NETWORK CONNECTIVITY TRACKING FOR A TEAM OF UNMANNED AERIAL VEHICLES

By

James Trimble

Daniel J. Pack
Dean of the College of Engineering and Computer Science and Professor of Electrical Engineering
(Chair)

Jin Wang
Professor of Applied Mathematics
(Committee Member)

Craig Tanis
Assistant Professor of Computer Science
(Committee Member)

Dalei Wu
Assistant Professor of Computer Science & Engineering
(Committee Member)

NETWORK CONNECTIVITY TRACKING FOR A TEAM OF UNMANNED AERIAL VEHICLES

By

James Trimble

A Dissertation Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy in Computational Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2019

The views expressed in this dissertation are those of the author and do not necessarily reflect the official policy or position of the Air Force, the Department of Defense or the U.S. Government.

ABSTRACT

Algebraic connectivity is the second-smallest eigenvalue of the Laplacian matrix and can be used as a metric for the robustness and efficiency of a network. This connectivity concept applies to teams of multiple unmanned aerial vehicles (UAVs) performing cooperative tasks, such as arriving at a consensus. As a UAV team completes its mission, it often needs to control the network connectivity. The algebraic connectivity can be controlled by altering edge weights through movement of individual UAVs in the team, or by adding and deleting edges. The addition and deletion problem for algebraic connectivity, however, is NP-hard.

The contributions of this work are 1) a comparison of four heuristic methods for modifying algebraic connectivity through the addition and deletion of edges, 2) a rule-based algorithm for tracking a connectivity profile through edge weight modification and the addition and deletion of edges, 3) a new, hybrid method for selecting the best edge to add or remove, 4) a distributed method for estimating the eigenvectors of the Laplacian matrix and selecting the best edge to add or remove for connectivity modification and tracking, and 5) an implementation of the distributed connectivity tracking using a consensus controller and double-integrator dynamics.

DEDICATION

This work is dedicated to my loving wife, Ginny, and our beautiful daughter, Julia.

ACKNOWLEDGEMENTS

I would like to express the utmost gratitude to my former commanders and leadership who have helped me along towards achieving this dream. I'd especially like to thank Colonel Land, Colonel Berg, Lt Col Julson, Lt Col Hawn, Lt Col Claborn, and Lt Col Walden who encouraged me and allowed me to pursue academic endeavors while working for them. A tremendous amount of gratitude goes to Colonel Butler and my advisor, Dr. Pack, for the incredible opportunity to pursue unique and interesting research. Finally, thank you to my family and friends for encouraging me and helping me through the challenging times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Cooperative robotic systems provide many advantages over monolithic robotic systems. They are fault-tolerant since a mission can often continue even if a subset of the robots in the system fail. They are more agile and flexible seeing that subsets of the team can be allocated for different tasks or for parallelizing a single task. They are scalable, since increasing the capability of the system involves adding more systems of the same type. They can be less expensive in terms of cost of ownership by using simpler hardware to achieve the same goals that would otherwise require a larger, more complicated system.

For all of their advantages, multiple robot systems face many challenges of coordination, sensing, and communication. Path planners and controllers must account for the increased complexity of deconflicting the motion of each robot to prevent collisions. They must also balance competing requirements and constraints amongst the robots, such as fuel levels and performance characteristics. In terms of sensing, different robots may have different sensors and their information must be fused into meaningful data. The position of each robot can also impact the performance of the sensor, so the path planner must account for that as well. These factors also impact communication since positioning a robot further away from others degrades its signal quality and response time, causing delays in the sensor processing and control algorithms. Furthermore, with multiple

robot systems, communication networks must be established to distribute information effectively. The quality and robustness of the network is the focus of this work.

As the number of robots in a cooperative system grows, the number of potential communication links grows quadratically and the number of possible configurations grows exponentially. Communicating with members that are far away requires high transmitting power or multiple-hop communications. Each of these bring with it additional downsides such as battery consumption, complicated routing protocols, packet loss and recovery, and latency. Although direct, low-power, low-latency connections between every UAV may be desirable, the reality is that a multi-agent system requires a network to communicate effectively.

The quality of the communication network impacts the team's ability to complete distributed tasks. The particular configuration with a desired connectivity is of interest since it influences the performance of certain algorithms, in terms of convergence rates, and it corresponds to the robustness of the network. Consensus occurs more rapidly in highly connected networks [1] [2]. This can be critical in high-speed maneuvering, high-threat environments, or high-risk situations.

It is important to be able to measure the connectivity of the network in order to know if requirements are being met and to provide feedback for control mechanisms. Algebraic connectivity, henceforth referred to interchangeably with connectivity, is the second-smallest eigenvalue of the Laplacian matrix and is a commonly-used metric in networked systems. It was developed by Fiedler [3] in his seminal paper published in 1973. Algebraic connectivity provides a metric for the robustness of a network, such as an airborne network amongst UAVs.

Connectivity is directly applicable to multi-robot systems including formations of UAVs. Algebraic connectivity has wide application in mobile robotics for wireless power management [4],

connectivity maintenance [5], and distributed algorithms [6], as well as in transportation networks [7], computing [8], and optimization [9]. It has been shown that the convergence rate of consensus algorithms is tied directly to the algebraic connectivity [1].

Much work has been done around connectivity for multi-robot systems and multiple methods of increasing connectivity exist. These include optimization-based approaches [10], search algorithms [11], [12], and heuristic approaches [10], [13], [14]. Methods have been developed to optimize the operational capabilities of the formation [15] [16] [17] (connectivity maximization) as well as maintain the integrity of the formation [18] [19] [20] (connectivity preservation or maintenance). However, little work has been done on tracking a connectivity profile.

Since the connectivity impacts the performance of various tasks, it should be controlled to ensure the connectivity level meets the requirement of the current operation. Throughout the course of a mission, the UAV team may be required to complete different tasks that require various levels of communication traffic. Therefore, the level of the connectivity should be dynamically changed throughout the flight. This gives rise to the idea of a connectivity profile that is tracked by a UAV team [21] [22].

A team's connectivity can be modified throughout a flight in two ways: edge addition/deletion and edge-weight modification. New edges can be added to or deleted from a network by moving members in and out of their neighbors' wireless range or enabling and disabling communication with neighbors. Furthermore, edge weights of a network can be modified by altering the signal strength through a change in distance to a neighbor or by a change in the power used to transmit with a neighbor.

1.1 Outline

The remainder of this dissertation is organized as follows. Chapter 2 provides a background and review of relevant literature surrounding algebraic graph theory, the addition and deletion problem, and connectivity tracking. Chapter 3 provides a comparison of methods for adding and deleting edges in order to maximize connectivity. It also develops a rule-based algorithm for tracking connectivity. Chapter 4 develops a hybrid method for selecting edges to be added to or deleted from a graph in order to best modify the connectivity of the graph. Chapter 5 discusses a method for estimating eigenvalues in a graph and provides an extension of this method allowing the estimation of the graph's eigenvectors. Chapter 6 develops a controller for tracking a connectivity profile. The remainder of the work covers the results and conclusions as well as suggestions for future work.

CHAPTER 2

BACKGROUND

2.1 Graph Theory Concepts

A graph is a mathematical structure consisting of a set of vertices, or nodes, and a set of edges. Edges always connect two nodes. A node is incident to an edge if it is one of the two nodes that edge connects. The degree of a node is the number of edges it is incident to. The two nodes incident to an edge are said to be adjacent. An edge may be directed, in which case, the edge $\{i, j\}$ defines a connection from $node_i$ to $node_j$, but not from $node_j$ to $node_i$.

A path is a sequence of connected edges. The diameter of a graph is the length of its longest path. A cycle is a closed path. A spanning tree is a connected, acyclic set of paths which reaches every node in the graph.

An undirected graph may have an orientation, which is an assignment of a direction to each edge. If a graph is directed such that no two vertices are connected in both directions then the graph is oriented.

Two graphs are isomorphic if there is a correspondence between their vertex sets which preserves adjacency [23]. A sub-graph is a graph on a subset of vertices and edges of an existing graph. The complement of a graph, $\bar{G}$, is the graph formed by removing all existing edges of G and adding new edges between all non-adjacent vertices in G.

There are many types of graphs. Some relevant ones are the complete graph (see Fig. 2.1), in which every node is connected to every other node, the k-regular graph (see Fig. 2.2), in which every node is connected to k other nodes, the star graph (see Fig. 2.3), in which a single node has $n-1$ neighbors and all other nodes have this node as a single common neighbor, and the line graph (see Fig. 2.4), a planar graph (one that can be drawn such that no edges cross) where every node, except two, has two neighbors and the graph has only a single spanning path.

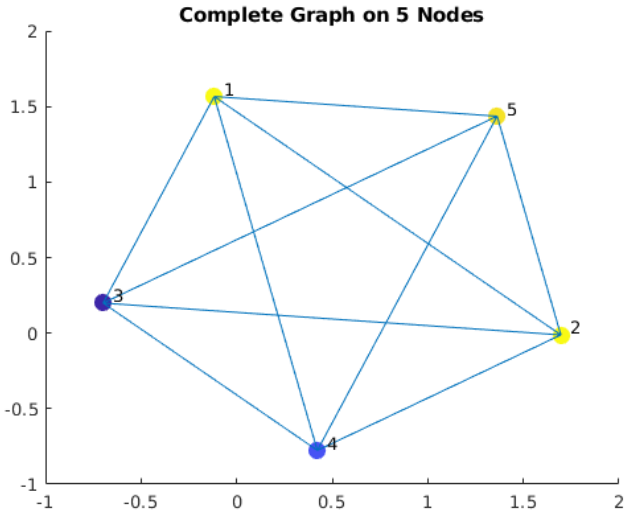

Figure 2.1 A Complete Graph on Five Nodes.

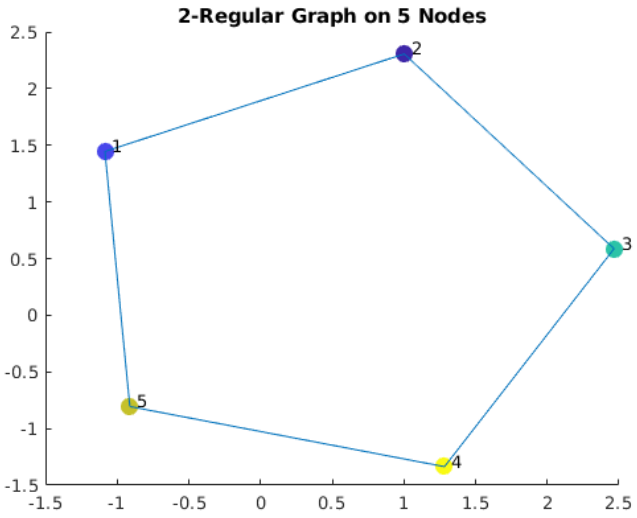

Figure 2.2 A 2-Regular Graph on Five Nodes.

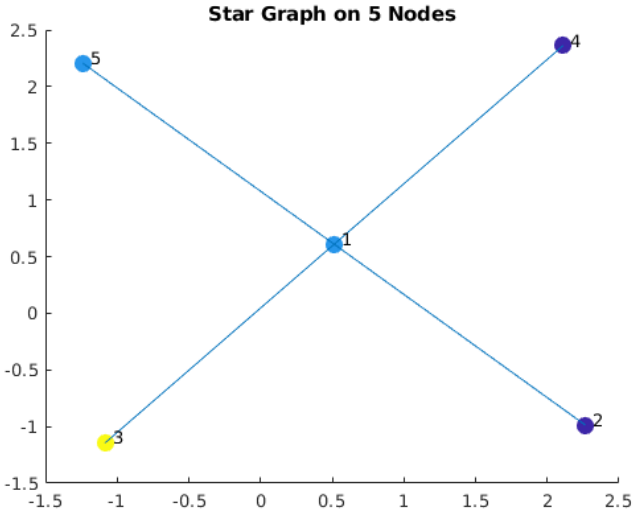

Figure 2.3 A Star Graph on Five Nodes.

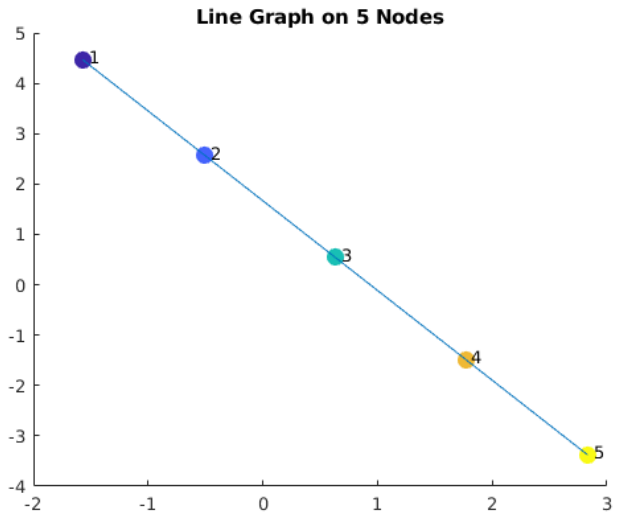

Figure 2.4 A Line Graph on Five Nodes.

2.2 Graph Matrices

Graphs may be represented by a variety of matrices. The different matrix representations of graphs have various uses. This work is primarily interested in the adjacency matrix, the degree matrix, the Laplacian matrix, and the incidence matrix. For the purposes of this work, graphs should be assumed to not include self-loops, i.e. edges from a node to itself.

The adjacency matrix is an $n \times n$ matrix where n is the number of nodes and each entry $A_{i,j}$ is non-zero if an edge exists between $node_i$ and $node_j$. The value of the adjacency matrix is typically binary, zero or one, for an unweighted graph or any continuous value for a weighted graph. The adjacency matrix for the graph shown in Fig. 2.3 is given here,

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The degree matrix is an $n \times n$ diagonal matrix where each entry, $D_{i,i}$, equals the degree of $node_i$ and each entry $D_{i,j}$, when $i \neq j$, is zero. In other words, $D_{i,i} = \sum_{j=1}^{n} w(i,j)A(i,j)$, where $j \neq i$. For unweighted graphs, the degree of each node is equal to the number of edges connected to it as the weight function always has a constant value of one. The degree matrix for the graph shown in Fig. 2.3 is given here,

$$D = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The ordinary Laplacian matrix, hereafter referred to simply as the Laplacian matrix, can be defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix for a graph. The Laplacian matrix for the graph shown in Fig. 2.3 is given here,

$$L = D - A = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The Laplacian matrix can also be formed using an incidence matrix. The incidence matrix consists of a matrix of size $n \times m$ where m is the number of edges and n is the number of nodes in the graph. Each column, e_i, of the incidence matrix corresponds to an edge in the graph and can be defined as follows,

$$e_{i,j} = \begin{cases} -1 & \text{edge } e_i \text{ leaves node } j \\ 1 & \text{edge } e_i \text{ enters node } j \\ 0 & \text{edge } e_i \text{ is not connected to node } j \end{cases}. \tag{2.1}$$

The above definition applies to an oriented graph, that is one in which a direction is assigned to each edge. When working with Laplacian matrices, the orientation is not important since the same Laplacian is obtained from any corresponding incidence matrix, regardless of orientation.

Therefore, the signs associated with entering and leaving a node may be reversed. The incidence matrix for the graph shown in Fig. 2.3 is shown here,

$$B = \begin{bmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

When an incidence matrix, B, is defined in this way, the Laplacian matrix, L, is obtained as $L = BB^T$. Furthermore, the addition or deletion of any edge or set of edges may be obtained by representing the set of edges to be added as an incidence matrix, S, and applying the following formula: $L' = L + SS^T$ for addition, or $L' = L - SS^T$ for deletion, where L is the Laplacian matrix and L' is the Laplacian matrix resulting from the addition or deletion of edges defined by the incidence matrix, S. Such representations are useful when working with edge sets.

Each of the previous matrices, adjacency, degree, and Laplacian, can be used to represent a weighted or unweighted graph. When the graph is weighted, the adjacency matrix is defined as follows,

$$A_{i,j} = \begin{cases} w(i,j) & node_i \text{ is adjacent to } node_j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where $w(i, j)$ is a function whose value is the weight of the edge connecting $node_i$ with $node_j$.

The weighted degree matrix is defined as,

$$D_{i,j} = \begin{cases} \sum_{k=1,k\neq i}^{n} w(i,k) & i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where $w(i, j)$ is an edge-weight function, and n is the number of nodes in the graph.

The weighted Laplacian matrix is still defined as $L = D - A$, however, the weighted degree and weighted adjacency matrices should be used. The incidence matrix can be weighted, however, the property $L = BB^T$, where L is the Laplacian matrix and B is the weighted incidence matrix, would no longer recover the original, ordinary weighted Laplacian.

2.3 Connectivity

The concept of connectivity as it applies to graphs describes not only how densely connected the graph is, but also the quality of the connectedness of the graph. At a basic level, a graph is connected if, for every pair of nodes i, j, there exists a path from $node_i$ to $node_j$, otherwise, it is disconnected. The idea of degrees of connectivity, however, is difficult to define without utilizing further criteria. There are several ways of defining connectivity.

A vertex cut is the removal of a set of one or more vertices which results in a graph that is disconnected [24]. Vertex connectivity is defined as the number of vertices in the minimum vertex cut [24]. A cut vertex is a vertex which, by itself, is a vertex cut [24].

An edge cut, similar to a vertex cut, is a set of edges which, once removed, results in a disconnected graph [24]. No loop can belong to an edge cut [24]. Edge connectivity defines connectivity as the number of edges in the minimum edge cut [24]. A cut edge is an edge which, by itself, is an edge cut [24]. A single edge of a connected graph is a cut edge if and only if it does not belong to a cycle [24]. The vertex connectivity is bounded above by the edge connectivity [25].

The algebraic connectivity of a graph is the second-smallest eigenvalue of the Laplacian matrix and was first proposed by Miroslav Fiedler [3]. When a graph is disconnected, i.e. there exists at least one pair of nodes between which there is not a path, the algebraic connectivity is zero. A non-zero connectivity indicates a path exists between every pair of nodes in the network. A higher connectivity indicates a more robust graph [26] [27] [28] [29] as a larger number of edges would need to be removed to result in the graph becoming disconnected. The algebraic connectivity ranges between zero and n, the number of nodes in the graph, inclusively.

The algebraic connectivity is monotonically increasing. That is, adding an edge to graph G to construct G', results in $\lambda_2 \leq \lambda_2'$. Additionally, λ_2' is bound by λ_2 and λ_3, the second- and third-smallest eigenvalues of the original graph Laplacian, so that $\lambda_2 \leq \lambda_2' \leq \lambda_3$ [10]. From this it is apparent that no edge will increase the connectivity when $\lambda_2 = \lambda_3$.

2.4 Addition/Deletion Problem

The addition/deletion problem answers the question of the best edge to add to or delete from a graph to cause the largest increase or decrease in connectivity. In the case of adding an edge, this can be stated as the following optimization problem [10]

$$\begin{aligned} &\text{maximize} \quad \lambda_2(L(E_{base} \cup E)), \\ &\text{subject to} \quad E \subseteq E_{cand} \end{aligned} \tag{2.4}$$

where λ_2 is the connectivity, L is the Laplacian matrix for the graph, E_{base} is the set of edges in the base graph and E is the set of edges selected from the set of candidate edges, E_{cand}.

The addition/deletion problem has been shown to be NP-hard by Mosk-Aoyama [30]. Therefore, several attempts have been made to develop heuristic methods of maximally increasing the connectivity.

2.5 Spectral Graph Theory

The sub-field of graph theory which is interested in the spectra of matrix representations of graphs is known as spectral graph theory. Primarily, the adjacency matrix and Laplacian matrix are used. There are several relevant results from spectral graph theory.

Since the Laplacian matrix is real, symmetric, and positive-definite, its eigenvectors associated with unique eigenvalues are orthogonal [25].

The eigenvector associated with the second-smallest eigenvalue of the Laplacian matrix is known as the Fiedler vector [3]. This vector can be used to approximately bisect the graph by separating the nodes with positive entries from those with negative entries [31] [32]. This provides a heursitic for the min-cut problem, where nodes should be divided into two groups such that the number of edges between them is minimized [31]. The Fiedler vector also acts as a super-gradient for algebraic connectivity and plays a key role in the greedy perturbation heuristic [10].

The multiplicity of the smallest eigenvalue of the Laplacian matrix, $\lambda_1 = 0$, indicates the number of disconnected components of the graph [3].

If the eigenvalues of a graph Laplacian G are given as λ_1 through λ_n, the eigenvalues of the complement of G, $\bar{G}$, are given as $\lambda_1 = 0$, $\lambda_k = n - \lambda_{(n-(k-2))}$, where k is an index s.t. $2 \leq k \leq n$, and n is the number of nodes in the graph.

If two graphs, G and G', are isomorphic, their Laplacian matrices are similar, i.e. they are co-spectral and share the same eigenvalues [25].

The eigenvector associated with the largest eigenvalue of the adjacency matrix is known as the Gould index [33] [34]. The value of a node's entry in the Gould index indicates its importance.

2.6 Representing UAV Teams

Any multi-UAV formation can be represented as a graph where each UAV in the formation is analogous to a node, or vertex, in the graph. The ability to communicate between pairs of robots is analogous to edges in the graph. One way to represent such a graph is using the Laplacian matrix, which is formed by subtracting the adjacency matrix from the degree matrix, $L = D - A$.

For weighted graphs, the weight value, w, is typically constrained to the range $w \in [0, 1]$. A typical weighting profile for Laplacians which describe multi-UAV formations is a function which is constant within a certain distance, to model a strong connection when the two communicating nodes are near each other, then decays exponentially beyond that distance to model a rapid degradation in signal strength due to path loss beyond the robust range [6].

