Investigating Methods of Reducing Algebraic Connectivity

Daniel Rodrigues *Harvard University*
rodriguesmd@college.harvard.edu

***Abstract*—Algebraic connectivity, as defined as the second smallest eigenvalue of the Laplacian, is a measure of connectivity. We study the problem of removing edges from a graph so as to approach a target connectivity and introduce a number of motivating use cases. We provide a simple framework[1] that takes in a heuristic, a graph, and a target connectivity, then uses this heuristic to reduce the connectivity to arrive near the target value. We also show that this problem is NP-complete.**

***Index Terms*—Algebraic connectivity, greedy algorithms; multi-agent robotic systems**

I. Note for Final Project

I tried to make this paper hyper focused on the spectral graph theory aspects of the project. I left most of my discussion of the parts of the project that involved robotics for my CS 286 Paper. I recognize that this can undermine the extent of how applied my project was. Please let me know if you would have liked to see more of that part of the project in this paper.

II. Introduction

Algebraic connectivity is a graph property of great importance. This quantity is related to the graph's number of connected components, diameter, average degree, and isoperimetric number. As a result, this quantity has impacts in a wide-range of fields and can informative for scenarios of partitioning, mixing, consensus, and network resilience to name a few. Thus, algebraic connectivity is often used to inform properties of a system and efforts to further understand this value and intelligently alter it have been made by researchers with a wide range of backgrounds. In this paper, we take notes from previous works that aim to increase the algebraic connectivity by adding edges and noticed a lack of exploration in the question of the methods and applications of reducing algebraic connectivity.

A. Motivation

We were first motivated to explore this problem when we realized that there was little investigation in slowing down consensus in multi-agent robotic systems. We considered how reducing the algebraic connectivity could be used to inform and design control mechanisms for consensus. As we did, we realized that reducing connectivity could be used to motivate approaches with applications in network traffic control and routing. As we further considered the applications, we began to dissect the problem at hand and noticed how variations could be related to a number of other prevalent problems. Reducing the algebraic connectivity was of particular interest because it is often easier to cut or omit edges than to add new edges. In the robotic context, we were specifically concerned with the communication topology; in this setting, adding edges often required rearranging the agents and thus tight integration with the motion planning algorithm; however, removing edges consisted of only ignoring the communication link. We leave the detailed study of this problem in a multi-agent robotic context for a separate paper [1], where we explored this problem with a dynamic graph and the robotic applications with a focus on consensus.

[1] https://github.com/Danieltech99/CS-229r-Final-Project

III. Related Work

We include related work in robotics that were key in confirming our original motivations and guided us to the topic of this paper. In the study of consensus it has been shown that the rate of convergence for consensus is strongly related to the Fiedler value of the network [2]. Multi-agent systems often require computationally complex graph-based algorithms and spectral graph theory has provided some guidance to the field; above we saw that algebraic connectivity was embraced early on to provide performance guarantees; sparsifiers have also made its way into the literature, providing more efficient decentralized algorithms such as that explored in [3]. [4] provides exploratory data on the relationship between the algebraic connectivity of a graph and the disagreement between the nodes in the graph, displaying convergence over time, and showing the relationship between the algebraic connectivity and the rate of reduction of the disagreement value, at low algebraic connectivity values.

As we begun investigating efforts in understanding algebraic connectivity, we arrived at classic works by Fiedler, as presented in [5]. Further investigation realized that at the time of writing, algebraic connectivity was not precisely understood intuitively and we have yet to understand how to reason about the exact connectivity of general graphs outside of bounds on the connectivity. However, efforts are being made to understand connectivity and graph configurations in specific contexts. [6] identifies a unique graph which attains minimum algebraic connectivity among the graphs which have complements that are trees, but not stars, and another unique graph that attains minimum connectivity among the graphs whose complements are unicyclic graphs, but not stars adding one edge. [7] and [8] investigated the bicyclic graphs with

two and three cycles and identify the unique graph with the minimum connectivity. Extended efforts have been made in understanding how one can increase connectivity by adding edges. [9] shows that adding a certain number of edges to maximize connectivity is NP-complete. [10] provides a heuristic to approximately solve the previous problem. These last two papers provide great insight for this paper.

IV. Definitions

Let $G = (V, E)$ be a graph on the vertex set $V = \{1, \ldots, n\}$ with an edge set of size $|E| = m$. In this paper, we assume the graph is undirected and simple. We denote the complement of the graph G as $G^C = (V, E^C)$, where $E^C = \{(u, v) \mid u, v \in V, u \neq v, (u, v) \notin E\}$. The Laplacian $L(G)$ of a graph G is the $n \times n$ matrix defined by

$$L(G) = D(G) - A(G)$$

$$L(G)_{i,j} = \begin{cases} \text{deg(i)} & \text{if } i = j \\ \text{-1} & \text{if } i \neq j \text{ and } (i,j) \in E \\ 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

where $deg(v)$ is the number of neighbors or degree of $v \in V$, $D(G)$ is the degree matrix, $A(G)$ is the adjacency matrix.

