AN ABSTRACT OF THE DISSERTATION OF

Carrie Melinda Rebhuhn for the degree of Doctor of Philosophy in Robotics and Mechanical Engineering presented on June 22, 2017.

Title: Adaptive Multiagent Traffic Management for Autonomous Robotic Systems

Abstract approved: _____

Kagan Tumer Geoff Hollinger

There is growing commercial interest in the use of unmanned aerial vehicles (UAVs) in urban environments, specifically for package delivery applications. However, the size, complexity and sheer numbers of expected UAVs makes conventional air traffic management that relies on human air traffic controllers infeasible. To enable UAVs to safely and efficiently operate in congested environments, it is essential to develop autonomous UAV management strategies.

We introduce a dynamic hierarchical traffic control model that reacts to traffic conditions instantaneously to reduce congestion in the airspace. An obstacle-filled airspace lends itself to a modelling as a graph structure similar to a road network. We introduce "controller agents", which set costs across the airspace. These agents control traffic similarly to adaptive metering lights in highway traffic. UAVs then plan their paths based on the costs (e.g. conflicts, or delays) they see for traversing particular parts of the airspace. This provides us a decentralized method for reducing traffic in an airspace.

Our hierarchical structure allows us to separate the traffic reduction problem from the individual robot navigation problem. Each robot does not explicitly coordinate with others in the airspace. Instead, robots execute their own individual internal cost-based planner to travel between locations. We then use neuro-evolution to provide incentives to these cost-based planners to reduce traffic in the environment.

Traffic quality can be expressed in several different ways. We first evaluate traffic our traffic reduction policies in terms of 'conflicts', which characterizes situations where an aircraft comes too close to another for safety in a physical space. We then examine

traffic in terms of the amount of 'delay' that all agents incur, which assumes that there is a structure to ensure only a safe number of UAVs occupy the same area. Finally, we look at the total travel time that a UAV can expect to take from the moment it enters the airspace until the time it gets to its destination.

To facilitate an exploration of the UTM problem without waiting for a full simulation of UAVS running with A* , we develop an abstraction of the UTM domain that preserves the core UTM problem. We then investigate performance under differing levels of traffic, a well as two different agent structures. Our results show similar performance for both agent definitions, with delay reduction of up to 68% in high traffic cases.

With a fast version of the UTM problem, we explore the effect of redefining the control structure such that *links*, or edges of the UTM graph, set costs individually. This shifts the control paradigm toward controlling directional travel rather than areas in the space, as was the case with *sector* agents used in previous approaches. Due to our graph structure, we find that there are far more control elements in the link agent approach than in the sector agent approach. We identify a tradeoff; link agents give finer control, but the coordination problem for the sector agents is easier because there are fewer sector agents. This indicates that we can improve performance out of a more distributed link-based setup if we address the challenges of multiagent coordination. However, the UAV traffic management domain presents a uniquely difficult coordination problem; each agent's action can affect the perceived value of every other agent's actions. This means that there is an excessive amount of noise in the system, as another agent's action can have a lot of impact on the reward an agent receives.

We reduce the amount of multiagent noise by reducing the *number* of agents that are capable of learning. We identify that some agents have more ability to influence traffic based on the topology and traffic profile of the graph. This metric we call *impactfulness*. We use this metric to improve the learning by removing less impactful agents from the learning process, making a more stationary system in which the impactful agents can learn.

The contributions of this work are to:

- Introduce a cost-based traffic management approach that is platform-agnostic and fast to implement.
- Develop a multiagent approach to setting costs in this traffic management system

that is adaptive to traffic conditions and learns long-term effects of management decisions.

- Create an abstraction of UAV traffic that captures key physical attributes, creating a fast and flexible simulation method.
- Quantify agent contributions to system performance by experimenting with single agent learning, single agent exclusion, and a sliding number of agents learning in the system.

Adaptive Multiagent Traffic Management for Autonomous Robotic Systems

by

Carrie Melinda Rebhuhn

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 22, 2017
Commencement June 2017

Doctor of Philosophy dissertation of Carrie Melinda Rebhuhn presented on June 22, 2017.

APPROVED:

Co-Major Professor, representing Robotics

Co-Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Carrie Melinda Rebhuhn, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

Table Page

LIST OF ALGORITHMS

Chapter 1: Introduction

Commercial use of unmanned aerial vehicles (UAVs) will drastically increase the traffic management burden in urban airspaces. Recent proposals to update legislation regarding UAV operation in the US airspace [36] have accelerated this discussion, and NASA NextGen has the goal of enabling safe low-altitude UAV flight within the next 5 years [52]. Current methods, which rely on human controllers to manage the airspace, become prohibitively costly and unsafe due to constraints on human multitasking and communication. If UAVs enter the airspace on a massive scale, it is clear that an automated approach is necessary.

Existing work in air traffic control has focused on the flow of airplanes through fixed geographical points and have typically used multiagent system methods to model the distributed system of traffic controllers [34]. A multiagent reinforcement learning framework was used in [7] to reduce congestion by setting the required separation between aircraft, ordering ground delays or ordering reroutes. A hierarchical system of management was considered in [20], with airlines negotiating for airspace to minimize chances of congestion at the higher strategic level, and at the lower tactical level, air traffic controllers could reroute aircraft by introducing no-go zones called "shocks".

While some aspects of these approaches can be applied to UAV traffic management, many new issues arise in this new domain that are not addressed by existing traffic management strategies. Most notably, the national airspace is a large and relatively obstacle-free environment, whereas this is not always the case in an urban setting with UAVs operating in close proximity to one another. If the flow of UAVs is not well-managed, congestion in cluttered environments may exceed a UAV's ability to plan around other UAV trajectories, potentially escalating to larger cascading effects throughout the system. However, it is impractical to model the behaviors of all the actors in the system. This is further complicated by the diverse range of planning software that may be used across the platforms present in the airspace, especially if they are competing for throughput and do not explicitly coordinate with foreign platforms to avoid conflict situations.

(a) (b)

Figure 1.1: An example of how a robot traffic management system would reduce downstream traffic. Robots (UAVs in this case) start from the location indicated by the UAV figure, with a goal indicated by the triangle. Planned paths are marked in a dashed line. Figure (a) shows robots that use a distance-optimal metric. Figure (b) shows how an agent could modify this metric to reduce downstream congestion. Map data ©2016 Google.

We define the UAV traffic management (UTM) problem, and model UAV traffic traveling across an airspace. We model individual UAVs as points planning across the space using A* path planners. UAVs first plan a path across an abstract graph-representation of the space, and then plan over the lower-level obstacle map to travel through the space. We then present several ways of quantifying the performance of a UAV traffic system.

This motivates the use of a high-level UTM learning framework that can map the salient features of the UAV airspace to control actions that can mitigate congestion, such as artificially increasing the cost of flying through particular areas. In this work, we propose a UTM strategy employing a decentralized system of neuro-evolutionary learning agents that manage the flow of UAV traffic through their designated sectors of the airspace.

We introduce "controller agents", whose actions are to set costs across the airspace. The agents control the cost map shown to UAVs in the system, and dynamically adjust the costs based on the current traffic profile. In a simple example, if many UAVs were

currently traveling through the agent's area, the agent will increase its perceived costs to incentivize UAVs to take alternative routes.

We first define control in a way based on current air traffic management approaches, with "sectors" of airspace under the control of one agent. We then identify cardinal directions within this sector that define the sector's control policy. Each sector agent learns a policy that assigns the cost of travel through the sector according to the current number of UAVs in the airspace in a particular direction. We perform policy evaluation by assessing the total number of UAV conflicts in the airspace during a single learning epoch.

Results from testing on a simulated urban environment, the team of sector agents was able to learn appropriate costing strategies to reduce conflicts in the entire airspace. This was despite using only local information regarding the number of UAVs in their sector and the intended heading of each UAV. Comparison to a uniform costing strategy showed on average a 16.4% reduction of conflicts in the airspace.

This approach is dynamic, distributed, human-independent, and can learn to reduce long-term congestion propagation. It is dynamic because we have agents observe the traffic conditions and dynamically respond with a cost appropriate to current traffic conditions. We provide distributed control by allowing agents to control divided *areas* of the airspace, rather than having a centralized controller. This permits arbitrarily large scalability in our implementation. The traffic control is human-independent as well, as it is able to develop a policy to set costs rather than rely on a human controller as is done in conventional air traffic control. Finally, we are able to reduce long-term congestion propagation because agents obtain rewards from long sequences of observed states, in which cascading congestion effects play a part. The agents aim to minimize this *cumulative group congestion* metric, rather than minimizing a local instantaneous congestion.

It also accommodates a wide variety of UAV traffic because *agents do not control individual UAV trajectories*. Instead, we model our approach after dynamic tolling or metering lights, and impose incentives or penalties to control the flow of traffic. By changing the observed cost of routes presented to the UAVs, we escape the computational complexity of dictating each path in the airspace, and accommodate a variety of path optimization algorithms. This is essential to a real-world implementation, where we cannot rely on homogeneity in the vehicles or planners moving through the airspace.

This setup divides the airspace into sectors with cardinal directions, and introduces artificial structure to our control problem. Such a discretization of control can lead to difficulty in incentivizing at the granularity needed to provide a good control structure. Therefore we introduce a second agent architecture that defines agent control over *links* between sectors, where agents set costs for each transition between two sectors. This removes the discretization into four cardinal directions, and allows a single agent to control traffic flow in a particular direction between two points in the airspace. These two different definitions give us agents with different challenges and benefits. The sector agents have access to more information, but sacrifice granularity of control. Link agents have the ability to more finely adjust the flow of UAVs across the airspace, but operate on less information. Link agents must also cope with greater coordination requirements; there can be many more links than sectors, as links represent the area of contact *between* sectors. This creates a much larger multiagent coordination problem.

In this work we also explore the use of our UTM system on physical robots. In this case we replace the simulation of A* across a given map with a ROS system operating across a real obstacle map built with laser scan data. We present a communication structure that allows UTM agents to take in traffic data, output costs, and then enforce a hierarchical path planning from the robots while still taking advantage of pre-built ROS navigation nodes.

In order to investigate the effects of many different traffic profiles, as well as other agent structure, we needed to develop a fast simulation that excluded low-level planning. We introduce an abstraction that maintains the key UTM problem characteristics to quickly evaluate these two approaches. We take the hierarchical traffic model and strip the low-level A* planning component from it. We replace this with a *capacity* for each link that restricts travel of UAVs, and delay any UAV that tries to travel across an at-capacity link. We then attempt to minimize delay in the system using neuro-evolution. This models propagation of delays, should a UAV system have regulated safe numbers of UAVs in the airspace. This could also model the propagation of stalls in the airspace caused by conflict-avoidance maneuvers executed in the physical space.

Using this abstract model, we scale our system to handle up to 400 UAVs, and over 100 controller agents. We demonstrate delay reduction of up to 68% in high-traffic scenarios. We also find that, despite having a much larger coordination problem and using no reward shaping techniques, link agents perform comparably with sector agents.

The contributions of this work are to:

- Introduce a cost-based traffic management approach that is platform-agnostic and fast to implement.
- Develop a multiagent approach to setting costs in this traffic management system that is adaptive to traffic conditions and learns long-term effects of management decisions.
- Create an abstraction of UAV traffic that captures key physical attributes, creating a fast and flexible simulation method.
- Quantify agent contributions to system performance by experimenting with single agent learning, single agent exclusion, and a sliding number of agents learning in the system.

Chapter 2: Background

UAV traffic management combines several fields. In Section 2.1 we will overview related work that has focused specifically on the state-of-the-art of multi-robot motion and routing. At a higher level, there is also the problem of separating the *paths* of these robots so that they do not come into physical proximity with one another, which relates to the problem of traffic control. We discuss traditional methods of traffic modeling and control in Section 2.2. Our work focuses primarily on this problem, which is to manage the traffic before a collision event ever occurs. We also overview the problem of autonomous navigation in Section 2.3, which is essential to understand in modeling a traffic system with autonomously-moving robots. We then give background on adaptive and learning approaches in Section 2.4, which is what we use to develop methods of managing this traffic. Finally, we overview work tackling the unique challenges of multiagent coordination in Section 2.5.

2.1 Robotic Traffic Management

For many robotic systems, a major focus is safely avoiding collisions, which becomes largely a geometric problem of avoiding where an aircraft or a set of aircraft will be in a space. In Section 2.1.1 we will discuss some of these approaches. In Section 2.1.2 we will discuss the related robotic routing or scheduling problem, where robots are ordered to move to certain locations at certain times by a centralized system.

2.1.1 Collision Avoidance

There are two main approaches to developing algorithms to ensure safety in the airspace; the first is to control the *density* of aircraft which are passing through different sectors in order to balance the load on regional air traffic controllers, and the second is to develop optimal *rerouting* procedures in order to avoid a predicted separation losses between aircraft. These problems are coupled, in that as the density of the aircraft

moving through sector decreases the probability of conflicts occurring (and therefore the difficulty of solving this problem) also decreases [14]. These approaches are called air traffic flow management (ATFM) and conflict-avoidance respectively. In this work we focus on the problem of conflict-avoidance.

Conflict-avoidance is currently performed at ground stations during plane flight. While many approaches to ATFM focus primarily on the airspace as an abstract scheduling problem, the problem of conflict-avoidance focuses instead on preventing temporal and spatial conflicts from occurring within an airspace by re-planning conflicted routes, not necessarily restricting this re-planning to passing through fixes. This approach focuses on safety at the route-planning level rather than the scheduling level, and agents have many more options for maneuvers to avoid conflicts. Optimality plays a major role as well; aircraft must also consider the cost of maneuvers taken to ensure the safety of the aircraft [12].

One approach is to treat this as a pairwise problem, where one aircraft will conflict with another in a given time horizon and a reroute or velocity change must occur in order to avoid this conflict. [63]. This unfortunately does not explicitly tackle the problem of 'cascading' conflicts, which are conflicts generated by rerouting between two planes. Approaches which address this include plotting courses for avoidance maneuvers using mixed integer programming [26, 17], distributed search strategies [72, 77], optimal control [89], game theory [45], and negotiation [78].

These approaches work well when the conflict between aircraft is well-defined and has no uncertainty. However in real flight there can be differences between the announced flight plans and the actual trajectory taken. The FAA separation restrictions accommodate this uncertainty in flight, however as the airspace becomes more crowded, it will become impossible to attain the desired throughput while maintaining the current safety restrictions. In order to maintain safety in the airspace while allowing a higher throughput, it is necessary to automatically identify which types of planes need higher safety allowances, and to predict ways these planes may deviate from their courses that may not be communicated to controllers in a timely manner. The information on air traffic necessary to make these more sophisticated decisions should be available to each plane through the (mandatory by 2020) ADS-B implementation, and the (optional) TIS-B subscriptions [35].

ADS-B stands for Automatic Dependent Surveillance - Broadcast, which uses satel-

lite navigation and periodically broadcasts its location, enabling it to be tracked by regulatory entities [1]. TIS-B stands for Traffic Information Service - Broadcast, and is a situational awareness tool for pilots to see real time positions of nearby aircraft [80]. In our work, we rely on the ability to observe aircraft in the space in order to determine control strategies, and these technologies would make this observability a feasible assumption.

2.1.2 Robotic Routing and Scheduling

Previous research has explored congestion as a centralized controller scheduling and routing problem. Qiu et al. give a comprehensive survey of centralized scheduling methods for automated vehicles [68]. A centralized controller tells every robot in the system when it can travel, and where it can travel. In order to compute a solution, these methods require full state information and are often slow and computationally complex. Dynamic routing methods [87, 21] manage the time-window of each robot through time expanded graphs without needing full state information [40]. However, these methods become computationally expensive when applied to a fast-changing system.

Other methods for traffic routing have focused on vehicle negotiation to resolve conflicts in congested areas. Large numbers of aircraft can use these schemes to resolve local conflicts resulting in improved system performance [93]. Some research has developed peer to peer collision avoidance protocols. More specifically, a software called AgentFly uses an agent-based distributed negotiation approach to the air traffic routing problem [64]. This technique works well in domains with standardized communication and vehicle abilities. However, UAVs are diverse in both hardware and software. The method that we propose in this work allows for these characteristics of the domain and still leaves open the possibility of future progression towards standardization.

Unlike previous research we use an incentivized routing technique instead of a central controller or vehicle negotiation scheme. Our proposed UTM system does not assign paths to each robot nor does it require the robots to act in a cooperative manner. Our algorithm assumes each UAV is using a cost-based planner and so manipulates the cost of travel through the airspace to incentivize the robots to avoid congested areas.

2.2 Traffic Modeling

Traffic models are useful in that they allow researchers to experiment with modifying traffic incentives without actually performing experiments on full-scale traffic. There are two main types of traffic; vehicular traffic, which relies on modeling human reaction and control for embodied vehicles, and network traffic, which assumes an algorithm is controlling the movement of traffic through the system. Our traffic type falls into the intersection of these two systems; we assume a set of embodied vehicles following an algorithmic control method.

Hoogendoorn and Bovy divide vehicular traffic into three broad categories based on level of modeling detail: submicroscopic, microscopic, mesoscopic, and macroscopic. Each of these approaches takes a varying level of abstraction of the motions of traffic throughout a system. Submicroscopic models focus on detailed physical models of each vehicle moving in the system. Microscopic models focus on general models of individual vehicles, and make heuristic rules for their interactions in the space. In contrast, mesoscopic and macroscopic models do not model individual vehicles; they focus on modeling the traffic density based on fluid physics analogs [107], or statistical data [61].

Under the classification structure offered by Hoogendoorn and Bovy [47], we follow a *microscopic* traffic model with *deterministic* driver modeling. This means that we model each UAV as a point moving in the space. However, instead of modeling the motion of each traffic entity based on following-distance or other human-based metrics, we make the assumption that autonomous UAVs will follow cost-optimal paths.

In contrast, much of the work in traffic research at the microscopic modeling level seeks to identify the behavior of individual drivers in the system. Toledo et al. model the driver actions at stop lights, and how this affects the traffic flow between these lights [92]. Miyajima et al. fit two kinds of models to real samples of driver behavior; an optimal velocity model and a Gaussian mixture model [60].

We assume that each of our 'drivers' is in fact a rational robot following a cost-optimal algorithm. This is a departure from some of the major methods of traffic modeling, and provides an algorithmic rather than instantaneous mathematical approach to following behavior. Because our traffic modeling must model a package delivery application, this encapsulates a somewhat higher level of reasoning and a significantly lower level of physical influence than traditional traffic modeling.

Our approach of assuming algorithmic behavior of each element, rather than fitting models to human behavior, may actually more closely resemble network traffic modeling. UAV traffic fits somewhere in the middle; autonomous and rational entities move through the space, but have the potential to interact physically with one another. Shortest-path algorithms are popular for network routing, however it has been shown in previous work that in a routing domain that objects being sent on a longer path may be beneficial to the system as a whole as it may alleviate downstream congestion [109].

2.3 Autonomous Navigation

A navigation task is the process of determining a path from a start point to a goal point that has sufficient safety and path-efficiency. Autonomous navigation has existed in the literature from the earliest stages of robotics, although the inputs that robots are now able to handle have increased substantially in complexity. Path planning research has focused mainly on abstract navigation problems across maps, and we will discuss algorithms for this in detail in Section 2.3.1. Of recent interest is the problem of applying this path planning to multiple robots, which we will discuss in 2.3.2. Finally, we will discuss approaches toward navigating in an unfamiliar environment in Section 2.3.3.

2.3.1 Path Planning

A planning algorithm locates a path through a space that a robot can travel. This may be in terms of pixel values or waypoints in the space. Autonomous systems rely heavily on path planning algorithms to move safely and efficiently through the space. There are a wide variety of algorithms to accomplish this, which can focus on maximizing time efficiency, path cost efficiency, or risk mitigation. Path planners can be deterministic or probabilistic in the method with which they search through the space, resulting in varying path cost optimality.

An *optimal* planning algorithm minimizes the cost across the planned path. A common optimal planning algorithm is the A* search algorithm. A* was introduced in 1968 by Hart et al. [44], and marked an improvement over Dijkstra's algorithm [32] by introducing a heuristic to truncate the search, making it more time-efficient. This heuristic estimates the cost of a proposed path through the space. Optimality guarantees are

maintained as long as the A* heuristic is *admissible*, meaning that it does not overestimate the cost of a path. Given the trivial heuristic of always estimating a path cost of zero, A* reduces back to Dijkstra's algorithm.

Algorithm 1 `A* (G, s, h)`

```
1: for each vertex u in V do
2:    d[u] := f[u] := infinity                          initialize vertex u
3:    color[u] = WHITE
4:    p[u] := u
5: end for
6: color[s] = GRAY
7: d[s] := 0
8: f[s] := h(s)
9: INSERT(Q, s)                                         discover vertex s
10: while Q! = ∅ do
11:    u := EXTRACT − MIN(Q)                            examine vertex u
12:    for each vertex v in Adj[u] do
13:       if w(u, v) + d[u] < d[v] then
14:          d[v] := w(u, v) + d[u]                     edge (u, v) relaxed
15:          f[v] := d[v]h[v]
16:          p[v] := u
17:          if color[v] = WHITE then
18:             color[v] := GRAY
19:             INSERT(Q, v)                           discover vertex v
20:          else if color[v] = BLACK then
21:             color[v] := GRAY
22:             INSERT(Q, v)
23:          else
24:                                                    edge (u, v) not relaxed
25:          end if
26:       end if
27:    end for
28:    color[u] := BLACK                                finish vertex u
29: end while
```

In this work, we use the Boost Graph Library version of A* , given in Algorithm 1, with a Euclidean heuristic, meaning that we estimate the straight-line distance from a given point in the space to another given point in the space. This will never overestimate the path-cost, because our graph restricts the UAVs to take longer hops.

We change the perception of optimality from *distance*, the traditional approach, to *predicted traffic reduction*. We do this by *adding* this value to the estimated Euclidean distance, therefore maintaining the admissibility of our Euclidean heuristic. This value used to modify the perceived distance is approximated through a learning process.

There have been many extensions to A* tailoring it specifically to the problem of controlling UAVs. Tseng et al. examined the use of A* for civilian UAV flight, and adapted it to consider the strength of signal available from base stations across the map [94]. Genetic algorithms have also been explored as a method of UAV path planning. Allaire et al. explored using a genetic algorithm for UAV path planning across a map using a Field Programmable Gate Array (FPGA) to parallelize genetic algorithm computation [9]. Qu et al. also explored compared the use of a genetic algorithm and a heuristic path planner for planning across a battlefield [69]. They looked at balancing the path cost with the threat level of a given path.

