future work to research planning algorithms for multiple UxVs.
We mentioned before that the performance of the Feature-Q algorithm is depending on the quality of the features. This algorithm is promising in the sense it is decentralised and robust in a dynamic environment. Research on new features could improve the performance of the algorithm and therefore make the algorithm competitive for centralised algorithms as MACS. Possible features are features that tell the agent about goals or locations of other agents, features about the spread of agents across the grid or features that determine the distance to the next nearest area of interest or no-fly zone.
Another interesting idea for further research on RL in multi UxV planning is the use of a cascade of RL agents, where one agent could for example determine the next destination and another agent could be used to navigate to this destination.
As for the MACS algorithm, the creation of a graph containing the distances is a time consuming process that leaves a lot of room for improvement. The use of multiple A-star algorithms to determine the distances is not of any practical use at larger grids. A solution to this lies in finding other ways to construct a graph. One approach that might be interesting to investigate is if the features used for the Feature-Q algorithm could be used for a biased search when creating the graph, creating a hybrid solution.

Bibliography

[1] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement learning and dynamic programming using function approximators*, vol. 39. CRC press, 2010.

[2] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels, "Self-organized shortcuts in the argentine ant," *Naturwissenschaften*, vol. 76, no. 12, pp. 579–581, 1989.

[3] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, 2011.

[4] P. J. Zarco-Tejada, J. A. Berni, L. Suárez, and E. Fereres, "A new era in remote sensing of crops with unmanned robots," *SPIE Newsroom*, pp. 2–4, 2008.

[5] L. Merino, F. Caballero, J. R. M. Dios, J. Ferruz, and A. Ollero, "A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires," *Journal of Field Robotics*, vol. 23, no. 3-4, pp. 165–184, 2006.

[6] P. H. Kourtz, "Advanced information systems in Canadian forest fire control.," 1994.

[7] I. Shames, B. Fidan, and B. Anderson, "Close target reconnaissance using autonomous UAV formations," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pp. 1729–1734, IEEE, 2008.

[8] H. bin Duan, X. yin Zhang, J. Wu, and G. jun Ma, "Max-min adaptive ant colony optimization approach to multi-UAVs coordinated trajectory replanning in dynamic and uncertain environments," *Journal of Bionic Engineering*, vol. 6, no. 2, pp. 161–173, 2009.

[9] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, "A ground control station for a multi-UAV surveillance system," *Journal of Intelligent and Robotic Systems*, vol. 69, no. 1-4, pp. 119–130, 2013.

[10] D. J. Bennet, C. MacInnes, M. Suzuki, and K. Uchiyama, "Autonomous three-dimensional formation flight for a swarm of unmanned aerial vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 6, pp. 1899–1908, 2011.

[11] Y. B. Sebbane, *Planning and decision making for aerial robots.* Springer, 2014.

[12] R. Aragues, E. Montijano, and C. Sagues, "Consistent data association in multi-robot systems with limited communications," in *Robotics: Science and Systems*, pp. 97–104, 2011.

[13] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, *Multi-task allocation and path planning for cooperating UAVs*, pp. 23–41. Cooperative Control: Models, Applications and Algorithms, Springer, 2003.

[14] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-art*, vol. 12. Springer Science and Business Media, 2012.

[15] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: An autocatalytic optimizing process," *Technical report*, vol. 16, no. 91, pp. 163–183, 1991.

[16] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[18] E. L. Thorndike, "The law of effect," *The American Journal of Psychology*, vol. 39, no. 1/4, pp. 212–222, 1927.

[19] A. M. Turing, "Intelligent machinery, a heretical theory," *The Turing Test: Verbal Behavior as the Hallmark of Intelligence*, vol. 105, 1948.

[20] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[21] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Temporal difference learning of position evaluation in the game of go," *Advances in Neural Information Processing Systems*, pp. 817–817, 1994.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," tech. rep., 1994.

[24] J. N. Tsitsiklis and B. V. Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, no. 1-3, pp. 59–94, 1996.

[25] C. Thiery and B. Scherrer, “Building controllers for Tetris,” *International Computer Games Association Journal*, vol. 32, pp. 3–11, 2009.

[26] P. Grasse, “La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La theorie de la Stigmergie: Essai d’interpretation du comportement des Termites Constructeurs.,” *Insectes sociaux*, vol. 6, pp. 41–80, 1959.

[27] M. Dorigo, E. Bonabeau, and G. Theraulaz, “Ant algorithms and stigmergy,” *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851–871, 2000.

[28] J. de Jong and M. Wiering, “Multiple ant colony systems for the busstop allocation problem,” in *Proceedings of the Thirteenth Belgium-Netherlands Conference on Artificial Intelligence*, pp. 141–148, 2001.

[29] L. M. Gambardella, É. Taillard, and G. Agazzi, *MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows*, pp. 63–76. McGraw-Hill Ltd., UK, 1999.

[30] Y. Gajpal and P. L. Abad, “Multi-ant colony system (MACS) for a vehicle routing problem with backhauls,” *European Journal of Operational Research*, vol. 196, no. 1, pp. 102–117, 2009.

[31] F. Vermeulen, “Disruption management at NS,” Master’s thesis, University of Groningen, 2011.

[32] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[33] M. Friedman, “The use of ranks to avoid the assumption of normality implicit in the analysis of variance,” *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[34] S. García, A. Fernández, J. Luengo, and F. Herrera, “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power,” *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[35] H. Finner, “On a monotonicity problem in step-down multiple test procedures,” *Journal of the American Statistical Association*, vol. 88, no. 423, pp. 920–923, 1993.