$$D(G)_{i,j} = \begin{cases} \text{deg(i)} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

$$A(G)_{i,j} = \begin{cases} 1 & \text{if and } (i,j) \in E \\ 0 & \text{if and } (i,j) \notin E \end{cases}$$

We enumerate the eigenvalues of the Laplacian $L(G)$ to be $0 = \lambda_1(G) \leq \lambda_2(G) \leq \cdots \leq \lambda_n(G)$ and the corresponding eigenvectors to be $\psi_1(G), \psi_2(G), \ldots, \psi_n(G)$. We also refer to $\lambda_n(G)$ as $\lambda_{max}(G)$. The algebraic connectivity of a graph G refers to $\lambda_2(G)$, which is also know as the Fiedler value. The Fiedler vector is the corresponding eigenvector $\psi_2(G)$.

We also mention a few useful properties of the Laplacian. First, the multiplicity of a 0 eigenvalue is equal to the number of connected components in the graph G, and thus only when the graph is connected is $\lambda_2(G) > 0$. Also, if $G = (V, E)$ and $H = (V, E' \subseteq E)$ then $\lambda_2(H) \leq \lambda_2(G)$. That is removing edges to a graph either maintains or decreases the algebraic connectivity of the graph; correspondingly adding edges either maintains or increases the algebraic connectivity of the graph.

A. Problem Statement

We study the following problem: given a graph $G = (V, E)$, a set of candidate edges $E_{cand} \subseteq E$, a non-negative number k, and a target connectivity t, can we choose a set of edges $A \subseteq E_{cand}$ to remove from G where $|A| \geq k$ and the result is a subgraph $G' = (V, E' = E - A)$ that leads to G' having the closest connectivity to t. That is we want to solve the problem

$$\begin{aligned} &\text{minimize } |\lambda_2(G') - t| \\ &\text{subject to } G' = (V, E') \\ &\qquad |A| \geq k \\ &\qquad A \subseteq E_{cand} \subseteq E \\ &\qquad E' = E - A \end{aligned} \tag{1}$$

This can be formulated as a decision problem by providing a range t_{min} and t_{max} instead of t. We then ask if there exists an edge set A following the above constraints, such that $G' = (V, E')$ has algebraic connectivity $t_{min} \leq \lambda_2(G') \leq t_{max}$.

To get the optimal solution for finding a subgraph G' of G such that $t_{min} \leq \lambda_2(G') \leq t_{max}$, we can set $E_{cand} = E$ and set $k = 0$. Our experiments focus on this case, where $E_{cand} = E$ and $k = 0$, but both our provided algorithm and our analysis of the problem, provided at the end of the paper, supports the full problem where $E_{cand} \subseteq E$ and k is any non-negative number. We often refer to a subgraph as a configuration where a set of equivalent configurations have the same edge set, while configurations are unique if their edge sets are different.

This problem can be solved exactly combinatorially by performing an exhaustive search where the connectivity of $\sum_{i=1,\ldots,|E_{cand}|} \binom{|E_{cand}|}{i}$ graphs. However this is not practical for large $|E_{cand}|$ where this can be the size of $m = |E|$.

B. Algorithm Definitions

1) Algorithm Model and Assumptions: In systems, sometimes it is useful to assign weights to edges in terms of value or efficiency. Thus, for the algorithm, we relax the assumption that the graph is simple. However, the experiments are still performed using simple graphs. The algorithm is designed to operate on one connected component since this paper mostly explores algebraic connectivity, thus the algorithm exits if the graph is disconnected; this is because after graph is disconnected, its algebraic connectivity is always exactly 0. The algorithm is designed to be flexible and composable framework, allowing for operations based on different target metrics and optimization heuristics. When using other metrics than algebraic connectivity, it is suggested to use a Depth-First-Search (DFS) algorithm to prepare the input and send each selected component separately to the algorithm. We will discuss later how this tactic can be used to apply the algorithm to even more applications by operating on graph complements.

2) Robotic Algorithm Context: In this paper, the algorithms were designed with both the original problem statement and the robotic context in mind. We design experiments with the "communication network" or "communication topology" of a robotic network in mind, although this approach can be applied to any graph. In the robotic context, a system consists of n independent mobile agents and m communication links. We allow this communication links to assigned between any two agents; this is to allow a wide-range applications; in the real-world links may be determined by communication frequency or physical constraints like euclidean distance between two agents. In robotic or networking applications, the candidate edge set may consist of any links that are allowed to be omitted and exclude links that are required for key functionality (for example, direct links between a leader and their followers); in a homogeneous system the candidate edge set is most likely the full edge set.

V. Preliminaries

After recognizing that removing one edge from a graph can have significant effects on the algebraic connectivity of the

graph depending on which edge was removed, we decided to do a cursory exploration of the problems result space and the configurations space. To do this we needed a way to explore many different graph configurations and to exactly solve the problem we proposed.