Risk-optimization is an important planning consideration for many UAV applications. Ant colony optimization has been used to plan across spaces with multiple different types of threat, such as no-fly zones, hazardous weather areas, and low-altitude control zones [110]. Özalp and Sahingoz looked at planning using parallelized genetic algorithms to find paths that minimize a scalarized cost metric that model threat zones around radar [62].

A hierarchical A* method was introduced by Wang et al. to handle high-level navigation across large map, focusing specifically on using OpenStreetMap [100, 43]. Looking across a large map hierarchically allowed autonomous navigation systems to plan across much larger spaces. This reduced the combinatoric costs of applying A* across an entire more detailed map. In our work, we use a similar hierarchical A* structure, although our intent is to apply our traffic control algorithm rather than mitigate computational costs.

A* is by no means the only path planning algorithm. Other algorithms offer even faster solutions, with bounded optimality. Rapidly Exploring Random Trees (RRT) was developed as a fast way of planning across the space by randomly selecting points and identifying connections between them [55]. RRT has many extensions, including CC-RRT* , which extends RRT* by approximating risks of constraint violation[57]. RRT has also been used in conjunction with a gaussian process in a UAV planning application to explore in an unknown environment [105].

We base our learning algorithm around modifying the cost map given to a cost-based planner. Though there are many alternatives in path planning, we have chosen the A* algorithm for planning low-level motions as well as high-level navigation through the space. We use A* because it is a widely known cost-based planner, has optimality guarantees, and a fast implementation is available through the Boost Graph Library. Future work could explore the use of other algorithms for planning across the space, but this is out of the scope of our work here.

2.3.2 Multi-UAV Path Planning

The planning problem becomes much more resource-intensive when applied to multiple UAVs. The problem of multi-UAV path planning is similar to the problem of collision-avoidance, but it looks at a longer time-horizon instead of investigating an imminent conflict event. Joint multi-UAV path planning relies entirely on centralized communication and coordination, the complexity of which grows nonlinearly with the number of UAVs participating in the system. Research has therefore focused on ways to increase the speed of these joint planning algorithms, or on applying them to small numbers of UAVs.

RRT has been investigated as one solution for generating multiple trajectories for UAVs in the system, considering kinematic restrictions on UAVs so that they did not interact in the space [53]. Non-conflicting paths based on differential geometry have also been proposed, which can be mathematically constrained to deconflict with one another [73]. Though these approaches provide valid non-conflicting paths, the experiments presented a maximum of three UAVs interacting in the space.

Multi-UAV path planning has also addressed the application of convoy protection, which adds restraints on UAVs moving through the space to be spatially near a convoy at all points in their plan [33]. In another military-based application, UAVs performing a surveillance task used a decentralized planning method combined with an auction-based system in order to cooperate in the same space [108]. UAVs have also coordinated for a tracking task using mixed-integer linear programming, although deconflicting paths were not considered in this case [8].

Singh et al. tackled an informative path planning problem with multiple robots using their algorithm *eMIP*, that used spatial decomposition and branch and bound

techniques combined with sequential allocation [76]. Stranders et al. mitigated the complexity of a multi-robot planning problem by introducing an adjustable look-ahead parameter to restrict plan length while coordinate a mobile sensor system [84]. Learning approaches have also been used in group informative path planning task. Zhang et al. introduced the cooperative and geometric learning algorithm (CGLA) to allow UAVs to learn informative paths across the space using Q-learning approach.

2.3.3 Learned Navigation

Deterministic path planning algorithms are not the only method of navigation through a space. In cases where the environment is entirely unknown, an iterative approach can be useful in navigating through a domain. The use of machine learning approaches can be applied to navigation tasks in order to process state information that is confusing or may not have a model to connect to the desired movement actions.

Navigation tasks can be unclear particularly when there is human interaction involved. The Fun Robotic Outdoor Guide (FROG) project has investigated the effects of this by building in a social cost function into its reinforcement learning method of navigation [65]. This can iteratively shape the robot's movements around people by observing their reactions to the robot.

The field of visual navigation has been the subject of recent study. The collection of large sets of images, including driving image databases such as ImageNet, have provided a wealth of data for research into enabling tasks such as autonomous driving [30]. Off-road robot navigation has used machine learning on driving image sets from ImageNet to learn paths through the space [67].

Deep learning has gained popularity for handling large amounts of image data, and has been used for navigation tasks as well. Deep learning was applied in a multi-agent setting in order to perform collision-avoidance based on observed images surrounding the robot [56]. A cognitive mapper and planner (CMP) was also developed using a neural architecture to learn to connect first-person views to a sequence of actions [42]. This could then be used to find a desired goal state.

Similar to the task of image recognition, the meaning of natural language navigation instructions provide occasionally unclear direction, and must be processed with machine learning techniques. This approach is termed *parser learning*. Chen and Mooney connect

natural language commands to a formalized plan for navigating through an environment using a lexicon learning algorithm [24]. Matuszek extends this work by removing the need for an a priori obstacle map [58].

Learning routes across a space is an important field of research. It is particularly important in unknown domains, or domains for which there is no map. The complexity of our work lies not in the map being unknown, but in the actions of the UAVs traversing the map being largely unknown. Instead of focusing on UAV navigation as the problem to learn, we look at the routing of many UAVs. This way our learning algorithm can handle the unknown portion, rather than focusing on navigating across a known map.

2.4 Learning and Optimization

In this section we will go over learning and optimization approaches. First in Section 2.4.1 we will cover reinforcement learning and Q-learning, which form some of the earliest approaches to learning. Next, in Section 2.4.2 we will cover evolutionary algorithms, which are a population-based method of iterative learning. Finally, in Section 2.4.3 we will cover coevolution, which is the multiagent method used in this work.

2.4.1 Reinforcement Learning and Q-Learning

Reinforcement learning is an iterative method of learning to map situations and actions to a reward signal [86]. Reinforcement learning is *model free*, which means that it updates the quality of its actions iteratively through exploration and feedback rather than querying a predictive model. This can provide substantial benefits when a model is unavailable.

The general method of performing reinforcement learning is for a autonomous entity, or *agent*, to sense its state, query its policy, take an action, receive a reward for this action, and then update the values of its policy. An agent can occasionally intentionally take an off-policy action for purposes of exploring the state space. There are two basic methods of doing this; ϵ-greedy [86] and soft-max [18]. The strategy behind ϵ-greedy action selection is to take an off-policy action ϵ percent of the time regardless of the quality of the action. Soft-max, in contrast, selects the action based on a soft-max activation function, where each action is chosen with the probability:

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}} \tag{2.1}$$

where $\mathbf{x}^T \mathbf{w}_j$ yields the current valuation of state $\mathbf{x}$ given action j. This results in high-valued actions being selected with higher probability and lower-valued actions to be selected with low probability.

Q-learning is a method of incorporating delayed feedback into a value function [102]. It operates on a *temporal difference* update, which looks ahead at potential future states that the agent could occupy. The update method is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \cdot (r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.2}$$

where $Q(s_t, a_t)$ is the current evaluation of the quality of action a_t at state s_t, α_t is the learning rate, and γ is the past-state discount factor. Q-learning has been shown to converge to the optimal policy for the single-agent case in a static domain [101], but this does not hold for the multiagent case due to the domain's more dynamic nature.

A common implementation of Q-learning is to use a Q-table, and to discretize states and actions to match elements in this Q-table [51]. States with continuous representations may not fit into this tabular approach, and approaches such as tile coding [74] and function approximation [59, 81] have been used.

Millán et al. introduced an incremental topology preserving map (ITPM) to serve as the function approximator for continuous Q-learning [59]. Smart and Kaelbling use locally weighted regression to approximate the Q-table [81]. Neural networks have also provided a method of function approximation for continuous Q-learning, as used by Tesauro for his work in temporal-difference backgammon [91]. Q-learning is not the only method that can be used to modify neural networks; in the next section we will discuss evolutionary algorithms that adapt populations of neural networks.

2.4.2 Evolutionary Algorithms and Neural Networks

Unlike standard reinforcement learning, evolutionary algorithms are based around having *populations* of policies, rather than adapting an individual policy [15]. This mimics an evolutionary process from nature, whereby population members produce new slightly

mutated population members, and then the members will compete with each other and the fittest will survive. Genetic algorithms are a specific kind of evolutionary algorithm that includes *crossover*, or recombinations of existing policies in the space [39]. This is combined with *mutation* in order to produce child solutions. Mutation is the process by which some elements of a policy in the population is changed.

Neuro-evolution consists of an evolutionary algorithm that has neural networks as its population members. Neural networks are function approximators that can be used to model continuous state-action control policies to arbitrary accuracy while only requiring a coarse approximation of the system state [48]. There are many recent implementations of neuro-evolutionary algorithms. Notably, the NEAT framework provides an adaptive way to evolve the network *topologies* as well as improving the netwrok weights [82]. This allows a population of networks to adapt their structure in order to accommodate the complexity of a given problem. Whiteson et al. present an algorithm for automatic feature selection, FS-NEAT that extends the NEAT framework to select important inputs [103].

Evolutionary algorithms have also been used for multi-objective optimization. The Strength Pareto Evolutionary Algorithm 2 (SPEA2) provides a method of using evolutionary algorithms with multiple objectives [112, 19]. There are many objectives to consider in the traffic management problem, such as safety, travel time, cost of fuel, and other real-world considerations. However, we leave the investigation of the traffic management problem as a multiobjective problem as a topic of future work.

2.4.3 Cooperative Coevolution

Cooperative coevolution refers to a set of agents in a system that learn together using an evolutionary algorithm. Typically these systems learn together toward a common goal, which is reflected in the fitness values given to these agents. As an extension of standard evolutionary algorithms, CCEAs have been shown to perform well in cooperative multiagent domains [37, 66]. Pseudocode for a cooperative coevolutionary algorithm is given by Algorithm 2.

Routing for a UAV team was evolved using cooperative coevolution in combination with game theory [104]. Zheng et al. also coevolved coordinated UAV paths [111]. UAV routing was also investigated using cooperative coevolution in conjunction with an ant

colony algorithm [85].

In this work we use cooperative coevolution for UAV traffic management, but we do not focus on coevolving individual UAV routes. Instead we focus on evolving a decentralized policy for incentivizing travel through parts of the *airspace*. UAV path planning is left to deterministic path planners, as discussed in Section 2.3.1.

Algorithm 2 `CCEA(agents,domain)`

```
1: Initialize k neural networks for every agent.
2: for epoch = 1 → total_epochs do
3:    Mutate population members to create k new population members for each agent.
4:    for i = 1 → 2k do
5:       Set ith population member as the active policy for every agent ∈ agents.
6:       team_fitness = domain.Simulate(agents)
7:       Assign every active population member their team_fitness score.
8:    end for
9:    Drop k population members with the lowest fitness for each agent.
10: end for
```

2.5 Multiagent Coordination

The canonical multiagent learning problem developed from being a game with two players. In the previous section we discussed methods of modeling agent behavior. The majority of these methods were applied to two-player domains. However, the field of multiagent learning is increasingly focusing on *large* multiagent systems, such as network routing and sensor combination [88, 99]. In this section we will discuss the challenging properties of scaling in multiagent systems. Unlike in the last section, we will also consider cooperative multiagent domains, as these tend to grow much larger in size than competitive domains.

2.5.1 Multiagent Air Traffic Control

Recent work in multiagent systems has focused on developing effective routing for commercial air traffic across the national airspace. In air traffic, delays and congestion can cascade throughout the system, and are difficult to mitigate and control. Using reinforcement learning agents to manage air traffic through geographical fixes, Agogino and

Tumer [7] were able to reduce airspace congestion by over 90% when compared to current traffic control methods. Cooperative coevolutionary algorithms provided similar demonstrations of the efficacy of a multiagent learning approach [106, 27]. The setup presented in [7] has agents control the separation between planes in the airspace. These works provide decentralized control of an airspace, but do not consider movement around obstacles as in our approach.

In this work, we apply a similar network of routing agents to control the flow of UAV traffic. However, we consider the UAV domain where platforms are not restricted to fly through particular fixes in the environment. Furthermore, our routing algorithm is based on the assumption that we have no direct control over the path planning aspect of the UAVs.

The problem of handling UAVs in the airspace is also emerging as a distinct field. Aubert et al.[13] proposed a framework for UAVs to communicate potential flight plans to a centralized air traffic management system that determines whether to accept or reject the plan. Recent work by Digani et al. has also focused on the use of UAVs in an industrial setting using ensembles [31]. They also use probabilistic path planning to prevent traffic jams within a warehouse [31].

Controlling the social aspect of UAVs is also relevant in recent work. Foina et al. [38] investigate a method for individual airspace owners to allow or disallow flights through their airspace, mitigating annoyance by property owners. While not directly related to a study of social acceptance of UAVs in the airspace, our approach addresses the issue of density of UAVs in the airspace. The safety and potential noise issues associated with a dense swarm of UAVs could be a significant limiting factor in long-term acceptance of UAV delivery.

2.5.2 UAV Swarms

UAV swarming behavior has become a popular topic of research. The main problem with practical UAV swarming behavior is positional awareness of other elements of the UAV swarm. Several centralized methods of communication have been used for early demonstrations, such as Vicon motion capture [49, 71, 98]. However, there is an increased focus on the use of UAV swarms in outdoor environments without the need for complex systems to be in place. We will focus primarily on these applications, as outdoor use is

the desired application for our work.

Swarms as mobile networking platforms have been a particularly active topic of research, termed flying ad-hoc networks, or 'FANETs' [16]. FANETs are distinct from MANETS (multiagent ad-hoc networks) and VANETs (vehicular ad-hoc networks) because of the higher degree of mobility in FANETS because of the generally higher speed and maneuverability that UAV platforms offer. This also means that typical distances in a FANET problem will be larger than with a VANET or MANET.

Broadly, swarming behavior to maintaining network connectivity can be divided into two categories: networks formed with the intention of providing informational relays between multiple ground stations, and networks formed with the intention of maintaining connectivity for multi-UAV coordination purposes. Sivakumar et al. investigated UAV swarms as a wireless communication relay solution [79]. This was proposed with the intention of UAVs autonomously detecting signal sources and self-organizing to connect points of communication. Physical UAV experiments toward wireless UAV networking were conducted by the SUAAVE project using a small UAV platform with a target payload under 100g [23].

Toward the goal of maintaining network connectivity within a self-assembling team, Teacy et al. focused on the problem of maintaining connectivity between elements of a swarm by incorporating this restriction into the multi-UAV planning algorithm used by the swarm elements [90]. Hsieh et al. proposed a reactive controller designed for communication link maintenance, which incorporates radio connectivity into a position controller's input [50]. The physical communication aspects of swarming behavior are also under research. Chlestil et al. investigated sensors that would enable networking between swarms of UAVs, and found that Free Space Optics (FSOs) provided the most feasible solution for fast transfer of data between swarming UAVs [25].

UAVs first gained popularity in the military sector, with large, complex drones. With the miniaturization of UAVs and an increase in focus on coordination of many less-complex elements, military applications of swarming UAVs have been proposed. Typical operation of a UAV requires more than one human per UAV, but with swarming algorithms the control of several UAVs simultaneously could become feasible for a single human. Swarm control for up to 50,000 UAVs using the pheremone control algorithm ADAPTIV was investigated for expanding the ability of a single soldier to control large swarms of UAVs [11]. Lamont et al. also investigated military mission planning for

swarms of UAVs using a swarm path planning algorithm, considering multiple mission objectives using an evolutionary algorithm [54]. Multiagent swarming behavior has also been investigated with the intent to automatically identify targets using computer vision and multiple cooperating UAVs [29].

2.5.3 Scaling with Large State or Action Descriptions

A common flaw of multiagent systems is known as the 'curse of dimensionality'. This refers to the fact that as the number of players in a system increases, the size of the problem formulation may grow nonlinearly.

Agent modeling suffers particularly when a set of agents must coordinate a joint action or consider state information of other agents. The number of joint actions and observations increases exponentially with the number of agents. Monte Carlo Tree Search (MCTS) has been used in cojunction with Bayesian reinforcement learning in large POMDPs [10]. 'Large' in this context, however, consisted of four agents and 15 states. It is not clear whether this approach would scale to hundreds of agents or states.

Not only joint action spaces, but large state spaces can slow learning performance. When a problem is framed as a multiagent Markov decision process (MMDP) the state becomes exponential in the number of agents. Use of generalized learning automata has been explored in order to reduce this dimensionality

2.5.4 Difference Rewards

Large multiagent systems have many challenges when performing a coordination task. A common approach to coordinating agents is to give the same score to the entire system for a task that they are completing. This gives some indication of how well the task is being completed as a result of their joint effort, but does not directly assess how each of them contributed. The amount of the reward that they are *not* responsible for is considered 'noise'.

A major focus in multiagent learning is reducing the effects of this 'noise' while ensuring a reward for working toward a group objective. Noise reduction has been approached by using *difference rewards* with counterfactuals to suppress the effects of other agents' actions [2, 3, 4, 6, 7, 95, 96]. Difference rewards reduce the noise in the

system by subtracting off the impact of other agents in the reward. This reward works by comparing the evaluation of the system to a system in which an agent is not there, or replaced by some counterfactual which refers to an alternative solution that the agent may have taken [4]. Mathematically this is expressed as:

$$D_i(z) = G(z) - G(z_{-i} + c) \tag{2.3}$$

where $G(z)$ is the full-system reward and $G(z_{-i} + c)$ is the reward for a hypothetical system in which agent i was removed and its impact was replaced by a counterfactual c.

The difference reward approach to reward shaping has been shown to improve the speed of convergence and performance in many multiagent problems, such as the El Farol Bar Problem and Multiagent Gridworld [5, 97]. The difference reward analytically decomposes a system-level reward via a trade-off between *alignment* and *sensitivity*. Alignment refers to the amount of information carried in the reward signal that indicates how an agents' actions impact the system as a whole. Sensitivity measures the reward signal information that is directly impacted by an agent [5].

Chapter 3: UAV Traffic Modeling

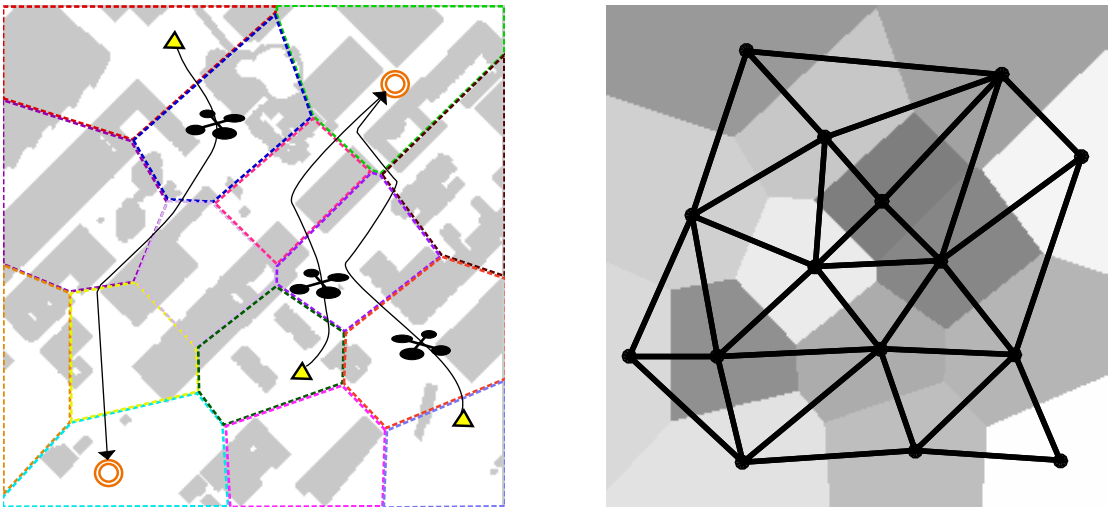
Consider an airspace containing multiple UAVs traveling between various start and goal locations. Each UAV plans its trajectory through the airspace according to a cost-based path planner aimed at minimizing travel cost. Furthermore, apart from low-level collision avoidance procedures, each UAV acts independently and does not coordinate with any other UAV in the airspace. Our goal in this work is to monitor and reduce congestion in the airspace by employing high-level UTM agents that manage the cost of traveling through sectors of the airspace.

We use an iterative simulation that consists of UAVs traveling across a graph by using A* . The graph consists of randomly-placed nodes, which represent sector centers. Edges are then constructed randomly between these nodes to make a planar graph. These edges represent the boundaries between the sectors. We construct our graph to have random planar edges, so that it can represent a connection map between physical areas of space.

A schematic of the problem setup is shown in Fig. 3.1a. The airspace is divided into sectors that are each controlled by a single UAV Traffic Management (UTM) agent, which controls traffic by assigning costs to travel through its portion of the airspace. UAVs flying from start to goal locations in the world must travel through a series of sectors and are subject to the cost of travel between each sector. The sector agents act as a decentralized system of traffic controllers that each individually learn to apply the appropriate costs in their respective sectors to minimize overall congestion in the entire airspace. The obstacle map shown in Fig. 3.1a is a section of San Francisco's South of Market area, with data gathered from a building height map [22].

Given the potentially large number of UAVs in each sector, it is impractical to specifically account for the individual states and policies of all UAVs. This motivates the use of a learning algorithm for developing costing strategies that map a reduced state space to the travel costs applied in the sector. In this work, we formulate the control policy of each sector agent as a single-layer neural network where the weights of the neural networks are learned using an evolutionary algorithm.

Each UAV generates an initial path to the goal by querying the initial cost to travel



(a) (b)

Figure 3.1: (a) The UAV traffic management problem. The airspace is divided into discrete sectors with a single UTM agent controlling the cost of travel through each sector. UAVs traveling from start (triangle) to goal (circle) locations plan and execute paths to reduce their individual travel cost without explicitly coordinating with other UAVs in the airspace. This can lead to potentially dangerous conflict incidents with significant cascading effects. The goal of the sector agents is to learn appropriate costing strategies to alleviate bottlenecks in the traffic and minimize congestion in the airspace. (b) Graph G_h of sector agent connections used in the experiments. Agent policies assign edge costs based on the current sector state s, and the discretized cardinal direction followed on the edge. Different colors on this graph refer to different sector memberships of areas of the space.

along all edges of the graph and performing a high-level A* search to determine the lowest-cost sequence of nodes to travel through to reach their goal location. Once a UAV reaches its goal location, it disappears from the airspace. The UAVs then independently calculate their trajectories through the space with this information. This approach for UAV traffic control manipulates the costs perceived by the UAVs rather than directly coordinating their trajectories, and therefore most of the computational cost can be distributed across the entire set of UAVs rather than at a single point.