A generic function for edge weights that resembles a path loss model is

$$w(i,j) = \begin{cases} 1 & \|p_i - p_j\| < R_{robust} \\ 0 & \|p_i - p_j\| > R_{max} \\ 2 - e^{\alpha(\|p_i - p_j\| - R_{robust})} & otherwise, \end{cases} \tag{2.5}$$

where R_{robust} represents the radius at which full signal strength is sustained and R_{max} represents the distance at which no usable signal is received. α is a user-defined parameter that was chosen so that w is continuous at R_{robust} and R_{max}. $\|p_i - p_j\|$ is the Euclidean distance between p_i and p_j, corresponding to nodes i and j, respectively.

Given a weighting function, w, the weighted Laplacian can be defined as follows,

$$L = \sum_{edge_{i,j} \in E} w(i,j) L_{i,j}, \tag{2.6}$$

where $L_{i,j}$ is the Laplacian matrix entry formed with a single edge between nodes i and j.

Undirected graphs are symmetric, indicating that if a path exists from $node_i$ to $node_j$, then a path also exists from $node_j$ to $node_i$. Directed graphs may not be symmetric since a directional edge, instead of a bi-directional edge, means that $A_{i,j} \neq A_{j,i}$. The undirected graph Laplacian is used in this work since it is assumed that each UAV is equipped with a homogeneous, bi-directional wireless network adapter. The assumption that all edges are bi-directional, or undirected, is important since it results in a real, symmetric, and positive semidefinite Laplacian matrix, which allows for use of a wider range of techniques when working with the Laplacian matrices.

CHAPTER 3

EDGE SELECTION AND CONNECTIVITY TRACKING

For an increasing number of UAV applications, varying levels of connectivity at different points throughout the duration of a mission are needed to meet operational requirements. Thus, the ability to track and obtain a desired connectivity profile over the course of a mission is critical. A connectivity profile is defined here as a sequence of desired levels of algebraic connectivity. The profile may have a time associated with it, so that the network's connectivity at time t should be equal to the desired connectivity, or it may be undefined, in which case the network should simply progress to the next desired connectivity level upon reaching the current target connectivity. As an initial step, the connectivity profiles used for this work do not impose a time requirement.

In this chapter four methods to adjust the connectivity of a networked system are presented: greedy perturbation heuristic (Fiedler method) [10], tabu search [11], [35], semidefinite programming (SDP) [10], and bisection [12]. A basic rule-based algorithm to track a desired connectivity profile through the addition and deletion of a sequence of single connections between two UAVs is developed. Furthermore, the effectiveness of methods for choosing the edge to be added or deleted using this algorithm is evaluated through MATLAB simulations.

3.1 Methods of Selecting an Edge

As discussed in Chapter 2, the addition/deletion problem is an NP-hard problem which answers the question of the best edge to add to or delete from a graph to cause the largest increase or decrease in connectivity. Four heuristic approaches are discussed here. Ghosh and Boyd [10] produced a semidefinite programming problem to be solved by an SDP solver and also developed the greedy perturbation heuristic which will be referred to as the Fiedler method. Wei and Sun [35] created the weighted tabu search, based on the tabu search algorithm developed by Glover [11], for solving the addition/deletion problem. They also implemented the Fiedler method/greedy perturbation heuristic and semidefinite programming methods and applied the techniques to air transportation networks. Kim [12] developed the bisection method of increasing connectivity based on a modified version of the secular equation and a bisecting search algorithm.

3.1.1 Fiedler Method or Greedy Perturbation Heuristic

The Fiedler method was developed by Ghosh and Boyd [10]. It is a greedy, local heuristic for solving the addition/deletion problem using the entries of the Fiedler vector.

If v is a unit eigenvector corresponding to λ_2 then vv^T is a supergradient of λ_2 [10]. For any symmetric matrix Y and graph Laplacian L the following holds,

$$\lambda_2(L + Y) \leq \lambda_2(L) + \mathbf{Tr}(Yvv^T) \tag{3.1}$$

[10].

When λ_2 is isolated, the supergradient is the gradient and this method yields a first-order approximation of the change in L [10]. When λ_2 is not isolated, no single edge addition will

increase λ_2 [10]. This follows from the principle of interlacing [31] [36]. Since the resulting connectivity, λ_2', obeys $\lambda_2 \leq \lambda_2' \leq \lambda_3$, if $\lambda_2 = \lambda_3$ then $\lambda_2 = \lambda_2' = \lambda_3$ and the connectivity does not change.

The Fiedler method involves finding the maximum squared difference between all entries in the Fiedler vector and selecting the pair of disconnected nodes whose corresponding entries produce the maximum value. This algorithm is described as follows,

Algorithm 1: Fiedler Method / Greedy Perturbation Heuristic

```
   Result: Node selection: node_1, node_2
 1 Given: v_2, the Fiedler vector
 2 max = 0
 3 for i ← 1 to n do
 4 |   for j ← i + 1 to n do
 5 |   |   if i ≠ j and L_{i,j} = L_{j,i} = 0 then
 6 |   |   |   if (v_2(i) − v_2(j))^2 > max then
 7 |   |   |   |   max = (v_2(i) − v_2(j))^2
 8 |   |   |   |   node_1 = i
 9 |   |   |   |   node_2 = j
10 |   |   |   end
11 |   |   end
12 |   end
13 end
```

In Algorithm 1, v_2 is the unit eigenvector corresponding to λ_2, the Fiedler vector, $v_2(i)$ is the ith element of the Fiedler vector, and L is the Laplacian matrix.

An advantage of this method is that if the graph is disconnected, the Fiedler method will always choose an edge which connects two connected components [10].

Ghosh et. al. provided the following upper and lower bound on the resulting connectivity after adding an edge.

$$\lambda_2(L + a_l a_l^T) \leq \lambda_2 + \frac{(v_i - v_j)^2}{1 + (2 - (v_i - v_j)^2)/(\lambda_n - \lambda_2)} \tag{3.2}$$

[10]

$$\lambda_2(L + a_l a_l^T) \geq \lambda_2 + \frac{(v_i - v_j)^2}{6/(\lambda_3 - \lambda_2) + 3/2} \tag{3.3}$$

[10]

Based on the bounds provided, the Fiedler method can be interpreted as maximizing both the upper and lower bounds [10].

Another interpretation is to consider the heuristic method of using the Fiedler vector in bisecting a graph such that the number of edges connecting vertices in the two separate halves is minimized [10] [37]. In other words, it attempts to strengthen the connection of two groups separated by a min-cut.

Nodes with positive entries in the Fiedler vector are grouped together. Similarly, nodes with negative entries are grouped together. Nodes with zero entries may be assigned to either group in approximately equal numbers [37] [38] [39]. Nodes whose entries in the Fiedler vector are most positive or most negative can be seen to be most strongly associated with their group. In light of this perspective, the Fiedler method can be thought to choose the edge which most strongly joins the bisected graph [10].

3.1.2 Bisection Method

The bisection method was developed by Kim [12] and was motivated by the goal of a computationally efficient (as compared to a brute-force search) and exact method of determining the best edge to add or remove to effect the largest change in algebraic connectivity. It requires knowledge of nearly all, if not all, eigenvalues and eigenvectors of the original Laplacian matrix.

The bisection method [12] takes advantage of the property $\lambda_2 \leq \lambda_2' \leq \lambda_3$, where λ_2' is the resulting connectivity after adding an edge, and the modified secular equation,

$$f(\lambda(G')^*) = 1 + \rho \sum_{k=2}^{n} \frac{(v_k(i) - v_k(j))^2}{\lambda_k - \lambda(G')^*}, \tag{3.4}$$

where $\lambda(G')^*$ is a hypothetical eigenvalue of $G' = G + ee^T$, e is an incidence vector for an edge to be added, and $v_k(i)$ and $v_k(j)$ are the ith and jth elements, respectively, of the kth eigenvector of G. λ_k is the kth eigenvalue of G. So long as $\lambda_k < \lambda_k(G')^* < \lambda_{k+1}$, (3.5) describes the relationship between an arbitrary value between λ_k and λ_{k+1} and the kth eigenvalue resulting from adding a specific edge to G [12].

$$f(\lambda(G')^*) = \begin{cases} > 0 & \lambda(G')^* > \lambda(G') \\ = 0 & \lambda(G')^* = \lambda(G') \\ < 0 & \lambda(G')^* < \lambda(G') \end{cases} \tag{3.5}$$

This secular equation allows one to choose an edge to add to a graph, then repeatedly guess the resulting eigenvalue and evaluate the equation to determine if the guess was too high, too low, or guessed correctly.

As an example, consider the five-node graph and corresponding secular equation plot shown in Fig. 3.1. The secular equation plot shows the result of adding an edge between nodes one and two. Green 'plus' markers show the zeros at the x-axis location corresponding to the resulting non-zero eigenvalues{1,2,4,5}. The asymptotes of the plot occur at the eigenvalues of the original graph, {0, 0.6972, 1.3820, 3.6180, 4.3028}, and are indicated by red 'circle' markers. For any value along the x-axis between any two asymptotes, the sign of the function indicates whether or not the actual eigenvalue resulting from the addition of that edge is greater than, less than, or equal to the corresponding x-value of that point. The eigenvalues resulting from the addition of the new edge will be the zeros of $f(\lambda(G')^*)$.

Kim [12] used this information to develop the bisection algorithm, which begins with the range λ_2 to λ_3 from the original graph and a set S of all candidate edges. Then, using (3.4), every candidate edge can be classified as producing a λ_2' that is greater than, equal to, or less than the midpoint, where λ_2' is the connectivity that would result from adding or removing the particular edge. Edges which produce a greater λ_2' are kept and the search space is reduced by half to include only the range where these edges reside. This process is repeated until a single edge remains, or the search range falls below an exit criteria, $|U - L| < \epsilon$.

Algorithm 2: Bisection Method

Result: Node selection: $node_1$, $node_2$

Given: λ_2 , and λ_3 , the second and third smallest eigenvalues of the initial graph, S, the set of candidate edges, and G, the initial graph

$x \leftarrow 0$;
$L \leftarrow \lambda_2, U \leftarrow \lambda_3$;
$d \leftarrow U - L$;
$M \leftarrow (L + U)/2$;
while $size(S_x) > 1$ *and* $x < log_2(d/\epsilon)$ **do**
    $S^+ \leftarrow \{\}, S^- \leftarrow \{\}$;
    for *each candidate edge,* $e_{i,j}$ **do**
        if $f(M) > 0$ **then**
            $S^+ \leftarrow S^+ \cup e_{i,j}$
        else
            $S^- \leftarrow S^- \cup e_{i,j}$
        end
    end
    if $size(S^-) = 0$ **then**
        $U \leftarrow M$
        $S_{x+1} \leftarrow S^+$
    else
        $L \leftarrow M$
        $S_{x+1} \leftarrow S^-$
    end
    $x \leftarrow x + 1$
end

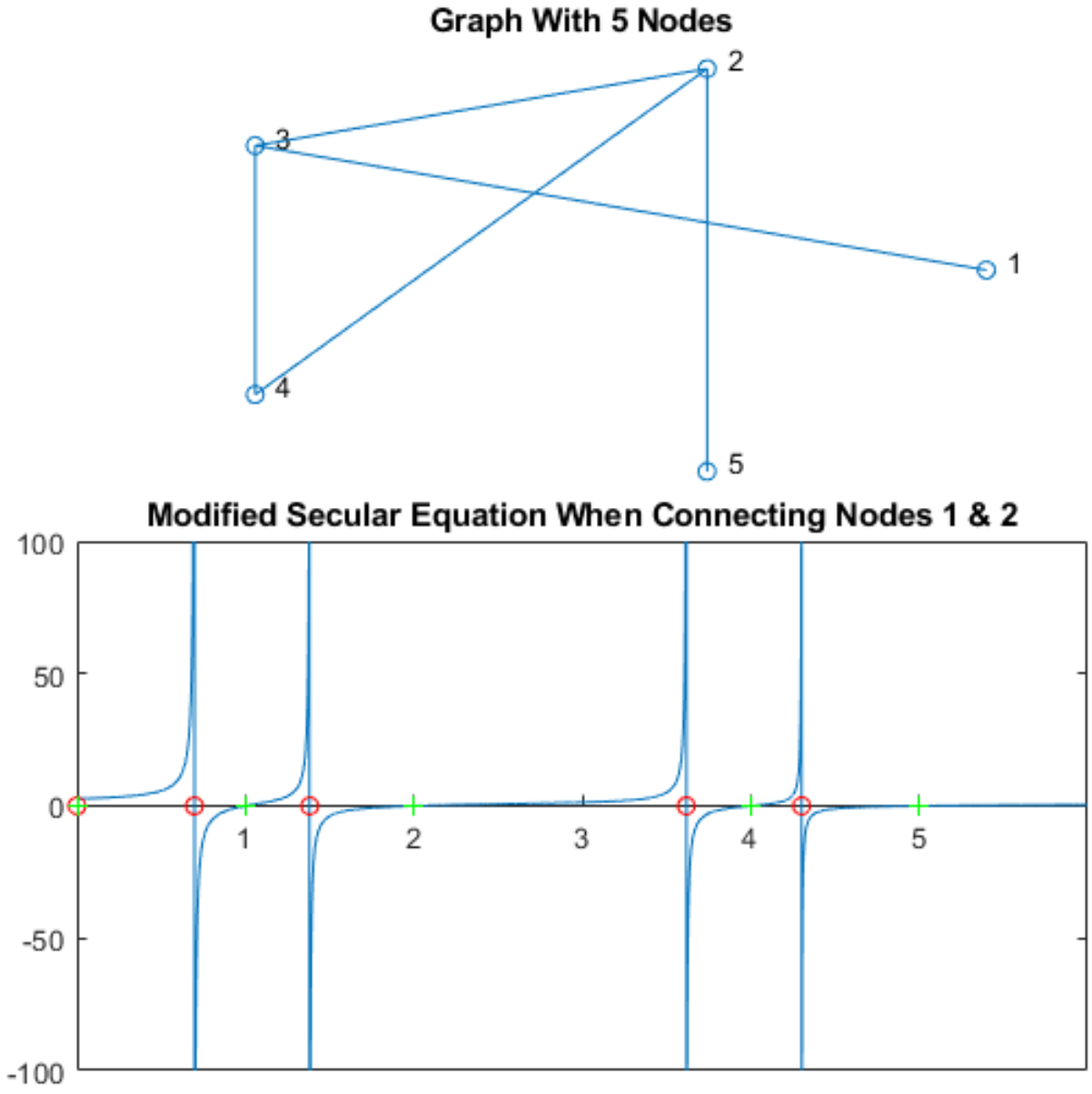

Figure 3.1 Top: Graph with Five Nodes Corresponding to the Plot Shown Below. Bottom: Modified Secular Equation When Connecting Nodes 1 and 2.

3.1.3 Tabu Search

The tabu search was developed by Glover in 1989 [11]. This performs a depth-first search with search trajectory tracking. It maintains a list, the tabu list, of recent moves to avoid redundant searches. Additionally, it can include random jumps in the search space to attempt to break away from local optimums. Wei and Sun [35] created a version of the weighted tabu search for the maximization of algebraic connectivity when adding routes to air transportation networks. This method was adapted for connectivity tracking in UAVs. Our method operated on the unweighted Laplacian matrix to remain consistent between methods and since the UAVs are continuously moving, affecting the weights at any given moment. The tabu search algorithm by Wei, Chen, and Sun [7] is shown in Algorithm 3.

Algorithm 3: Tabu Search Method

Result: Node selection: node1, node2

Given: G, the original graph, P, the candidate edges, Φ, the number of iterations to perform, and k, the number of edges to be added

Randomly pick k edges from P to construct s_0

$s = s_0, \lambda_2^* = 0, s^* = s_0, T = \{\}$

for $iteration \leftarrow 1$ **to** Φ **do**

  for $p \leftarrow 1$ **to** k **do**

    construct N(s,p) of the pth edge in s

  end

  while *1* **do**

    Pick one edge p' from each N(s,p) to construct s'

    if $\lambda_2(s') > \lambda_2^*$ **then**

      $s = s'$

      Update T

      $\lambda_2^* = \lambda_2(s)$

      $s^* = s$

    end

    if $s' \notin T$ **then**

      $s = s'$

      Update T

    end

  end

end

output λ_2^* *and* s^*

3.1.4 Semidefinite Programming Method

The semidefinite programming method [10] uses the fact that the space of $n \times n$ positive semidefinite matrices, which includes the Laplacian graphs, is a cone [40], allowing convex optimization methods.

The addition and deletion problem is shown in (3.6) as a binary programming problem,

$$\begin{aligned} \text{maximize} \quad & \lambda_2(L(E_{base} \cup E)) \\ \text{subject to} \quad & |E| = k, \\ & E \subseteq E_{cand}, \end{aligned} \tag{3.6}$$

[10] where L is a starting graph Laplacian, E_{base} is the initial set of edges in L, E is a selected set of edges from E_{cand}, the set of candidate edges, and k is the number of edges selected from E_{cand}.

This problem, however, is not suitable for convex optimization solvers. A relaxation is necessary to restructure the problem as a convex optimization problem. To that end, Ghosh and Boyd [10], removed the requirement for a binary selection of edges in favor of a continuous selection value. In this way, the selection vector could contain any value in the range [0, 1] instead of strictly binary values of either zero or one.

Consider the selection variable $x \in \{0, 1\}^{m_c}$, where m_c is the number of candidate edges. Then,

$$L(x) = L_{base} + \sum_{l=1}^{m_c} x_l a_l a_l^T, \tag{3.7}$$

[10] where x is a selection vector, l is an index to the set of candidate edges, x_l is the binary selection value applied to a candidate edge, and a_l is the incidence vector for edge l gives $L(x)$, the new Laplacian after adding the selected edges.

Replacing the binary selection $x \in \{0, 1\}$ with a linear weighted selection $x \in [0, 1]$ results in the following relaxation,

$$\begin{aligned} \text{maximize} \quad & \lambda_2(L(x)) \\ \text{subject to} \quad & \mathbf{1}^T x = k, \\ & 0 \leq x \leq 1, \end{aligned} \tag{3.8}$$

[10] where $L(x)$ is the Laplacian resulting from adding k edges selected by x. $\mathbf{1}^T x = k$ indicates that the sum of the selection vector x is k, which means k edges are selected. The problem shown in 3.8 is a convex optimization problem with linear constraints and a concave objective function [10].

With this relaxation comes the need to further address the selection of edges. In the binary program, the selected edges were clearly identified with a one indicating the edge is selected and a zero indicating it is not, but in the relaxed convex problem the solution is a vector of decimal values for each candidate edge. There are multiple methods that could be used for selecting the edges based on the non-binary selection vector. The simplest approach, and the one used for our simulation, is to choose the edge with the maximum value in the solution vector.

To solve this problem CVX, a package for specifying and solving convex programs [41], [42], was used with the default, free solver SDPT3.

3.2 Simulation

A Graphical User Interface (GUI) and simulation environment was developed in MATLAB. A screenshot from the GUI environment is shown in Fig. 3.2.

In the lower right corner of the graph window current connectivity value is shown. In the lower left corner a drop-down menu is available to select which algorithm to use in the simulation. Below that are play and step buttons to play the simulation or step through it one movement at a time. Below that is the button that presents a window where the user can enter a desired connectivity value or profile, as a scalar or vector, respectively. Next are options to operate in weighted or unweighted mode. Finally there is a delete button and an export button at the bottom.

On the right side of the screen there are histories of the connectivity after every movement and after every click. On the bottom is a button to clear the graph window of all nodes and reset the displays. Below that is a button which will show a plot with the history of the connectivity along with the desired connectivity. On the left are the locations of the nodes. Finally, in the file menu, the user can save or open .gph files which contain the graph data in an ascii-text format.

A rudimentary connectivity profile tracking algorithm was developed to move simulated UAVs in a manner which allowed their overall connectivity to track a desired profile. The effectiveness of this algorithm using each of the four connectivity modification methods (Fiedler, SDP, tabu search, and bisection), as well as a brute-force search, was tested.