A. Exhaustive Search: The Decision Tree Algorithm

To allow for precise exploration and solving of algebraic connectivity on subgraphs, we presented an algorithm we refer to as the "Decision Tree" algorithm. This algorithm explores the set of possible subgraphs by removing edges in a Depth-First-Search (DFS) style; this can also be done in a Breadth-First-Search fashion if preferred. Starting with a graph $G = (V, E)$, the algorithm calculates the algebraic connectivity $\lambda_2(G)$, adds $(\lambda_2(G), G)$ to a state set s, and recurses removing each edge $e \in E$. When the algorithm recurses, it passes $G' = (V, E' = E - e)$; all recursions share the same state set; if the edge set of the input is a configuration in the state set, meaning $(\lambda_2(G), G) \in s$, then it terminates without recursing. This prevents the algorithm from exploring a branch for a configuration that was previously explored. As a result, the algorithm explores all combinations of removing edges such that $|S| = 0, \ldots, m$, and does not concern itself with permutations in the ordering of removing edges. This is extremely important to prevent the already computationally intensive algorithm from being even less efficient. We can obviously do this because two graphs with the same vertex set and edge set have the same algebraic connectivity no matter how edges were removed prior to consideration. When solving for the configuration $G' = (V, E' \subseteq E)$ that has algebraic connectivity closest to the target t, we perform a search for $min_{G'}|\lambda_2(G') - t|$ in the state set s. It is clear that this is a very computational expensive algorithm but it guarantees the correct answer. To reduce computation, the algorithm takes in an optional parameter that indicates that the target algebraic connectivity is a one-sided lower bound; this allows us to terminate the tree search whenever the current configuration has $\lambda_2(G') < t$; as a result, the number of explored steps and leaves of the decision tree can be drastically reduces as t increases.

B. Initial Exploration

To better acquit ourselves with the configuration space, we used the Decision Tree algorithm to generate all subgraph configurations for three initial graphs as seen in "Fig. 2". We then plotted the algebraic connectivity for each configuration in "Fig. 1". From the plot, we see a number of patterns. As expected, the more edges a graph has the higher the maximum subgraph's connectivity (where the maximum subgraph is the input graph). We also notice that the fewer the edges, the smaller the change in algebraic connectivity, as visible by fewer configuration as you go to the higher end of the range. Another takeaway is that there are fewer unique Fiedler values (algebraic connectivity values) than unique configurations, meaning different configurations or subgraphs can have the same algebraic connectivity. This is inline with a previous observation: removing an edge does not always change the

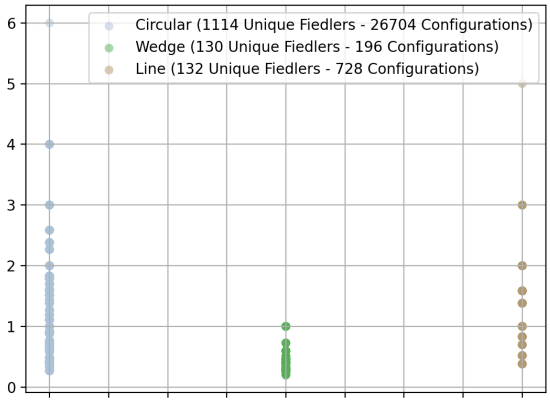


Fig. 1: Each formation's subgraphs plotted as a dot with the y-value equal to the algebraic connectivity. Tests were performed on the formations shown in "Fig. 2"

algebraic connectivity; sometimes after removing an edge, the connectivity is the same. This also follows from the quadratic form of the Laplacian: $x^T Lx = \sum_{(a,b)\in E} w_{a,b}(x(a) - x(b))^2$; by analyzing this form, we see that whenever $x(a) = x(b)$ the Laplacian is not always affected by removing an edge; thus when considering the algebraic connectivity λ_2, we can remove an edge without affecting the algebraic connectivity (Fiedler value) if and only if $\psi_2(a) = \psi_2(b)$ where ψ_2 is the Fiedler vector; this motivates a future configuration "Small" and "Big Gradient". Lastly, the obvious observation that there are more unique Fiedler values for a graph with more edges provides a useful takeaway for the problem studied in this paper: a graph with more edges has a larger set of achievable connectivity values; therefore reaching a target connectivity value is more likely possible with a graph with more starting edges.

VI. Algorithm

A. Motivation

Since this paper was originally inspired with a motivation for multi-agent robotic systems, we wanted to present an algorithm that was cognitively simple (meaning it was easy to reason about), flexible to different heuristics or a growing understanding of connectivity, and extensible to other spectrum (like λ_{max}) or connectivity metrics.

We also recognized that this was an exploratory study and furthermore, robotic systems often ran algorithms that were at least $O(n^3)$ in time complexity, thus our algorithm was not designed to be computationally efficient beyond being $poly(n)$. We leave it to future works to optimize this algorithm or provide alternatives in the same spirit that leverage specific spectral properties to gain computational advantages. Two obvious routes for optimization include reducing the complexity of calculating an edge's value (this approach is used with "Small Gradient" and "Big Gradient") and reducing the complexity of choosing which edge to choose without

calculating a value for each edge (this approach is used for "Randomized").