3.1 Airspace Construction

There were two ways in which we generated map; generating a map from a human analysis of a pixel-level physical map, and a random map generation algorithm. We wanted to examine both of these methods because each had a distinct advantage. The human-created map has a clearly-explainable logic behind control, and represents a real physical space. However, construction of the airspace by a human is labor-intensive and subject to bias. The randomly-created map allowed us to generate any complexity of planar map with little time cost. However, the random maps do not directly map to an obstacle-filled space, and do not necessarily reflect a distinct physical system. In this section, we will explain the method behind constructing each of these maps.

3.1.1 Physical Map

We first looked at constructing an airspace across a physical obstacle map drawn from buildings in a real city. We looked up height maps for San Francisco, and identified a downtown area in the South of Market (SoMa) district that we would use to generate our obstacle map. We then applied a threshold to the height map data to obtain a binary obstacle map.

We partitioned the space by visually identifying intersection points in the space. We determined 15 suitable intersection points, which would define 15 different agents across the space. We then used a Voronoi partitioning in order to divide the space into sectors of control for the agents. Traversal across each pixel in the 256x256-pixel took 1 simulated step for each UAV.

This method depended highly on human factors; for one, the map area was hand-selected from a larger map of San Francisco. This was not only labor-intensive, but involved selecting an area that was biased toward a map that would provide a human-identifiable traffic scenario. Additionally, the points were human-selected rather than randomly-selected, which shows us control for a scenario that is based on what 'looks like' intersections to a human observer. Because it is not certain that intersections are a good analogy for this problem, and we wanted a less labor-intensive way of generating maps, we also explored a random map generation method. We will discuss this in detail in the next section.

3.1.2 Random Map

We wanted to have a testing airspace that could be arbitrarily complex, and independent of human design. To do this, we developed an algorithm to generate a random map for us to test. We made the additional restriction that the airspace be planar, so that it could be a valid abstraction of a 2-dimensional pixel-level obstacle map, as we investigated using the previous physical map airspace definition. Though 3-dimensional travel is a potential option for UAVs, we restricted ourselves the planar case so that we would have a valid abstraction of the human-generated height map method as explained in the previous section.

Algorithm 3 `GenerateRandomAirspace(nodes)`

1: N = |`nodes`|
2: Initialize graph G with `nodes`
3: $possible\_connections = \binom{N}{2}$
4: randomize($possible\_connections$)
5: **for** $connection \in possible\_connections$ **do**
6: **if** *connection* does not overlap any $edge \in G$ **then**
7: Add *connection* to edge list in G
8: **end if**
9: **end for**
10: Return G as high-level airspace graph

Though we selected a planar map generation method here, this is not strictly necessary for our control algorithms. We merely desired this to connect more firmly with the physical space that we were abstracting. If we permitted non-planar graph generation, we could model a space with three dimensions. This would effectively enable UAVs to pass above or below one another in the space. Because this does not change the nature of the learning algorithms that we apply in this work, we do not address here the possibility of having more than two dimensions of movement in the airspace.

3.2 Traffic Generation and Elimination

In this section, we will go over the different parameters regarding the generation and elimination of traffic in the system. This section deals with how traffic originates in the system, how its endpoint is defined, and what happens to the traffic when it has reached

this endpoint.

Deterministic traffic (ρ): Traffic was generated at every point in the system, at a pre-set interval. This generation rate, ρ, determined how frequently traffic was generated, with a lower value of ρ corresponding to a higher rate of traffic generation. The units of ρ are $UAVs/s$.

Multiple generation: Under the deterministic method of traffic generation, a node had the potential of generating more than one traffic element at the specified interval. In later experiments, this was increased from one to several in order to show the effects of a high amount of traffic entering the system all at once.

Probabilistic traffic: Traffic was generated probabilistically at each step, with probability p_{gen}. This differs from deterministic generation because there could be asynchronous introduction of new traffic by each node in the system.

Static destinations: Destinations were chosen from a list of pre-defined destinations. Traffic would be sent to these destinations as it was generated, cycling through this destination list. This provided even traffic flow across the entire map, and allowed us to reduce randomness in the system if desired.

Random destinations: Destinations were selected randomly at the time of generation. A UAV could select any node in the airspace, apart from the node at which it had just appeared.

Removed at destination: When a UAV reached its destination, it could be removed from the simulation, and would no longer contribute to traffic in the system. This would correspond to a package delivery application where a UAV landed and was stored or recharged at its destination.

Rerouted at destination: Alternatively, a UAV could reach its destination and be rerouted to a different node immediately. This models a physical system where a UAV makes several sequential deliveries in the airspace.

3.3 Performance Metrics

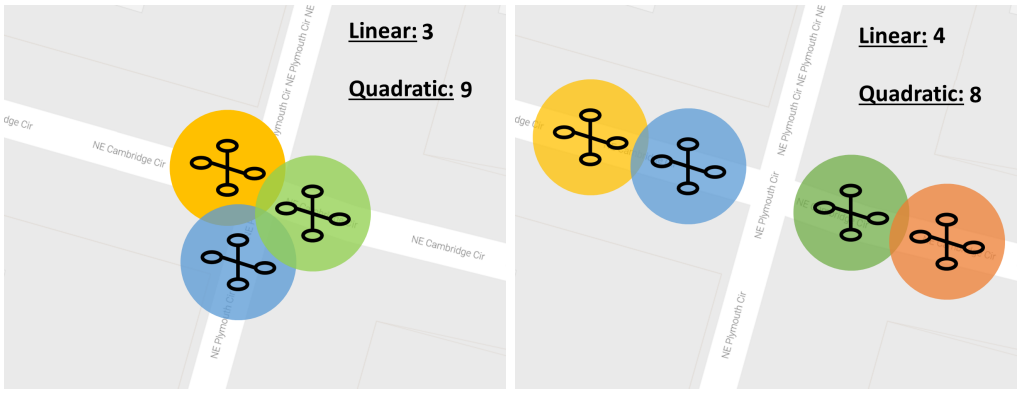
There are various ways to report traffic; we report on three different metrics. The first, *conflict*, describes a physical proximity of more than one UAV in the space (Section 3.3.1). The next metric,*delay*, assumes enforcement of a certain restriction of travel across the airspace, and counts the number of times a UAV must wait in order to accommodate this enforcement (Section 3.3.2). The final metric, *travel time*, assumes the same restriction of travel, but aims to measure the average time that a UAV takes to traverse the system (Section 3.3.3).

3.3.1 Conflict

We would like to train the agents to reduce congestion across the airspace. However, there are many definitions of 'congestion' in traffic research. We draw one definition from the field of air traffic management. In air traffic management, 'conflict' events indicate that two or more aircraft have come within unsafe proximity to one another.

As the UAVs execute their planned trajectories, conflicts occur when two or more UAVs pass within some distance, $d_{conflict}$, of one another. The number of conflicts detected in the entire airspace over a single learning epoch is then used to calculate the team fitness evaluation. There are two different ways of viewing the severity of a conflict event. One way of viewing conflict is to look at the number of UAVs that come into close proximity with one another. This is our *linear* conflict penalty. Another way is to penalize the severity of the conflict, or number of UAVs involved in a conflict instant. We do this by adding a squared term to calculate the penalty for co-located conflicts, which we call our *quadratic* conflict penalty. The differences between these two methods are shown in Figure 3.2

This approach models concerns in air traffic management, but does not model possible propagation of physical delays caused by conflict-avoidance maneuvers. We consider it bad for a UAV to come too close, but assume that lower-level conflict-avoidance maneuvers will handle these interactions if there are sufficiently few of them. Later, in Section 3.3.2, we will introduce a performance metric that models the physical delay propagation caused by UAVs moving in the system, but the results presented in this section focus only on conflict as the reported metric.



(a) (b)

Figure 3.2: Two examples of conflict situations. Figure 3.2a shows three UAVs coming into proximity with one another simultaneously. With our *linear* conflict count, the fitness penalty would be 3. With our *quadratic* count, the fitness penalty would increase by 9 due to the squared term. In Figure 3.2b there are two areas of conflict with only two UAVs in conflict. Under a linear count, the fitness penalty would increase by 4, and under a quadratic count the fitness penalty would increase by 8. The situation in Figure 3.2a is preferable under the linear conflict count method, and the situation in Figure 3.2b is preferable under the quadratic method.

3.3.2 Delay

Removing our pixel-level simulator removes also our ability to count 'conflict' penalties as they occur in the space. Instead of counting this metric, we limit the number of UAVs that may travel along an edge at once. The objective is to reduce the cumulative delay experienced by all UAVs in the system. In our abstracted problem, we arbitrarily select the capacity of each edge i to be a constant $\kappa_i = 2$. This edge capacity would be a calculable feature of the physical space in a real-world implementation. When an edge has $\zeta_i \geq \kappa_i$ UAVs traveling across it, it is unsafe to add any more UAVs to that edge. The UAVs that are ready to enter that edge must wait until the current traffic on the edge drops below capacity before they may continue.

The number of UAVs that travel across an edge is a function of the UAVs' starting locations, ending locations, generation rate, and edge costs along the map. Since UAVs are using cost-based planners to find the cheapest path, and agents may only apply an edge cost that is greater than zero, UAVs will plan cycle-free paths. This prevents trivial solutions where UAVs develop 'holding patterns' that do not incur delay but drastically increase the time that it takes for the UAV to get to its destination.

The cumulative delay in the system is the sum of the number of UAVs waiting to move to the next edge in their path at each timestep. The number of UAVs in the system, as well as the paths they take, affects the quantity of delay. The costs assigned to the edges of the graph by the learning agents directly affect the planned UAV paths. Thus, the goal of the multiagent UTM team is to learn costing strategies that minimize the cumulative delay.

In this problem, each agent independently learns a control policy for assigning costs to incentivize UAV planning behavior given the current traffic profile. Training uses the global system evaluation signal, which is the cumulative delay in the system over a single learning epoch. The optimal team policy Π^* is given by:

$$\Pi^* = \underset{\Pi}{\operatorname{argmin}}\, G\left(\Pi, \zeta\right) \tag{3.1}$$

where $G\left(\cdot, \cdot\right)$ is the total system delay, and ζ is the traffic.

3.3.3 Travel Time

Though conflict and delay counting provide two useful metrics for accounting for traffic, they don't necessarily measure potential learned behavioral problems in the system. Both of these promote taking diverse paths in the system, which is desirable for managing traffic, but they do not promote taking *short* paths in the system. This can be managed in some respects because the A^* paths that the UAVs select are guaranteed to include the destination node. However, if replanning is allowed sufficiently often, there is a possibility that a UAV could end in an infinite cycle.

The *total travel time* metric allows us to measure the total time an average UAV exists in the system. There are three modes a UAV can occupy in the system:

Waiting to launch ($\mathcal{W}$): A UAV is generated at a node, but must wait for the first link in its path to become open.

In transit ($\mathcal{T}$): A UAV is currently traveling on a link.

Arrived ($\mathcal{A}$): A UAV has arrived at its destination, and exits the system.

Given the function $t(x)$, which returns the time since introduction in the system until the time of exit from the system (or the time of simulation end), our average total travel time is given by:

$$\frac{t(\mathcal{W}) + t(\mathcal{T}) + t(\mathcal{A})}{|\mathcal{W}| + |\mathcal{T}| + |\mathcal{A}|} \tag{3.2}$$

We consider UAVs waiting to launch onto a link as having entered the system, though they do not contribute to the link capacity. This is why $t(\mathcal{W})$ is included in the count of UAV time. This metric effectively represents the average wait time for any UAV generated into the system, including the grounded time of those that wait initially.

3.4 Physical Space Assumptions

There are many ways to model an airspace, and we explore the effects of learning while modeling it with varying assumptions. In this section, we go over some key physical assumptions that we make when modeling, as well as identify assumptions that we vary

between modeling methods.

Collision Modeling: In this work we do not model collisions between UAVs in a physical space, even for our pixel-based map navigation. There is a lot of research done on exactly how to avoid collision in a one-on-one setting, and therefore we make the assumption that a physical implementation of this algorithm would have some onboard collision-avoidance built into their path-planning algorithm. Furthermore, we assume that this would have little impact on the total path length taken by the UAV, and we therefore neglect to model this interaction in our pixel-level simulations.

Safe Operating Capacities: As a feature of the environment, the airspace will have a certain number of UAVs that it can reliably allow through a space at once. This feature, which we term *capacity*, would rely on many real-world elements; weather, flight speed, sophistication of UAV navigation algorithms, and obstacle topology would all contribute to this safe operating parameter. In this work, because we are largely working on random maps, we use a flat value for safe operating capacity on each link. However if this were applied to a real-world domain, this value would need to either analytically or empirically be determined.

Launch Waiting: There are two approaches we take to introducing UAVs into the system. One mode is having an *immediate launch*, where links have no ability to prevent UAVs from being generated into the system, disregarding the current traffic status of a link. In this way, UAVs are generated directly on a link, and immediately contribute to the traffic in the system. The second mode is where UAVs *wait to launch* until their desired first link is open. UAVs will exist in a state where they do not actively contribute to the traffic in that moment, but they will be the next in line to enter the system. This prevents links from exceeding their designated capacity. This effectively enforces safety requirements in the system.

3.5 Simulation

First, each *fix* will determine on whether to generate a UAV (Line 3). This is different based on whether it has *deterministic* traffic generation or *probabilistic* traffic generation, as described in Section 3.2. It then generates a UAV into the system (Line 4). This has

a destination set either randomly or from a static list, as also described in Section 3.2.

Each agent will then observe their own state (Line 8), and from this state, they will compute their action output based on this state (Line 9). At this point, they modify the graph available to all agents to reflect the new costs across the high-level graph.

Every UAV in the system with then take this newly-changed graph into account, and plan its high-level path across the airspace (Line 13). Each UAV then plans a pixel-level path across the space that has the same sector ordering as given in their high-level path (Line 14). The UAVs then execute their paths (Line 15).

After all UAVs have moved in the system, conflicts are detected by identifying and UAVs that have come within a distance of $d_{conflict}$ with one another (Line 17). This is then counted up for the episode for later use in the fitness evaluation. After conflicts have been identified, any UAV that has made it to its destination is then either removed from the system or rerouted (Line 19), according the the methods explained in Section 3.2. Once this is complete, the number of conflicts are either summed up across all time steps or squared and summed (Line 22).

3.6 Control Methodology

In this section, we discuss our method of hierarchical path planning, with two different map levels for A^* planning. We then analyze how effective this control methodology can be dependinding on the topology of a set of graphs.

3.6.1 Hierarchical Path Planning

In this work, UAVs use two levels of path planners; a sector-level planner and a pixel-level planner. The role of the *sector-level* planner was to find the lowest-cost sequence of sectors to visit according to the costs assigned by the sector agents (this cost-assignment process explained in detail in Section 4.1.1). The A* search was performed over the spatial connection graph, G_h, generated by identifying the neighboring Voronoi sectors shown in Fig. 3.1b. This graph did not consider obstacles across the map, and it was assumed that a clear path existed from points in one sector to the edge of another.

The *pixel-level* planner represented the connectivity of the unit cell graph with the obstacle map, G_{obs}. An 8-connected graph was constructed with unit traversal costs in

Algorithm 4 `Simulate(agents)`

while $t < T$ **do**
  for each $f \in fixes$ **do**
    if `ShouldGenerateUAV(t)` **then**
      generate a UAV
    end if
  end for
  for each $m \in agents$ **do**
    $m.\mathbf{s} \leftarrow countUAVs()$
    $m.\mathbf{a} \leftarrow getActions(m.\mathbf{s})$
  end for
  Turn agent actions into edge costs and update graph G_h
  for each $n \in UAV$ **do**
    $n.path_{high} = sectorPath(G_h, \mathbf{m.a})$
    $n.path_{low} = traversalPath(n.path_{high}, G_{obs})$
    $n.x = executePath(n.path_{low})$
  end for
  $conflicts$ += UAVs within a distance $d_{conflict}$ of others
  if any $UAVs$ have reached goal **then**
    Remove respective $UAVs$ from airspace
  end if
end while
$F = f(conflicts)$
return F

Algorithm 5 `ShouldGenerateUAV(t)`

if probabilistic and random $< p_{gen}$ **then**
  return true
else if deterministic and $t\%\rho = 0$ **then**
  return true
else
  return false
end if

any direction. The connectivity of the graph was further restricted by the high-level path. The pixel-level path was restricted to only those sectors present in the high-level plan. Furthermore, sectors could only be traversed in a particular order and the connectivity of the graph reflected this fact. For example, if a high-level plan specified a path visiting the sequence of sectors $\{1, 5\}$, there would be connections between cells along the border of sectors 1 and 5, but only in the direction of sector 5. Backward traversal into sectors in the high-level map was not permitted.

Path failure occurred when it was impossible to satisfy the high-level plan using a low-level path. This could occur when the path to the specified sector was blocked by an object. In the case that no path was available that satisfied the high-level planner, the UAV did not move in the airspace. Multiple occasions for replanning were available, due to the fact that high-level edge costs changed each time a UAV changed sector membership, so the UAV may have a chance to generate a different high-level plan if a particular one was infeasible.

3.6.2 Impactfulness

In order to influence traffic, a node must be able to pull or push traffic away. This depends on three things: the graph topology, traffic profile, and the costs of the other traffic elements. Here we present an *impactfulness* metric, which identifies the amount that an agent can influence paths in the space. By identifying how much agents within the system can impact generated paths, we can verify that we have an effective control methodology for this system. If none of the paths can be impacted, our multiagent system will not be able to take meaningful actions to influence traffic.

We experiment with two measures of impactfulness; *blocking* impactfulness, and *attracting* impactfulness. Blocking impactfulness refers to the degree to which an edge can impact traffic if it is 'blocked', or has a disproportionately high cost of travel. Attracting impactfulness refers to the degree to which an edge can pull traffic toward itself if it offers essentially free cost of travel.

Assume that each link on the graph has a unit cost. There is an A^* optimal path calculated from each node to each other node in the graph. We call this set of optimal paths assuming unit costs U^*. For each edge i in the graph, we then raise the cost to an arbitrarily high amount. We then calculate the optimal paths for each edge in the

graph, and call this H_i^*. For each edge i we also lower the cost to 0, then calculate the optimal paths for each edge in the graph, and call this L_i^*

There is a 'blocking impactfulness' metric for each edge, B_i, which is based on the difference between the optimal paths with unit costs and the optimal paths with a high replacement cost, divided by the number of possible paths.

$$B_i = \frac{|U^* \cap H_i^*|}{|U^*|} \tag{3.3}$$

This replacement of a link's cost with a high value is given on the left figure in Figures 3.3-3.5 and termed 'high replacement'. This represents the number of paths that would be impacted if a link charged an exorbitant amount to travel through it.

There is also a 'attracting impactfulness' metric A_i for each edge, which is based on the difference between unit cost optimal paths and the lower replacement edge cost:

$$A_i = \frac{|U^* \cap L_i^*|}{|U^*|} \tag{3.4}$$

This replacement of a link's cost with a low value is given on the right figure in Figures 3.3-3.5, and termed 'low replacement'. This represents the number of paths that would be impacted if a link allowed free travel through it.

In some cases, the agents are not able to influence traffic. When a single agent primarily influences the traffic, we expect local rewards or global rewards to perform well. When no agents have influence over the traffic, or they have little influence, we don't expect learning to produce any improvement.

We analyzed three main graphs to see the effect that a learning system could have on the graph. Figure 3.3 shows a graph topology hand-designed graph will 11 nodes, arranged in a way that allows for varying numbers of hops between nodes. If traffic is generated only traveling back and forth between node 1 and node 2, there is a clear shortest-path option between them. The cost can then be adjusted by agents in the system in order to incentivize or disincentize longer route alternatives, moving through the 'suburb' nodes in the grid in the bottom right. We see that when a blocking impactfulness is calculated for this graph, the 'suburb' links have the maximum number of impacted paths, with the maximum being 10 and the minimum being 2 impacted paths. We see conversely that when an 'attracting' impactfulness is calculated, the long direct-

connection link impacts many more paths, with a maximum number of paths impacted of 10, and a minimum number of paths impacted being 4. This indicates that the long link can have a greater effect on the system if it attempts to attract UAVs toward it rather than block UAVs from going through it.

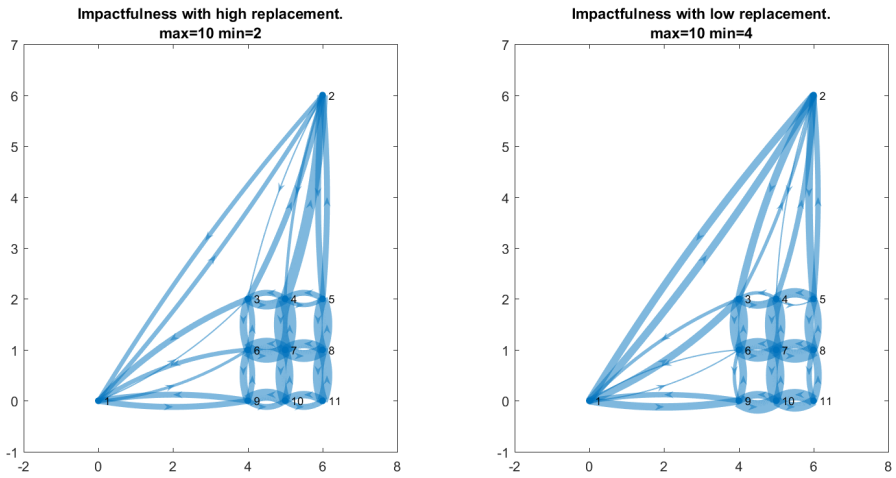


Figure 3.3: Impactfulness metric for test graph topology with 11 nodes, 38 edges. This demonstrates a symmetrical graph with a direct route between two nodes, with many optional alternative routes through the grid graph at the bottom right.

Figure 3.4 shows the impactfulness calculated for a graph topology with 25 nodes. This graph was designed to show paths where there was an area with no alternative routes, which is the middle multi-hop connection between the two grid-style graphs. In both graphs, the connecting links between nodes {2,6,13,20,24} have an impactfulness of zero. This means that a learning agent would have no power in the system, and could set its values arbitrarily high or arbitrarily low and still not impact the traffic flow through the system. This is because if a UAV must travel from one side of the graph to the other side of the graph, it has no alternative but to follow the connecting path between them. Additionally, if a UAV must travel within one of the subgraphs, there is no permissible incentive that would make it cost-optimal to travel through the connecting links.