In order for these methods to be useful in connectivity tracking, several implementation details needed to be addressed. Additionally, all methods were originally intended to address the problem of maximally increasing the connectivity of a graph by adding an edge, however, for the purpose of connectivity tracking, three more cases were desired. The ability to maximally decrease

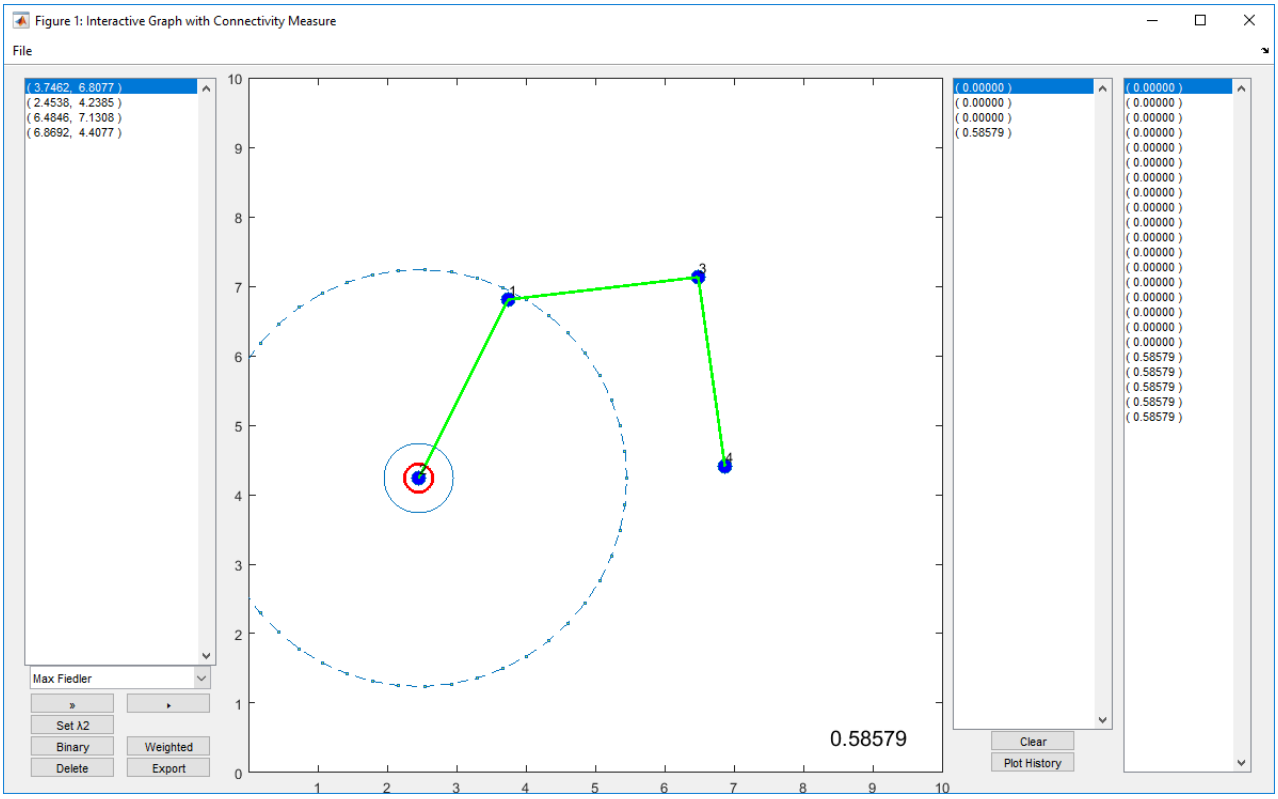

Figure 3.2 Connectivity Simulation Environment.

the connectivity by deleting an edge, the ability to minimally increase the connectivity by adding an edge, and the ability to minimally decrease the connectivity by deleting an edge. These four cases formed the foundation upon which the connectivity profile tracking algorithm was developed, as will be described later. In the following section, the necessary modifications for each algorithm are described.

3.2.1 Modification and Implementation of Methods

3.2.1.1 Fiedler

The Fiedler method has been extended to cover the four cases. The first case, as described in Chapter 2, is the edge to be added for a maximum increase in connectivity. The rationale behind this method is that the Fiedler vector provides an approximate gradient for the connectivity [10]. Based on this rationale, it is reasonable to expect a small squared difference between entries in the Fiedler vector to correspond to a small increase in the connectivity. Similarly, a large or small squared difference between entries corresponding to nodes with an edge to be deleted should correspond to a large or small decrease, respectively, in connectivity. This has been found to be true in practice and through simulations described below.

3.2.1.2 Bisection

In the bisection method, the secular equation used can be adapted for the deletion problem by changing the value of ρ from one to negative one [12] and searching for the edge which gives the largest negative value for $f(\lambda(G')^*)$ between λ_1 and λ_2 (instead of between λ_2 and λ_3). Additionally, as with the Fiedler method, the bisection method has been further modified to produce two additional cases, minimum increase and minimum decrease. To find the minimum increase

and decrease, one only has to search for the smallest values of $f(\lambda(G')^*)$ and choose the set of edges falling within the opposite extreme of the midpoint as compared to the maximum increase and decrease.

3.2.1.3 Tabu Search

The tabu search required only changing the criteria for which the algorithm was searching. For example, instead of searching for the edge which would result in the maximum increase, it could search for the edge which would result in the minimum increase. The algorithm itself remains unchanged otherwise.

3.2.1.4 Semidefinite Programming

The semidefinite programming method was adapted to provide a minimal increase by selecting the minimum value in the results vector, rather than the largest, as was the case for the maximum increase. To accommodate a decrease rather than an increase in connectivity, the SDP was modified to include a subtraction of an edge rather than an edge addition. The maximum decrease was found by selecting the minimum value in the result vector of the edge deletion SDP. The maximum increase was found by selecting the maximum value in the result vector of the same SDP.

3.2.2 Comparison of Each Method

As a precursor to our connectivity tracking simulation, a comparison of the methods for solving the addition/deletion problem was conducted. Each method was tested on all combinations of graphs up to size seven. Due to the rate of growth of the number of combinations for each single

increase in size, evaluating all graphs of each size above size seven quickly became impractical. Evaluating all graphs up to size seven allows a complete comparison for small UAV teams, which are more common due to the reduced resource requirements. It also provides an opportunity to begin identifying trends in the performance of each algorithm as the graph size grows. More importantly, it demonstrates the effectiveness of each algorithm relative to each other. The fully-connected and empty graphs were not considered since either no edges could be added or no edges could be removed.

To test each method, every combination of initial graph was generated. Through brute force, every possible edge addition or deletion, for each initial graph, was evaluated and the connectivity computed. This gave the true maximum connectivity change for each initial graph. Then, each method was applied to the same initial graph and the best edge to add or remove based on that method was compared with the results of the brute-force search. If the method produced the same level of connectivity, although not necessarily the same exact edge choice, as the brute-force method, it was counted as successful. The number of successful trials for each graph size and method are shown in Table 3.1 and Table 3.2.

Table 3.1 Number of Successful Edge Selections in Determining Maximum Increase in Connectivity.

Method	**3**	**4**	**5**	**6**	**7**
SDP	6	50	732	22,056	1,284,126
Fiedler	6	55	788	23,599	1,301,448
Tabu Search	6	62	1,022	32,397	2,008,810
Bisection	6	62	1,022	32,766	2,097,150
Total	6	62	1,022	32,766	2,097,150

Table 3.2 Number of Successful Edge Selections in Determining Maximum Decrease in Connectivity.

Method	**3**	**4**	**5**	**6**	**7**
SDP	6	62	901	24,153	1,872,214
Fiedler	6	62	1,017	30,655	1,884,122
Tabu Search	6	62	1,022	32,452	1,995,938
Bisection	6	62	1,022	32,766	2,097,150
Total	6	62	1,022	32,766	2,097,150

3.2.3 Connectivity Profile Tracking

Using the four methods previously described, a rudimentary connectivity tracking algorithm, shown in Algorithm 4, was developed. First, a decision on whether to add or remove an edge or to simply expand or contract the graph is made. At one extreme, a weighted graph becomes equivalent to an unweighted graph when the edge distances are small and all edge weights become one. At the other extreme, expanding the graph until the weight edges approach zero will become equivalent to a fully disconnected graph. By expanding and contracting the graph, any level of connectivity can be achieved, however, this is not a very useful technique in practice. Therefore, this algorithm makes a determination on whether the graph should expand or contract, or if an edge should be added or removed.

Two issues which were discovered while creating this algorithm are firstly, without taking into account existing edges, it is easy for a move which is made to connect a new edge to disconnect one or more existing edges. This has the very real possibility of leaving the graph disconnected with a connectivity of zero. In order to account for this, a recursive check of all existing edges was incorporated to ensure that they were not broken during the move. If an edge was broken, the node

Algorithm 4: Connectivity Tracking

```
Given: Connectivity profile P as an array
λ2 and λ3, the second and third smallest eigenvalues of the initial weighted graph
λ2,upper, the second smallest eigenvalue of the initial unweighted graph
λ2,lower, the connectivity obtained by a maximum decrease deletion
for i ← 1 to size(P) do
    λ2,desired = Pi
    while λ2 ≠ λ2,desired do
        if λ2 < λ2,desired && λ2,desired ≤ λ2,upper then
            Contract edge with largest sum of distances
        else if λ2 > λ2,desired && λ2,desired > λ2,lower then
            Expand edge with smallest sum of distances
        else
            if λ2 < λ2,desired then
                if λ2,desired > λ2 + (λ3 − λ2)/2 then
                    Maximal increase based on current method
                else
                    Minimal increase based on current method
            else
                if λ2,desired < λ2/2 then
                    Maximal decrease based on current method
                else
                    Minimal decrease based on current method
        Move selected nodes, checking for broken links, stagnation, and repeated moves
        while Broken Link do
            Move nodes to re-establish link
        end
        if Stagnation or Repeated Moves then
            Add a random offset to the move
    end
end
```

connected to that edge which did not move, would be moved to re-establish the connection. It is possible for this to create a daisy-chain effect of breaking connections and re-establishing them.

A second issue that occurred was when a series of moves would inadvertently connect new edges, causing an immediate jump in connectivity beyond the target connectivity. On the next move, the same nodes would move apart to reduce the connectivity. This led to cycling in some cases which prevented the algorithm from completing the profile.

To account for both of these issues, a check for cycling was implemented. Whenever the connectivity remained unchanged for an extended number of moves, or the same nodes were repeatedly moved, a small random offset would be added to the movement of the nodes. This provided enough variation to allow the algorithm to break out of cycles.

3.3 Experiment Design

In order to evaluate the effectiveness of the four methods in tracking a desired connectivity profile with the proposed tracking algorithm, an experiment was designed in which a series of connectivity profiles were simulated with each method at a range of graph sizes. Connectivity profiles fell into the following five categories: increasing, decreasing, sine wave, square wave, and random (See Fig. 3.3). These profiles were selected to provide a variety of connectivity transitions which might occur.

Graph sizes of one to twenty-five nodes were tested. Twenty-five was chosen as a large graph size that was still within practical limits of the simulation. Although conclusions regarding the performance of the algorithms at larger and larger graph sizes cannot be made completely based on smaller size simulations, it is reasonable to expect these trends to continue as the size grows.

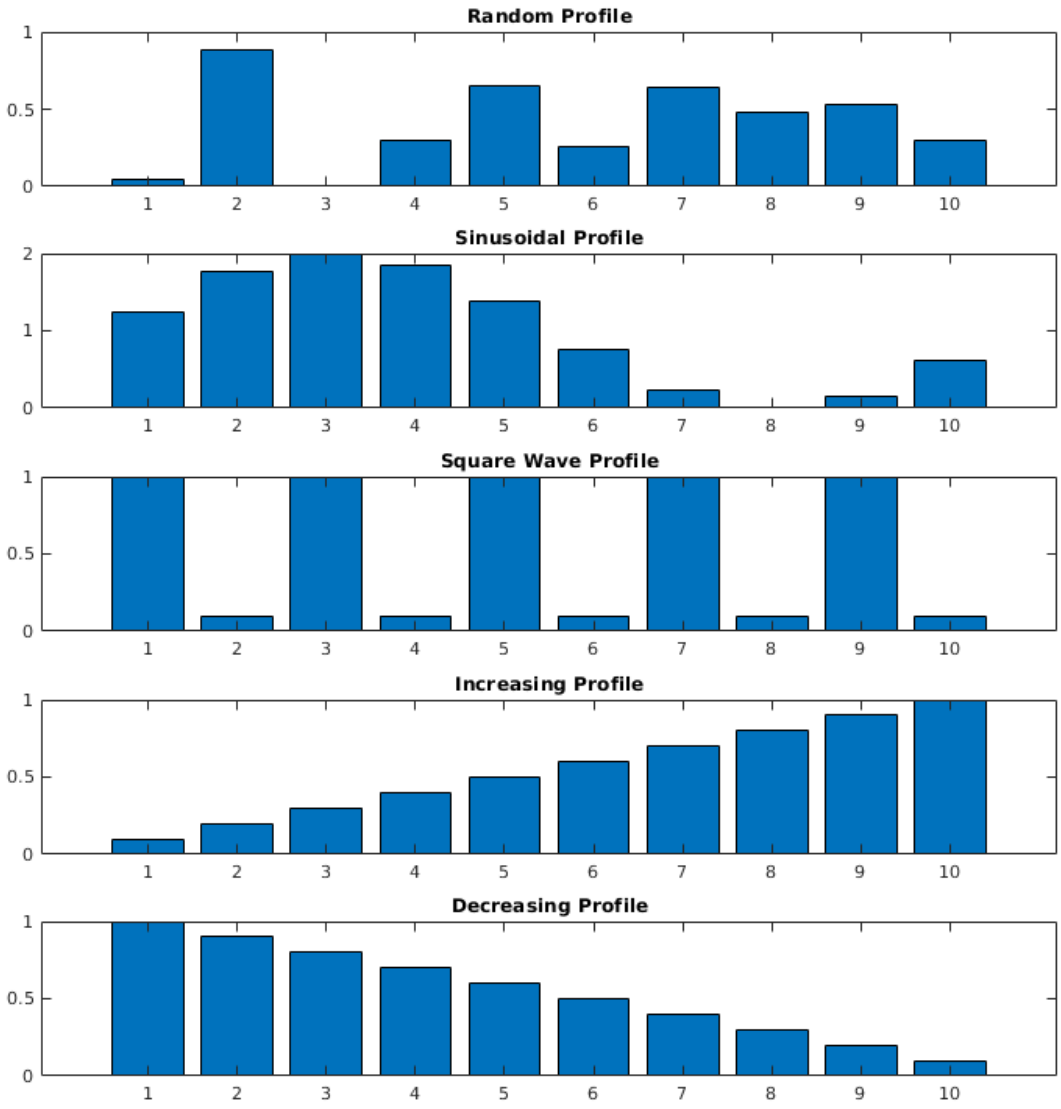

Figure 3.3 Top to Bottom: Random Profile, Sinusoidal Profile, Square Wave Profile, Increasing Profile, Decreasing Profile.

Nevertheless, these simulations and results do give a reliable indication of performance for the graph sizes within their scope and UAV teams of size twenty-five and below can be used in a large number of cooperative UAV scenarios.

Within each graph size, five randomly generated graphs were tested. The same five graphs for a given size were used with each method. For each random graph of each size, the five profiles were tested using the tracking algorithm with each of the four methods for selecting the best edge to add or delete. The same test was also completed using a brute-force method as a basis for comparison.

3.4 Results and Discussion

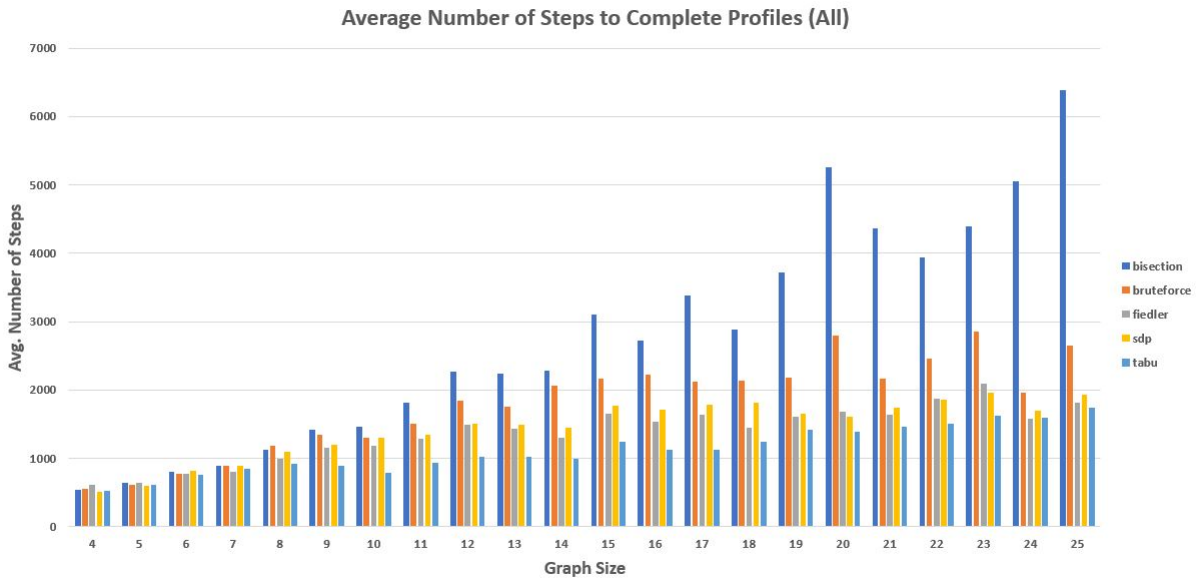

Figure 3.4 Average Number of Steps to Complete All Graphs and Profiles for Each Graph Size.

The graph shown in Fig. 3.4 shows the average number of steps required to complete all graphs of a given size, using a given method. From these results, it is immediately obvious that the bisection method is under-performing the other methods in being able to track the profile as it requires almost twice the number of steps to complete the profile. The reason for this comes from the behavior of this implementation of the bisection method for the cases where there are repeated eigenvalues.

At the beginning of the profile, the graph is typically disconnected, which means that the first- and second-smallest eigenvalues are zero. If there are more disconnected components, there may be even more eigenvalues which are zero. A known limitation of the bisection method, as discussed by Kim in [12], is the case where the eigenvalues are repeated. In this case, there is not a range within which the bisection can conduct its search. Although all edges will yield the same resulting connectivity in this case, some edges are better than others in the long term.

Another interesting aspect of the results comes from the brute-force method. The brute-force method does a complete search of all possible edge choices and returns an edge that yields the largest or smallest increase or decrease in the connectivity value, as desired. However, the selected edge is not guaranteed to be a good choice in terms of how quickly a connectivity level is attained over time as multiple edges are added or removed. Additionally, the brute-force method suffers from a similar situation as the bisection method does at the beginning in that it cannot differentiate between edges if there are more than two disconnected components in the graph. It also cannot make a good choice when there is a repeated λ_2. In these cases, no edge choice will result in an increase in connectivity. The brute-force method in this implementation returned the last edge evaluated, which is the edge with the largest node index. This causes this method to select many

non-optimal choices when starting from a connectivity of zero and to make non-optimal choices throughout whenever a repeated eigenvalue occurs.

Since the initial graphs were formed by randomly placing the UAVs within a rectangular area, many UAVs are not connected initially. To determine if starting with a connected graph impacted the results, the average number of steps required to complete all profiles was adjusted by the number of steps required to reach a non-zero connectivity for each run of the simulation and is shown in Fig. 3.5. In this case, the brute-force method was close to the number of steps required for other methods, however, the bisection method still significantly under-performed the other methods. A possible reason for this is the type of structure created by the bisection method during the zero connectivity phase. While the brute-force constructs a graph that is nearly complete, the bisection method creates a star arrangement.

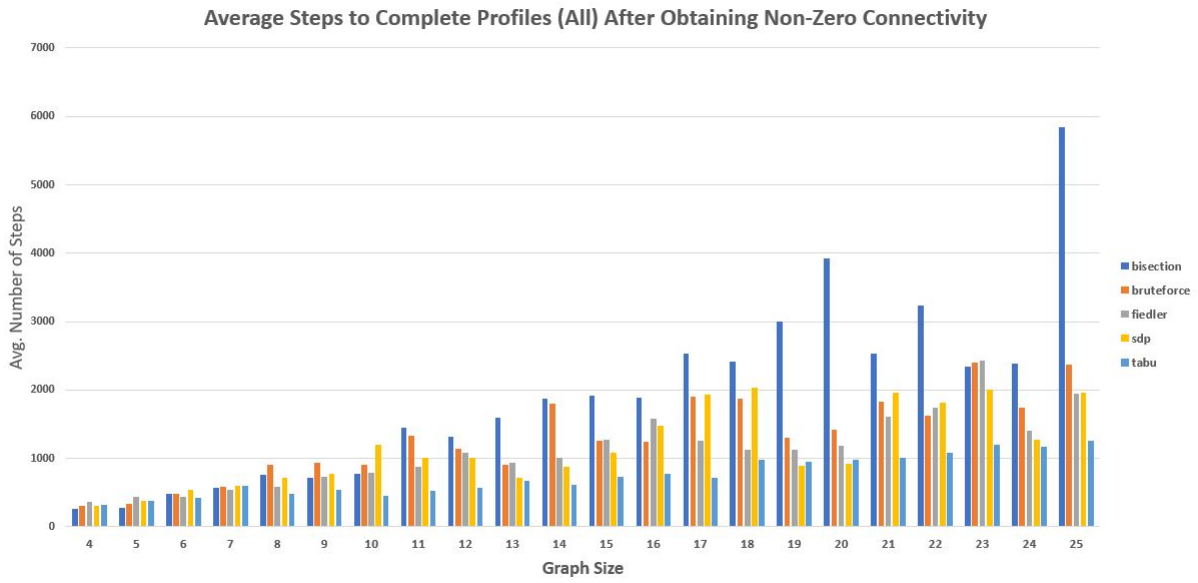

Figure 3.5 Average Number of Steps to Complete All Graphs and Profiles for Each Graph Size, Adjusted by Number of Steps Required to Reach Non-zero Connectivity.

When looking at the change in connectivity achieved by starting with an empty, 25-node graph and repeatedly adding the ideal edge according to each method, it becomes apparent that the brute-force method is building in a way that takes many steps before achieving connectivity (see Fig. 3.6).