As a result, we built a general framework program that accepts a metric and heuristic as a parameter and approaches our original problem and that can be adapted to other problems by providing specific metrics and heuristics. We first introduce the algorithm and then discuss a number of heuristics we explored.

In our primary investigation and experiments we set the metric calculation function to return the Fiedler value, the edge action to cut an edge, the candidate set to be the edge set of the graph, and $k = 0$.

B. The Algorithm

The algorithm (as shown in "Algorithm 1") takes in two three high level configuration parameters: a metric calculation function, a heuristic-based edge selection method, and an edge action. The calculation function defaults to a function that returns the second smallest eigenvalue of Laplacian of the input graph, since this paper is most interested in algebraic connectivity; however this can be a function that returns any number; the function takes in the current augmented graph and the edge being considered; it is also used to measure progress where it is passed just the augmented graph to be compared with the target value; more advanced applications can return other connectivity metrics (e.g. λ_{max}) or even operate on the compliment graph. The heuristic-based edge selection method is used to determine which edge to choose to perform an edge action on; as our understanding of connectivity evolves, we can use more precise methods here; in this paper we explore different heuristics; this method takes in a list of edges to choose from and either returns one edge or no edge to indicate to terminate. The last parameter is the edge action and can either be to cut the edge or add the edge.

The algorithms take in, a graph G representing the original network, a set of candidate edges, a k_{min} to indicate the minimum number of edge actions (used mainly for removing edges), a k_{max} to indicate the maximum number of edge actions (used mainly for adding edges), and a target metric value t, and outputs a subgraph G'. At a high level, the algorithm starts with $G' = G$ and then evaluates performing an edge action on each edge to bring the desired metric value of G' closer to t. This process loops, modifying G' to continue bringing the metric value closer to the desired target, until no action could bring the metric value closer to our target. Then the algorithm stops and returns G'. The challenge is in choosing the correct edges to perform an action on. Thus the program accepts a heuristic-based method to identify which edge act on.

Each algorithm has a number of extra flags. We include a $allow\_disconnected$ boolean flag; when $false$ (default), this prevents the algorithm from disconnecting the graph. We also include a parameter $bound$ which can be "one" or "two"; this indicates if the desired metric value is a lower bound or two sided target; in the case of a one sided bound, the algorithm will not return a graph below the desired metric value. This ensures that any performance guarantees that may be associated with being at or above the desired metric value are protected. Our experiments have $bound =$ "one".

Algorithm 1 Algorithm

```
function FindEdge(g,cand_edges, f, target)
    options ← []
    for all (u, v) ∈ cand_edges do
        g_next ← EdgeAction(g, u, v)
        f_next ← CalcMetric(g,g_nextu, v)
        ▷ Consider all edges that bring us closer
        if |f_next − target| < |f − target| then
            options.append((g_next,(u,v)))
    return HeuristicDecider(options)
function Augment(g,cand_edges,k,target)
    f ← CalcMetric(g)
    while f > target and  do
        res ← FindEdge(g, cand_edges, f, target)
        if res == null then break
        f_next, g_next, edge = res
        cand_edges.remove(edge)
        f, g = f_next, g_next
    return (f, g)
function CreateGraph(g, target)
    return CutEdges(g, target)
```

C. Algorithm Configurations

In our paper, we explored a number of configurations and compared them in our experiments. These configurations give guidance as to how to implement the final algorithm. They also provide inspiration for ways which this algorithm can be used to explore other problems.

1) Removing Edges by $\Delta\lambda_2$: This is the default configuration of the algorithm. In this configuration the metric in question is the algebraic connectivity of the graph: $\lambda_2(G')$. The edge action is cutting an edge. The edge selection heuristic considers the graph without the edge and measures its Fiedler value. This then has two variations **"Small Cut"** where the edge that changes the Fiedler value the least is removed, and **"Big Cut"** where the edge that brings the Fiedler value closest to the target is chosen. This can be presented as "Algorithm 2" which was explored and extended in [1].

2) Removing Edges by Leverage Score: This configuration is the same as the previous configuration but with a different HeuristicDecider. This configuration showcases how a different heuristic can be used with the same metric; specifically we are still interested in algebraic connectivity but we are testing a different method of choosing which edges to cut. In this method, we use the leverage score of the edge in the augmented graph as defined by [11]. The leverage score of an edge is

$$\ell_e = w_e R_{eff}(e)$$

where w_e is the weight of the edge and $R_{eff}(e = (a, b)) = (\delta_a - \delta_b)^T [L(G')^+](\delta_a - \delta_b)$ where δ_a is the indicator vector where $\delta_a(i) = 1$ iff $i = 1$. This has two variations, choosing

Algorithm 2 Remove Edges by $\Delta\lambda_2$ ("Small" Version)

```
function FindEdge(g,e_set, f, target)
    options ← []
    for all (u, v) ∈ e_set do
        f_next, g_next ← WithoutEdge(g, u, v)
        ▷ Consider all edges that bring us closer
        if |f_next − target| < |f − target| then
            options.append((f_next,g_next,(u,v)))
    dist ← [|f − target| for (f,g) in options]
    return min(dist, key= o[0], default= null)
function CutEdges(g, target)
    e_set ← EdgesAsList(g)
    f ← CalcFiedler(g)
    while f > target do
        res ← FindEdge(g, e_set, f, target)
        if res == null then break
        f_next, g_next, edge = res
        e_set.remove(edge)
        f, g = f_next, g_next
    return (f, g)
function CreateGraph(g, target)
    return CutEdges(g, target)
```

the edge with the largest leverage score (labeled **"Big Leverage"**) and choosing the edge with smallest leverage score (labeled **"Small Leverage"**).