We also analyzed the the impactfulness of a map that we generated randomly, given Algorithm 3 given in Section 3.1.2. Figure 3.5 shows a map generated using this random algorithm. Largely, for either a high or low cost replacement, we see that most links can

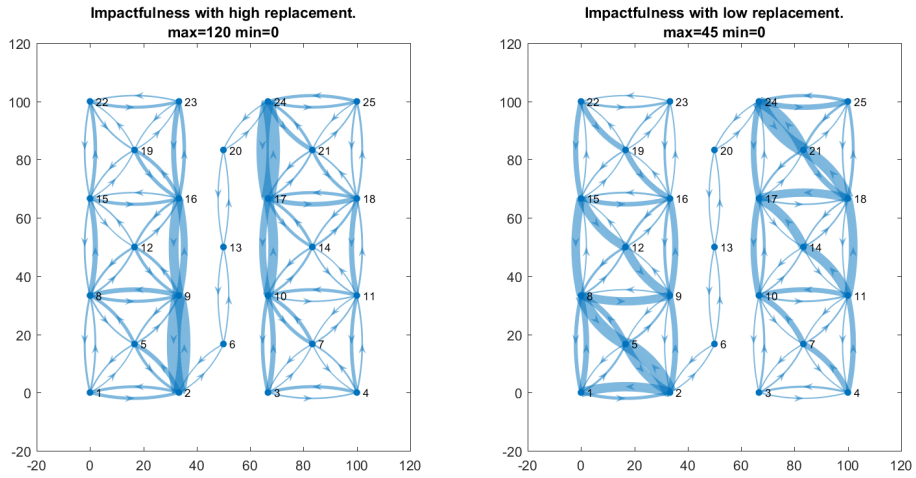


Figure 3.4: Impactfulness metric for test graph topology with 25 nodes, 96 edges. This represents two symmetrical subgraphs connected by links connecting nodes {2,6,13,20,24}. The connecting links can have no impact on the paths planned throughout the graph.

have some impact on the paths in the system. This validates us using a random map generation technique for creating maps, as it indicates that the maps that we generate with this method will likely respond to control by agents setting costs across the map. This indicates the relative ability of different agents to influence traffic based on their position on the graph.

3.7 UTM Agent Control

In this section we discussed how we can model UTM traffic as a system of robots all using A* to plan their routes. We also discussed how a UTM agent can plan a hierarchical route through the space, using A* to plan its route at the sector level rather than only across the pixel level. This setup was in preparation for the introduction of agents to control the system. In this work we use agents to *manipulate the high-level cost map* across which UAVs plan.

This approach is unique in that it does not directly dictate paths for UAVs to take in the system. Instead, it chances the appearance of optimality of paths. Agents learn

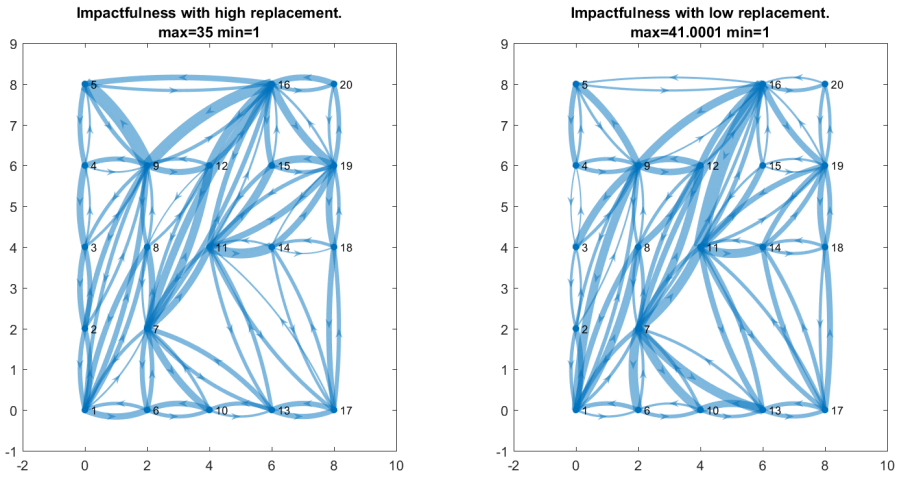


Figure 3.5: Impactfulness metric for test graph topology with 20 nodes. This graph was generated using the *random* graph generation algorithm as defined by Algorithm 3. This shows a fairly even level of impactfulness across the map, although some nodes are not able to change more than one path. This graph indicates that most agents have at least some power to change paths generated in the system.

costing strategies based on current traffic conditions, and get performance metrics as feedback, as discussed in Section 3.3. This enables them to adapt their policies to minimize traffic. Because we change only the appearance of optimal paths, our computational complexity scales with the size of the high-level graph, not with the amount of traffic that goes through it.

In Section 3.6.2 we test the possibility of altering optimal paths in the system by adjusting the cost map. This is the same thing that our agents do; based on a current traffic profile, they adjust the appearance of optimal paths in the system. Our investigation in impactfulness verified that this control approach would be effective for several graphs, as the appearance of optimality can be controlled for many edges of the graph. In the next chapter, we will go over the exact method of our UTM agent construction, including two different approaches to a UTM agent state and action space.

Chapter 4: Learning UAV Traffic Management Agents

Given the high-level graph structure that we have outlined in the previous section, we now must control UAV movement between the sectors to reduce traffic. In this section we introduce a conceptual framework to connect learning UAV traffic management agents into the planning process of UAVs. We then define an agent that controls the apparent cost of UAV movement entering a sector in Section 4.1.1, and we define an agent based on edges of the graph in Section 4.1.2. Finally, we present simulated results and a brief discussion in 4.2.1 and 4.2.2 respectively.

4.1 Agent Definition

We had two definition of agents in our system. One grouped the control of agents around the nodes, and the other looked at individual edges as control elements. This section explains each agent definition, and how they control and learn in the system. Then, we explain how neuro-evolution is implemented using both of these agent definitions.

The sector agent formulation groups edge traversal costing strategies together based on their connected nodes. This may simplify agent coordination, but it also imposes potentially artificial structure onto the problem. We explore the impact of using link agents that control costs only along a single directed edge. This simplifies the agent structure and the individual agent's control strategy. However, the multiagent coordination aspect becomes much more difficult as there are many more agents in the system with a reduced perception of the state space.

4.1.1 Sector-Based Learning Agents

Sector agents are defined by nodes on a graph. They control the cost of traversal of all edges that are connected at the node and pointed at that node. Sector agents assign cost of travel across these edges, discretized by direction, such that each sector assigns a cost of travel in each cardinal direction.

Given the potentially large number of UAVs in each sector, it is impractical to specifically account for the individual states and policies of all UAVs. We therefore define a sector state as the *count* of UAVs traveling in a particular direction. This allows our approach to scale independently of the amount of traffic that passes through it. In this work, we formulate the control policy of each sector agent as a single-layer neural network where the weights of the neural networks are learned using an evolutionary algorithm. The following section describes the neuro-evolutionary algorithm used to develop the control policies for each sector agent.

As in the previous work, sector agents control the cost of travel across a sector of the airspace. In the abstraction of the domain to the graph representation, sector agents are defined as the nodes on the graph and have control over the cost of traversal of all outgoing edges. To maintain generality across all sector agents, regardless of the number of connected edges, all incoming edges are discretized by direction, such that each sector agent assigns a cost of travel for traffic arriving from each cardinal direction.

Sector agents assign costs to the sector-level graph that UAVs use to plan their sector traversal order. Sector agents represent the nodes on the graph and have control over the cost of traversal for all outgoing edges. The state of each sector agent is a discretized representation of all potentially incoming UAVs. Agents consider UAVs that are currently managed by neighboring sectors to be in their state. To maintain generality across all agents, regardless of the number of neighbors for each agent, UAVs are discretized and summed according to the bearing of each neighbor with respect to four cardinal directions.

The control policy of each sector agent is modeled by a single layer neural network. For the neural network controller of agent i, the four-dimensional input state is defined as,

$$\mathbf{s}_{i_{sector}} = [\zeta_{N_i}, \zeta_{E_i}, \zeta_{S_i}, \zeta_{W_i}]\,. \tag{4.1}$$

where ζ represents the traffic on a particular edge. Note that sectors only count traffic traveling *into* their node, so this directional aspect is important. The output action is also a four-dimensional vector specifying the cost of travel for UAVs on outgoing edges, again discretized into each of the cardinal directions,

$$\mathbf{a}_{i_{sector}} = [c_{N_i}, c_{E_i}, c_{S_i}, c_{W_i}]\,. \tag{4.2}$$

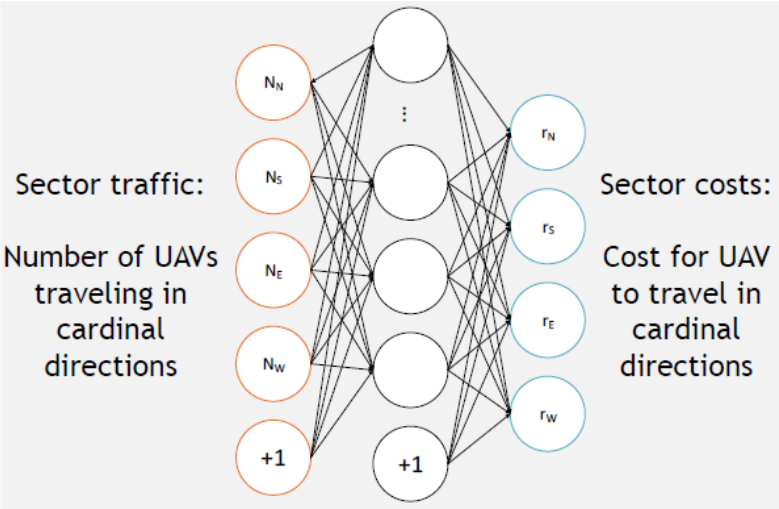


Figure 4.1: The structure of the neural network for sector agents. There are four inputs, one for the traffic count in each direction, and there are four outputs, one to denote the cost the sector will assign to each direction.

where c represents the cost attributed to a particular edge, specifically those that are traveling *into* the node representing the sector. Therefore the neural network has a structure as shown in Figure 4.1

This cost is assigned simultaneously for all agents, and UAVs are permitted to change their high-level plan at any point during the simulation.

4.1.2 Link-Based Learning Agents

Link agents control the cost of travel along each directed edge of the graph. Since there is an agent per link, there is no need to discretize edges into cardinal directions for this agent formulation. Thus, the state space for the link agent is much simpler than for the sector agent and simply consists of the current traffic traveling along the designated edge in the link agent's direction of control. This is ζ_i for link i, such that,

$$s_{i_{link}} = \zeta_i. \tag{4.3}$$

Furthermore, the agent's action is to assign a single cost to its designated link,

$$a_{i_{link}} = c_{i_{link}}. \tag{4.4}$$

where $c_{i_{link}}$ is the cost assigned to a particular link i. This is broadcast to the UAVs as the cost of traversal on each edge of the abstract graph.

4.1.3 Neuro-Evolution

This is suitable for the UTM problem considered in this work since it allows a simplification of the system state to include local information available to each agent; a count of UAVs traveling into the sector.

The agent takes this state information, which is given as a count of UAVs traveling toward the sector in the case of sector agents (explained further in Section 4.1.1) or on an individual sector transition (as explained in Section 4.1.2). A forward pass through the agent's neural network maps the given state to a cost that will be applied to each of the directed edges in the graph. The edge costs are broadcast to UAVs in the airspace that then recompute their flight trajectories to reduce their individual cost of travel.

The weights of the neural network controllers of each sector agent are trained using a cooperative coevolutionary algorithm (CCEA). This allows multiple populations to evolve in parallel such that each population develops a policy for a single sector agent in the UTM system. In the following experiments, there are M coevolving populations of NN controllers, one for each sector agent and each with a population size of k. For mutation of the NNs, 50% of the network weights are chosen at random and values drawn from a normal distribution (zero mean and unit variance) are added to the weights.

4.2 Experimental Results for Learning Hierarchical UTM

We tested our UTM learning algorithm on a simulated urban airspace derived from a 256×256 unit cell map of San Francisco, which is shown in Fig. 3.1a. We discretized simulation timesteps into 1s time intervals with each learning epoch defined as an evaluation of each random team in the population over 100 simulated timesteps.

We divided the tested airspace into 15 Voronoi partitions with each sector controlled

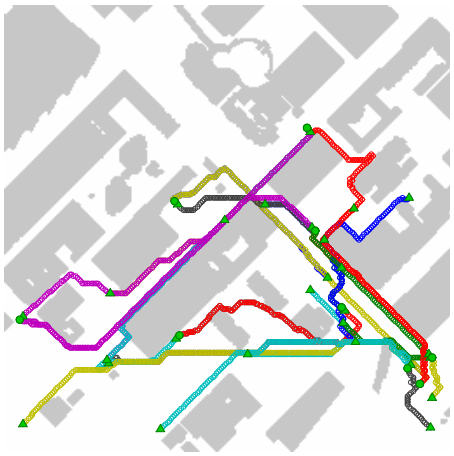
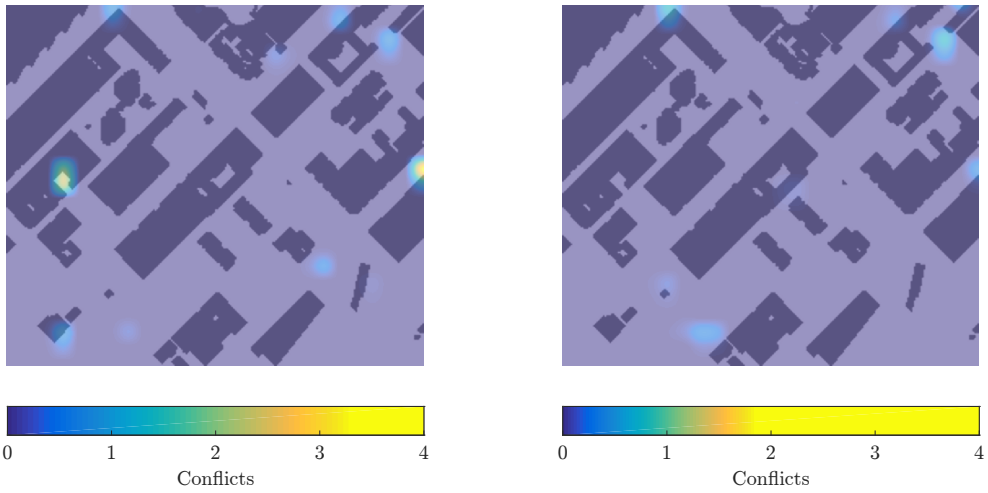


Figure 4.2: Initial planned paths for 27 UAVs in the airspace travelling from start (solid green triangle) to goal (solid green circle) locations. It can be seen that high levels of congestion may be expected in some thoroughfare regions of the airspace, such as the bottom right area where there are a group of close fixes.

by a single agent in the UTM system. Voronoi partitioning was hand-selected to place sector agents in locations where congestion was expected due to the obstacle layout of the region. The airspace sectors are shown in Fig. 3.1a by the dashed lines.

At the start of learning, each sector agent initializes with a population of $k = 10$ random NN control policies. As described in Equation (4.1), UAV headings are discretized to the nearest cardinal direction such that at each timestep and in each sector, the total north-, south-, east- and west-bound UAV traffic is tallied to form the input state vector of the NN policy. The output action of each agent, that is, the costs to travel in each direction as shown in Equation (4.2), were restricted to lie within $c = [0, 1]$.

At each timestep and at each start location, or *fix*, a new UAV is generated with probability $p_{gen} = 5\%$ with a randomly-selected goal *fix* and executes an A* plan over the sector-level graph and over the pixel-level graph. Figure 4.2 shows an example of planned paths over the pixel-level graph. This value was selected to produce about 100 UAVs in the airspace in a single run, demonstrating the effectiveness of the algorithm in a high-traffic situation. Once a UAV reaches its goal location, it disappears from the airspace. Note that in this implementation the *fixes* that generate UAVs across the airspace are not necessarily colocated with sector centers.



(a) (b)

Figure 4.3: Change in congestion observed over one run of the *linear* conflict evaluation trials. The overlaid heat map shows the number of conflicts experienced in an area for (a) the best evaluation trial for agents with random initialized sector travel costs and (b) the best evaluation trial for agents with evolved sector travel costs after 100 epochs. The overall number of conflicts has reduced with some high congestion regions removed completely.

In the following experiments, each UAV traveled at a constant speed of 1 cell per timestep and had a conflict radius $d_{conflict} = 2$ unit cells.

4.2.1 Results

We performed three sets of simulation experiments, the first tested the proposed UTM learning algorithm trained using the total number of conflicts in the airspace (*linear* conflict evaluation). The second set of trials tested the same algorithm, however in this instance, policies were evaluated according to the sum of squared collocated conflicts in the airspace (*quadratic* conflict evaluation). As a baseline, we performed a set of simulations using uniform costs for travel between the sectors. This meant that the UAVs took the path that traveled through the smallest number of sectors. Each set of experiments contained 20 runs of 100 learning epochs. The algorithm was implemented

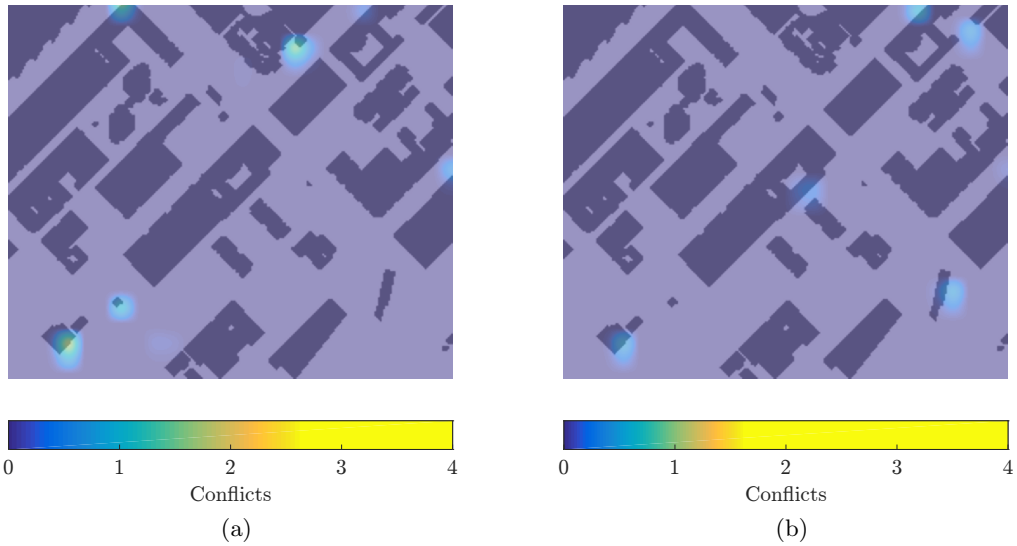


Figure 4.4: Change in congestion observed over one run of the *quadratic* conflict evaluation trials. The overlaid heat map shows the number of conflicts experienced in an area for (a) the best evaluation trial for agents with random initialized sector travel costs and (b) the best evaluation trial for agents with evolved sector travel costs after 100 epochs. As with the *linear* trials, the overall number of conflicts has been reduced.

in C++ with the high-level and low-level planners using the A* search algorithm provided by the Boost Graph Library [75]. Planners re-planned whenever the cost map generated by the agents changed.

Figure 4.3a shows the congestion observed in the first learning epoch of one *linear* trial. Sector agents initially assign travel costs randomly, so there are areas of high congestion that occur. Figure 4.3b shows the performance at the end of evolution. Comparing Fig. 4.3a and Fig. 4.3b we can see a reduction in the number of conflicts.

In a realistic UAV routing scenario, the number of close calls doesn't necessarily matter as much as the severity of the congestion in the system. For example, if two UAVs get within a dangerous distance of one another, a backup safety mechanism may be able to divert one and avoid a collision. However, if more UAVs get within dangerous distance of one another, the backup safeties must then deal with the cascading effects of the collision-avoidance maneuver. This could lead to a failure of the lower-level collision-avoidance systems if the conflict density is too high. To address this fact, we also tested

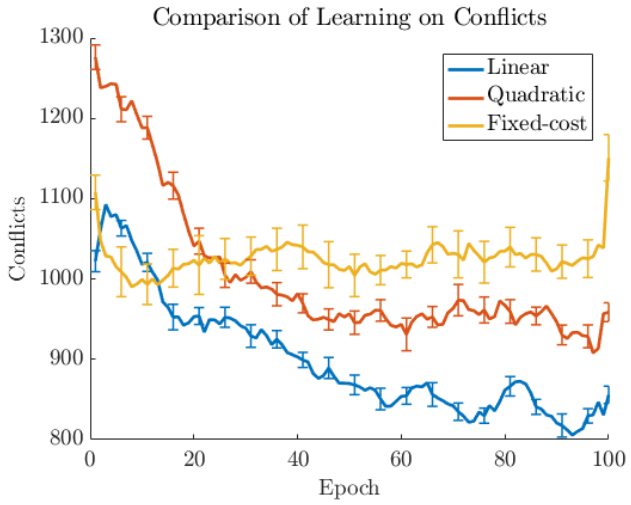


Figure 4.5: Comparison of conflict occurrence over epochs using fixed costs versus evolved costs with two different fitness functions. The linear performance metric reduces conflicts substantially better than the quadratic method, although both outperform the fixed-cost method.

a *quadratic* fitness evaluation function that summed the squared number of conflicts for *each cell*, which correlates to a pixel value on the map. This attempts to decrease the likelihood that there will be large numbers of conflicts that the UAV's low-level planner cannot handle. This resulted in a larger reduction in the *severity* of congestion across the map. By comparing Fig. 4.4a and Fig. 4.4b we see a larger reduction in the *severity* of congestion across the map, that is, there are fewer conflicts in each hotspot.

We also quantitatively compare the reduction of conflicts using evolution with the *linear* and *quadratic* fitness evaluations to the conflicts arising in the fixed-cost map. We see in Fig. 4.5 that the number of conflicts using the fixed-cost map is higher than the conflicts using the learned costs after evolution has been performed for 100 epochs. The number of conflicts in the baseline comparison algorithm remains high, with an overall average of 1023 conflicts, while the neuro-evolutionary system is able to reduce the number of conflicts seen in the system. Note that the evolution process does not monotonically decrease the occurrence of conflicts due to the fact that there is randomness in the population generation as well as in the goal selection of the UAVs. However evolution still significantly decreases the occurrence of conflicts in the system, partic-

ularly compared with the fixed-cost method. Comparing to the fixed-cost method, we see an average of 856 conflicts at the end of learning with the *linear* fitness evaluation, which represents a 16.4% reduction in total conflicts after 100 epochs using the linear fitness evaluation with evolution. Learning with the *quadratic* fitness evaluation does not outperform the *linear* fitness evaluation in terms of total *number* of conflicts, however this is to be expected since it is not what the fitness function uses to determine survival. Nonetheless we see an average performance of 950 conflicts after 100 epochs, giving us a 7.1% improvement over the fixed-cost method. Our heat map results indicate additionally that these are less severe than the conflicts that are present in the learning that does not consider co-location of conflict events (Figure 4.4).