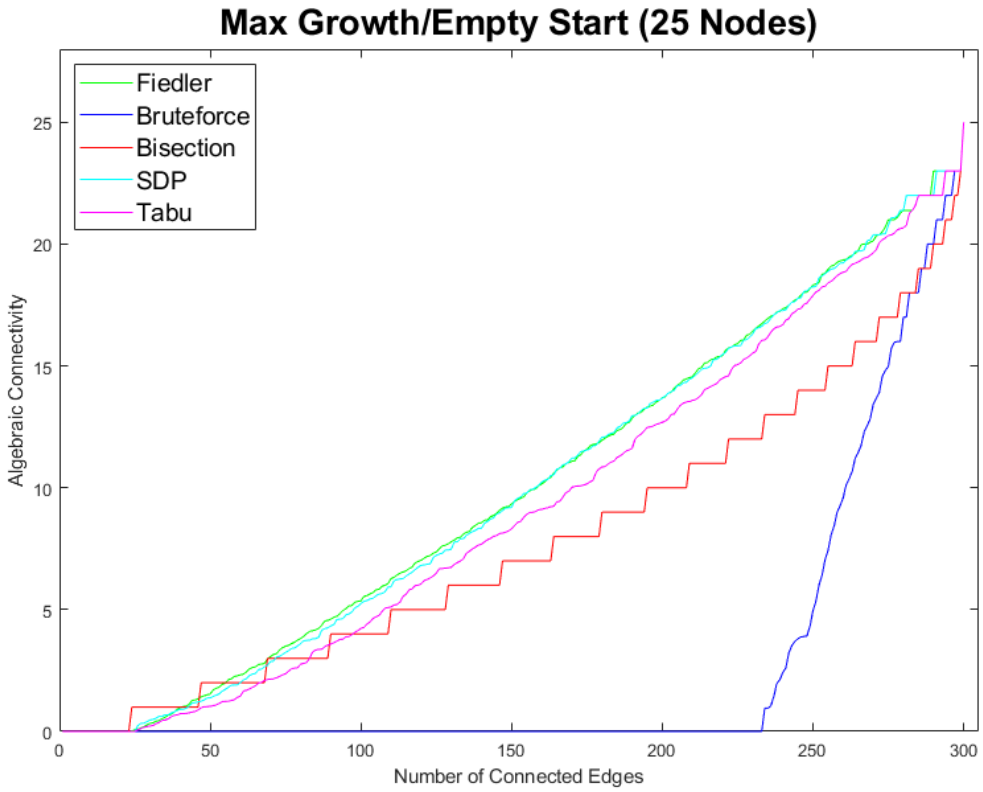

Figure 3.6 Graphs Grown by Selecting the Recommended Edge Using Each method, Starting with an Empty, 25-node Graph.

Interestingly, when starting with a 25-node line graph, the results are much closer to each other (see Fig. 3.7). This suggests that the starting configuration can have a significant impact on how connectivity should be adjusted. Additionally, how edges are chosen at each step can

drastically change the resulting connectivity profile. Kim discussed the potential for a graph to deviate from an optimal configuration by repeated edge additions [12]. This highlights the fact that optimizing a single edge addition or deletion does not guarantee the optimality of a sequence of edge additions or deletions.

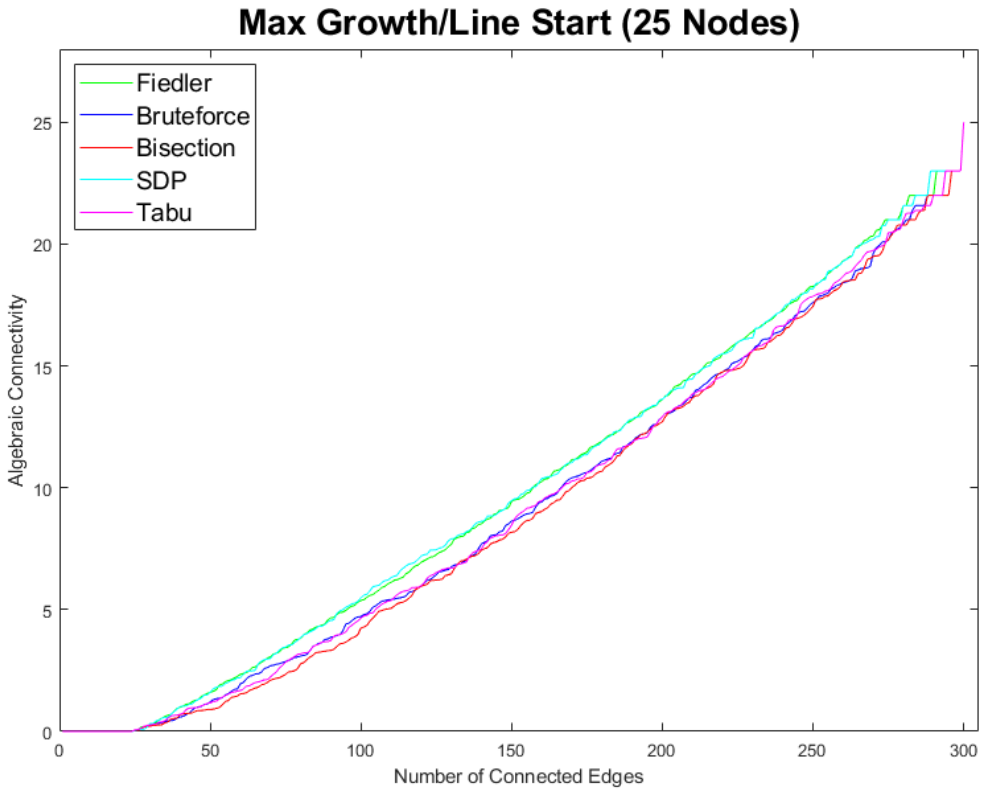

Figure 3.7 Graphs Grown by Selecting the Recommended Edge Using Each Method, After Starting with a 25-node Line-graph.

The semidefinite programming method and the Fiedler method gave similar results in terms of the number of steps required to complete the profiles, with the Fiedler method tending to be

slightly better. However, computationally, the Fiedler method executed much more quickly than the semidefinite programming solver, as was suggested by Ghosh and Boyd [10].

The tabu search consistently performed well. For all profiles and graph sizes, it was either on par or better than the other methods. The reason for this is that the tabu search covered a good portion of the search space, but rather than choosing the first edge from the options in some sorted order, its edge selection had a component of randomness to it. This turned out to give it a large advantage over the other methods, effectively making it a near-brute-force search with random selection.

Looking at the results profile by profile, it is clear that the steadily increasing and decreasing profiles required fewer steps to complete. As an example, for size 25 graphs, on average, the random profile required 3,491 steps, the square profile required 3,936, and the sine profile required 3,091 steps, while the increasing profile required 1,790 steps and the decreasing profile required 2,234 steps.

3.5 Summary

Four methods of tracking connectivity profiles were presented. The connectivity tracking problem was identified as the addition and deletion of links among nodes and modification of edge-weights via repositioning as a connectivity is incrementally changed. The tracking methods were implemented and evaluated for a variety of graph sizes and connectivity profiles using each method.

Strengths and weaknesses of this approach were demonstrated through the simulation study. The effectiveness of the four methods in determining the best edge for addition or deletion was

compared for all graphs up to seven nodes with the bisection method, followed by the tabu search, performing the best in both cases. The impact of the initial configuration on the ability of the formation to track a profile was also shown. It was found that different edges may result in the same connectivity after addition or deletion but have differing impact on later addition or deletion steps. In this respect, the SDP and Fiedler methods out-performed the other methods. Finally, simulation suggested that a connectivity profile which is continually increasing or decreasing may reach its targets more quickly than profiles with more variation (square wave, sine wave, or random profiles).

Some limitations of the simulation that were conducted include the use of a heuristic algorithm and simple motion model rather than a formalized controller and dynamics model which can track connectivity over time. Also, many of the methods can be implemented in a variety of ways using a range of parameters. It is possible to achieve different performance using a different implementation of the same method or using different parameters.

CHAPTER 4

HYBRID METHOD OF SELECTING EDGES

In this chapter a new hybrid algorithm which uses a random forest classifier to blend the greedy perturbation heuristic [10] and the bisection method [12] is developed. The hybrid method chooses links to add or remove in order to adjust the connectivity of a networked system. The effectiveness of methods for choosing the edge to be added or deleted using this algorithm is evaluated using a brute-force comparison on graphs up to size seven, a sampling of Erdős-Rényi random graphs up to size 25, and a connectivity tracking simulation [22].

Consider the graph shown in Fig. 4.1. According to the Fiedler method, the edge to add which will have the largest increase in connectivity would be between $node_2$ and $node_3$, $node_2$ and $node_4$, $node_6$ and $node_3$, or $node_6$ and $node_4$, since all of these edges correspond to a squared difference of 0.7887. However, all of these edges result in an algebraic connectivity of 0.5188 when added to the graph while the edge between $node_7$ and $node_1$ results in a higher algebraic connectivity of 0.5505.

Despite a relatively low accuracy, this method was the most efficient as only the Fiedler vector needs to be computed. The efficiency can also be improved by not computing the Fiedler vector to a high degree of accuracy, since only the values relative to other pairs of nodes are important [10]. Finally, by noting that the maximum squared difference will occur between the largest and smallest nodes in the vector, one only has to iterate through the vector one time to

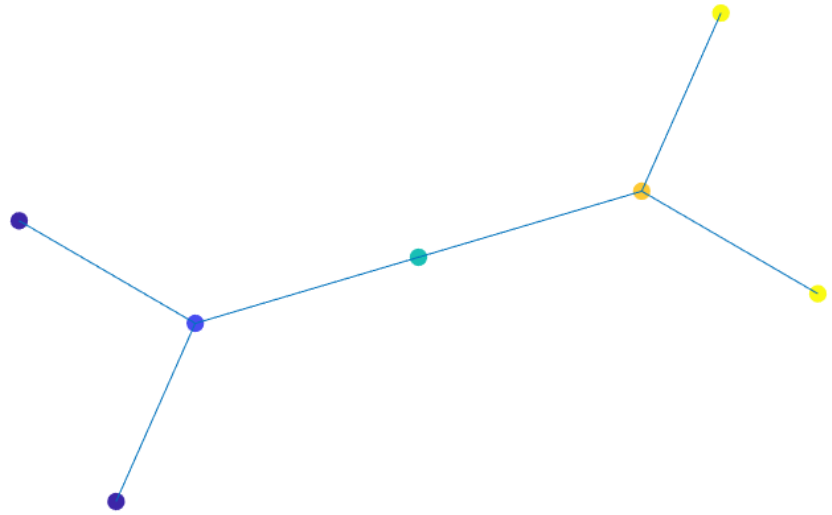

Figure 4.1 A Graph with Seven Nodes On Which the Fiedler Method Fails.

find the maximum and minimum values, resulting in an $O(n)$ operation. The overall complexity is therefore $O(\frac{4}{3}n^3 + n)$ if all eigenvectors are computed.

The bisection method, as discussed in Chapter 3, is an algorithm developed by Kim [12] for choosing an edge based on the modified form of the secular equation derived from a rank-one modification. To determine a maximal increase in connectivity by adding an edge, the algorithm performs the following: It first specifies a range with the lower boundary being $lower = \lambda_2$ and the upper boundary being $upper = \lambda_3$. Then, the midpoint is defined as $m = \frac{\lambda_2 + \lambda_3}{2}$. The algorithm repeatedly sorts all possible edges as giving rise to a connectivity that will be below or above the midpoint. If at least one edge is above the midpoint, the edges below it are discarded and the range is updated to be from m to the upper boundary. If no edges are above the midpoint, the lower range

of the lower boundary to m is used. The process is then repeated until a single edge remains or a maximum number of iterations have occurred.

The bisection method is exact and will yield the best edge choice but it is less efficient than the Fiedler method, having a complexity of $O(4n^3/3 + 4mn\,log_2(d/\epsilon))$ [12] where n is the number of nodes, m is the number of candidate edges, d is the distance between the upper and lower search boundary, and epsilon is a user chosen parameter. Like the Fiedler method, this method requires computation of eigenvectors and eigenvalues of the graph. In addition, a calculation must be performed each time an edge is evaluated, with every edge being calculated at least once, and the best edges being calculated multiple times.

Chapter 3 introduced the concept of connectivity tracking [22]. Connectivity tracking is based on two tenets. First, the desired level of connectivity for a multi-robot team is situation dependent. Second, the desired level of connectivity will change over time. These two fundamental ideas provide motivation for a connectivity tracking algorithm which can control a team to maintain levels of connectivity appropriate for their current position within a desired profile.

4.1 Random Forest

A tool which is relied upon heavily in this chapter is the random forest. The random forest is built on the decision tree classifier, where classifications are performed based on a series of questions, or discriminating decisions [43]. The classification process begins at the top, root node of the decision tree. At each node the branch corresponding to the current decision is followed to the descendent node. Upon reaching the final node, or leaf-node, the final classification is made.

An extension of the decision tree is the random forest. A random forest consists of an ensemble of decision trees that vote on the classification [44]. Random forests can result in significant improvements over single trees while avoiding over-fitting [44]. There are various methods for growing the ensemble of decision trees. Bootstrap-aggregating (also known as bagging) decision trees is one method [44] and is the method used in this work.

The random forest, and decision trees in general, have several advantages. First, they are easier to interpret than other types of classifiers, although a dense random forest may also be difficult to interpret. They are also efficient, requiring only a number of operations proportional to the depth of the tree. Finally, they are able to work with non-linear problem sets.

4.2 Hybrid Algorithm

The bisection method is highly accurate, but suffers from an unfavorable rate of growth. The Fiedler method, on the other hand, is much more efficient, but also much less accurate. These two facts have motivated the development of a hybrid algorithm for selecting the best edge to add to or delete from a graph to reach a desired change in connectivity.

4.2.1 Algorithm

The proposed algorithm is shown in Algorithm 5. This algorithm first performs a prediction, using a bootstrap-aggregated random forest, to predict if the Fiedler method of selecting an edge will give an optimal result.

If the Fiedler method is predicted to be successful, the algorithm checks if the choice is ambiguous. An ambiguous choice, with respect to the Fiedler method, occurs whenever more than one edge results in a maximum squared difference between its corresponding node's entries in the

Fiedler vector. If there is an ambiguous choice, the bisection method is performed on only the reduced set of candidate edges which have the same maximum squared difference. If the choice is not ambiguous, the result of the Fiedler method is returned.

If the Fiedler method is predicted to be unsuccessful, then the result of the Bisection method is returned.

Algorithm 5: Hybrid Algorithm for Selecting an Edge

```
Result: Node selection: node_1, node_2
Given: G, the graph laplacian
   mdl, a pre-trained bootstrap-aggregated random forest model
Using mdl, predict if the fiedler method will be successful
if Fiedler is Predicted to be Successful then
    if Choice is ambiguous then
        Return Bisection on Reduced Set of Candidate Edges
    else
        Return Fiedler
    end
else
    Return Bisection
end
```

4.2.2 Decision Tree Design

A key component of the hybrid algorithm is the random forest classifier which is used to predict if the Fiedler method will be successful. The random forest was chosen for several reasons. First, the way that it makes its prediction is able to be interpreted. The decision trees consist of a series of discriminating choices based on the values in the feature vector which can be traced to determine how a prediction was made. Second, the additional computational overhead is very low since it only requires following a fixed number of branches. Third, the classification performance

is generally quite good. This section will discuss the development of the random forest classifier.

4.2.2.1 Feature Selection

Twenty-six features to be used with the model were explored. It is important to use features that are based on the original Laplacian matrix and its eigenvectors and eigenvalues, since these are required for both the Fiedler and bisection methods. The additional cost of computing the feature vector is minimal when using such features. The features that are used are described in the list below.

1. The maximum squared difference between any two entries in the Fiedler vector.
2. The difference between the maximum and minimum value in the Fiedler vector.
3. The sum of all values in the Fiedler vector.
4. The minimum value in the Fiedler vector.
5. The maximum squared difference between any two entries in the eigenvector associated with the third-smallest eigenvalue (i.e. the third eigenvector).
6. The difference between the maximum and minimum value in the third eigenvector.
7. The sum of all values in the third eigenvector.
8. The minimum value in the third eigenvector.
9. The maximum squared difference between any two entries in the eigenvector associated with the largest eigenvalue (i.e. the nth eigenvector).

10. The difference between the maximum and minimum value in the nth eigenvector.

11. The sum of all values in the nth eigenvector.

12. The minimum value in the nth eigenvector.

13. The spectral gap (i.e. $\lambda_n - \lambda_2$).

14. A boolean flag that is set if any eigenvalue is repeated.

15. The algebraic connectivity, λ_2.

16. A boolean flag that is set if any entry in the Fiedler vector is zero.

17. Average of the Fiedler vector.

18. Variance of the Fiedler vector.

19. The sum of the degrees of the nodes selected by the Fiedler method.

20. The maximum degree of any node.

21. The minimum degree of any node.

22. The average degree of the graph.

23. The number of negative values in the Fiedler vector.

24. The number of positive values in the Fiedler vector.

25. The multiplicity of the maximum squared difference in the Fiedler vector (i.e. The number of edges which have the same, maximum squared difference).

26. The difference between the sum of all positive values in the Fiedler vector and the absolute value of the sum of all negative values in the Fiedler vector.

Features 1-4 were chosen due to the importance of the Fiedler vector to the Fiedler method as well as the vectors' association with algebraic connectivity. There is a relationship between the Fiedler vector and the nth eigenvector (associated with the largest eigenvalue). In particular, the nth eigenvector is the Fiedler vector of the graph's complement [45]. Due to this relationship, the same set of features (5-8) was used for the nth eigenvector. Similarly, some characteristics of a graph have been associated with properties of the third eigenvector [46]. For this reason, the same set of features (9-12) based on the third eigenvector were also included.

Feature 13, the spectral gap between the largest and second-smallest eigenvalues was chosen based on intuition and the relationship between λ_n and λ_2.

Feature 14, a boolean value indicating if an eigenvalue is repeated, was chosen in part due to the properties of repeated eigenvalues for graph Laplacians. In particular, when the smallest eigenvalues λ_1 through λ_k are zero, there are k disconnected components in the graph. Additionally, when λ_2 has a multiplicity of r, at least r edges must be added to increase λ_2 [10].

Feature 15, the algebraic connectivity, was chosen since the algebraic connectivity is associated with the Fiedler vector.

Features 17 and 18, the mean and variance of the Fiedler vector were chosen since the Fiedler method is based on the entries in the Fiedler vector and these properties describe the entries.

Features 19, 20, 21, and 22 correspond to the sum of the degrees chosen by the Fiedler method, the maximum and minimum degree of any node in the graph, and the average degree of

the graph. These features were chosen due to the relationships between the degrees of the nodes, maximum and minimum cuts, and algebraic connectivty [47].

Features 23, 24, and 25 are the number of negative entries in the Fiedler vector, the number of positive entries in the Fiedler vector, and the number of edges with a maximal squared difference between entries in the Fiedler vector. These features were chosen due to the Fiedler method's reliance on the Fiedler vector entries.

Feature 26, the difference between the absolute value of the sum of positive entries and the absolute value of the sum of negative entries in the Fiedler vector was chosen based on the Fiedler method's reliance on the Fiedler vector entries and the ability of the signs of the entries to partition the graph.

4.2.2.2 Hyper-parameters

The bootstrap-aggregated decision trees were constructed and executed using the MATLAB $TreeBagger$ function. This function accepts as inputs the number of decision trees to use in the random forest, the feature vectors, and the class labels. Additional parameters allow for sampling with replacement and specifying a minimum leaf size. For this research, the default values of sampling with replacement turned on and a minimum leaf size of one were used.

The number of decision trees to use was set to 25. The generalization error of random forests converges to a limiting value as the number of trees is increased [44], therefore increasingly large values for the number of trees used in the random forest tend to give little to no improvement in classification. Twenty-five was chosen as a reasonably large number.

4.3 Analysis

To study the effectiveness of this approach, random forests were trained for each graph size to be tested. The training data was generated as a set of Erdős-Rényi graphs with edge probability of 0.5 so that every graph of size n was equally probable. The training data size was the smaller of 50,000 or 20% of the total number of graphs of a given size. The hybrid method was tested on all graphs up to size seven and compared with previous results for the Fiedler method and the Bisection method. Additionally, a random sampling of 1,000 graphs of each size from 8 to 25 nodes (UAVs) were tested.

A simulation was also conducted to track five connectivity profiles using five graphs with 25 nodes. The profiles used were a random profile, a sinusoid, a square wave, a linearly increasing profile, and a linearly decreasing profile. The graphs and profiles used were the same as those used in Chapter 3. The method of tracking connectivity was also discussed in that chapter.

The tracking method involved evaluating the target algebraic connectivity relative to the current algebraic connectivity level. If the target connectivity was much greater or smaller than the current connectivity, then an edge was added or deleted to reach the target connectivity. For both addition and deletion, the ability to make a maximum or a minimum increase or decrease was developed. This allows, for example, edges to be added to maximize the connectivity until the target connectivity is passed then remove edges that result in a minimum change in connectivity until the target connectivity is reached.

The number of steps to complete the profile for the same graph and same profile was determined and compared with the previous results using the Fiedler method and the bisection method.

4.4 Results and Discussion

The number of successful edge selections for each direction and type (minimum or maximum), for all graphs of size four through seven is shown in Table 4.1. From these results it is clear that the hybrid method yields a great improvement over the Fiedler method alone in choosing an edge that maximizes or minimizes the connectivity change.

Table 4.1 Number of Successful Edge Selections in Determining Maximum and Minimum Increase and Decrease in Connectivity for All Graphs Size 4 Through 7.

Size	Dir.	Type	Hybrid	Bisection	Fiedler	Total
4	-	Min.	62	62	62	62
5	-	Min.	1,018	1,022	946	1,022
6	-	Min.	32,716	32,766	25,437	32,766
7	-	Min.	2,096,803	2,097,150	1,471,795	2,097,150
4	-	Max.	62	62	62	62
5	-	Max.	1,022	1,022	1,016	1,022
6	-	Max.	32,707	32,766	30,680	32,766
7	-	Max.	2,096,671	2,097,150	1,883,974	2,097,150
4	+	Min.	62	62	58	62
5	+	Min.	1,022	1,022	930	1,022
6	+	Min.	32,745	32,766	27,885	32,766
7	+	Min.	2,097,134	2,097,150	1,825,276	2,097,150
4	+	Max.	62	62	50	62
5	+	Max.	1,022	1,022	788	1,022
6	+	Max.	32,766	32,766	23,651	32,766
7	+	Max.	2,096,717	2,097,150	1,300,302	2,097,150

The methods were also compared using 1,000 Erdős-Rényi graphs with equal probability for each size 8 through 25. The results of these tests are shown in Table 4.3 for increasing connectivity and Table 4.2 for decreasing connectivity. From these results, it is clear that the hybrid algorithm again improves on the Fiedler method.