3) Removing Edges by Gradient Heuristic: Again this configuration is the same as the first but with a different HeuristicDecider. This configuration using a gradient heuristic based on the Fiedler vector and presented in [10]. Here the calculated of the heuristic is

$$(\psi_2(G'_{last})(i) - \psi_2(G'_{last})(j))^2$$

for an edge $e = (i, j)$ and where $\psi_2(G)$ is the eigenvector that corresponds to the second smallest eigenvalue of the Laplacian of G; G'_{last} refers to the augmented graph before the action is taken, in the first iteration this is the input graph G. This should be reminiscent of the quadratic form of the Laplacian and is actually an upper bound on $\Delta\lambda_2$, meaning it is an upper bound on the change of connectivity. This has two variations, choosing the edge with the largest gradient (labeled **"Big Gradient"**) and choosing the edge with smallest gradient (labeled **"Small Gradient"**).

An implementation of this is shown in "Algorithm 3". This algorithm showcases how a heuristic can be used to assign a value to an edge more efficiently; this method only requires a subtraction and squaring for each edge by using the Fiedler vector of the current graph and then assigning a value based on the vector which was calculated once per action loop. There is still expense in testing and adding potential edges to a list; however, if this were removed and edges were not validated before being put into the list, instead validated when being chosen form the list, then this configuration would be the most efficient out of all those presented; this tactic can be applied to other configurations, the most natural next one being leverage; these efficiencies reduce the runtime up to a factor of $O(|E_{cand}|)$, allowing such an algorithm to be close to $O(|E_{cand}| \cdot n^3)$ instead of $O(|E_{cand}| \cdot |E_{cand}| \cdot n^3)$; runtime is explained more later in "Algorithm: Brief Note on Runtime".

Algorithm 3 Remove Edges by Gradient Heuristic ("Small" Version)

```
function FindEdge(g,e_set, f, target)
    options ← []
    for all (u, v) ∈ e_set do
        f_next, g_next ← WithoutEdge(g, u, v)
        ▷ Consider all edges that bring us closer
        if |f_next − target| < |f − target| then
            options.append((f_next,g_next,(u,v)))
    fv = FiedlerVector(g)
    val ← [(fv[u] − fv[v])^2 for (f,g,(u, v)) in options]
    return min(val, key= o[0], default= null)
```

4) Adding Edges by $\Delta\lambda_2$: Here we discuss an additional configuration. This was not included in our experiments as it solves a different problem than what the paper originally studied. However the problem this does study, a heuristic for adding edges to graphs, is a well studied one. In fact, future works can apply it to [9], which we discuss in our analysis of the problem, and can be compared to [10].

This configuration is the same as the first three except with an edge action of Edge Addition. This is a widely applicable use case. This can be used as a baseline for the next configuration.

5) Adding Edges by Removing Edges with $\Delta\lambda_{max}$: This one we did not include in our experiments either, for the same reason as the previous configuration.

In this configuration, we set the metric in question to be $\lambda_{max}(G'^C)$. Any of the above heuristics could be used on the complement graph or min/max $\lambda_{max}(G'^C)$ could be used in spirit of the first configuration. The edge action is removing edges from the complement graph. Each iteration a DFS algorithm is used to split the matrix for G^C into the connected components and then pass each connected component into the algorithm described above. Perform this algorithm until all connected components are close to the threshold.

By a property presented in [9], we know that $\lambda_2(G) = n - \lambda_{max}(G^C)$ where $\lambda_{max}(G^C) = max_{i=1,\dots,p}\lambda_{max}(C_i)$ for connected components $C_1, \dots, C_p$ in G^C. Thus by removing edges from the connected components of G^C we attempt to lower λ_{max} for each connected component and thus the overall G^C, thereby increasing the algebraic connectivity of G. This makes sense since removing edges from the complement graph means adding edges to the original graph. But it may mean a more predictable/controlling way to increase the connectivity of G. We leave it to future works to rigorously evaluate this.

D. Randomized Algorithm

The randomized algorithm (as presented with the default configuration in "Algorithm 4") attempts to bypass the greedy trap of making a wrong choice that leads to a suboptimal result. Thus instead of implementing a heuristic, this algorithm takes in the candidate edge set and removes any invalid edges

(for example edges that lower the fielder value past the desired target) and then chooses a random edge from the remaining set. Note that since this filters "invalid" edges, this algorithm requires a one-sided lower bound or two target parameters that act as a target range, in which case the lower target in the range will serve as the one-sided lower bound.