In a realistic UAV routing scenario, the number of close calls doesn't necessarily matter as much as the severity of the congestion in the system. For example, if two UAVs get within a dangerous distance of one another, a backup safety mechanism may be able to divert one and avoid a collision. However, if more UAVs get within dangerous distance of one another, the backup safeties must then deal with the cascading effects of the collision-avoidance maneuver. This could lead to a failure of the lower-level collision-avoidance systems if the conflict density is too high. To address this fact, we also tested a *quadratic* fitness evaluation function that summed the squared number of conflicts for *each cell*. This attempts to decrease the likelihood that there will be large numbers of conflicts that the UAV's low-level planner cannot handle. In Fig. 4.4b we see a larger reduction in the *severity* of congestion across the map, that is, there are fewer conflicts in each hotspot. However Fig. 4.5 shows that we do not outperform the linear fitness evaluation in terms of total *number* of conflicts, because this is not what the fitness function uses to determine survival. Nonetheless we see an average performance of 950 conflicts after 100 epochs, giving us a 7.1% improvement over the fixed-cost method.

The number of UAVs in the system varied during the simulation. Because of the probabilistic generation process, there could be a variable number of UAVs present in the system, but generally this number was close to 100. If the generation rate is lowered, it is possible to achieve zero conflicts in the system without learning intervention, with the tradeoff of severely reduced throughput. If the generation rate is increased, it is also possible to saturate the system with UAVs such that a learning policy cannot effectively reduce the number of conflicts. However, these extremes would not demonstrate realistic scenarios, and would not show the improvements that the algorithm offers. We explore

the scalability of this approach in the next section.

4.2.2 Discussion

The results presented in this work show that a distributed UTM system can learn appropriate costs to apply to UAVs traveling in the airspace to incentivize implicit cooperation between the UAV planners. Using our method, we achieved an 16.4% reduction in conflicts compared to the baseline method with uniform sector costs. We also showed a reduction in severity of conflict occurrences using a quadratic fitness function. The agents were able to take in directional sector congestion information and appropriately weight the cost of travel to promote safety in the system. It is also worth noting here that agents do not require a model of the available airspace or knowledge of the obstacles in the sector, the number of conflicts experienced in the sector was sufficient information by which to evaluate the performance of the current policy.

This structure gives a method by which neural networks may be trained for managing conflict occurrences through an obstacle-filled area. Because we introduced random traffic throughout the run, the policies generalize to a variety of different traffic scenarios. The computation time in the evolved policy is therefore a combination of the time to calculate the output of a neural network. The UAVs then independently calculate their trajectories through the space with this information.

The ability to manipulate high-level planners allows us to reduce occurrences of potentially dangerous congestion in the system. UAVs in the real world can handle encounters with other UAVs using low-level collision avoidance procedures, but by reducing the congestion in the airspace at a high level we can permit safer travel by avoiding many of these conflicts before they occur.

This work forms an important step toward the formulation and exploration of the UAV Traffic Management (UTM) problem. Handling the challenges associated with free flight in an obstacle-filled environment will become critical as UAVs become more prolific in the airspace. Shortest-path algorithms for UAV routing may lead to unsafe UAV density. Intelligent high-level coordination must occur in conjunction with low-level collision-avoidance in order to ensure safety in the airspace.

Chapter 5: Learned Abstract Policies Applied to Real Robots

Our approach has the benefit that it is platform-agnostic, meaning that it should be able to be used by any robot navigating the space that implements a cost-based planner. Our routing method is specifically designed to route robots through a physical space. We wish to show that our evolved policies will provide incentives for robots to take diversified paths, but still get to their destinations. In this section, we demonstrate an end-to-end application of our multiagent system, demonstrating its real-world feasibility.

We set up a traffic management system using a ROS framework to interface with Pioneer-3DX ground robots. We trained neural network agents and to reduce traffic on our high-level simulator, and then saved the converged neural network weights. We then wrote a ROS node to take in the number of robots in a particular area of the space, feed it to its respective agent, and this agent will adjust a broadcast map. We then tested this in a high-fidelity simulator, as discussed in Section 5.3. Then we applied this ROS framework on robots in a physical maze, which we will present in Section 5.4.

5.1 Neural Network Controller Evolution

Our research into traffic routing through a simulated airspace suggests that we can reduce traffic with many robots in the system. However, we can also develop policies for smaller numbers of robots. In order to train the neural networks, we generated four robots in the system, and had them randomly select destinations. When they reached their destination, they rerouted to another node in the simulation. This caused continuous traffic for the duration of the simulation. For training the neural networks, we used a delay-based performance metric.

Robots provide the underlying dynamics of the system on which our controller agents act. Because our approach focuses on *platform-agnostic controller agents*, and we are routing on a 2-dimensional plane, we substitute pioneers for UAVs in our Gazebo experiments. UAVs may have a different speed or reaction time, but the substitution of Pioneers for UAVs does not change the core density management problem. Toward this,

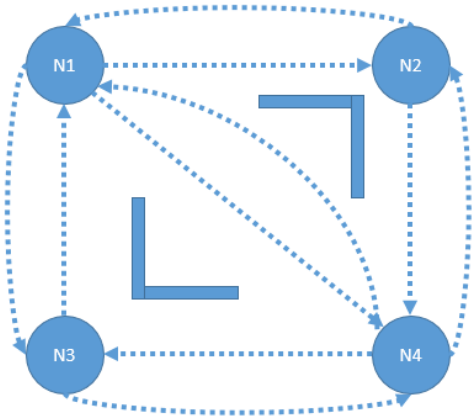


Figure 5.1: The high-level map used for training the neural network policies the Pioneer experiments. Shown for reference are barriers (square brackets) that would create this high-level topology. For training, all traffic moves between N1 and N4, requiring controller agents to make non distance-optimal routes look attractive to some traffic.

we set up a physical room with obstacles and four points of interest (Figure 5.4). This graph structure has 5 physical edges; 4 edges at the edges of the room, and a fifth forming a connection between the two obstacles in the room. This corresponds to 10 links. One pioneer running moved between these points of interest in order to build a map of the room. This map was then copied over to the other robots.

The map was used to create 10 link agents in a simulation, with graph topology identical to the physical system. Neural networks then evolved across 200 epochs on this small system. Figure 5.2 shows the reduction in delay over 100 epochs of training, for 10 statistical runs. A difference between this simulation and the scalability simulations were that the traffic was kept in the system continuously rather than removed and re-generated randomly, and assigned new destination points when they arrived at their target. Additionally, the capacity of each link in the system was set to 1, as the corridors in the physical space could only safely accommodate one Pioneer at a time.

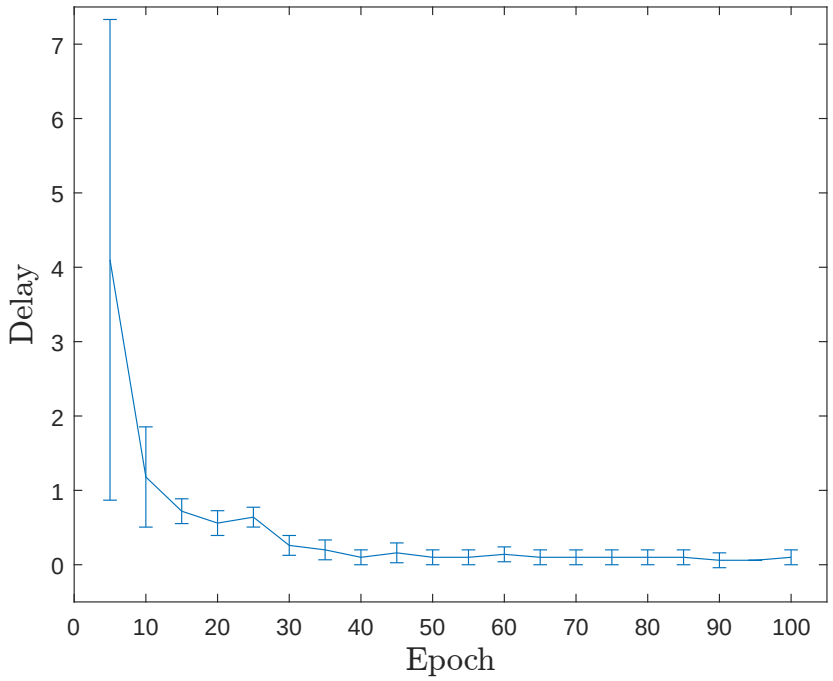


Figure 5.2: Learning performance using a graph with 4 nodes and 10 edges. 10 runs ar shown for statistical significance. Agents are able to completely eliminate delay in the system.

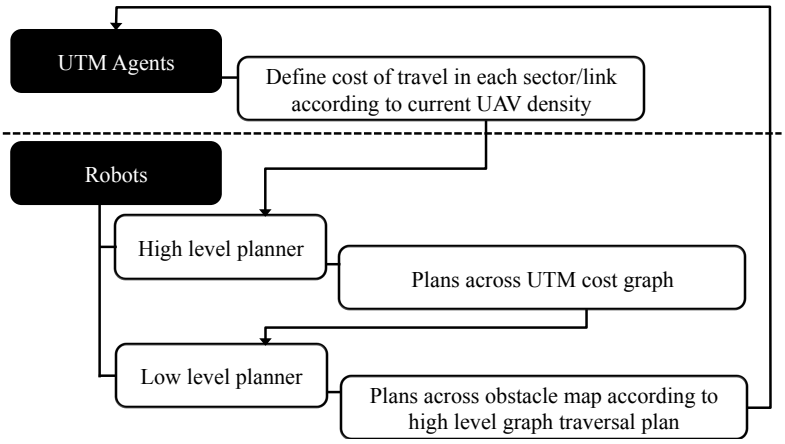


Figure 5.3: System architecture of the ROS experiments. The UTM agents publish a cost graph over which the robots plan their high level paths. The low level planning and navigation is handled by the standard ROS navigation packages.

5.2 ROS UTM Architecture

We use the Robot Operating System (ROS) architecture to add our UTM algorithm onto a real robot platform [70]. ROS offers a communication protocol wherein computational entities called *nodes* communicate with each other by publishing strictly-typed messages. ROS nodes can also subscribe to other nodes in the system, which passes this information through the computational step.

This framework allows for a large degree of abstraction away from fundamental hardware programming. Because of this level of abstraction, as well as a high level of community support, the ROS website is able to offer a wide variety of highly modular packages to handle standard algorithms. Additionally, ROS packages are available for many standard hardware elements. The basic controllers for the Pioneer-3DX ground robots used in our experiments, as well as the sensors used, are available without needing additional programming.

5.3 Gazebo Experiments

We tailored this approach to coordination of robots in a physical space. The target of our evolved control policies are to incentivize robots to take diversified paths but still

get to their destinations. In our scalability tests, we demonstrate that we can learn to reduce delay in a large system with many robots moving around. However we also must demonstrate that this could apply to a smaller physical system with imperfect information. Toward this end, we use Gazebo, a high-fidelity physics simulator.

Figure 5.3 describes the system architecture. UTM agents, which are a set of neural networks, take in information about the positions of the robots in the system. These UTM agents then define the cost at the high-level connection map. Robots then compute A* paths using their high-level planners to plan across the UTM cost graph. Robots use this cost graph to then plan across the obstacle map. This is also shown in the RQT graph in Firuge 5.8.

Our implementation creates artificial walls between the spaces to enforce travel through the optimal link path. We do this by first having the robot calculate the its optimal high-level path, then a node takes this high level path and modifies the walls perceived by the robot. This enables us to use pre-built waypoint commands in ROS without directly modifying them.

This domain is extremely constrained for the set of pioneers. There are many points at which the pioneers can get 'stuck' because they are too big to physically pass each other in the space. The simulation that we train our agents on is more permissive of allowing traffic through the space, because there are two directional links connecting each point, as shown in Figure 5.1. However we are still able to see a incentive strategy that reduces the congestion in the space. A simulation that is closer to the true constraints of the test problem would likely provide better learned results.

5.4 Real Routing in a Maze

We used Pioneers running *rocon* [83] to demonstrate the physical end-to-end application of this work. Pioneers received a *rtm_graph* object that allowed them to plan shortest paths across the space. The pioneers communicated their positions on the 'airspace graph'

Map Design: We had a directional graph, which means that we need the corridors to be able to accommodate two robots at once. We selected a distance of at least $1m$ to design the map. We constructed a map for the robot (Figure 5.6) which would result in

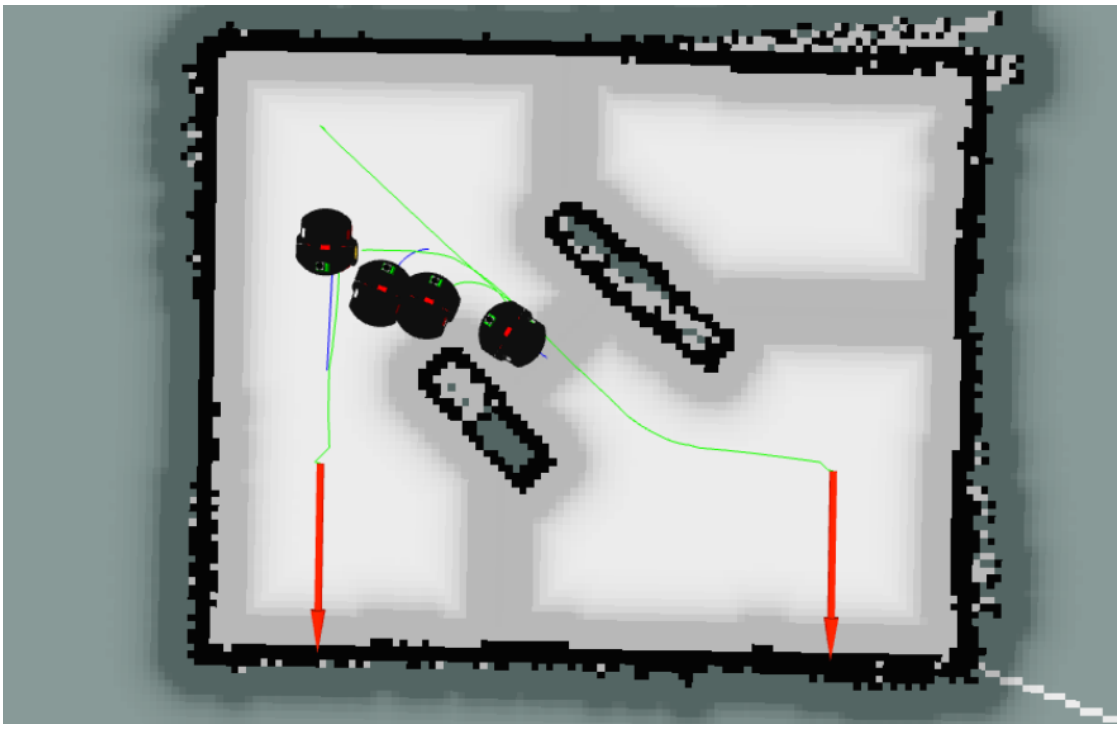


Figure 5.4: A simulated Gazebo environment with two obstacles. This map was created by driving a Pioneer around a real environment with two square obstacles, although multiple robots are introduced in simulation. This graph has four nodes, each at a corner of the graph, and the arrows denote current target points of the Pioneers.

the same topology that has been trained on.

Traffic Design: In order to demonstrate the impact of our algorithm, we set it so that all traffic would move from the top left node to the bottom right node. This makes the direct route distance-optimal, but if all robots take this path there will be congestion. The high-level graph is shown in Figure 5.1

For this section, we will manage ground robot traffic rather than UAV traffic. Therefore here we will refer to our traffic management multiagent system as *robot traffic management* (RTM), rather than UAV traffic management (UTM), which is used for the other sections of this work.

Figure 5.6 shows the SLAM map that was generated by driving a robot through the

Figure 5.5: A Pioneer-3DX robot.

environment and this map was shared across each Pioneer robot. We used the *amcl* [41] localization package to feed into the ROS navigation stack. This allowed us to localize from any point in the room. We first moved the robots a short distance to allow them to localize within the saved map. Nodes were located at the corners of the map. These were given as destination points for the robots to route to. Robots would autonomously sense when they had reached a waypoint, and would choose a new waypoint at random from one of the other nodes.

Robots coordinated using a central computer as the base of communications. Each robot was initialized using a script that began robot networking and connected to the hub. To initialize the planning, a (unit) cost graph was published to all robots by the RTM. An initial waypoint was then sent to the each robot, which defined which link it belonged to. Membership on a link was defined by calculating which sector center was closest to the originating point, and then which sector center was closest to the final point. This created an online Voronoi partitioning of the space to define the graph. The traffic on this link was communicated back to the RTM agent node in ROS. This allowed the RTM agent node to adjust the cost map and re-broadcast it to all of the robots.

The robots ran an A* planning algorithm across the 4-node 10-link high-level map.

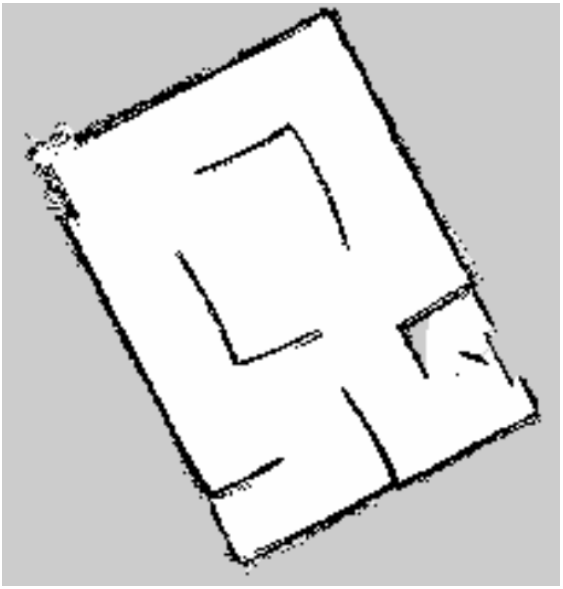


Figure 5.6: The map generated from running SLAM across a room. This was shared among all the robots.

They used this algorithm to plan a trajectory across space designed for certain sectors. Artificial 'walls' were then constructed in their map to force the low-level path-planning in ROS to find a path that adhered to the sector ordering recommended by the A* planning algorithm. After walling off the sectors that were not in the given sector ordering, the robots then ran Dijkstra's algorithm to plan a path across the nearby space. Robots moving using paths planned under this architecture are shown in Figure 5.7.

Our setup allowed the robots to observe cost maps with higher costs on occupied links, which impacted their path planning decisions. However, hardware testing also demonstrated the necessity to combine this approach with accurate low-level sense-and-avoid systems.

5.5 Discussion

In this section we have demonstrated an end-to-end application of our algorithm on real robots. This demonstration is important to this work because it shows the feasibility of our approach to a real system. The neural network, trained prior to the application

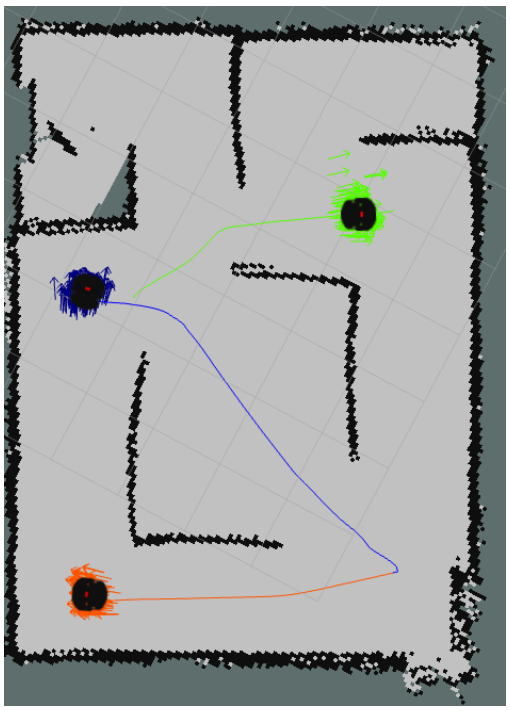


Figure 5.7: Robots moving between waypoints in the space using the RTM framework. This map was generated by driving a robot around a physical obstacle maze. Robots then used the *amcl* package to localize within this maze. The robots then autonomously moved throughout the maze, planning their routes considering the high-level RTM graph.

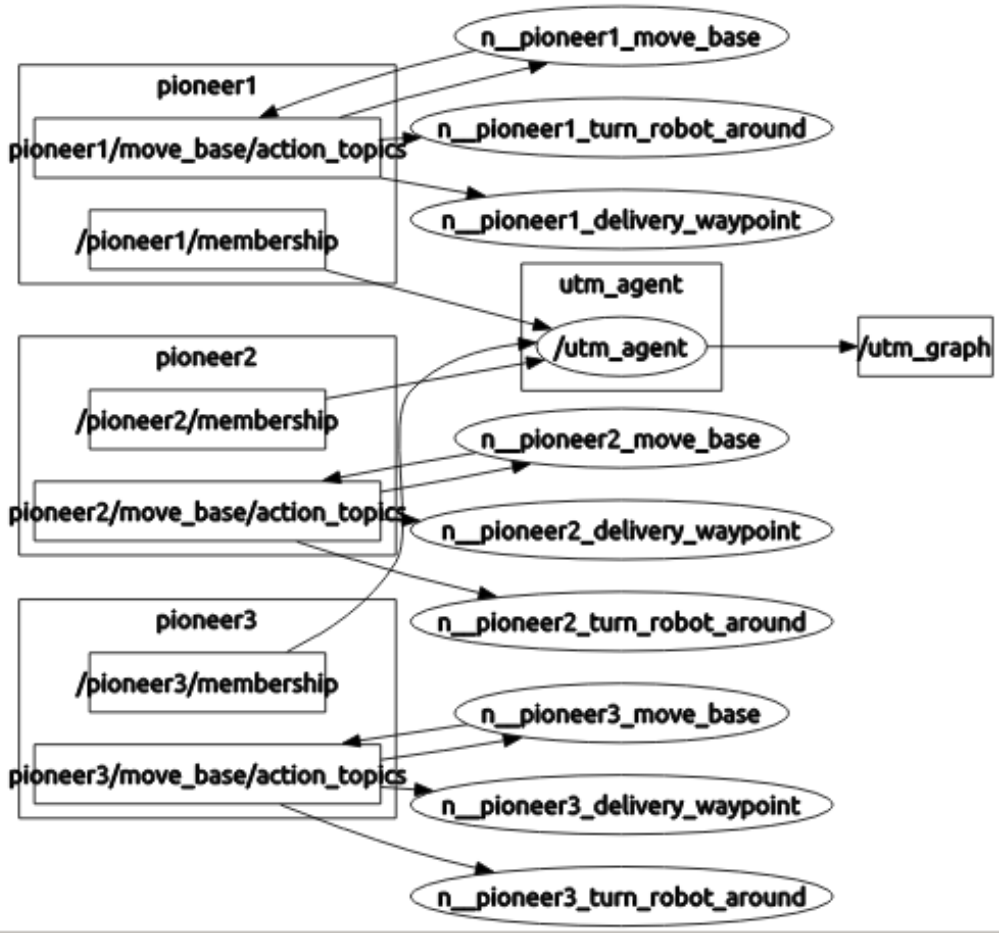


Figure 5.8: RQT graph for three pioneers in a Gazebo simulation. Pioneers communicate their position in the map with respect to which agent controls their motion (the /membership) topic). This traffic information is then processed by the utm agent, which is a group of evolved neural networks. This information is then used to update the /utm_graph topic, which is the high-level cost map set by agents in the system. This is then used to update the appearance of walls for each robot topic according to its A* optimal plan.

here on a simulated traffic profile, can be used by constructing a ROS take in traffic information and use the trained neural network algorithm to assign weights.