Table 4.2 Number of Successful Edge Selections in Determining Maximum and Minimum Decrease in Connectivity for 1,000 Random Graphs of Each Size 8 Through 25.

Size	Dir.	Type	Hybrid	Bisection	Fiedler	Total
8	-	Min.	997	1,000	691	1,000
9	-	Min.	981	1,000	737	1,000
10	-	Min.	931	1,000	743	1,000
11	-	Min.	937	1,000	784	1,000
12	-	Min.	945	1,000	790	1,000
13	-	Min.	963	1,000	816	1,000
14	-	Min.	966	1,000	842	1,000
15	-	Min.	976	1,000	851	1,000
16	-	Min.	969	1,000	851	1,000
17	-	Min.	975	1,000	878	1,000
18	-	Min.	983	1,000	880	1,000
19	-	Min.	981	1,000	889	1,000
20	-	Min.	984	1,000	919	1,000
21	-	Min.	982	1,000	922	1,000
22	-	Min.	990	1,000	940	1,000
23	-	Min.	992	1,000	955	1,000
24	-	Min.	996	1,000	958	1,000
25	-	Min.	997	1,000	975	1,000
8	-	Max.	1,000	1,000	841	1,000
9	-	Max.	991	1,000	838	1,000
10	-	Max.	923	1,000	819	1,000
11	-	Max.	897	1,000	789	1,000
12	-	Max.	878	1,000	785	1,000
13	-	Max.	868	1,000	770	1,000
14	-	Max.	849	1,000	756	1,000
15	-	Max.	826	1,000	732	1,000
16	-	Max.	826	1,000	707	1,000
17	-	Max.	821	1,000	725	1,000
18	-	Max.	810	1,000	686	1,000
19	-	Max.	824	1,000	691	1,000
20	-	Max.	820	1,000	712	1,000
21	-	Max.	817	1,000	689	1,000
22	-	Max.	795	1,000	667	1,000
23	-	Max.	821	1,000	668	1,000
24	-	Max.	792	1,000	656	1,000
25	-	Max.	801	1,000	644	1,000

Table 4.3 Number of Successful Edge Selections in Determining Maximum and Minimum Increase in Connectivity for 1,000 Random Graphs of Each Size 8 Through 25.

Size	Dir.	Type	Hybrid	Bisection	Fiedler	Total
8	+	Min.	999	1,000	876	1,000
9	+	Min.	985	1,000	892	1,000
10	+	Min.	962	1,000	894	1,000
11	+	Min.	962	1,000	886	1,000
12	+	Min.	970	1,000	900	1,000
13	+	Min.	974	1,000	878	1,000
14	+	Min.	979	1,000	883	1,000
15	+	Min.	982	1,000	882	1,000
16	+	Min.	993	1,000	917	1,000
17	+	Min.	992	1,000	913	1,000
18	+	Min.	996	1,000	916	1,000
19	+	Min.	997	1,000	938	1,000
20	+	Min.	999	1,000	948	1,000
21	+	Min.	997	1,000	960	1,000
22	+	Min.	996	1,000	965	1,000
23	+	Min.	997	1,000	983	1,000
24	+	Min.	999	1,000	983	1,000
25	+	Min.	1,000	1,000	989	1,000
8	+	Max.	999	1,000	574	1,000
9	+	Max.	971	1,000	541	1,000
10	+	Max.	911	1,000	528	1,000
11	+	Max.	924	1,000	527	1,000
12	+	Max.	870	1,000	486	1,000
13	+	Max.	905	1,000	520	1,000
14	+	Max.	882	1,000	512	1,000
15	+	Max.	895	1,000	499	1,000
16	+	Max.	894	1,000	534	1,000
17	+	Max.	892	1,000	522	1,000
18	+	Max.	901	1,000	505	1,000
19	+	Max.	900	1,000	489	1,000
20	+	Max.	865	1,000	525	1,000
21	+	Max.	902	1,000	512	1,000
22	+	Max.	859	1,000	477	1,000
23	+	Max.	849	1,000	490	1,000
24	+	Max.	874	1,000	486	1,000
25	+	Max.	880	1,000	505	1,000

It is important to understand the cases in which an edge selection is successful or unsuccessful. Successful edge selections indicate either a correct prediction was made or a false negative occurred. In the case of a false negative, the classifier will predict that the Fiedler method does not work when it in fact does. In this case, it will fall back to the bisection method, which always works, resulting in a successful edge, so the overall results for the algorithm will improve when a false negative occurs. When the algorithm is unsuccessful in selecting the best edge, the classifier has given a false positive, predicting that the Fiedler method would work when it actually would not.

In a test of 1,000 edge selections each on random, size eight graphs for maximum increase, maximum decrease, minimum increase, and minimum decrease, the hybrid method correctly predicted that the Fiedler method would work 3,000 times. The hybrid method also correctly predicted that the Fiedler method would fail 546 times. It incorrectly predicted that the Fiedler method would work only 5 times, and incorrectly predicted that it would fail 449 times. In this case, it made correct predictions 88.65% of the time and in the instances in which it made an incorrect prediction, 98.9% of those were false negatives. A false negative is preferred over a false positive since the method falls back to the bisection method in those cases, which always works.

Table 4.4 shows the average number of steps required in simulation to complete the five profiles (random, sine wave, square wave, increasing, and decreasing) for five random graphs of size 25. Size 25 was chosen since it had the largest difference between methods [22] and, in general, the larger the graphs become, the more pronounced the difference between methods. From this data it is apparent that the hybrid method gives performance between that of the Fiedler method and the Bisection method.

Table 4.4 Average Number of Steps to Complete All Five Profiles for All Five Size 25 Graphs.

Hybrid	**Bisection**	**Fiedler**	**SDP**	**Tabu**	**Bruteforce**
4983.64	6391.6	1820.68	1935.84	1748.12	2647.56

The growth in connectivity from an empty graph as well as a line graph was evaluated. In the first case, edges were repeatedly added to an initially empty graph with 25 nodes. The connectivity after each new edge addition is plotted in Fig. 4.2. The same process was repeated with the initial graph being a line graph and is shown in Fig. 4.3. From these two plots, it is apparent that the hybrid method again strikes a balance between the Fiedler method and bisection method.

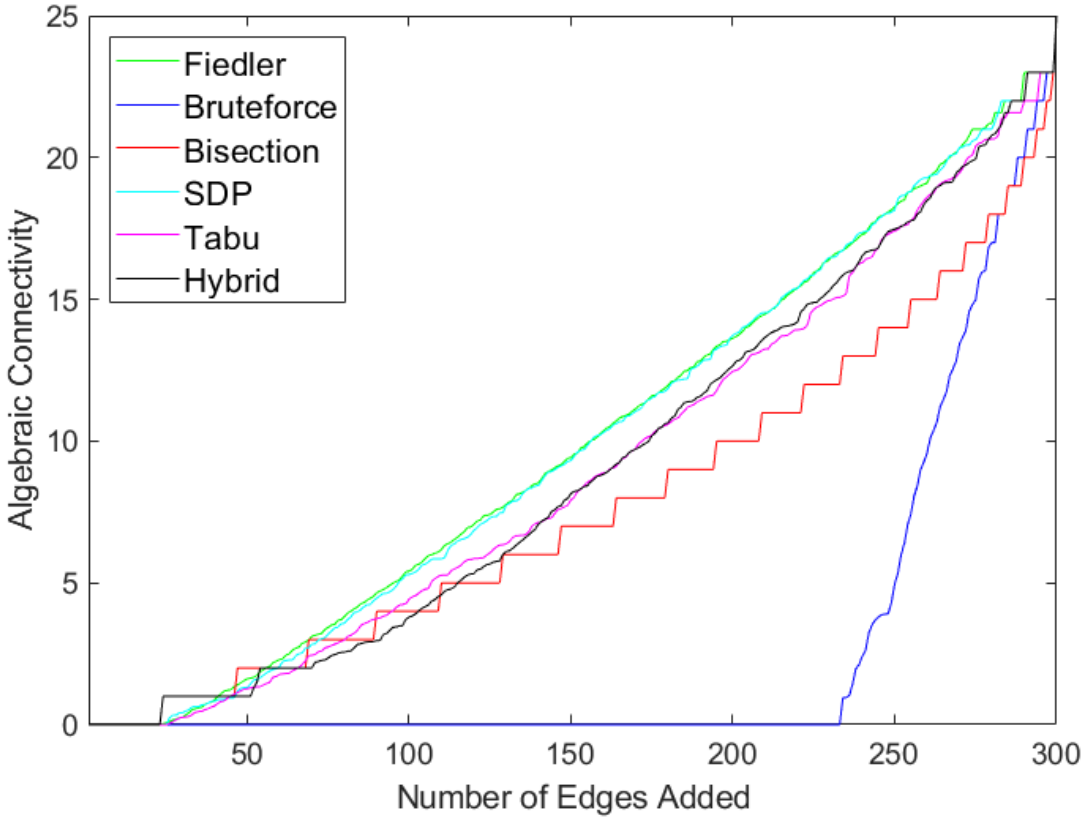

Figure 4.2 Algebraic Connectivity of An Initially Empty Graph as Edges are Added.

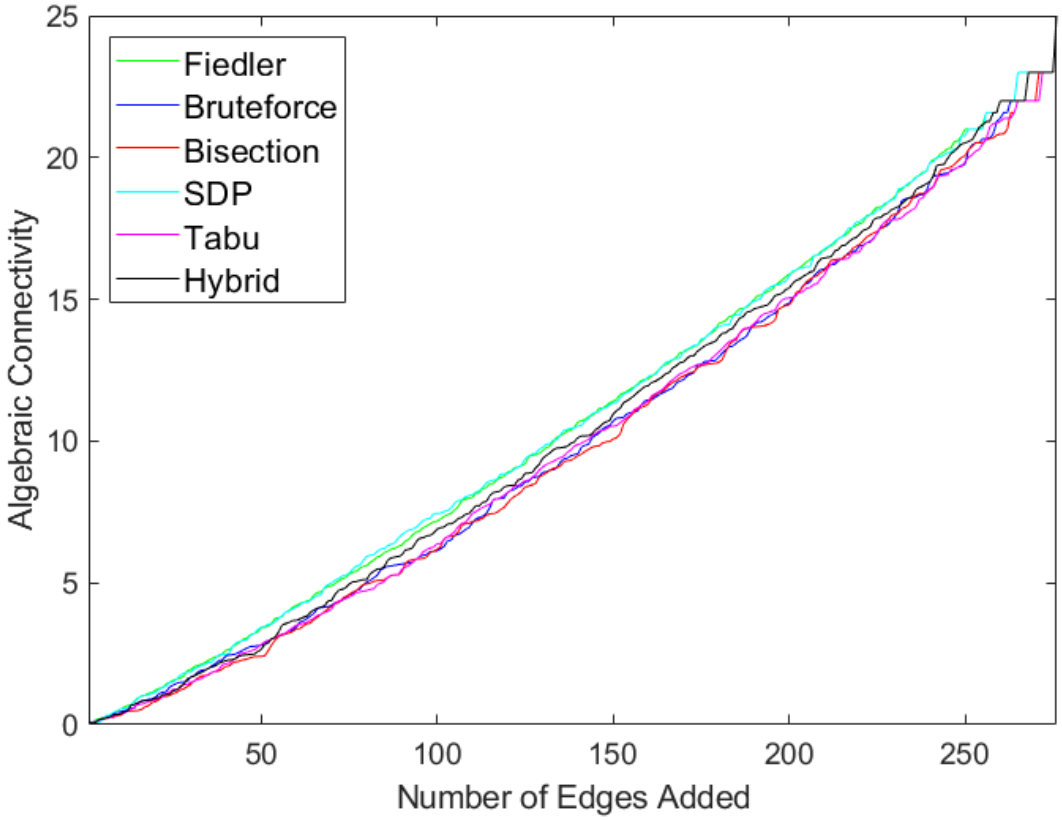

Figure 4.3 Algebraic Connectivity of An Initial Line Graph as Edges are Added.

4.5 Summary

In this chapter a new, hybrid method was developed which improves the accuracy of the efficient Fiedler method by predicting cases in which the Fiedler method does not work, and instead employing the more computationally expensive, but accurate, bisection method. This hybrid method relies on the development of an accurate classifier, in this case a random forest, to predict the success or failure of the Fiedler method. A set of features has been developed which can be used for such a classifier and this classifier has been incorporated into a method for selecting an edge. This method has been demonstrated in a connectivity tracking simulation and compared with the standalone methods by evaluating all graphs from size 4 to 7, and samples of graphs from size 8 to 25. It has also been shown how a size 25 graph can be grown starting from an empty graph or a line graph to a complete graph, using the hybrid method as well as other standalone methods.

The hybrid method could be improved by reducing the number and complexity of the features used and by reducing the size of the random forest. Several experiments were conducted using smaller feature sets based on the correlation between the features and the success indicator, however, these reduced feature sets under-performed the current, larger set of features. One likely reason for this is that some combination of the features is more important than the individual features themselves. Further experimentation, possibly with new, untested features, could potentially yield a smaller and/or more effective feature set.

Additionally, different types of classifiers could be explored to possibly improve the prediction performance or efficiency. Furthermore, a study into the modification of connectivity over multiple additions or deletions is needed.

CHAPTER 5

A DISTRIBUTED METHOD FOR CONNECTIVITY ESTIMATION AND CONTROL

The connectivity of a multiple robot team can be computed from the graph Laplacian when the entire network topology is known. It is desirable, however, for each node to know the connectivity of the network without explicitly communicating all of the connections. This allows each robot to make decisions that affect the connectivity of the team while limiting communication that spans the entirety of the network.

Franceschelli, et al. [48] developed a decentralized method of estimating the spectrum of the graph Laplacian at each node using only messages exchanged between neighboring nodes. This system is described here.

Consider a system given by

$$\begin{bmatrix} \dot{x}(t) \\ \\ \dot{z}(t) \end{bmatrix} = A \begin{bmatrix} x(t) \\ \\ z(t) \end{bmatrix}, A = \begin{bmatrix} 0 & I+L \\ \\ -I-L & 0 \end{bmatrix} \tag{5.1}$$

where x and z are state variables, I is the identity matrix, and L is the graph Laplacian of a network.

For a row i this gives

$$\begin{aligned} \dot{x}_i(t) &= z_i(t) + \sum_{j \in N_i} z_i(t) - z_j(t) \\ \dot{z}_i(t) &= x_i(t) + \sum_{j \in N_i} x_i(t) - x_j(t) \end{aligned} \tag{5.2}$$

[48] where $\dot{x}_i$ and $\dot{z}_i$ describe the change in a given row i for the state variables x and z, x_i and z_i are the ith entries in the x and z state vectors and correspond to $node_i$, and N_i is the set of indices for the neighboring nodes which are connected to $node_i$.

Thus, each node in the graph can compute its entries $x_i(t)$ and $z_i(t)$ by exchanging its state with its neighbors.

The solution [48] of this system is given as

$$\begin{aligned} x_i(t) = \sum_{j=1}^{n} [v_j(i)(cos((1+\lambda_j)t)v_j^T x(0) + \\ sin((1+\lambda_j)t)v_j^T z(0))] \\ z_i(t) = \sum_{j=1}^{n} [v_j(i)(-sin((1+\lambda_j)t)v_j^T x(0) + \\ cos((1+\lambda_j)t)v_j^T z(0))] \end{aligned} \tag{5.3}$$

where x_i and z_i are entries in the state vectors x and z, $v_j(i)$ is the ith entry in the jth eigenvector, λ_j is the jth eigenvalue, and n is the number of nodes. The initial state should be chosen uniformly at random with $|x_0| = 1$ and $|z_0| = 1$.

By numerically solving the differential equation at each time step t and recording a history for the duration of the time window, sufficient data can be gathered for spectral analysis. As seen in Equation 5.3, the frequencies present in the state trajectory correspond to the eigenvalues of the Laplacian plus one. Therefore, an analysis of the frequency spectrum of the state history will reveal the eigenvalues for the network, including the algebraic connectivity.

As an example, consider the graph shown in Fig. 5.1 and the corresponding time-domain plot of the state vector at $node_1$, $x_1(t)$, is shown in Fig. 5.2.

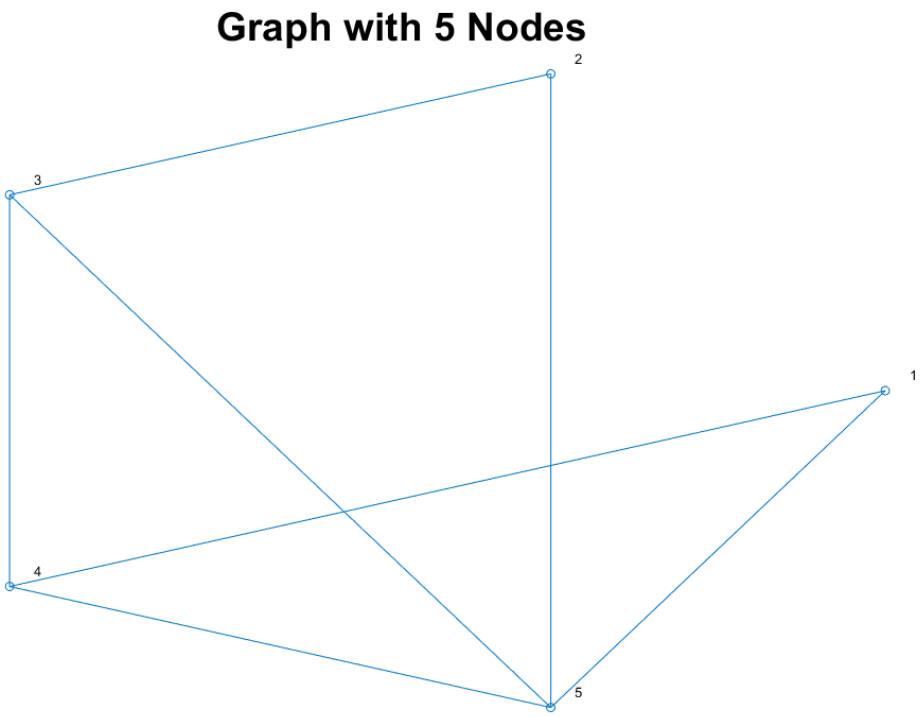

Figure 5.1 A Graph with Five Nodes.

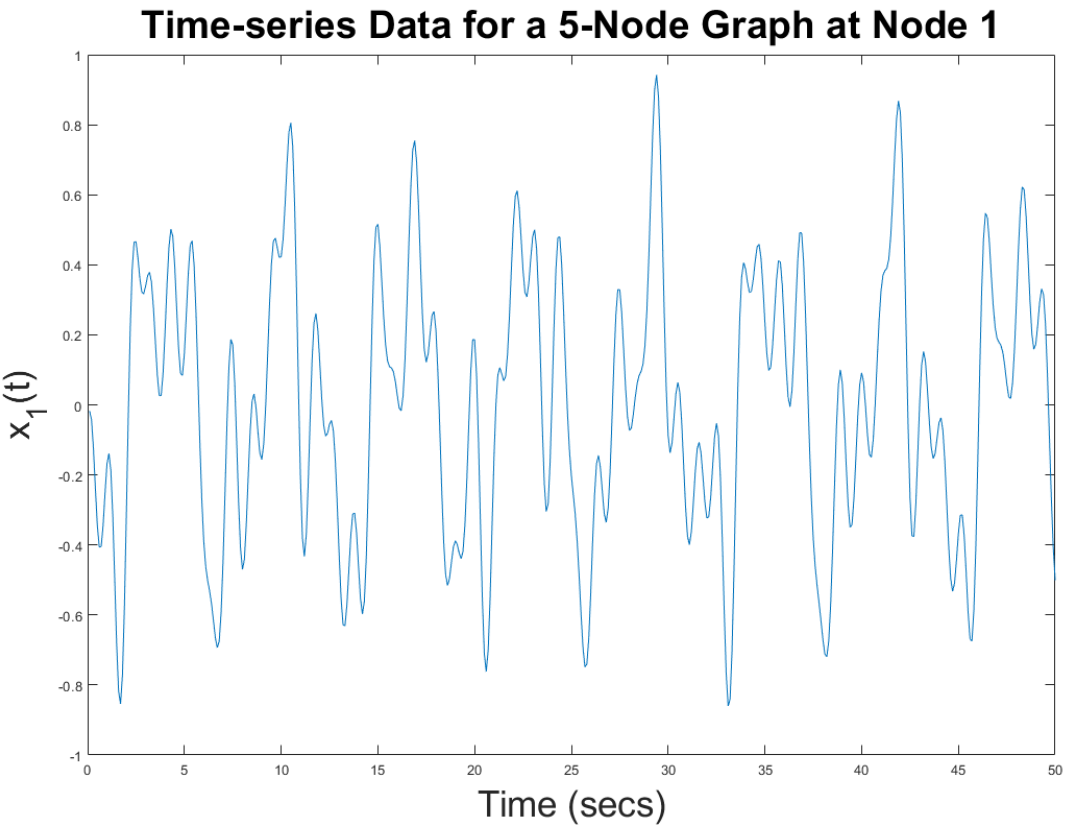

Figure 5.2 Time-domain data for the Graph Shown in Fig. 5.1.