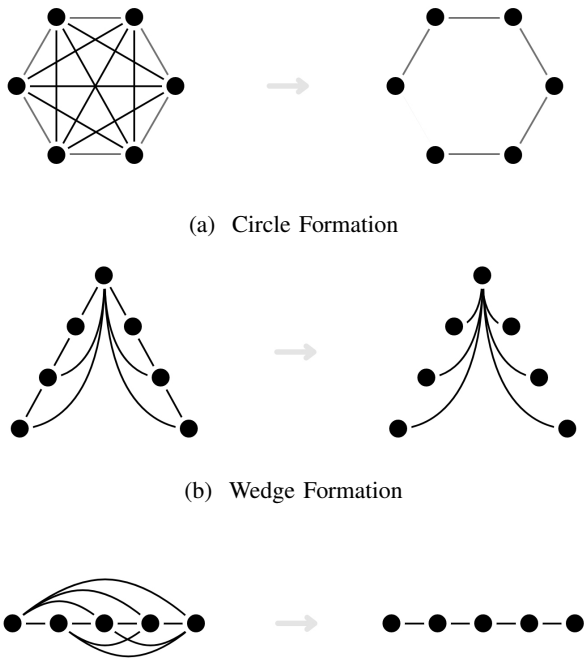
For probabilistic amplification, the algorithm runs the routine multiple times, then returns the G' that has a metric value closest to the desired value. Probabilistic amplification is well studied concept; a larger number of runs increases the algorithm runtime (most of the time by a lower order factor or a constant), but results in a significantly higher probability of success. The number of times is an algorithm parameter; in the experiments, we set this parameter to $runs = c \cdot n$ where n is the number of nodes/agents; we use $c = 3$ to show the possibilities of failure with this algorithm; as this number increases, it becomes much harder to find suboptimal results.

Algorithm 4 Randomized Algorithm

```
function FindEdge(g,e_set, f, target)
    edges_considering ←copy_of(e_set)
    while len(edges_considering) do
        u, v ← random_edge(edge_considering)
        f_next, g_next ← WithoutEdge(g, u, v)
        ▷ Take edge if it brings us closer to the target
        if |f_next − target| < |f − target| then
            return f_next, g_next, (u,v)
        edges_considering.remove((u,v))
    return null                    ▷ No edge brought us closer
function CreateGraph(g, target, runs)
    graphs ← []
    for i do1runs
        ▷ CutEdges same as "Algorithm 2"
        graphs.append(CutEdges(g, target))
    dist ← [|f − target| for (f,g) in graphs]
    return min(dist, key= o[0], default= null)
```

E. Brief Note on Runtime

The final runtime of the algorithm depends on the high level parameters. It was also mentioned that this algorithm prefers flexibility and simplicity over computation efficiency. We leave full optimization to future works. However, we do point out that if the metric can be calculated in polynomial time, the edge selection heuristic runs in polynomial time, and we assume the edge action is only adding or removing edges which is constant work, then the algorithm runs in polynomial time. We also point out that all our deterministic configurations above run in order $O(m_c * m_c * \ell)$ where $m_c = |E_{cand}|$ is the size of the candidate edges and ℓ is the time it takes to solve for the metric, which in the above cases is solving for the Laplacian. Thus the above configurations have a loose bound of $O(m_c^2 * n^3)$. An obvious opportunity for optimization is using a metric or calculating a metric in a way that does not require recalculation on each edge calculation or can be modified instead of recalculated for each edge; we presented a configuration of this type in "Small Gradient" and "Big Gradient".



(a) Circle Formation

(b) Wedge Formation

(c) Line Formation

Fig. 2: The original formation and its corresponding tree for all three formations: Circle (a), Wedge (b), and Line (c).

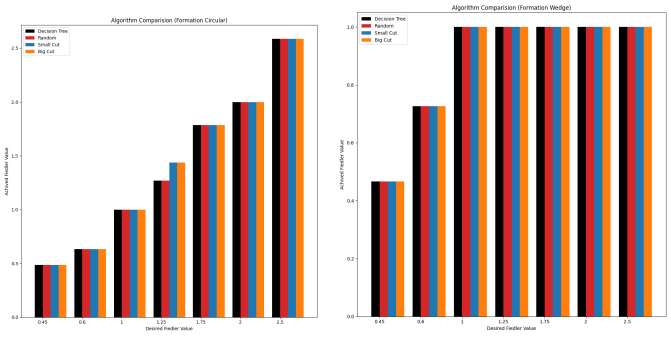
VII. Experiments

The initial inspiration for the paper was to explore different methods reducing algebraic connectivity. We conjectured (and then showed in "Further Analysis of the Problem") that the problem was NP-complete. Yet, we still wondered if we could relax the problem where $E_{cand} = E$ and $k = |E|$, could we make headway with a greedy approach. Thus we came up with the configurations above. We determined that even if a greedy approach did not provide the optimal answer, it could be used to reach approximate solutions and then utilized in a number of applications.