We have also demonstrated one way of implementing a hierarchical control structure in a real robot system. When a robot plans its path across the high-level map, we can introduce artificial walls around the edges of sectors that do not fall into the high-level path plan. This is a small modification to our navigation method, and allows us to still take advantage of many of the navigation functions available through the ROS community.

We demonstrate also that the communication requirements between robots is practical. Robots must be aware of and communicate their location to UTM agents. This is not an undue restriction, particularly because of the assumption that in an aircraft-based system, all elements would be equipped with ADS-B technology [1]. This enables accurate calculation of a traffic profile, which serves as an input to our neural networks. Self-reporting of position based on localization in a map additionally means that there are no external structures, such as Vicon, required for monitoring robot movement through the space. This puts outdoor experiments within the realm of possibility.

We additionally show that we can manage manage traffic even on a small scale with this approach. Our Gazebo results show traffic routing for 4 robots under an ideal sensing environment. Gazebo allowed us to test the predicted motion of a ROS infrastructure without using a physical system. Gazebo also gave us insight into the structure of our simulated network structure; in many cases, robots would interact in the space while trying to pass one another. Because these robots were considered to be on separate 'links' they were not combined into the same traffic count. In the Gazebo simulation, this frequently meant that robots would get stuck in too-small corridors while attempting to pass one another. A way of solving this is to either modify the graph so that it has only one directional link through a space, or to apply this same topology to a space that is large enough to accommodate robots passing one another.

Our real robot experiments showed this control structure working for 3 robots moving through the space. In this case, we had sufficiently large passages to allow movement of robots past one another, as would be allowed in the high-level graph topology. Additionally, we observed that UTM agents were able to take feedback from localized robots moving in the physical space, and compute a recommended cost map. The only downside of these experiments was that many of them were truncated due to sensor error.

Sensors were placed high on the robot, and therefore did not sense the main body of other robots.

Chapter 6: Fast UTM Learning

We develop an abstraction of the UTM problem that preserves the core learning elements for agents controlling costs on a sector-level graph (Section 6.1). We then present experimental results for extensive testing of the UTM learning system under different traffic conditions, as well as different agent formulations, in Section 6.2. Finally, we discuss the implications of these results and some trends that we observe in Section 6.3.

6.1 Abstraction of the UTM Coordination Problem

In the problem we consider here, we remove the pixel-level planning across the obstacle map. We instead introduce an abstraction of these dynamics by having a 'capacity' parameter for each edge on the graph, as illustrated in Figure 6.1. This abstracts away the dynamics of traversing across the map because we assume that conflicts are a function of the density of the UAVs in a particular area. We preserve the core routing problem, with a significant reduction in simulation time.

We represent the airspace using a graph, where UAVs generate at (and travel to) nodes of the graph, and graph edges represent their physical path through the airspace. We construct a planar graph with random connections between these nodes and edges. Nodes have a physical xy-location in the world, and the time that it takes to travel across the edge is the Euclidean distance between the connected nodes. A UAV occupies some amount of space while it is traveling across an edge, and this contributes to the traffic ζ_i on that edge.

In this work we refer to agents controlling the cost of travel on edges surrounding a node as *sector agents*. Agents that control the cost of travel across an individual directed edge are *link agents*. Figure 6.1 shows the two multiagent formulations.

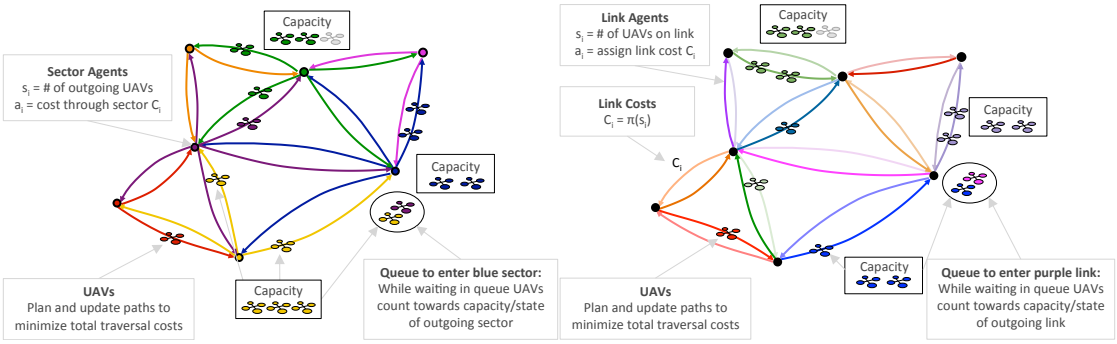


Figure 6.1: The sector agent (a) and link agent (b) problem formulation. Sector agents control groups of edges rather than a single edge. We divide these groups of edges into four cardinal directions, and determine traffic and costs based on this discretization.

6.2 Experimental Results for Varying UTM Traffic Profiles

There are two definitions of agents that we test: *sector agents* (Section 4.1.1) and *link agents* (Section 4.1.2). Simulation timesteps were discretized into 1s time intervals with each learning epoch defined as an evaluation of each random team in the population over 100 or 200 simulated timesteps. The algorithm was implemented in C++ with the UAVs using the A* search algorithm provided by the Boost Graph Library. We repeated each experiment 10 times for statistical significance.

Figure 6.2 shows the delay reduction in a system with 20 different nodes, which corresponds to 102 different edges. We tested both sector agents and link agents in this domain, with a generation rates of 20 UAVs every $\{10, 20, 50\}$ s. This means that in the high-traffic scenario we could have as many as 400 UAVs generated. Both agent formulations were able to reduce the delay across the system as learning progressed. As the generation rate increases, we see a superlinear relationship with number of delays accumulating in the system. In a system with lighter traffic (such as a system that generates new UAVs every 50s), the delay can be reduced to zero. In scenarios with heavier traffic, both the sector agents and link agents produced significant delay reductions. However, the minimum delay still converged to a nonzero value, suggesting an upper bound on the system throughput with respect to the traffic density.

The number of timesteps for each learning epoch also plays a role in the delay per-

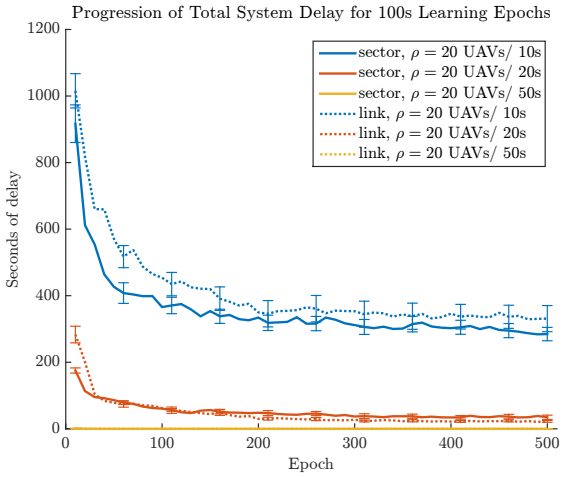


Figure 6.2: Sector and link agent performance on a map with 20 nodes, 102 edges. We tested three different generation rates for learning epochs of length 100s. The low-traffic scenarios ($\rho = 20UAVs/50s$) is zero in most of the graph.

formance. When the steps for simulation increase by a factor of 2 there is a nonlinear increase in the delay. This is because there is nonlinear delay in the system that rolls over and continues to cause congestion in the system. The results in Figure 6.3 demonstrate that the sector and link agents are capable of learning policies for reducing system delay to zero for a UAV generation rate of 20 new UAVs every 50s. However, traffic congestion compounds for both the $\{10, 20\}$ s generation rates.

Table 6.1 shows the percentage delay reduction across several different maps and generation parameters. Percentage improvement compares the first epoch's system delay performance to the last epoch's performance. The improvement varies greatly with the length of the learning epoch and the generation rate. These results also include analysis for a sector agent learning system with 100 nodes and 580 edges to compare the effect of a finer discretization of the sector space versus applying a similar number of link agents along graph edges. The traffic generation rate in the 100 node graph was higher for two of the three tested cases (100 UAVs every $\{10, 20, 50\}$ s) and the reduced percentage improvement for those traffic profiles reflects this. However, comparing the 100 UAVs per 50s case to the link agent scenario with 20 UAVs per 10s, which both generate the same total number of UAVs, it is clear that the link agent formulation provides a

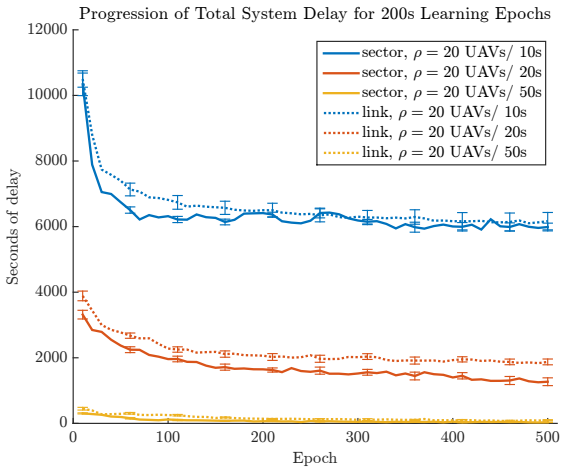


Figure 6.3: Sector and link agent performance on a map with 20 nodes, 102 edges. We tested three different generation rates for learning epochs of length 200s.

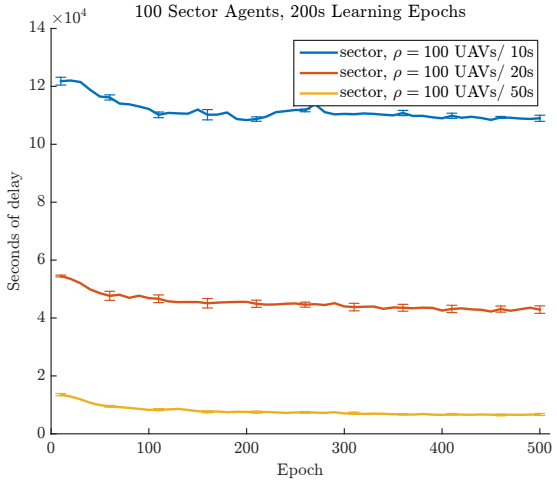


Figure 6.4: Sector agent performance on a map with 100 nodes, 580 edges. We tested three different generation rates for learning epochs of length 100s.

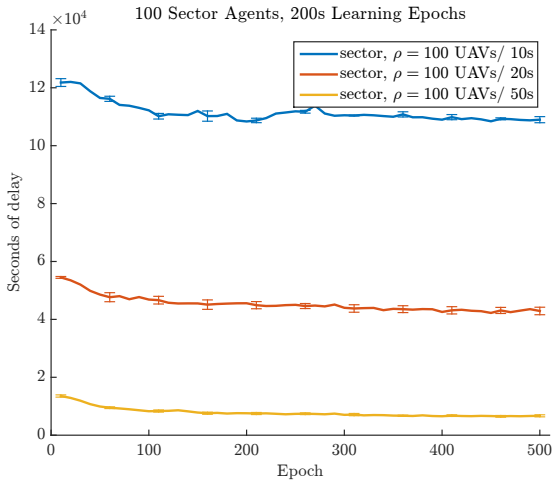


Figure 6.5: Sector agent performance on a map with 100 nodes, 580 edges. We tested three different generation rates for learning epochs of length 200s.

greater improvement in the overall delay reduction.The link agents improve system delay by $\{67.78\%, 41.47\%\}$ for the $\{100, 200\}$ s learning epochs, respectively, while the sector agent formulation provides a delay reduction of $\{53.37\%, 24.42\%\}$.

The results presented in this work show that a distributed UTM system can learn appropriate costs to apply to UAVs traveling in the airspace to incentivize implicit cooperation between the UAV planners. Using our method, we achieved a 68% and 67% reduction in delays for a high-traffic scenario using sector and link agents respectively. The agents were able to take in directional sector congestion information and appropriately weight the cost of travel to promote safety in the system.

6.3 Discussion

In order to do more extensive testing of our UTM system, we had to develop a control structure that did not require motion across a low-level map to perform simulation. Because the UAV motion was divided hierarchically, we abstracted the lower-level motion away by restricting edge *capacities* rather than counting up spatially close interactions between UAVs. This allowed us to massively reduce computation, as A* no longer needed to be computed by agents for the lower-level graph.

Table 6.1: Comparison of performance for different agent control, graphs, traffic generation rates, and epoch length (T). Percentage improvement is calculated between the delay experienced in the first and last learning epoch.

Agent	Graph	ρ	T	Improvement
Sector	20 Nodes 102 Edges	20 UAVs/10s	100s	68.93%
			200s	41.18%
		20 UAVs/20s	100s	81.39%
			200s	61.71%
		20 UAVs/50s	100s	100%
			200s	87.28%
	100 Nodes 580 Edges	100 UAVs/10s	100s	22.16%
			200s	12.11%
		100 UAVs/20s	100s	29.95%
			200s	21.80%
		100 UAVs/50s	100s	53.37%
			200s	24.42%
Link	20 Nodes 102 Edges	20 UAVs/10s	100s	67.78%
			200s	41.47%
		20 UAVs/20s	100s	92.27%
			200s	53.12%
		20 UAVs/50s	100s	100%
			200s	75.44%

This fast simulation enabled us to investigate the effects of two different parameters; the traffic profile and the agent structure. We were able to substantially increase the tested graph size by generating a random graph with up to 100 different nodes and 580 edges.

In terms of a percentage of traffic reduction over the beginning traffic, the lower the generation rate was typically the better the reduction of traffic. In two cases (sector and link formulations on the 20-node graph with UAV generation every 50 seconds) all traffic was able to be reduced in the system. However, looking at the absolute reduction of delay rather than as a percentage, we see that the more traffic that exists in the system the more impact our algorithm is able to have in reducing it.

This set of experiments suggests that our approach to learning traffic reduction can apply in cases where there is high amounts of traffic as well as low. Our agents can learn to reduce traffic in the system even in systems where a routing solution cannot eliminate

delay entirely. This scalability to handle increased amounts of traffic without becoming computationally intractable was one of our original design requirements.

In addition to comparing different traffic profiles, we also compare different agent formulations. Sector agents essentially control collections of edges, discretizing costing strategies into different directions. This had a natural parallel to the area-based sector distinctions in traditional air traffic management. However, we hypothesized that we could gain finer control over the policies of agents if we allowed each edge to learn. This would enable more specific policies to be developed independently.

Contrary to our assumption, link agents performed comparably to sector agents, and in some cases underperformed. We identified key differences in the two formulations; the first was that the link agents had a single traffic input and a single traffic output, whereas the sector agents had four inputs and four outputs, each for a cardinal direction. This lower state lost some information about other UAVs traveling nearby the link agent. However, the greater issue appeared to be the disparity between the number of agents in each case; for a random map, the number of links tended to be four to five times the number of sectors on the map.

The introduction of link agents makes this a much more complex multiagent coordination problem. There are many more agents, with much less state information available to them. Using link agents should allow us a finer degree of control over traffic routing, but this benefit is overshadowed by the noisiness in this large multiagent problem. The fact that we obtained comparable performance with five times more agents indicates that if this noise were to be mitigated, we could get better performance.

Chapter 7: Scalable UTM Learning

In the previous abstract experiments, we found similar performance with different agent definitions, even though there was a significant increase in the number of learning agents present. We expected to have a finer degree of control using the link agents, and thus obtain better performance overall when more agents were present in the system. Because this was not the case, we suspect that *multiagent noise* played a part in this.

7.1 The Multiagent Noise Problem

Large learning multiagent systems have many challenges when performing a coordination task. A common approach to coordinating agents is to give the same score to the entire system for a task that they are completing. This gives some indication of how well the task is being completed as a result of their joint effort, but does not directly assess how each of them contributed. The amount of the reward that they are *not* responsible for is considered 'noise'.

A major focus in multiagent learning is reducing the effects of this 'noise' while ensuring a reward for working toward a group objective. Noise reduction has been approached by using *difference rewards* with counterfactuals to suppress the effects of other agents' actions [2, 3, 4, 6, 7, 95, 96]. Difference rewards reduce the noise in the system by subtracting off the impact of other agents in the reward.

Another kind of noise which has recently been explored in large multiagent systems is *exploratory* noise. This noise is caused by agents taking off-policy moves in order to explore their environment. This can be ignored as an infrequent environmental noise in many cases and learned around, but this off-policy selection can introduce a constant source of noise in a large system of learning agents. CLEAN (Coordinated Learning without Exploratory Action Noise) was developed in a similar manner to difference rewards, in order to suppress noise from agents taking their exploratory actions [46]. This approach uses the difference reward framework in that it subtracts off the impact of other agents on the global reward. However, CLEAN specifies that the global rewards

be calculated based on the agents following their policy-specified action and a potential counterfactual action that they could have taken.

Larger multiagent systems have many problems, but sometimes the large number of agents can be an advantage. D'Ambrosio and Stanley point out the *problem of reinvention*, which refers to the fact that many agents in the system must re-learn policies that may be similar or identical to other policies developed in the system [28]. They suggest approaching this by using an algorithm that represents the *pattern of policies*, which can train much larger teams by extracting roles from these policy patterns.

7.2 Learning with Subsets

One way of reducing the problem of noise in a highly coupled multiagent system is to reduce the variability of actions in the multiagent system. CLEAN accomplishes this by making agents take their policy-optimal actions rather than exploratory actions. We take this one step further; we look at cases where only subsets of agents are actually learning in the system, and other agents take a base action. In this section, we experimentally identify which agent is the *best* single agent to perform in the system alone, and we identify which agent is the *worst* agent in the system by removing agents individually and comparing global performance.

7.2.1 Include-One

To test whether noise could be reduced by excluding agents, we performed several tests isolating a single agent. This agent would learn while the actions of the other agents were kept static, outputting the baseline value of 1. This would be allowed to learn for 100 epochs and the maximum value obtained would be computed.

Figure 7.1 shows the performance of all agents learning at once compared to learning with a single agent (which we discuss in this section), as well as learning excluding a single agent (which we discuss in Section 7.2.2). The traffic conditions for this domain were a generation rate of $\rho = 5UAV/s$, on an 11-node and 38-edge domain with the graph topology shown in Figure 3.3.

We ran 38 experiments focusing on single-agent performance; each where an individual link was learning and active in the domain. The single *active* agent learned its policy

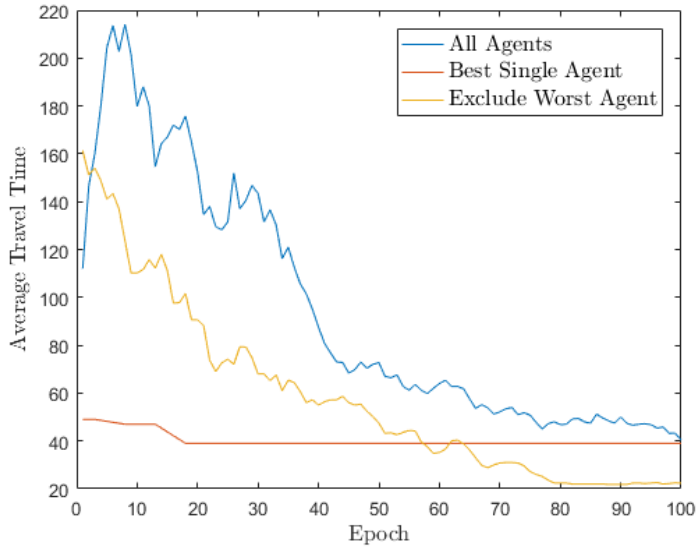


Figure 7.1: Learning with all agents, learning using only the best individual agent, and learning excluding the worst individual agent.

using neuro-evolution, while the other 37 *inactive* agents generated a flat output of 1 at each timestep. Inactive agents did not learn, and did not respond to traffic conditions. Each experiment had a different single agent active in the system. We then identified the single agent that attained the best global performance in the system. This was the *best*-performing agent, and its performance is shown on Figure 7.1.

Figure 7.1 indicates that the value of a single agent learning can be higher than all agents learning at once. The single agent trial shows a fast convergence, compared to all agents learning simultaneously, because there is no noise from other agents within the system. It converges to its final value around epoch 18, while it takes all agents learning simultaneously the full 100 epochs to approach a similar performance.

Performance at the beginning of the episode starts better for the single-agent test than with the all-agent test. This is because there is not random policies of agents at the beginning, but instead output a static policy of 1. This baseline provides better than random performance, but converged learned performance is ultimately preferable with any subset of learning agents.