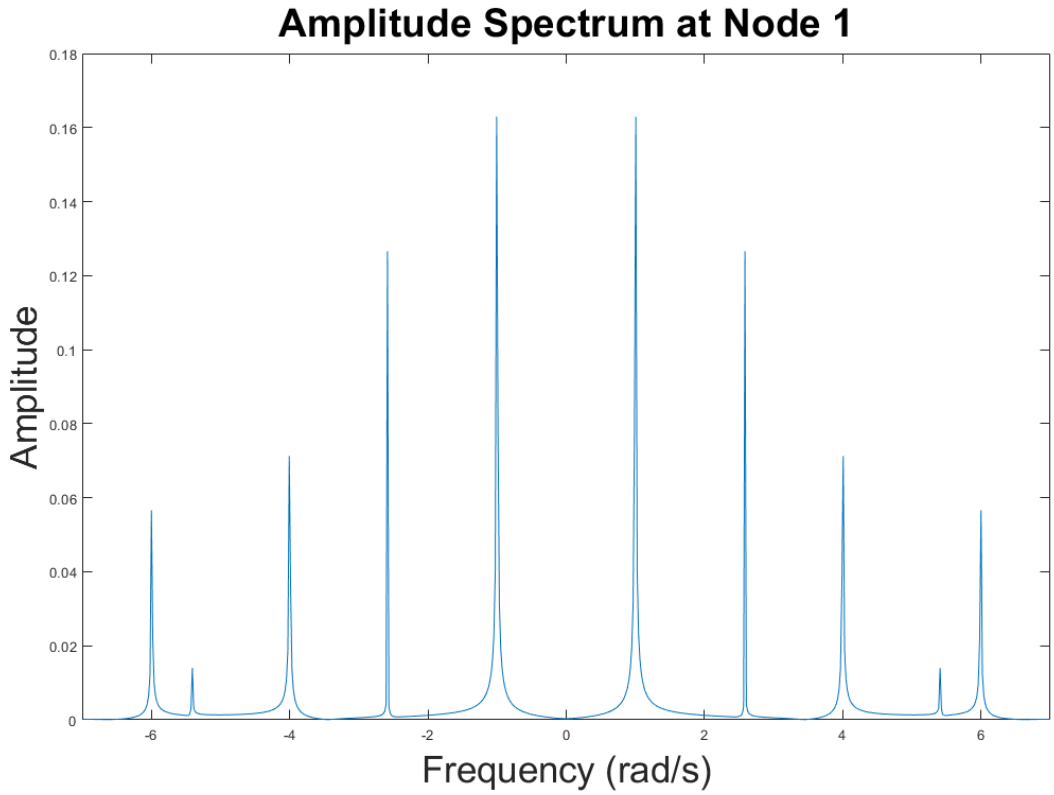

Figure 5.3 Amplitude Spectrum for the Graph Shown in Fig. 5.1 at Node 1.

The eigenvalues of the graph are $\lambda_k = \{0, 1.5858, 3, 4.4142, 5\}$. As seen in the amplitude spectrum shown in Fig. 5.3, the corresponding frequencies are at $1 + \lambda_k$.

A team's connectivity can be modified throughout a flight by either adding or removing edges, or by increasing or decreasing the edge-weights. When a UAV in the team moves within the wireless range of their neighbor a new edge will be added. Conversely, when a UAV moves outside of the wireless range of a connected neighbor, the edge is removed. The same effect can also be achieved, without requiring physical movement, by enabling and disabling communication with neighbors. Furthermore, edge weights can be modified by altering the signal strength either by moving and changing the distance to a neighbor or by adjusting the power used to transmit to a neighbor.

As mentioned in Chapter 2, modifying the connectivity through the addition and deletion of edges has been shown to be NP-hard [30]. Four heuristic methods were described in 3.1, one of which was the greedy perturbation heuristic [10]. This method relies on knowledge of the Fiedler vector, the eigenvector associated with the algebraic connectivity.

In this chapter an extension to the decentralized Laplacian spectrum estimation in [48] is developed which allows estimation of the entries of the Fiedler vector at each node in the graph. Furthermore, the decentralized Fiedler vector estimation was used to implement a distributed version of the greedy perturbation heuristic [10], or Fiedler method, of selecting an edge to be added or deleted from a graph. This new distributed method is then used as the basis of an algorithm which allows a team of UAVs to track a connectivity profile. The effectiveness of this decentralized algorithm is compared against the centralized Fiedler method [10] and evaluated

using a brute-force comparison on graphs up to size seven, a sampling of Erdős-Rényi random graphs up to size 15, and a connectivity tracking simulation [22].

5.1 Distributed Fiedler Vector Estimation

Based on the work in [48], an extension is now provided which allows recovery of the eigenvectors of the graph Laplacian.

Observing Equation 5.3 reveals that the state trajectory of x is a pair of sinusoids with amplitudes $v_j(i)(v_j^T x(0))$ and $v_j(i)(v_j^T z(0))$. By choosing the same initial vector for both x_0 and z_0 the amplitudes of both sinusoids become equal and proportional to $v_j(i)$. The amplitude of every node is scaled by the same value, $v_j^T x(0)$, which means that the amplitude detected by any node is proportionate to its entry in the Fiedler vector by the same amount as any other node, allowing for a comparison between entry values. By evaluating the amplitude peaks, the magnitude of the Fiedler vector entries can be recovered. As can be seen in Fig. 5.4, the peaks associated with λ_2 are at [0.1266; 0.1273; 0.0531; 0.0520; 0.0071]. The normalized vector then becomes [0.6511; 0.6547; 0.2731; 0.2674; 0.0366] which is a reasonable estimate for the magnitudes of the true eigenvector entries, [-0.6515; 0.6552; 0.2733; -0.2675; 0].

All that remains is to determine the sign of each entry. This is accomplished by inspecting the imaginary portion of the spectrum obtained by the Fourier transform. Since all sinusoids in the system have the same amplitude and sign before scaling by their respective entries in the Fiedler vector, entries with opposite signs in the Fiedler vector will have a phase angle 180° apart. The choice of sign given is arbitrary so long as entries with opposite signs in their imaginary components maintain opposite signs in their estimated Fiedler vector entries.

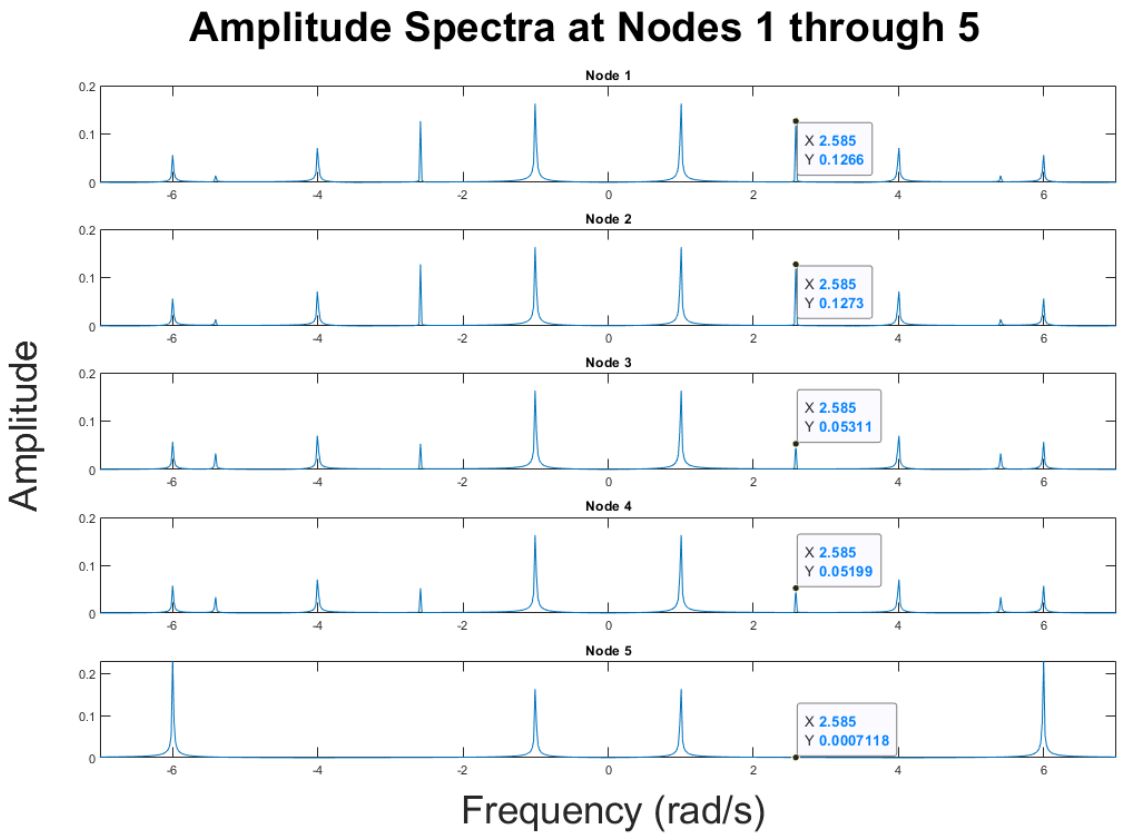

Figure 5.4 Amplitude Spectrum of All 5 Nodes for the Graph Shown in Fig. 5.1.

In the case of a repeated eigenvalue, the Fiedler vector estimation is still valid. However, the result will be a linear combination of two or more eigenvectors associated with the algebraic connectivity, which is still an eigenvector associated with λ_2. The resulting vector will be

$$\tilde{v_2} = \sum_{j=2}^{1+k} v_j^T x(0) v_j, \tag{5.4}$$

where k is the multiplicity of λ_2.

5.2 Algorithm

The decentralized Fiedler vector estimation was developed into an algorithm which could determine, in a distributed manner, an edge to add or remove from the graph for a maximal or minimal effect in terms of the connectivity. The algorithm is based on the Fiedler method [10] of selecting an edge and the distributed Fiedler vector estimation.

Initially, each node is given a unique identification number from 1 to n. It is assumed that the graph is connected and a means of communication between nodes exists. Each node should know the total number of nodes in the team and parameters for the system, the time step increment, Δt, and duration, T. The parameters chosen have an impact on the ability to effectively process the signal that is produced. A primary consideration is to sample, or record, state values at a rate greater than or equal to the Nyquist rate. Since the largest possible value of λ_n is the number of nodes, n, it is recommended to sample or compute the states of each node with a frequency of $2n$ or greater.

The first step in the algorithm is to run the local interaction rule according to the decentralized Laplacian eigenvalue estimation method [48]. This begins by initializing each node's value of $x_i(0)$

and $z_i(0)$, its entries in the x and z state vectors. This will be a random value between negative one and one, inclusive. Next, the nodes begin performing the interaction rule. This requires each node to exchange its state with its neighbor, compute $\dot{x}$ and $\dot{z}$ according to Equation 5.2, then numerically integrate the result to obtain x and z. The numerical implementation used for this research was a fourth-order Runge-Kutta method.

The interaction rule is executed for a pre-defined number of steps. As it is executed, the value of $x_i(t)$ at each step are recorded. See the example in Fig. 5.2. It is not necessary to store both x and z state histories as both signals contain the desired information. Once the interaction rule has completed, the Discrete Fourier Transform (DFT) for the state history is computed. Using the resulting amplitude spectrum (see Fig. 5.3), each node then executes a peak-finding algorithm. In its most basic form, this involves finding all points $f(k)$ where $f(k) > f(k-1)$ and $f(k) > f(k+1)$. The location of these peaks indicates an eigenvalue for the graph shifted by one. The amplitude at the peak should be stored as the magnitude of the entry in the corresponding eigenvector. The sign of the entry is determined as the sign of the imaginary portion of the frequency-domain data at that point.

Once each node has determined its entry in the Fiedler vector, it will take an action based on the type of desired connectivity change. Determining a decrease in connectivity is more accurate since every possible edge can be evaluated. This is because every existing edge connects two nodes who know their entry in the Fiedler vector. In these cases, the nodes exchange their entries with their neighbors and compute the squared difference. If a minimum decrease is desired, the nodes keep only the information corresponding to the edge with the smallest squared difference. If a maximum decrease is desired, then the edge with the maximum squared difference is used.

Determining an increase in connectivity is more challenging since it involves evaluating edges where the two nodes cannot share their Fiedler vector information. Sharing information with existing neighbors does not help the node find a new potential edge. The solution used in this work is to exchange information with two-hop neighbors. This allows each node to determine the best new edge between itself and a node which shares a neighbor. In a similar manner to the decrease method, each node determines the two-hop edge information which has a maximum squared difference or a minimum squared difference, depending on the desired change in connectivity.

Once each node has made a local determination of the best edge to add or remove, based on the Fiedler method, a system-wide minimum or maximum reduction operation is completed. Each node is considered as a node in a logical binary tree structure formed based on the node indices. The root node is $node_1$. Each node has up to two children. If the parent is $node_i$ then the children are $node_{2i}$ and $node_{2i+1}$. The orphan nodes of the tree immediately send their optimal edge information to their parent node. Each parent gathers up to two edge choices from its children, determines the best choice among the children's choices and the parent's information, and then sends that to its parent. This process repeats until it reaches $node_1$, which makes the final determination. Once the decision is made, a command to move is sent to the two nodes involved.

5.3 Simulation

To study the effectiveness of this approach, the decentralized method and the Fiedler method were tested on 100 Erdős-Rényi random graphs of size 4 to 10 with edge probability of 0.5 so that every graph of size n was equally probable. The resulting connectivity was compared to the optimal connectivity that could be obtained based on a brute-force search.

A simulation was also conducted to track a linearly increasing and a linearly decreasing connectivity profile using a random graph with 10 nodes. The graph and profile used was selected from those used in Chapter 3. The method of tracking connectivity was also discussed in that chapter.

The number of steps to complete the profile for the same graph and same profile was determined and compared with the previous result using the Fiedler method.

5.4 Results and Discussion

The results of the edge selection tests are shown in Table 5.1 for increasing and decreasing connectivity. In many cases, the decentralized system performs well. In determining the edge to delete from the graph it is able to evaluate all possible choices distributively. For this reason, the performance of the decrease functionality almost matches the centralized Fiedler method. The edge addition functionality however is limited due to the nature of the problem. The optimal edge to maximally increase connectivity is often one which connects distant portions of the graph. Since the distributed method is limited to evaluating the two-hop neighbors, it does not consider edges with large reach. This is a primary reason that the edge addition performance lags behind the centralized method, which uses global information. However, in real systems, creating edges based on large movements may not be practical.

The system was also used to run two profiles on size 10 graphs. One linearly increasing and one linearly decreasing. An example of a profile track based on the centralized Fiedler method is shown in Fig. 5.5. Table 5.2 shows the number of steps required for each method to complete

Table 5.1 Number of Successful Edge Selections in Determining Maximum and Minimum Increase and Decrease in Connectivity for a Sampling of Graphs Size 4 Through 10.

Size	Dir.	Type	Fiedler Method	Decentralized	Total
4	-	Max.	64	59	64
5	-	Max.	100	92	100
6	-	Max.	94	85	100
7	-	Max.	91	79	100
8	-	Max.	87	81	100
9	-	Max.	90	76	100
10	-	Max.	78	73	100
4	-	Min.	64	50	64
5	-	Min.	91	54	100
6	-	Min.	78	49	100
7	-	Min.	77	43	100
8	-	Min.	78	54	100
9	-	Min.	76	54	100
10	-	Min.	76	59	100
4	+	Max.	53	54	64
5	+	Max.	74	74	100
6	+	Max.	73	47	100
7	+	Max.	73	40	100
8	+	Max.	52	41	100
9	+	Max.	51	27	100
10	+	Max.	55	27	100
4	+	Min.	62	57	64
5	+	Min.	95	66	100
6	+	Min.	85	59	100
7	+	Min.	89	65	100
8	+	Min.	90	59	100
9	+	Min.	90	63	100
10	+	Min.	89	71	100

Table 5.2 Number of Steps Required to Complete Profile.

Size	Profile	Fiedler Method	Decentralized
10	Increasing	492	554
10	Decreasing	937	1175

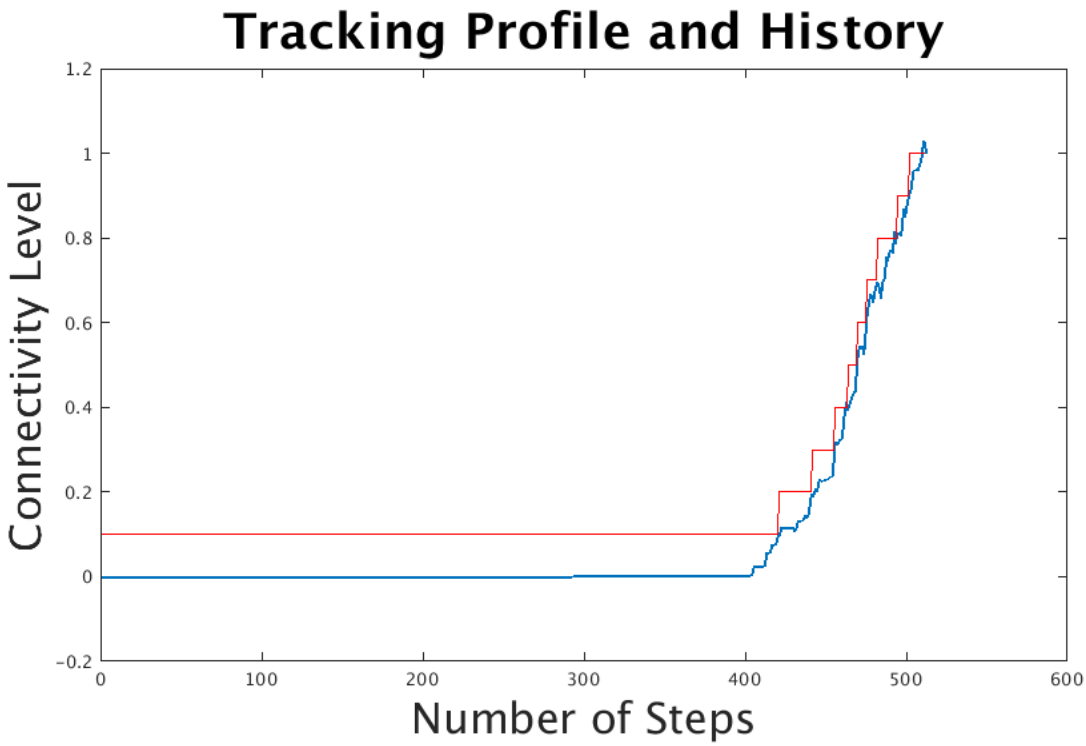

Figure 5.5 Linearly Increasing Profile and Tracking History for the Centralized Fiedler Method.

the profile. Although the distributed method did not perform as well as the centralized Fiedler approach, it was not far off.

One aspect to consider is the amount of communication necessary to implement this system. Although many messages must be transmitted to distriubtively compute the state and trajectory of the system, the messages are almost exclusively between immediate neighbors. Franceschelli et. al. [48] provided an upper bound on the communication rounds necessary to collect a sufficient amount of data as $4\Delta_{max}T_{min}f_s$ based on using the fourth-order Runge-Kutta method in conjunction with this system. Δ_{max} is the maximum degree of any node in the system, T_{min} is the largest period of the sinusoid with the smallest frequency, and f_s is the sampling frequency.

Several limitations exist with this approach. First, when the initial state vector is orthogonal to any eigenvector, it will attenuate the signal associated with that eigenvector's eigenvalue. When randomly selecting the initial state vector it is possible to select an orthogonal or near-orthogonal vector which makes it difficult to accurately extract information from the signal spectrum. Furthermore, a zero entry in an eigenvector will also attenuate the associated signal. This is a large factor in the performance of the distributed method compared to the centralized method.

The first attenuation problem could be solved by executing the system twice. In the first execution, a random state vector would be used. Then, the result could be evaluated to determine if an orthogonal vector was used, in which case a new random vector would be generated and the system would execute again.

If the results instead suggested that some entries contained zeroes, a simple voting protocol could determine the most likely position for λ_2, which could be shared via a broadcast to all nodes. The nodes could then accurately determine their Fiedler entry values without the need to execute the system again.

Running the system a second time can also be used to greatly increase the accuracy. When an eigenvector of the graph is used as x_0, the peaks associated with the other eigenvectors are attenuated and the peak associated with the eigenvector used as x_0 is amplified. If an approximate Fiedler vector and λ_2 is determined on an initial run, it can be used in the second execution to amplify the peak associated with λ_2. The approximate location of the peak would be known, allowing the nodes to locate it more precisely and more easily determine if the signal was attenuated due to a zero entry.

Another limitation is the use on large graphs. Very large graphs will require higher frequencies and longer time windows to accurately analyze the system.

Finally, the system must remain connected to function. Disconnecting a node would result in obtaining the spectrum of a graph with $n - 1$ nodes, rather than the true spectrum for the graph with a disconnected node.

CHAPTER 6

A CONNECTIVITY TRACKING CONTROLLER

In the previous chapters, a rule-based connectivity tracking method was used with a basic motion model. In those experiments, each agent was able to move in any direction with a constant velocity. The model disregarded the necessity of turning, accelerating, and decelerating. Although a simple model, it is not without merit. Many UAV applications involve multi-rotor vehicles which are capable of moving in any direction. They are also very responsive, and can make sharp changes in direction. Even so, a higher fidelity dynamics model is desired. With it, comes the need for a true controller. In this chapter, a consensus based controller [49] [50] which can be used to drive agents to relative positions, optionally around a target, is adapted for the use of connectivity tracking. A simulation was built in Mathworks Simulink and controlled from MATLAB vis a Robot Operating System (ROS) interface to demonstrate its effectiveness.

6.1 Consensus Controller

Consider a team of UAVs or other agents in a multi-agent network. Each agent can communicate directly with its neighbors, and indirectly with all other agents, so long as the graph remains connected. Assume that the UAV team remains at a constant altitude. Then, the states of the ith UAV, where $i \in [1, n]$ are as follows,

$$p_i = \begin{bmatrix} x_i \\ y_i \\ v_i \\ \theta_i \end{bmatrix}. \tag{6.1}$$

The equations of motion, then, are as follows.