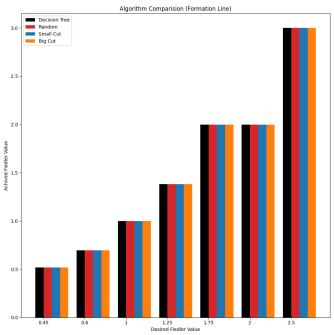
A. Round 1 of Extended Investigation

For our first round of experiments we started with three formations (as shown in "Fig. 2") that represent common multi-agent formations and that have varying algebraic connectivity. We selected these formations because these formations were ones that are commonly used in practice, and could more accurately represent realistic scenarios in robotics.

To compare the different algorithm configurations and heuristics, we tested across 7 different desired target algebraic connectivity values and measured the resultant connectivity value to examine the precision of each algorithm. In "Fig. 3", we can see that as a whole, all 4 algorithms perform very comparably, all resulting in very similar connectivity values.



(a) Circle Formation (b) Wedge Formation



(c) Line Formation

Fig. 3: The comparison between the algorithms first two configurations on each formation. The x axis is the target Fiedler value, the y axis is the achieved target Fiedler value.

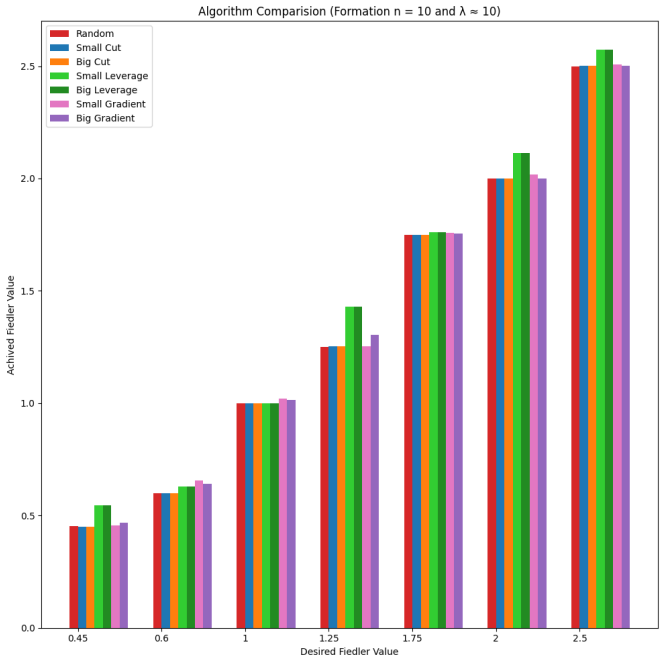
This is evident by the bar for each algorithm having the same height as the Decision Tree algorithm (black bar) and as each other. The exception is on the Circle formation, with target connectivity value as 1.25, where the randomized (red bar) and the decision tree algorithm (black bar) both outperform the other two algorithms. The randomized algorithm tends to be the most precise out of the algorithms. The exception in performance was enough to validate our hypothesis that these deterministic algorithms were not immediately optimal and motivated a round 2 of testing.

In these first round of tests we confirmed that multiple edge sets that reach the true closest Fiedler value by having each algorithm print the edges it cuts, and realizing that the different algorithms returned a graph with different edges but the optimal connectivity value. Conversely, looking at the results for the Line Formation, as seen in "Fig. 3c", we see that not all target connectivity values are achievable; referencing "Fig. 1" where we did our exploration of the target space, we see that 1.75 was not in the space of unique Fiedler values and thus it was not reachable even by the Decision Tree which gets the exact answer; instead because we were using the one-bound/lower bound setting, the algorithm returned the graph with the next closest connectivity which was a graph with a connectivity of 2. We also confirmed that removing a edges in a greedy approach according to the tested heuristics can make it impossible to then reach the true closest connectivity value as a result of "going down the wrong path". This is clear in the Circle formation, where the Randomized (red bar) algorithm is able to get closer to the true closest Fiedler value, while neither of the tested deterministic algorithms were able to reach the true optimal. There may be another metric that enables a greedy algorithm to reach the optimal answer. We conjecture is that, as the space of possible connectivity values increases and the starting number of edges increases, there is a higher probability of removing the "wrong" edge and missing the true optimal; however, we have also discussed that as the number of edges increases, the number of achievable connectivity increases and there are multiple configurations with this target connectivity value because there are fewer unique connectivity values than configurations/subgraphs; thus while the probability of removing the "wrong" edge increases for a given configuration, as the number of equivalent configurations grows or the number of edge actions required grows, the probability of arriving at a configuration with the target connectivity increases; these conflicting notions require further investigation in future works. Looking at the Line formation and the Circle formation, which are both fully connected, we see that the deterministic algorithms (blue and orange bars) do not arrive at the true closest Fiedler value as found by the Decision Tree (black bar) on the Circle but it does on the Line. We see that the Randomized algorithm (red bar) reaches the true closest value; this is not a performance guarantee but significantly more probable than the deterministic algorithm because this algorithm choosing different edges and using amplification to take the best configuration.