The success of the single-agent testing gives us a strong indication that the multiagent

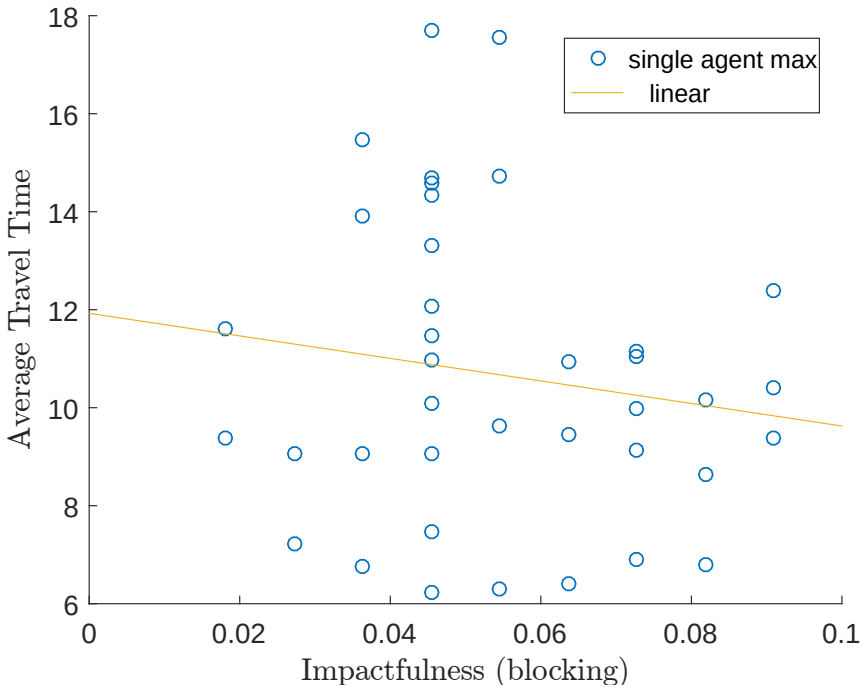


Figure 7.2: The maximum performance of a single agent graphed against that agent's blocking impactfulness. The linear trend line shows that an agent's blocking impactfulness has a negative correlation with increasing average travel time, indicating agents that can block traffic tend to have better performance.

system has difficulty learning quickly because other agents cannot tolerate randomness introduced by concurrent agent learning. However selecting a single agent to learn cuts out the possible contribution of other agents learning in the state space. Though the noise in the system is eliminated, and results in fast learning, selecting a single agent to learn could hurt the overall long-term converged performance of the system.

Testing with a single agent at a time requires a full simulation for each individual to find the best performing agent. In this case, this caused us to run 38 separate tests in order to determine the best agent in the system. This is an exhaustive pre-processing step that may not be feasible for larger systems. To avoid this exhaustive step, we looked at the impactfulness of an agent, and attempted to correlate this metric, which does not require a full simulation, with the converged performance of a full simulation.

In the case of blocking impactfulness, we see in Figure 7.2 that there is a weak linear correlation with better performance given a higher impactfulness. However, Figure 7.3 indicates that attracting impactfulness has a weak linear correlation with *worse* performance given a higher impactfulness. This lack of a strong correlation between either impactfulness metric indicates that we cannot use it as a substitute for an exhaustive simulation to find which agent will perform the best acting alone.

The deeper insight is that the measure of *ability to change optimal paths* from distance-optimal does not necessarily indicate better performance. This means that due to the topology of the graph, some of the agents are situated such that they *should not* try to attract or deflect traffic in the simulation, and instead should output a value that will *not* pull or push traffic from it. This is related to the policies that the agents developed based on the topology of the map.

7.2.2 Exclude-One

The previous section showed that learning with a single agent could perform better than all agents together. This presented a static learning problem which was entirely decoupled from the policies of other agents. However, we wanted to investigate whether substantial gains could be acquired by *excluding* an agent that contributed a large amount of noise to the system.

In a similar way to how we performed the single-agent tests, we selected a single agent to output a value of 1 instead of learning a policy. The 37 active agents would then learn their policy as normal. We would then investigate the converged performance of the system without this agent. The same graph topology provided in Figure 3.3 was used in this case as well. We again ran 38 experiments, each deactivating a single agent at a time.

Figure 7.1 shows that when the correct agent is *deactivated*, a substantial amount of noise can be reduced in the system, and the converged performance can improve. We call this agent the *worst* agent, as it has the worst performance in the system.

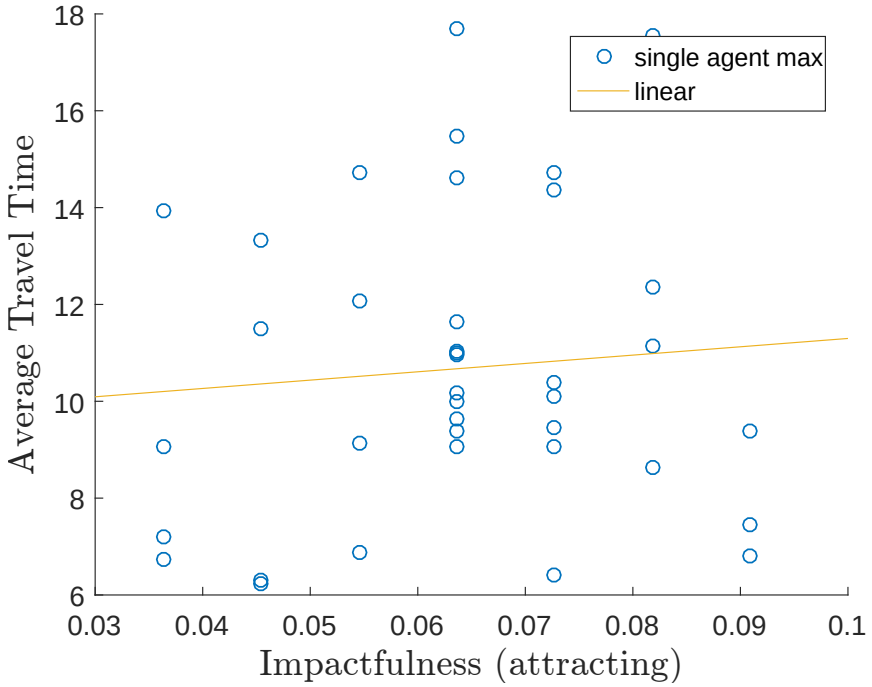


Figure 7.3: The maximum performance of a single agent graphed against that agent's attracting impactfulness. The linear trend line shows that an agent's attracting impactfulness has a positive correlation with increasing average travel time, indicating agents that can attract traffic tend to have worse performance.

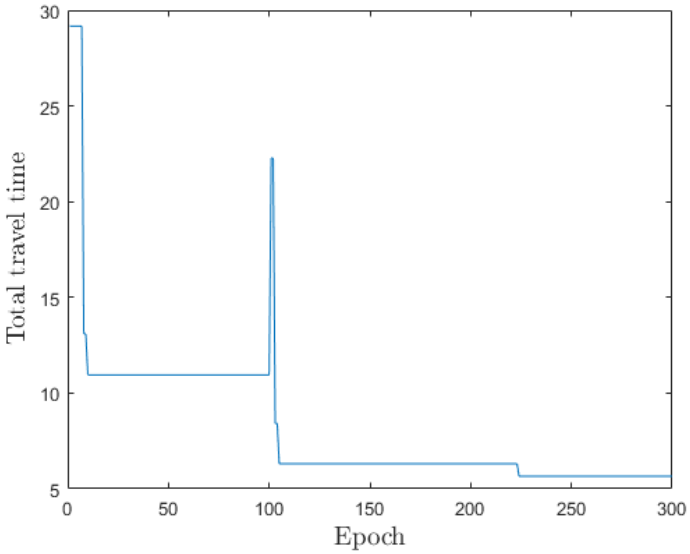


Figure 7.4: Learning with 3 randomly-selected agents learning sequentially in the system. At step 100 the second agent is activated, and the first agent is deactivated. At step 200 the third agent is activated and the second agent is deactivated.

7.2.3 Sequential Learning

In order to reduce the largest amount of noise, but still get adaptation from several elements in the system, we looked at using subsets of agents learning. Similar to the include-one trials, we looked at what happened when a single agent was added into the system and all others output a value of 1. After a 100 epochs, this agent would stop learning, fix its policy to the best it had found, and then another agent would begin to learn in the system. The policy found by the first agent would remain fixed (but not 1) while the second agent learned in the system. Though the actions of this agent in the system was no longer static, its *policy* is static, and therefore it presents a more learnable problem for the other agents in the system.

For these trials, we initialized the neural networks to have zero weights instead of random. This would mean that they would output a constant amount before mutation, and would only select a mutation if it were better than outputting a value of 1.

Figure 7.4 shows the effect of selecting 3 agents randomly from the system for learn-

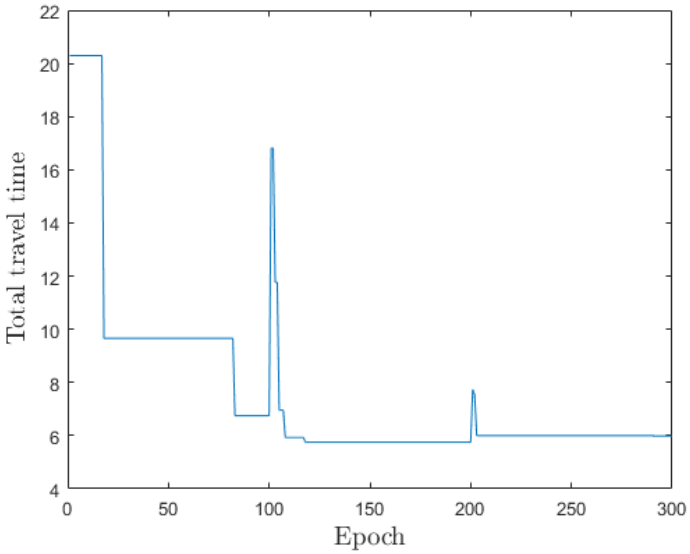


Figure 7.5: Sequential activation of three agents in the system. We see a slight decrease in performance after the third agent is added into the system.

ing, and having each learn sequentially. The graph shows spikes of poor performance during the transition phases, as a neural network is introduced with zero policy. Having a zero policy is slightly different than having a 1-output, which causes some increase in total travel time, until the agent can learn to handle this. We see here that we have quick convergence when learning with a single agent at once.

This approach depends entirely on the agents that are activated in the system. We see that when three different agents are active, as in Figure 7.5, the converged performance is above 6 aerate steps of total travel time, as opposed to the converged performance in Figure 7.4, which is near 5. In fact, this marks an increase from the global minimum travel time in the graph, which is found before the third agent begins learning. This indicates that sometimes the agent does not learn to shut off entirely when it is not actually needed.

Figure 7.6 shows a sequence of three agents that do not provide substantial improvement over the first agent learning alone. The performance does not substantially decrease when the other agents are added, but it does not decrease either. When the second agent is added, it adds a substantial decrease in performance, performing much

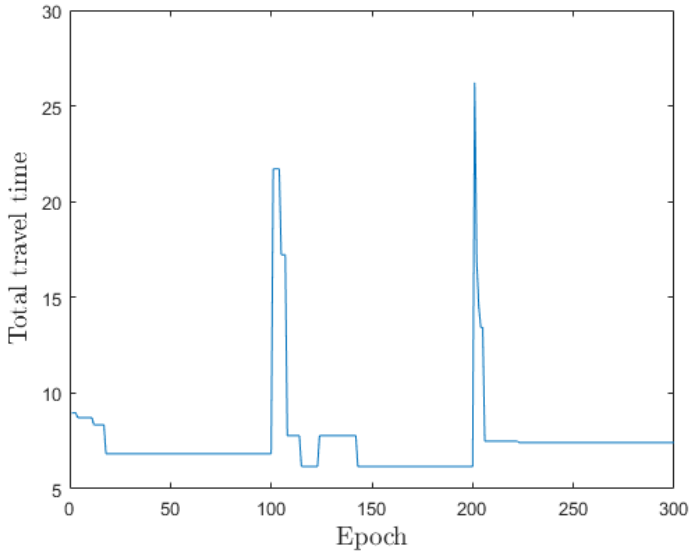


Figure 7.6: Sequential activation of three agents in the system. We see that in this case, there is largely no improvement when more than one agent learns in the system.

worse than the first agent without training. This is a negative result for this approach. However, it should be notes that all agents still find solutions that outperform the entire system working together.

Our results from sequentially activating agents in the system show that we can substantially increase convergence speed in the system by reducing the number of agents that act. This is one way of mitigating the multiagent noise that the full system experiences.

We also look at the effect of learning with varying numbers of agents in the system. To perform these tests, we sorted the agents by impactfulness, then performed tests to find which subset of n agents with top impactfulness would perform well. Figure 7.7 shows that there is an improvement in performance up to adding between 2-5 agents into the system. Beyond this, however, the improvement decreases, as multiagent noise becomes a problem. The multiagent noise means that the problem takes much longer to converge to a solution, and the agents are given only 100 epochs to learn. If the learning process were to continue for longer, there might be a shift to favor more agents in the system.

A natural extension of this approach would be to identify convergence of an agent, and

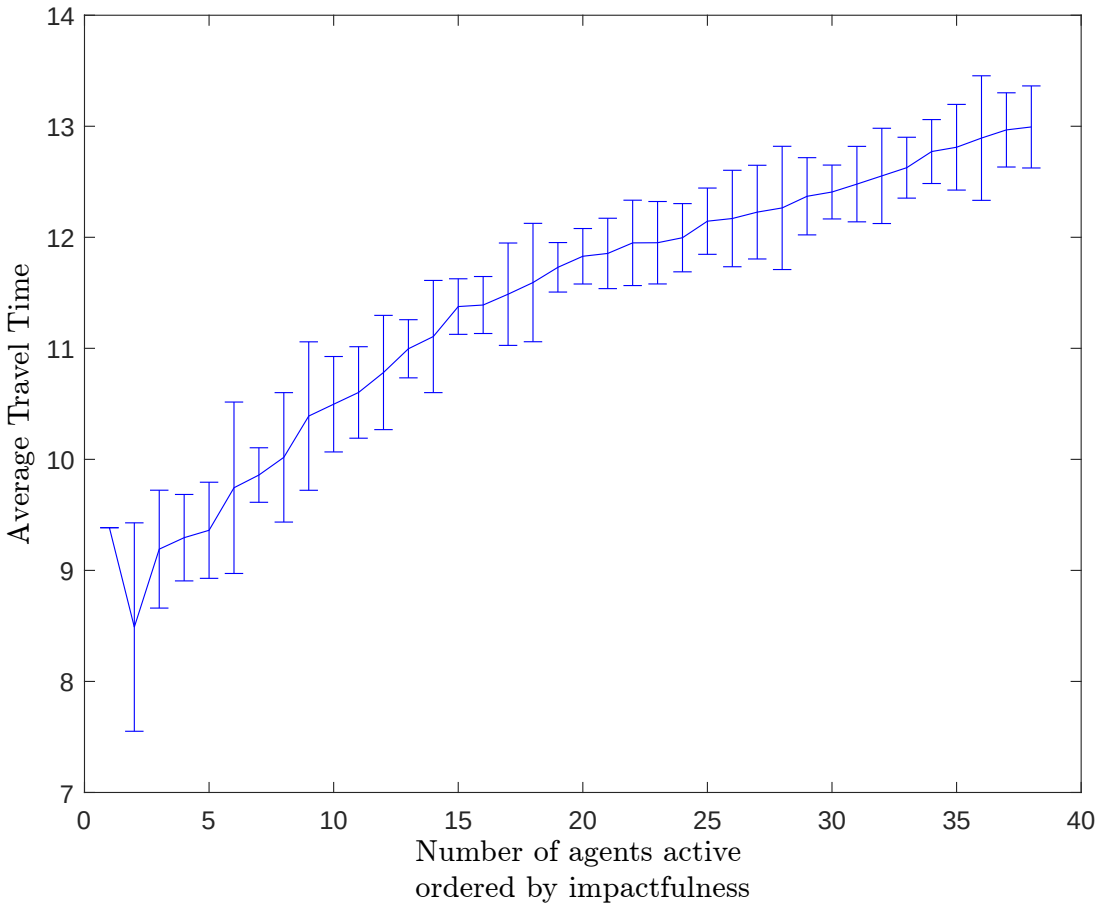


Figure 7.7: Testing with different numbers of agents, by impactfulness. We see an increase in total travel time as more agents are added as the multiagent learning problem becomes harder.

then switch to learning with another single agent when convergence has been reached. This could lead to much faster discovery of a good routing solution. Additionally, our results indicate that there is a specific subset and ordering of agents learning in the system that produce good results. Being able to detect these subsets a priori would be a beneficial extension to this work.

7.3 Discussion

We have investigated the problem of multiagent noise in the UAV traffic management domain. Multiagent noise occurs when there are many agents acting at once in a system. In the UAV traffic management domain, we would like to be able to scale to systems, and therefore would like to mitigate this multiagent noise problem as much as possible.

Multiagent noise can be reduced if fewer agents act in the system. Our experiments with impactfulness indicated that some agents had more *ability* to influence the system than others. By excluding some agents from the learning process aimed to reduce the noise in the multiagent system. This could potentially be done with minimal effect on system performance if the agents being removed had a small ability to influence the system anyway.

We started by isolating single agents in the system, and assess their learning performance in the absence of other agents acting in the system. This represents how an agent would act without any multiagent noise. We found that in some cases, an agent acting along in the system could have better learning performance than all agents learning concurrently. This is significant because it illustrates the degree to which other agents can confuse the learning process. The advantages of having other agents with the *ability* to adapt better policies in the system is overshadowed by the noise that interferes with the learning process. This is counterintuitive; the baseline policy of outputting one tends to be highly suboptimal. It would seem that the advantages of learning to output something different should outweigh a noisy reward signal.

Impactfulness did not have a strictly linear correlation with the performance of a single agent acting alone in the system. We expected that if an agent could impact more paths of UAVs in the system, it would correlate to having a greater reward in the system. By definition, an agent with an impactfulness of zero should have zero improvement over baseline when learning, because it would not be able to affect the

performance of the system in any way, no matter how high or low it set its value. We do see a loose correlation of impactfulness with global reward attained by a single learning agent, but impactfulness turns out not to be a good predictor of received reward.

This indicates that there is another parameter besides impactfulness that matters in the ability to produce a good global reward; whether an agent *should* attract or deflect paths through it. The topology of the map may be such that it is counterproductive to shift the flow of traffic from an agent. This would not be captured by our impactfulness metric, and if it is not a good policy for an agent to *use* its ability to attract or deflect traffic, it will still have little impact on the system. The highest global value score would therefore correspond to a high impactfulness, and a general positioning in the system where it was beneficial to draw or deflect many different UAV paths. As of now, we have no way of quantifying this position in the graph, and this could be a source for future investigation.

Similar to the formulation of Difference Rewards, which exclude an agent during the calculation of a reward, we also experimented with excluding single agents in the system. This experiment gave us an insight into the working of the multiagent system; some individual agents produce much more noise than others in the learning process. These *deceptive* agents can signficantly reduce performance. We saw substantial improvement when the correct agent was removed from the learning process.

The exclusion trials gave us the best performance in the system compared with all agents learning or a single agent learning. This validates our multiagent approach; when we remove a noisy agent, we can get the benefit of all agents adapting in the system, while substantially reducing the noise from learning.

Excluding one agent also did not strictly linearly correlate to the impactfulness. Again we expected higher impactfulness to correspond to higher performance when an agent was removed, for the exact opposite reason of the include-one trials; we assumed that an impactful agent would have the most ability to vary the learning signal in the system. Again, we find a loose correlation, but this is not sufficient to predict the global reward value that an agent will produce when acting in the system. The reason for this is slightly more obvious: an impactful agent may be needed in the system to provide routing. If it is able to learn a good policy, removing it will be detrimental for the system, and not remove a substantial amount of noise.

Finally, we looked at sequentially adding agents into the system. In these experiments

we aimed to get the fast convergence observed in the single-agent trials, but to add adaptability for other agents once that single agent had learned. This provided mixed results. Certain subsets of agents would perform better than others when learning in sequence. In some cases, we found fast perfomance to good values when the correct sequence of agents was introduced.

Our expectation was that if agents learned sequentially, and an unhelpful agent was added in later, it would be able to learn to output a flat value as it had been forced to output before. In some cases we observed this behavior, but it appeared that sometimes the agents could learn to reduce their effect, but not completely learn the optimal output value.

We successfully showed that we can reduce the agent noise substantially, and attain better performance than all agents learning together. In this section we identified these subsets of agents experimentally, and were able to achieve substantial reduction in total travel time. Our goal was to find the optimal mixture of agents working in the system. We moved toward this by isolating the impact of individual agents, and by introducing agents sequentially, but we did not find an a priori method that would allow us to design a smaller multiagent system. Impactfulness was a promising metric, but it did not correlate entirely to the global reward that an agent could attain (or detract from) in the system.

Chapter 8: Conclusion

In this work we present an end-to-end construction of a UAV traffic management problem. We first propose a method to manage UAV traffic at a high level by adding a two-step process to a UAV's planning strategy, forcing a UAV to plan the *sector order* that it visits before planning a finer-grained physical path across the space. We then use this planning over the sector order to incentivize or disincentivize travel of UAVs through sections of the airspace, using a learning multiagent system to assign edge costs.

First we outline the UAV traffic management problem, a new problem that mixes elements of traditional ground traffic modeling with airspace-based traffic management approaches. We then develop several performance metrics, and a control methodology to manipulate air traffic in our problem. We verify that our control methodology is effective by quantifying the *impactfulness* of elements in a graph. We then outline two agent formulations to provide multiagent traffic reduction in our problem.

Our multiagent framework has many advantages. The first is that it is platform-agnostic. We coordinate UAVs by incentivizing alternate routes when a UAV's path may become congested. The UAVs independently calculate their trajectories through the space with this information. This frees us from assuming anything about the UAV's internal structure. Our management strategy is also dynamic. Agents evolve *policies* that convert observed traffic to air travel costs. The costs will therefore change throughout the simulation as different numbers of UAVs travel through the space. A converged solution represents a real-time costing strategy that responds dynamically to traffic conditions.

We also consider time-extended effects. Our agents learn to reduce delays *over an entire run*. This means that downstream delay propagation is captured in the final learned costing policy. Agents can thus respond proactively to observed traffic patterns in order to curtail future traffic. Finally, computing a forward pass through a neural network is fast. The training time may be significant to produce these policies, but an evolved policy can be used in real time.

Experiments using robots demonstrated an end-to-end application of our routing approach. Using the ROS framework we were able to implement the UAV traffic man-

agement system on Pioneer 3-DX robots on a real maze. In doing this, we show to feasibility of this approach toward a real application. The application presented largely in this work is geared toward UAVs. However, in showing our work on a Pioneer, we demonstrate that our approach is platform- and planner-independent at the low level.

In this work we find a tradeoff between the two different UAV traffic management agent definitions. Sector agents are able to use a much larger amount of the information in the system, but are unable to provide fine-grained control. Link agents offer a way to modify individual transitions between the sectors, but observe much less of the state space and suffer from multiagent noise. The next step in this research is to develop new multiagent coordination techniques (e.g. difference rewards combined with deep learning approaches) that can provide effective reward shaping in this complex domain.