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} v_i cos\theta_i \\ v_i sin\theta_i \end{bmatrix}, \tag{6.2}$$

$$\ddot{p} = \begin{bmatrix} cos\theta_i & -v_i sin\theta_i \\ sin\theta_i & -v_i cos\theta_i \end{bmatrix} \begin{bmatrix} \dot{v}_i \\ \dot{\theta}_i \end{bmatrix}. \tag{6.3}$$

Let

$$M_i = \begin{bmatrix} cos\theta_i & -v_i sin\theta_i \\ sin\theta_i & -v_i cos\theta_i \end{bmatrix}, \tag{6.4}$$

and,

$$u_i = \begin{bmatrix} \dot{v}_i \\ \dot{\theta}_i \end{bmatrix}. \tag{6.5}$$

Then, u_i is an input to the system and M_i is a matrix which relates the input to the acceleration of the x and y position.

The control objective is to drive each agent to a desired position, relative to the center of the formation, and can be expressed as

$$\begin{matrix} p_i(t) - p_{avg}(t) \rightarrow P_i \\ \dot{p}_i - \dot{p}_{avg} \rightarrow 0 \end{matrix}, \tag{6.6}$$

where p_i is the position vector representing the location of the ith UAV, p_{avg} is the average position of all UAVs in the team, P_i is the desired position for the ith UAV, $\dot{p}_i$ is the velocity of the ith UAV, and $\dot{p}_{avg}$ is the average velocity of the team.

In other words, the relative distances between pairs of agents should converge to a desired relative displacement and the velocities of each agent should converge to the same value.

The control law to move the ith UAV to achieve the desired separation with the jth UAV is given in Equation 6.7 [51].

$$M_i u_i = M_j u_j - \alpha_1(\hat{p}_i - \hat{p}_j) - \alpha_2(\dot{p}_i - \dot{p}_j) \tag{6.7}$$

The ideal control input would satisfy Equation 6.7 for all $i, j \in [1, n]$, however, there is no such u which can satisfy all of these equalities as the system is overdetermined [49]. The best solution is to average all u_i which satisfy the equations separately. Therefore, the control input u_i for each UAV is given as

$$u_i = \frac{M_i^- 1}{\sum_{j \in N_i} a_{ij}} \sum_{j \in N_i} a_{ij}[\ddot{p}_j - \alpha_1(\hat{p}_i - \hat{p}_j) - \alpha_2(\dot{p}_i - \dot{p}_j)] \tag{6.8}$$

[51].

This is a consensus-based control method which is decentralized since each control input u_i requires only information from the neighbors of the ith UAV. Proof of convergence for this method is given in [49].

An example of this control method executed on a team of four UAVs is shown in Fig. 6.1

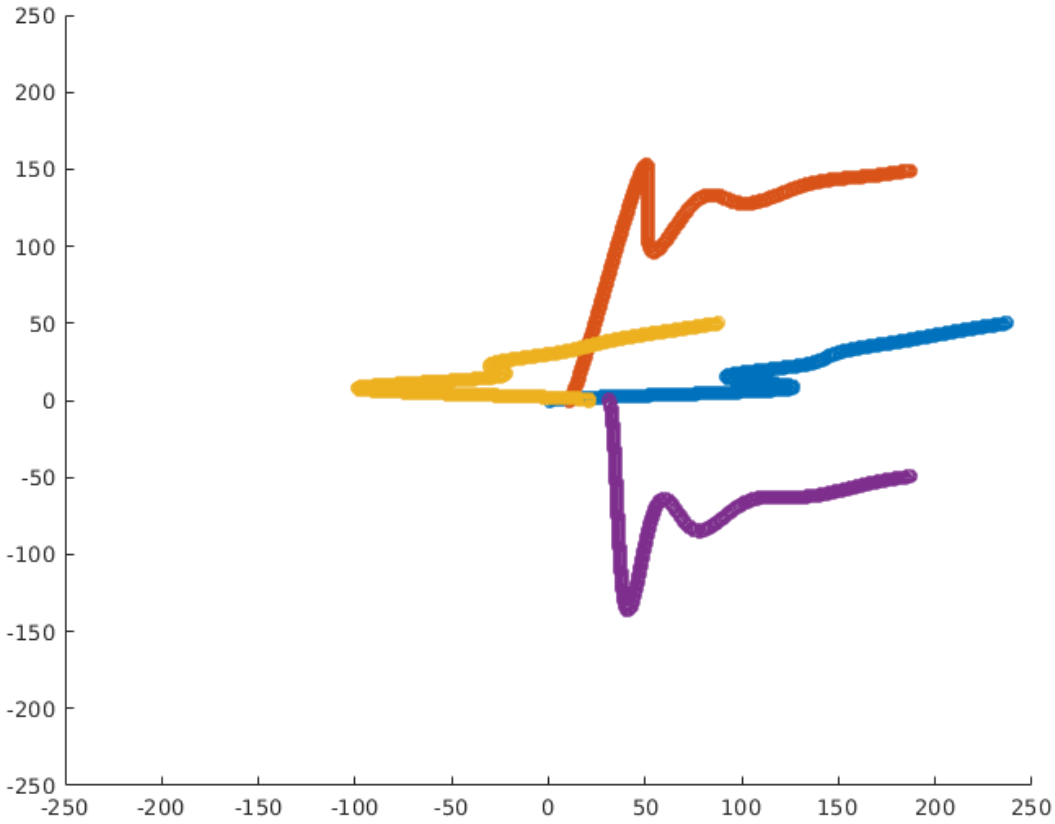

Figure 6.1 Control of Four UAVs Driven to Equal Spacing Around a Center Point.

6.2 Connectivity Tracker

The previously described controller was adapted for connectivity tracking using the rule-based connectivity tracking algorithm shown in Algorithm 4 in Chapter 3. The UAV dynamics and consensus controller were simulated using Simulink (see Appendix). The desired positions, current connectivity (λ_2), and current states were communicated with the MATLAB-based connectivity tracker via a ROS interface. This interface allows a physical UAV team to take the place of the simulation during later testing.

The connectivity tracker sent desired relative positions to the controller in the Simulink simulation via a ROS message. The controller then continually computes control inputs for each UAV based on the current states of the UAVs as well as the desired positions. The connectivity tracker monitors the states of the UAVs as well as the connectivity of the team by subscribing to ROS topics running within the Simulink simulation. Since the controller is a consensus controller, it drives each UAV to the same heading and velocity. Once the connectivity tracker detects that the mean squared error between each UAVs heading and velocity and the average heading and velocity of the entire team drops below a threshold, it can consider that the move has been completed and re-evaluate the status of the team. The connectivity tracker will continually give inputs in the form of desired positions based on either contracting and expanding edges or adding and removing edges as discussed in Chapter 3. Once the team has reached the current target connectivity, a new target connectivity is set within the connectivity tracker and the rule-based connectivity tracking algorithm again proceeds to give inputs for edge modification, addition, or deletion until the profile is completed.

6.3 Simulation

As a proof of concept, the connectivity tracking simulation was conducted twice, once with the Hybrid Algorithm discussed in Chapter 4 and once with the Distributed Fiedler Method discussed in Chapter 5. The output of these simulations is shown below. The profile used for both simulations was $\lambda_2(k) = [1, 2, 3, 4, 3, 2, 3, 4]$, where k is an index to the desired profile array.

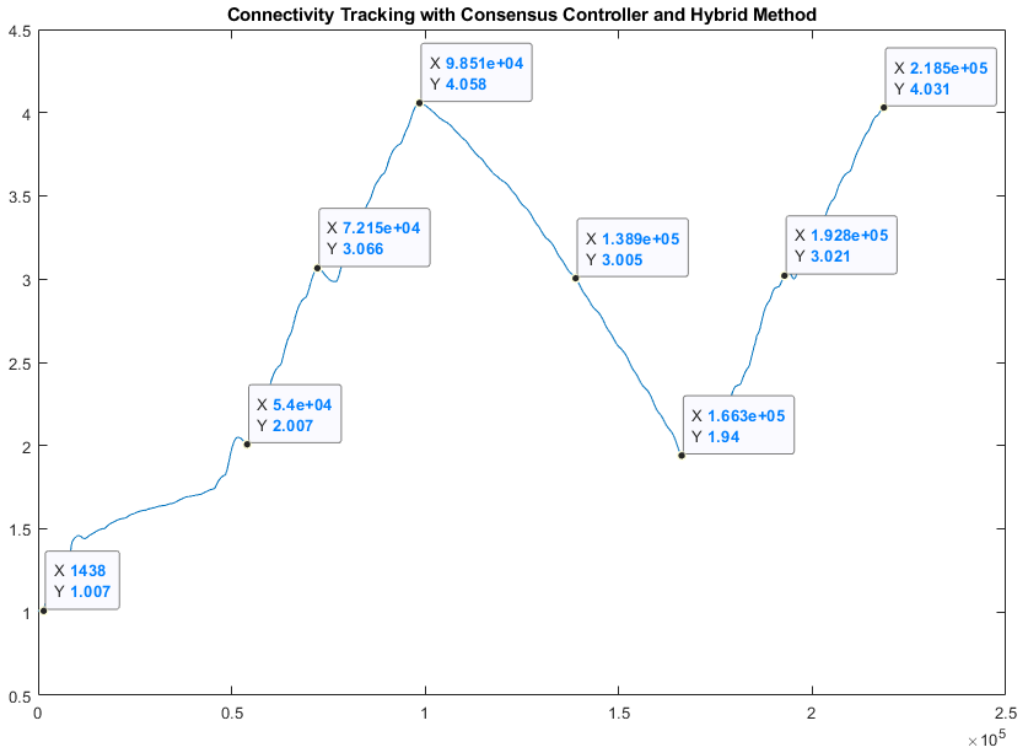

Figure 6.2 Connectivity History of a Five UAV Team Using the Consensus Controller and the Hybrid Method.

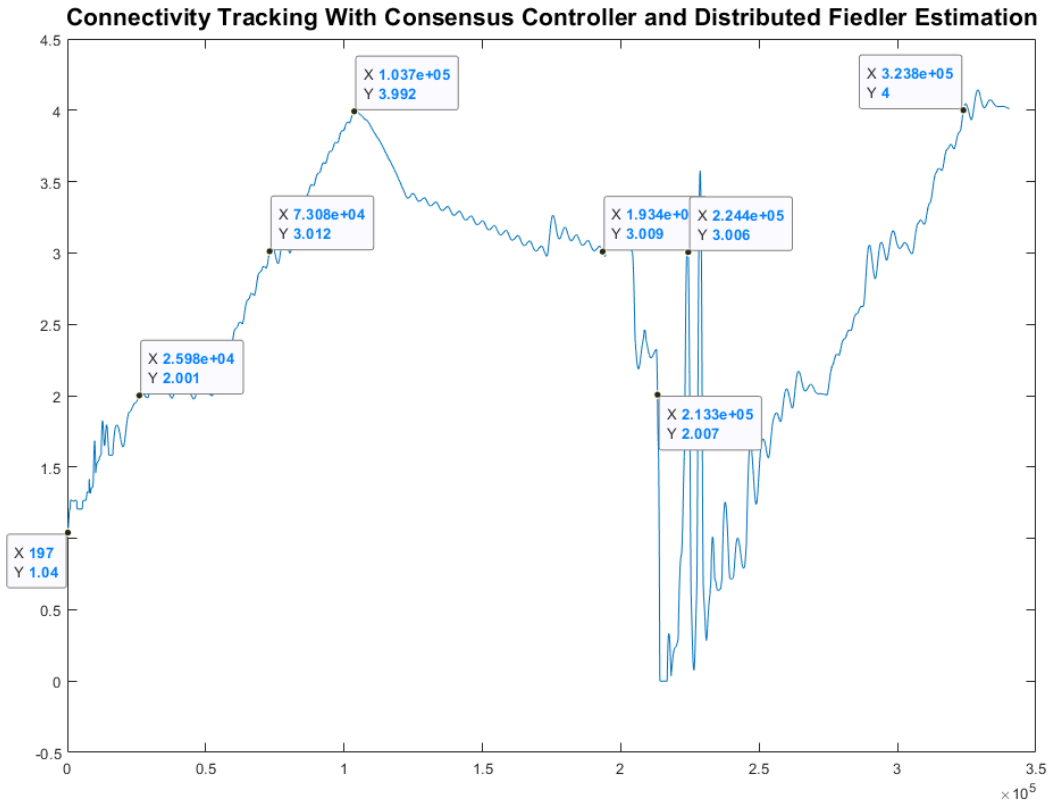

Figure 6.3 Connectivity History of a Five UAV Team Using the Consensus Controller and Distributed Fiedler Estimation Method.

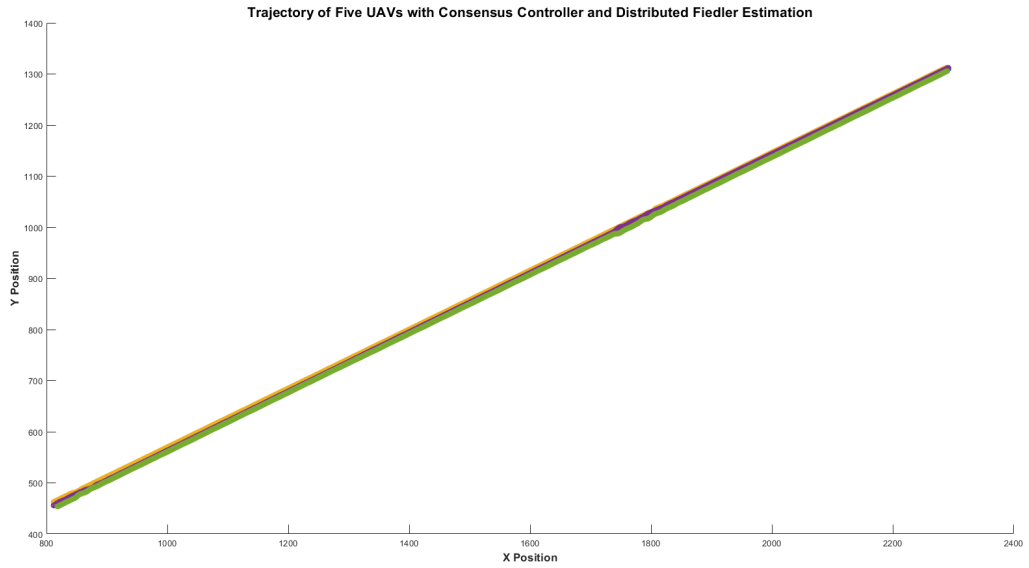

Figure 6.4 Trajectory of a Five UAV Team.

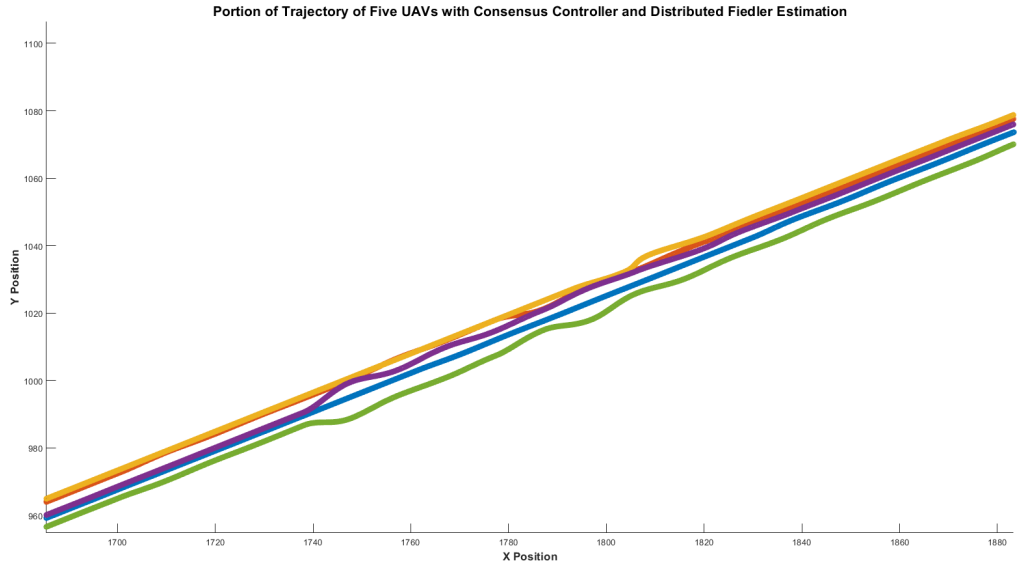

Figure 6.5 Trajectory of a Five UAV team, Zoomed-in.

6.4 Results and Discussion

The consensus controller performed well in the demonstrations using both the hybrid method and the distributed Fiedler estimation method. The consensus controller was successfully used to continually place the UAVs at desired positions relative to each other, based on inputs from the connectivity tracker. This allowed the system to create the desired graph structure and resulting connectivity at each step during the course of the profile.

The controller could be improved by removing the requirement for all UAVs to be moving and reaching a common velocity. While a common velocity may be desirable in fixed wing UAVs, multi-rotor UAVs often hover in place. The connectivity tracker or the controller should also account for the potential to inadvertently break or create links during moves. This would help prevent some of the drastic spikes seen in the connectivity history. Next, the controller should incorporate more constraints and objectives. Finally, the controller could be made to minimize distance traveled when moving a UAV to a new position.

6.5 Summary

In this chapter a consensus controller with double-integrator dynamics was implemented to simulate and control a team of UAVs. The rule-based connectivity tracking algorithm was used to direct the UAV team in following a desired connectivity profile. The controller and UAV dynamics were simulated in MATLAB and simulink. The connectivity tracking algorithm was executed with both the hybrid method and the distributed Fiedler method of selecting edges to add and remove from the graph.

Demonstrations showed that the consensus controller, combined with the connectivity tracker, allowed the UAV team to effectively complete the desired connectivity profile using both algorithms.

CHAPTER 7

CONCLUSION

This work presented a comparison of four methods for modifying connectivity via edge addition or deletion. A rule-based connectivity tracker was developed and a simulation was conducted using a variety of graph sizes, starting configurations, and profiles. A hybrid algorithm was developed based on two leading methods, the greedy perturbation heuristic [10] (Fiedler Method), and the bisection method [12]. A distributed method of estimating the eigenvectors of the graph Laplacian was developed and incorporated into a distributed greedy perturbation heuristic (Fiedler Method) for selecting edges for addition and deletion. Finally, a consensus-based controller was adapted and simulated with a two-dimensional dynamics model. A connectivity tracker was created to command the consensus controller using either the distributed Fiedler Method or the centralized Hybrid Method.

7.1 Recommendations for Future Work

Connectivity is non-differentiable and attempts to modify the graph result in many discontinuities as edges are added and removed, either intentionally or unintentionally. The selection of edges is NP-hard [30] and while methods exist to select edges for moderately sized graphs, the problem is compounded when multiple edges need to be added or removed, and no good methods currently exist.

Future work should focus on smoothly transitioning between desired connectivities. Current research into combinatorial-type games may be beneficial in developing a more advanced tracking planner. Additionally, the use of selectively connecting and disconnecting to nodes, rather than relying on a disc model, will be beneficial in giving more control over the network.

In terms of distributive control, a cluster-based approach, such as that used in [52] may present new opportunities for decentralization. Furthermore, a framework for working with connectivity from the perspective of the end user as well as the application developer is needed so that connectivity can be clearly defined as a mission requirement and end users can have an understanding of the performance associated with a given connectivity. Variations of the Laplacian matrix, such as the normalized Laplacian, may be beneficial in establishing a common framework to be used among any multi-agent network. This framework should be portable in the sense that it allows an end user to apply the same understanding of connectivity to any multi-agent system, regardless of properties such as the network size or particular hardware used.

Finally, practical aspects such as the estimation of link quality, complications of loss of link or loss of a UAV, time delays and dropped packets in communication, among others should be examined and developed into a robust implementation.

REFERENCES

[1] Ren, W. and Beard, R. W., *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications*, Springer Publishing Company, Incorporated, 1st ed., 2007.

[2] Kar, S. and Moura, J. M. F., "Distributed Average Consensus in Sensor Networks with Random Link Failures," *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, Vol. 2, April 2007, pp. II–1013–II–1016.

[3] Fiedler, M., "Algebraic Connectivity of Graphs," *Czechoslovak Mathematical Journal*, Vol. 23, 01 1973, pp. 298–305.

[4] Xu, M., Yang, Q., and Kwak, K., "Topology Control with Energy-Connectivity Tradeoff for Wireless Ad Hoc Networks," *ETRI Journal*, Vol. 39, 10 2016.

[5] Schuresko, M. and Cortes, J., "Distributed Motion Constraints for Algebraic Connectivity of Robotic Networks," *2008 47th IEEE Conference on Decision and Control*, Dec 2008, pp. 5482–5487.

[6] Zavlanos, M. M., Egerstedt, M. B., and Pappas, G. J., "Graph-theoretic Connectivity Control of Mobile Robot Networks," *Proceedings of the IEEE*, Vol. 99, No. 9, Sept 2011, pp. 1525–1540.

[7] Wei, P., Chen, L., and Sun, D., "Algebraic Connectivity Maximization of an Air Transportation Network: The Flight Routes Addition/Deletion Problem," *Transportation Research Part E: Logistics and Transportation Review*, Vol. 61, 2014, pp. 13 – 27.

[8] Zhang, Z., Wang, X., and Xin, Q., "A New Performance Metric for Construction of Robust and Efficient Wireless Backbone Network," *IEEE Transactions on Computers*, Vol. 61, No. 10, Oct 2012, pp. 1495–1506.

[9] Cvetković, D., "Applications of Graph Spectra: an Introduction to the Literature," *Zbornik Radova*, Vol. 13, No. 21, 2009, pp. 7–32.

[10] Ghosh, A. and Boyd, S., "Growing Well-connected Graphs," *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 6605–6611.

[11] Glover, F., "Tabu Search - Part I," *ORSA Journal on Computing*, Vol. 1, No. 3, 1989, pp. 190–206.

[12] Kim, Y., "Bisection Algorithm of Increasing Algebraic Connectivity by Adding an Edge," *IEEE Transactions on Automatic Control*, Vol. 55, No. 1, Jan 2010, pp. 170–174.

[13] Wang, H. and Van Mieghem, P., "Algebraic Connectivity Optimization via Link Addition," *Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems*, BIONETICS '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, 2008, pp. 22:1–22:8.

[14] Li, G., Hao, Z. F., Huang, H., and Wei, H., "Maximizing Algebraic Connectivity via Minimum Degree and Maximum Distance," *IEEE Access*, Vol. 6, 2018, pp. 41249–41255.

[15] Simonetto, A., Keviczky, T., and Babuška, R., "Constrained Distributed Algebraic Connectivity Maximization in Robotic Networks," *Automatica*, Vol. 49, No. 5, 2013, pp. 1348 – 1357.

[16] Zheng, Y., Zhao, S., Liu, Y., Li, Y., Tan, Q., and Xin, N., "Weighted Algebraic Connectivity Maximization for Optical Satellite Networks," *IEEE Access*, Vol. 5, 2017, pp. 6885–6893.

[17] Simonetto, A., Keviczky, T., and Babuška, R., *Distributed Algebraic Connectivity Maximization for Robotic Networks: A Heuristic Approach*, Springer, Berlin, Heidelberg, 2013, pp. 267–279.

[18] Fang, H., Wei, Y., Chen, J., and Xin, B., "Flocking of Second-Order Multiagent Systems With Connectivity Preservation Based on Algebraic Connectivity Estimation," *IEEE Transactions on Cybernetics*, Vol. 47, No. 4, April 2017, pp. 1067–1077.

[19] Dong, Y. and Huang, J., "The Leader-following Rendezvous with Connectivity Preservation via a Self-tuning Adaptive Distributed Observer," *International Journal of Control*, Vol. 90, No. 7, 2017, pp. 1518–1527.

[20] Sabattini, L., Gasparri, A., Secchi, C., and Chopra, N., "Enhanced Connectivity Maintenance for Multi-Robot Systems," *IFAC Proceedings Volumes*, Vol. 45, No. 22, 2012, pp. 319 – 324, 10th IFAC Symposium on Robot Control.

[21] Dutta, R., *Cooperative Control of Autonomous Network Topologies*, Ph.D. thesis, The University of Texas at San Antonio, 2016.

[22] Trimble, J., Pack, D., and Ruble, Z., "Connectivity Tracking Methods for a Network of Unmanned Aerial Vehicles," *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2019, pp. 440–447.

[23] Bollobas, B., *Graph theory : An Introductory Course*, Springer Verlag, 1979.

[24] R Balakrishnan, K. R., *A Textbook of Graph Theory*, Springer Verlag, 1979.

[25] Godsil, C. and Royle, G., *Algebraic Graph Theory*, Vol. 207 of *Graduate Texts in Mathematics.*, Springer, 2001.

[26] Jamakovic, A. and Uhlig, S., "On the Relationship Between the Algebraic Connectivity and Graph's Robustness to Node and Link Failures," *2007 Next Generation Internet Networks*, May 2007, pp. 96–102.

[27] Ghedini, C., Secchi, C., Ribeiro, C. H., and Sabattini, L., "Improving Robustness in Multi-robot Networks," *IFAC-PapersOnLine*, Vol. 48, No. 19, 2015, pp. 63 – 68, 11th IFAC Symposium on Robot Control SYROCO 2015.

[28] Panerati, J., Minelli, M., Ghedini, C., Meyer, L., Kaufmann, M., Sabattini, L., and Beltrame, G., "Robust Connectivity Maintenance for Fallible Robots," *Autonomous Robots*, Vol. 43, No. 3, Mar 2019, pp. 769–787.

[29] Liu, W., Pawlikowski, K., and Sirisena, H., "Algebraic Connectivity Metric for Spare Capacity Allocation Problem in Survivable Networks," *Computer Communications*, Vol. 34, No. 12, 2011, pp. 1425 – 1435.

[30] Mosk-Aoyama, D., "Maximum Algebraic Connectivity Augmentation is NP-hard," Vol. 36, Nov 2008, pp. 677–679.

[31] Arsić, B., Cvetković, D., Simić, S. K., and Škarić, M., "Graph Spectral Techniques in Computer Sciences," *Applicable Analysis and Discrete Mathematics*, Vol. 6, No. 1, 2012, pp. 1–30.

[32] Rocha, I., "Spectral Bisection with Two Eigenvectors," *Electronic Notes in Discrete Mathematics*, Vol. 61, 2017, pp. 1019 – 1025, The European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'17).

[33] Straffin, P. D., "Linear Algebra in Geography: Eigenvectors of Networks," *Mathematics Magazine*, Vol. 53, No. 5, 1980, pp. 269–276.

[34] Gould, P. R., "On the Geographical Interpretation of Eigenvalues," *Transactions of the Institute of British Geographers*, , No. 42, 1967, pp. 53–86.

[35] Wei, P. and Sun, D., "Weighted Algebraic Connectivity: An Application to Airport Transportation Network," *IFAC Proceedings Volumes*, Vol. 44, No. 1, 2011, pp. 13864 – 13869, 18th IFAC World Congress.

[36] "Rank One Perturbation and its Application to the Laplacian Spectrum of a Graph," *Linear and Multilinear Algebra*, Vol. 46, No. 3, Aug 1999, pp. 193–198.

[37] Rocha, I., "Spectral Bisection with Two Eigenvectors," *Electronic Notes in Discrete Mathematics*, Vol. 61, 2017, pp. 1019 – 1025, The European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'17).

[38] Urschel, J. C. and Zikatanov, L. T., "Spectral Bisection of Graphs and Connectedness," *Linear Algebra and its Applications*, Vol. 449, 2014, pp. 1 – 16.

[39] Fiedler, M., "A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory," *Czechoslovak Mathematical Journal*, Vol. 25, No. 4, 1975, pp. 619–633.

[40] Barker, G. P., "Theory of Cones," *Linear Algebra and its Applications*, Vol. 39, 1981, pp. 263 – 291.

[41] Grant, M. and Boyd, S., "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," `http://cvxr.com/cvx`, March 2014, (accessed 12/01/2018).

[42] Grant, M. and Boyd, S., "Graph Implementations for Non-smooth Convex Programs," *Recent Advances in Learning and Control*, edited by V. Blondel, S. Boyd, and H. Kimura, Lecture Notes in Control and Information Sciences, Springer-Verlag Limited, 2008, pp. 95–110, `https://web.stanford.edu/~boyd/papers/graph_dcp.html` (accessed 12/01/2018).

[43] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification (2nd Edition)*, Wiley-Interscience, New York, NY, USA, 2000, pg. 395.

[44] Breiman, L., "Random Forests," *Machine Learning*, Vol. 45, No. 1, Oct 2001, pp. 5–32.

[45] Merris, R., "Laplacian Graph Eigenvectors," *Linear Algebra and its Applications*, Vol. 278, No. 1, 1998, pp. 221 – 236.

[46] Pati, S., "The Third Smallest Eigenvalue Of The Laplacian Matrix," *Electronic Journal of Linear Algebra*, Vol. 8, 01 2001, pp. 128–139.

[47] Li, G., Hao, Z. F., Huang, H., and Wei, H., "Maximizing Algebraic Connectivity via Minimum Degree and Maximum Distance," *IEEE Access*, Vol. 6, 2018, pp. 41249–41255.

[48] Franceschelli, M., Gasparri, A., Giua, A., and Seatzu, C., "Decentralized Estimation of Laplacian Eigenvalues in Multi-agent Systems," *Automatica*, Vol. 49, No. 4, 2013, pp. 1031 – 1036.

[49] Sharma, R., Kothari, M., Taylor, C. N., and Postlethwaite, I., "Cooperative Target-capturing with Inaccurate Target Information," *Proceedings of the 2010 American Control Conference*, June 2010, pp. 5520–5525.

[50] Dutta, R., Sun, L., and Pack, D., "A Decentralized Formation and Network Connectivity Tracking Controller for Multiple Unmanned Systems," *IEEE Transactions on Control Systems Technology*, Vol. 26, No. 6, Nov 2018, pp. 2206–2213.

[51] Dutta, R., Sun, L., Kothari, M., Sharma, R., and Pack, D., "A Cooperative Formation Control Strategy Maintaining Connectivity of a Multi-agent System," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 1189–1194.

[52] Whitehead, J. R., *Cluster-based Trust Proliferation and Energy Efficient Data Collection in Unattended Wireless Sensor Networks with Mobile Sinks*, Master's thesis, The University of Tennessee at Chattanooga, 2016.

[53] Wei, P., Spiers, G., and Sun, D., "Algebraic Connectivity Maximization for Air Transportation Networks," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 15, No. 2, April 2014, pp. 685–698.

[54] Brownlee, J., *Clever algorithms : Nature-Inspired Programming Recipes*, Lulu, [Place of publication not identified], 2011.

[55] Kirkland, S., Oliveira, C. S., and Justel, C. M., "On Algebraic Connectivity Augmentation," *Linear Algebra and its Applications*, Vol. 435, No. 10, 2011, pp. 2347 – 2356, Special Issue in Honor of Dragos Cvetkovic.

[56] Li, X., Zhang, S., and Xi, Y., "Connected Flocking of Multi-agent System Based on Distributed Eigenvalue Estimation," *Proceedings of the 30th Chinese Control Conference*, July 2011, pp. 6061–6066.

[57] Dutta, R., Sun, L., and Pack, D., "Multi-agent Formation Control with Maintaining and Controlling Network Connectivity," *2016 American Control Conference (ACC)*, July 2016, pp. 1036–1041.

[58] Simonetto, A., Keviczky, T., and Babuška, R., "On Distributed Maximization of Algebraic Connectivity in Robotic Networks," *Proceedings of the 2011 American Control Conference*, June 2011, pp. 2180–2185.

[59] Di Lorenzo, P. and Barbarossa, S., "Distributed Estimation and Control of Algebraic Connectivity Over Random Graphs," *IEEE Transactions on Signal Processing*, Vol. 62, No. 21, Nov 2014, pp. 5615–5628.

[60] Hager, M., "Path-connectivity in Graphs," *Discrete Mathematics*, Vol. 59, No. 1, 1986, pp. 53 – 59.

[61] Bunch, J. R., Nielsen, C. P., and Sorensen, D. C., "Rank-one Modification of the Symmetric Eigenproblem," *Numerische Mathematik*, Vol. 31, No. 1, Mar 1978, pp. 31–48.

APPENDIX A

UAV DYNAMICS AND CONSENSUS CONTROL SIMULATION

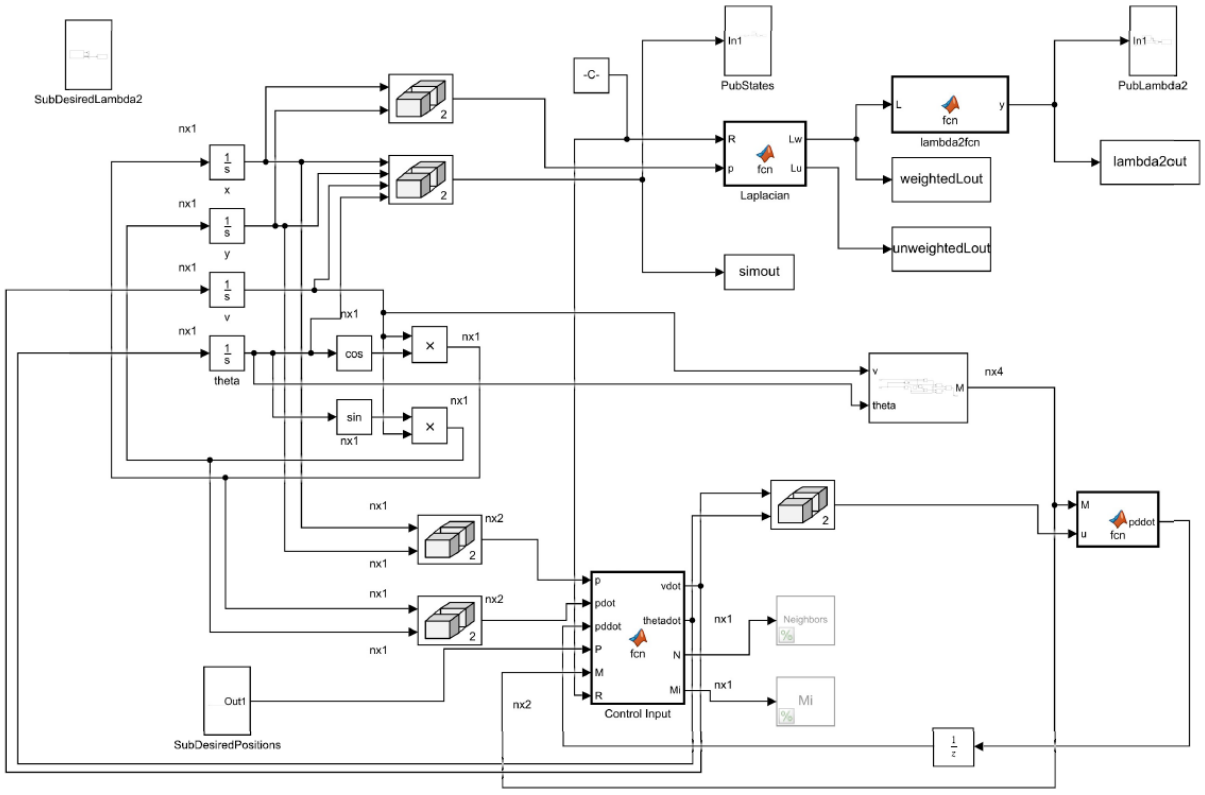

Figure A.1 Simulink model incorporating UAV dynamics, a consensus controller, and a Robot Operating System interface

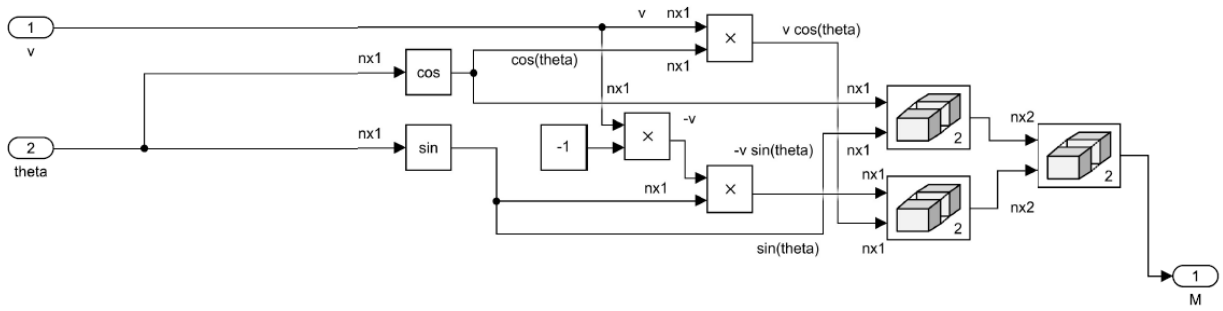

Figure A.2 A subset of Fig. A.1 showing the computation of the M control input parameter

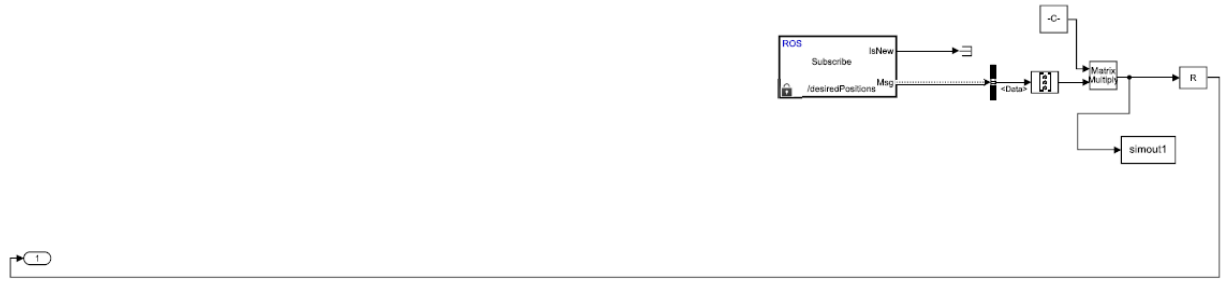

Figure A.3 A subset of Fig. A.1 showing the ROS interface which inputs desired UAV positions

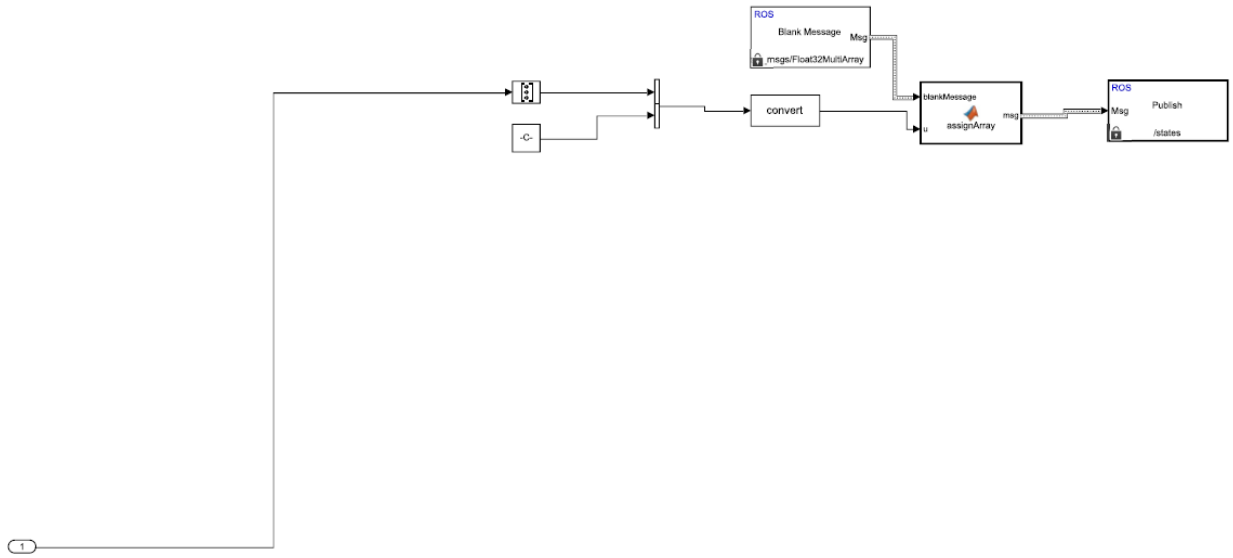

Figure A.4 A subset of Fig. A.1 showing the ROS interface which publishes the state of the UAVs

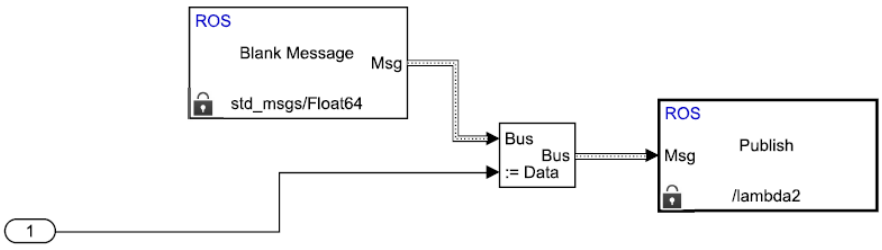

Figure A.5 A subset of Fig. A.1 showing the ROS interface which publishes the current algebraic connectivity of the system

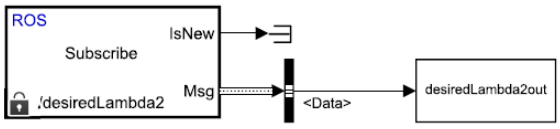

Figure A.6 A subset of Fig. A.1 showing the ROS interface which receives the desired connectivity input

VITA

James Trimble was born in Hollywood, FL, on October 2nd, 1983 to his parents, Carl and Nell Trimble. Growing up he always had an interest in tinkering, figuring out how things worked, and figuring out how to use that knowledge to make something new. He got his first computer, an Apple II, before starting elementary school. During elementary school, he started programming in Logo and QBasic. In middle school he spent much of his time programming Texas Instruments graphing calculators in assembly and C++. In high school he first learned about microcontrollers and started working with the Zilog Z8 MCU. He also started his high school's first Robotics Club. His love for engineering took him to the United States Air Force Academy where he majored in Electrical Engineering and Computer Engineering, served as the president of the school's student chapter of the IEEE, did independent research on shape-changing robots, and worked on a capstone project developing a system of multiple UAVs. Following graduation, he attended Joint Specialized Undergraduate Pilot Training, logging over 200 hours in the T-6 Texan II and T-1 Jayhawk over the course of a year. His first assignment following pilot training was as a reconnaissance pilot in the E-8 Joint Surveillance Target Attack Radar System, an air-to-ground radar intelligence platform. After 4.5 years culminating as an Aircraft Commander with over 1,000 combat hours and serving as the Operations Group Executive Officer, he returned to the Air Force Academy as a flight instructor in the TG-16 Glider. While at the academy he began serving as an instructor in Electrical Engineering as well as Computer Science. At the same time, he launched the Academy's first

flight training program for freshman cadets in support of the Superintendent's directive to instill airmanship in all cadets from an early stage. The program was a huge success. Following his time as the Airmanship 250 Program Manager he moved to the position of Chief of Flight Safety, overseeing the safety program for all of the Air Force Academy's manned flight training programs. While there, he developed an innovative software program that could be used to analyze flight data recorder logs for mishap investigations, noise abatement, and trend analysis. After serving in the flight safety position, he became the Chief of Training for the Flight Training Group, administering and overseeing all manned flight training programs at the Air Force Academy. In the summer of 2016, he moved to Tennessee to pursue doctoral studies in Computational Engineering with a research focus in unmanned aircraft systems.