Though the link agents performed had similar performance to the sector agents, it is contrary to intuition that they were able to learn so effectively. In the 20-node case, there were 20 agents in the sector case and 102 agents in the link agent case. The link agent case presents a much larger multiagent system, with agents that could observe much less of the environment than the sector agent case. There are many techniques in the field of multiagent learning that can improve the coordination between agents in large systems. This indicates that there is a promising area of future work in gaining further system improvement by developing multiagent coordination techniques for the link agent controlled UTM domain.

We also show that we can gain substantial convergence improvements by looking at subsets of agents in the multiagent system. We found that a single agent acting alone in the multiagent system can provide faster convergence and better performance than the entire multiagent system acting together. By examining a single agent in the system, we can reduce all multiagent noise. Even better performance is attained when we exclude a particularly detrimental agent from the system. In this work, we find these subsets experimentally, but we leave it to future work to develop an autonomous method of identifying these subsets a priori.

The policies developed at this high level are useful, but we are ultimately testing on an abstraction of a physical domain. This abstraction may not capture some of the dynamics that could occur when modeling the airspace. Previous work included a simulation of pixel-level paths across an obstacle-filled airspace. It may be valuable to investigate more complex system dynamics using the algorithms presented in this work,

as the presence of obstacles may introduce further challenges in coordination.

Chapter 9: Future Work

There are many extensions that were out of the scope of our current investigation, but could provide a research opportunity to others looking to use this multiagent system. Though we have investigated the variance of many different parameters including graph structure, traffic levels, fitness definitions, and agent structure, there are numerous extensions in the direction of better traffic modeling or agent construction. In this section we provide an overview of some possible extensions to the work presented in this dissertation.

9.1 Airspace Construction

There is a significant amount of work available in setting up the optimal architecture for the airspace of a multiagent system. In particular, our results showed a significant disparity between the routing ability of different agents in the system based on the graph topology at a high level.

We began our investigation of UAV traffic management by looking at a height map, and then hand-picking intersections to define areas of agent control. We then used Voronoi partitioning to define the bounds of agent control. Both the hand-selection of points and the points and the Voronoi partitioning are aspects that invite future work into this topic, as they were chosen arbitrarily. For example, if the Voronoi partitioning is the ideal method of sectioning the space for agent control, is there a mathematical method of choosing points in the space that yields optimal learning performance? Finding a better partitioning of the space could result in a reduction of conflicts.

The approach of using a Voronoi partition not be well-tailored to this domain. It provided an impartial way of dividing the space, which reduced the human element of the state partitioning. However, focusing on an approach that would provide a more even area assigned to each space could result in performance improvements.

9.2 Physical Trials

Future work in the real robot domain will include improving the ability of robots to sense one another in the space. Additionally, scaling this to a much larger map, with a potentially more complex topology, could better demonstrate the feasibility of this method on a large scale for a real-world application. Making a larger map will also necessitate creating more traffic to manage. Future work will look at adding 'ghost' traffic in the system, to study the behavior of robots when they perceive many other robots in the system.

Another important direction for future work is outdoor trials. Outdoor trials mean a much less controlled environment, which could cause a problem for laser scan data and affect localization. However, a large advantage of outdoor experiments is the possibility of using GPS equipment. This could potentially give a more reliable position signal that could be communicated to UTM agents. Outdoor trials would also enable the use of actual UAV testing, which would be an important step for a UAV traffic management framework.

9.3 Heterogeneous Planners

Our UAV traffic management system manipulates a cost-based planner to route traffic. The only specification for using the learning approach for this work was that we assume that UAVs rely on cost-based planners. In this work we selected A* planners to plan paths for the UAVs. This was due to the widespread use and optimality guarantees of an A* planner, the availability of a fast implementation on Boost, and the suitability of A* for use on a pixel-based graph.

There are other cost-based planning algorithms that are feasible for use by UAVs. RRT* is a fast, sampling-based path planner that has gained popularity in robot applications. This provides paths that are not necessarily optimal, but have fast generation. This algorithm could be used to replace motion planning at the pixel level to give us faster simulations for physical conflict testing.

9.4 Tolerance Testing

Our learning approach has been tested in the accommodation of different traffic conditions, different graph scales, and different agent architectures. However in each of these, our focus was on learning with rational UAV motion. However this is not always the case in real-world scenarios. A UAV, particularly if it is remotely operated and not autonomously guided, may take suboptimal paths through the space. Investigating the gracefulness of the performance of our agents when in the presence of non-compliant agents is an important step toward actual application of this approach as a regulatory method.

There may also be substantial differences in the models of UAVs encountered in the real world. Investigating the effect of heterogeneous mixtures of UAVs would be a possible extension to this work. Heterogeneous speed, for example, would provide a less predictable element to routing UAVs. This could complicate the learning of policies by agents in the system. Additionally, the safety and predictability of some UAVs could differ. Different platforms mean that there will be different sensing capabilities, and it could be that reaction times allow some UAVs to require much more distance than others to safely interact in the space. An important extension of this work would be to find a way to incorporate this heterogeneity into the learning process.

A* is also not the only or necessarily the most common solution to path planning in an obstacle-filled environment. Different graph-based planning algorithms may be explored, using the same evolutionary infrastructure for setting the graph weights. Additionally, the routing algorithm currently does not account for possible heterogeneity in the system. Further improvements could be made to this algorithm if heterogeneity in UAV type or priority could be accommodated.

Bibliography

[1] ADS-B Technologies. What Is ADS-B? `http://www.ads-b.com/`.

[2] A. Agogino. Evaluating evolution and monte carlo for controlling air traffic flow. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 1957–1962, New York, NY, USA, 2009. ACM.

[3] A. Agogino and K. Tumer. Regulating Air Traffic Flow with Coupled Agents. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, May 2008.

[4] A. K. Agogino and K. Tumer. Reinforcement Learning in Large Multi-agent Systems. In *AAMAS-05 Workshop on Coordination of Large Scale Multiagent Systems*. Springer-Verlag, Utrecht, Netherlands, July 2005.

[5] A. K. Agogino and K. Tumer. Analyzing and Visualizing Multiagent Rewards in Dynamic and Stochastic Environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[6] A. K. Agogino and K. Tumer. Learning Indirect Actions in Complex Domains: Action Suggestions for Air Traffic Control. *Advances in Complex Systems*, 12(4-5):493–512, 2009.

[7] A. K. Agogino and K. Tumer. A Multiagent Approach to Managing Air Traffic Flow. *Autonomous Agents and MultiAgent Systems*, 24:1–25, 2012.

[8] A. Albert, F. S. Leira, and L. Imsland. Uav path planning using milp with experiments. *Modeling, Identification and Control*, 38(1):21, 2017.

[9] F. C. Allaire, M. Tarbouchi, G. Labonté, and G. Fusina. FPGA implementation of genetic algorithm for UAV real-time path planning. In *Unmanned Aircraft Systems*, pages 495–510. Springer, 2008.

[10] C. Amato and F. A. Oliehoek. Scalable planning and learning for multiagent POMDPs. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[11] B. and Walter, A. Sannier, D. Reiners, and J. Oliver. Uav swarm control: Calculating digital pheromone fields with the gpu. *The Journal of Defense Modeling and Simulation*, 3(3):167–176, 2006.

[12] J. K. Archibald, J. C. Hill, N. A. Jepsen, W. C. Stirling, and R. L. Frost. A Satisficing Approach to Aircraft Conflict Resolution. *Trans. Sys. Man Cyber Part C*, 38(4):510–521, July 2008.

[13] M. C. Aubert, S. zmc, A. R. Hutchins, and M. L. Cummings. Toward the development of a low-altitude air traffic control paradigm for networks of small, autonomous unmanned aerial vehicles. In *AIAA Infotech @ Aerosapce, SciTech Conference*, 2015.

[14] M. A. Azzopardi and J. F. Whidborne. Computational Air Traffic Management. In *Proceedings of the 30th AIAA/IEEE Digital Avionics Systems Conference (DASC2011), Best Paper Session*, pages 1.B.5–1, Seattle, WA, USA, 2011.

[15] T. Bäck, D. B. Fogel, and Z. Michalewicz. Handbook of evolutionary computation. *New York: Oxford*, 1997.

[16] I. Bekmezci, O. K. Sahingoz, and Ş. Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

[17] D. Bertsimas and S. Patterson. The Air Traffic Flow Management Problem with Enroute Capacities. In *May-June 1998, pp. 406-422*, 1998.

[18] C. M. Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[19] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: Reducing bloat using spea2. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 536–543. IEEE, 2001.

[20] C. Bongiorno, G. Gurtner, F. Lillo, L. VAlori, M. Ducci, B. Monechi, and S. Pozzi. An agent based model of air traffic management. In *Proceedings of the Third SESAR Innovation Days*, 2013.

[21] F. Bullo, E. Frazzoli, M. Pavone, K. Savla, and S. L. Smith. Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, 99(9):1482–1504, 2011.

[22] D. Burgett. SF Building Heights, 2014.

[23] S. Cameron, S. Hailes, S. Julier, S. McClean, G. Parr, N. Trigoni, M. Ahmed, G. McPhillips, R. De Nardi, J. Nie, et al. Suaave: Combining aerial robots and wireless networking. In *25th Bristol International UAV Systems Conference*, 2010.

[24] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, volume 2, pages 1–2, 2011.

[25] C. Chlestil, E. Leitgeb, N. Schmitt, S. S. Muhammad, K. Zettl, and W. Rehm. Reliable optical wireless links within uav swarms. In *Transparent Optical Networks, 2006 International Conference on*, volume 4, pages 39–42. IEEE, 2006.

[26] M. A. Christodoulou and S. G. Kodaxakis. Automatic commercial aircraft-collision avoidance in free flight: the three-dimensional problem. *IEEE Transactions on Intelligent Transportation Systems*, 7(2):242–249, 2006.

[27] W. J. Curran, A. Agogino, and K. Tumer. Addressing hard constraints in the air traffic problem through partitioning and difference rewards. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1281–1282. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[28] D. B. DAmbrosio and K. O. Stanley. Scalable multiagent learning through indirect encoding of policy geometry. *Evolutionary Intelligence*, 6(1):1–26, 2013.

[29] P. Dasgupta. A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(3):549–563, 2008.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[31] V. Digani, L. Sabattini, and C. Secchi. A Probabilistic Eulerian Traffic Model for the Coordination of Multiple AGVs in Automatic Warehouses. *IEEE Robotics and Automation Letters*, 1(1):26–32, 2016.

[32] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[33] X. C. Ding, A. R. Rahmani, and M. Egerstedt. Multi-uav convoy protection: An optimal approach to path planning and coordination. *IEEE Transactions on Robotics*, 26(2):256–268, 2010.

[34] H. Emami and F. Derakhshan. An overview on conflict detection and resolution methods in air traffic management using multi agent systems. In *16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP)*, pages 293–298. IEEE, 2012.

[35] Federal Aviation Administration. NextGen Implementation Plan, 2016.

[36] Federal Aviation Administration. Operation and Certification of Small Unmanned Aircraft Systems, June 2016.

[37] S. G. Ficici, O. Melnik, and J. B. Pollack. A game-theoretic and dynamical-systems analysis of selection methods in coevolution. *Evolutionary Computation, IEEE Transactions on*, 9(6):580–602, 2005.

[38] A. G. Foina, C. Krainer, and R. Sengupta. An Unmanned Aerial Traffic Management solution for cities using an air parcel model. In *2015 International Conference on Unmanned Aircraft Systems*, pages 1295–1300, 2015.

[39] A. Fraser. Simulation of genetic systems by automatic digital computers vii. effects of reproductive ra'l'e, and intensity of selection, on genetic structure. *Australian Journal of Biological Sciences*, 13(3):344–350, 1960.

[40] E. Gawrilow, E. Khler, R. H. Mhring, and B. Stenzel. Dynamic routing of automated guided vehicles in real-time. In H.-J. Krebs and W. Jger, editors, *Mathematics - Key Technology for the Future*, pages 165–177. Springer, 2008.

[41] B. P. Gerkey. amcl. `http://wiki.ros.org/amcl`.

[42] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

[43] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

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

[45] J. C. Hill, F. R. Johnson, J. K. Archibald, R. L. Frost, and W. C. Stirling. A cooperative multi-agent approach to free flight. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1083–1090, New York, NY, USA, 2005. ACM.

[46] C. HolmesParker, A. Agogino, and K. Tumer. CLEAN Rewards for Improving Multiagent Coordination in the Presence of Exploration (extended abstract). In *Proceedings of the Twelveth International Joint Conference on Autonomous Agents and Multiagent Systems*, Minneapolis, MN, May 2013.

[47] S. P. Hoogendoorn and P. H. Bovy. State-of-the-art of vehicular traffic flow modelling. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 215(4):283–303, 2001.

[48] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[49] J. P. How, B. BEHIHKE, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE control systems*, 28(2):51–64, 2008.

[50] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor. Maintaining network connectivity and performance in robot teams. *Journal of Field Robotics*, 25(1-2):111–131, 2008.

[51] J. R. Kok, N. Vlassis, et al. Sparse tabular multiagent q-learning. In *Annual Machine Learning Conference of Belgium and the Netherlands*, pages 65–71, 2004.

[52] P. Kopardekar. Safely Enabling Low-Altitude Airspace Operations: Unmanned Aerial System Traffic Management (UTM). Technical report, NASA Ames Research Center, Mountain View, CA, Apr. 2015.

[53] M. Kothari, I. Postlethwaite, and D.-W. Gu. Multi-uav path planning in obstacle rich environments using rapidly-exploring random trees. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 3069–3074. IEEE, 2009.

[54] G. B. Lamont, J. N. Slear, and K. Melendez. Uav swarm mission planning and routing using multi-objective evolutionary algorithms. In *Computational Intelligence in Multicriteria Decision Making, IEEE Symposium on*, pages 10–20. IEEE, 2007.

[55] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11*, October 1998.

[56] P. Long, W. Liu, and J. Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[57] B. D. Luders, S. Karaman, and J. P. How. Robust sampling-based motion planning with asymptotic optimality guarantees. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 5097, 2013.

[58] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.

[59] J. D. R. Millán, D. Posenato, and E. Dedieu. Continuous-action q-learning. *Machine Learning*, 49(2):247–265, 2002.

[60] C. Miyajima, Y. Nishiwaki, K. Ozawa, T. Wakita, K. Itou, K. Takeda, and F. Itakura. Driver modeling based on driving behavior and its evaluation in driver identification. *Proceedings of the IEEE*, 95(2):427–437, 2007.

[61] P. Munjal and J. Pahl. An analysis of the boltzmann-type statistical models for multi-lane traffic flow. *Transportation Research*, 3(1):151–163, 1969.

[62] N. Özalp and O. K. Sahingoz. Optimal uav path planning in a 3d threat environment by using parallel evolutionary algorithms. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 308–317. IEEE, 2013.

[63] R. A. Paielli. Automated Generation of Air Traffic Encounters for Testing Conflict Resolution Software. *AIAA Journal of Aerospace Information Systems*, 10(5), May 2013.

[64] M. Pechoucek and D. Sislak. Agent-based approach to free-flight planning, control, and simulation. *Intelligent Systems, IEEE*, 24(1):14–17, 2009.

[65] N. Pérez-Higueras, R. Ramón-Vigo, F. Caballero, and L. Merino. Robot local navigation with learned social cost functions. In *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on*, volume 2, pages 618–625. IEEE, 2014.

[66] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[67] A. Provodin, L. Torabi, B. Flepp, Y. LeCun, M. Sergio, L. D. Jackel, U. Muller, and J. Zbontar. Fast incremental learning for off-road robot navigation. *arXiv preprint arXiv:1606.08057*, 2016.

[68] L. Qiu, W.-J. Hsu, S.-Y. Huang, and H. Wang. Scheduling and routing algorithms for AGVs: a survey. *International Journal of Production Research*, 40(3):745–760, 2002.

[69] Y.-h. Qu, Q. Pan, and J.-g. Yan. Flight path planning of uav based on heuristically search and genetic algorithms. In *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, pages 5–pp. IEEE, 2005.

[70] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[71] E. Saad, J. Vian, G. Clark, and S. Bieniawski. Vehicle swarm rapid prototyping testbed. In *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*, page 1824, 2009.

[72] J. Samek, D. Sislak, P. Volf, and M. Pechoucek. Multi-party Collision Avoidance among Unmanned Aerial Vehicles. In *Intelligent Agent Technology, 2007. IAT '07. IEEE/WIC/ACM International Conference on*, pages 403–406, 2007.

[73] M. Shanmugavel, A. Tsourdos, B. White, and R. Żbikowski. Co-operative path planning of multiple uavs using dubins paths with clothoid arcs. *Control Engineering Practice*, 18(9):1084–1092, 2010.

[74] A. A. Sherstov and P. Stone. Function approximation via tile coding: Automating parameter choice. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 194–205. Springer, 2005.

[75] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001.

[76] A. Singh, A. Krause, C. Guestrin, W. J. Kaiser, and M. A. Batalin. Efficient planning of informative paths for multiple robots. In *Proceedings of IJCAI 2007*, pages 2204–2211, 2007.

[77] D. Sislak, P. Volf, and M. Pechoucek. Agent-Based Cooperative Decentralized Airplane-Collision Avoidance. *Intelligent Transportation Systems, IEEE Transactions on*, 12(1):36–46, 2011.

[78] D. Sislak, P. Volv, M. Pechoucek, and N. Suri. Automated Conflict Resolution Utilizing Probability Collectives Optimizer. *IEEE Transactions on Systems, Man, and Cybernetics*, 41(3):365–375, May 2011.

[79] A. Sivakumar and C. K.-Y. Tan. Uav swarm coordination using cooperative control for establishing a wireless communications backbone. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3*, pages 1157–1164. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[80] SkyRadar Radenna LLC. Ads-b technology (tis-b, fis-b). `http://www.skyradar.net/skyradar-system/adsbtechnology.html`.

[81] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *ICML*, pages 903–910, 2000.

[82] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[83] D. Stonier, J. Lee, and H. Kim. rocon. `http://wiki.ros.org/rocon`.

[84] R. Stranders, A. Farinelli, A. Rogers, and N. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of IJCAI 2009*, pages 299–304, 2009.

[85] F. Su, Y. Li, H. Peng, and L. Shen. Multi-ucav cooperative path planning using improved coevolutionary multi-ant-colony algorithm. *Emerging Intelligent Computing Technology and Applications*, pages 834–845, 2009.

[86] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[87] F. Taghaboni-Dutta and J. M. A. Tanchoco. Comparison of dynamic routeing techniques for automated guided vehicle system. *International Journal of Production Research*, 33(10):2653–2669, 1995.

[88] N. Tao, J. Baxter, and L. Weaver. A multi-agent, policy-gradient approach to network routing. In *In: Proc. of the 18th Int. Conf. on Machine Learning*. Citeseer, 2001.

[89] T. Tarnopolskaya and N. Fulton. *Synthesis of Optimal Control for Cooperative Collision Avoidance for Aircraft (Ships) with Unequal Turn Capabilities*. Springer, 2009. DOI: 10.1007/s10957-009-9597-1.

[90] W. L. Teacy, J. Nie, S. McClean, and G. Parr. Maintaining connectivity in uav swarm sensing. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1771–1776. IEEE, 2010.

[91] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[92] B. Toledo, V. Munoz, J. Rogan, C. Tenreiro, and J. A. Valdivia. Modeling traffic through a sequence of traffic lights. *Physical Review E*, 70(1):016107, 2004.

[93] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *Automatic Control,IEEE Transactions on*, 43(4):509–521, 1998.

[94] F. H. Tseng, T. T. Liang, C. H. Lee, L. Der Chou, and H. C. Chao. A star search algorithm for civil uav path planning with 3g communication. In *Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2014 Tenth International Conference on*, pages 942–945. IEEE, 2014.

[95] K. Tumer and A. Agogino. Distributed Agent-Based Air Traffic Flow Management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 330–337, Honolulu, HI, May 2007.

[96] K. Tumer and A. Agogino. Adaptive Management of Air Traffic Flow: A Multiagent Coordination Approach. In *Proceedings of the Twenty Third AAAI Conference on Artificial Intelligence, Nectar Track*, Chicago, IL, July 2008.

[97] K. Tumer and N. Khani. Learning from Actions Not Taken in Multiagent Systems. *Advances in Complex Systems*, 12(4-5):455–473, 2009.

[98] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian. The mit indoor multi-vehicle flight testbed. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2758–2759. IEEE, 2007.

[99] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides. A survey on sensor networks from a multiagent perspective. *The Computer Journal*, page bxq018, 2010.

[100] H. Wang, J. Zhou, G. Zheng, and Y. Liang. Has: Hierarchical a-star algorithm for big map navigation in special areas. In *Digital Home (ICDH), 2014 5th International Conference on*, pages 222–225. IEEE, 2014.

[101] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[102] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[103] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1225–1232. ACM, 2005.

[104] P. Yan, M.-Y. Ding, and C.-P. Zhou. Game-theoretic route planning for team of uavs. In *Proceedings of the 2004 International Conference on Machine Learning and Cybernetics*, volume 2, pages 723–728. IEEE, 2004.

[105] K. Yang, S. Keat Gan, and S. Sukkarieh. A Gaussian process-based RRT planner for the exploration of an unknown and cluttered environment with a UAV. *Advanced Robotics*, 27(6):431–443, 2013.

[106] L. Yliniemi, A. K. Agogino, and K. Tumer. Evolutionary agent-based simulation of the introduction of new technologies in air traffic management. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, pages 1215–1222. ACM, 2014.

[107] H. Yserentant. A new class of particle methods. *Numerische Mathematik*, 76(1):87–109, 1997.

[108] H. Yu, K. Meier, M. Argyle, and R. W. Beard. Cooperative path planning for target tracking in urban environments using unmanned air and ground vehicles. *IEEE/ASME Transactions on Mechatronics*, 20(2):541–552, 2015.

[109] M. Zhang, R. Batta, and R. Nagi. Modeling of Workflow Congestion and Optimization of Flow Routing in a Manufacturing/Warehouse Facility. *Management Science*, 55(2):267–280, Feb. 2009.

[110] M. Zhang, C. Su, Y. Liu, M. Hu, and Y. Zhu. Unmanned aerial vehicle route planning in the presence of a threat environment based on a virtual globe platform. *ISPRS International Journal of Geo-Information*, 5(10):184, 2016.

[111] C. Zheng, M. Ding, C. Zhou, and L. Li. Coevolving and cooperating path planner for multiple unmanned air vehicles. *Engineering Applications of Artificial Intelligence*, 17(8):887–896, 2004.

[112] E. Zitzler, M. Laumanns, L. Thiele, et al. SPEA2: Improving the strength Pareto evolutionary algorithm, 2001.