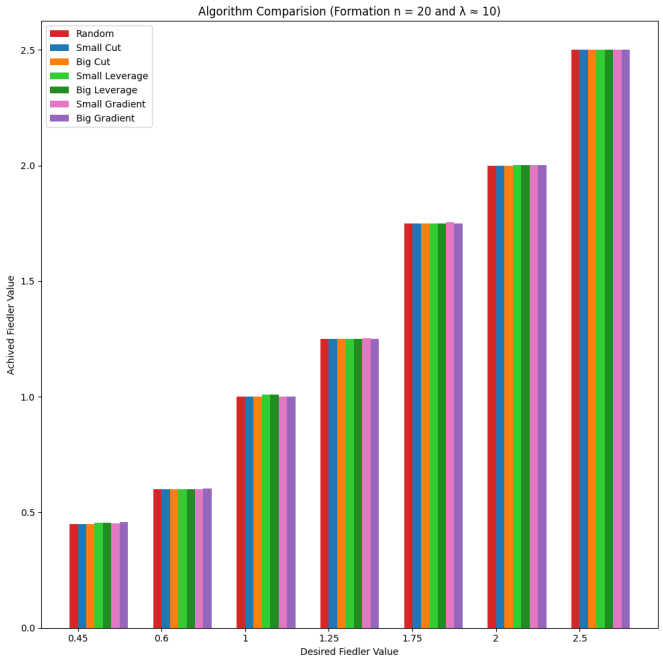
B. Round 2 of Extended Investigation

In our second round of experiments, we increased the tested graphs and the tested configurations to include the Leverage and Gradient configurations. We used the Randomized algorithm to generate a number of semi-sparse graphs; the size of these graphs ranged from 4 nodes to 100 nodes with a starting connectivity of around $\lambda_2(G) \approx 10$; note that a graph with fewer than 10 nodes cannot reach that connectivity, so in this case the fully-connected/maximum connectivity subgraph is used. These graphs were generated by starting with a fully connected graph of different numbers of nodes, which were passed to the Randomized algorithm (presented above) with a target algebraic connectivity parameter higher than the target connectivity parameters in our tests. We did this so we would have a wide range of graphs and the Randomized algorithm provided the most variation in graph "shape"/edge sets (this is explained in the discussion). Since we used a one-sided bound/lower bound in our experiment, no algorithm returned a graph with a connectivity below the target; thus, the best performing algorithm for each target value, is the algorithm that returned the graph with the lowest connectivity as visualized by the lowest bar. It is also worth noting that whenever a bar is equal height across all algorithms, this indicates that all algorithms did equally well.

After graphing the six configurations and the Randomized algorithm (red bar) on more graphs, it became clear that all the heuristics studied were suboptimal with small graphs. We tested around 15 different input graphs where each input graph was a plotted as a different figure. On each input graph and representative figure, we tested and plotted 7 connectivity



(a) Input Graph: $n = 10$ and $\lambda_2 \approx 10$



(b) Input Graph: $n = 20$ and $\lambda_2 \approx 10$

Fig. 4: The comparison between the algorithms size configurations on randomly generated semi-sparse graph. The number of nodes and connectivity of the input graph is noted in the subfigure caption. The x axis is the target Fiedler value, the y axis is the achieved target Fiedler value. The lowest bar is the best performing algorithm.

targets. We handpicked a few figures and set the number of amplification runs lower to show where the algorithms failed. We chose "Fig. 4b" and "Fig. 4**??**" because they showed a large number of failures across all configurations; but they also show a more important pattern. As the size of the graph increases, all algorithms are able to perform better; by around 30 nodes (as shown in "Fig. 5a), all algorithms showed no errors or suboptimal results, which is evident by all performing equally and resulting in a bar of equal height that had the desired Fiedler value (x-axis) as the achieved Fiedler value (y-axis); this trend continued in our tests which tested graphs where $n = 30, 40, 45, 50, 55, 75, 100$ (we show all these charts in our appendex). We also tested a case with an input graph with $n = 40$ and $\lambda_2(G) \approx 38$ with tested desired eigenvalues $\lambda' = 36.5, 37, 37.5, 38, 38.5, 39, 39.5$ to investigate the case of a larger graph where the connectivity was close to the input graph, meaning few edges were being removed; even in this case, all algorithms reached the desired Fiedler value; this leads us to the important conclusion: **heuristic greedy algorithms can reliably be used to remove edges from a graph to reach a target connectivity when the graph is large enough (empirically this means $n \gtrapprox 30$).** Across all the tests, the Randomized algorithm (red bar) did best, especially when the number of amplification runs was not reduced. Next best were the configurations that used $\Delta\lambda_2$ ("Small Cut"/blue bar and "Big Cut"/orange bar). We noticed that if you ordered the graph by nodes and input connectivity, that the leverage and gradient configurations failed more earlier and more often, leading us to hypothesize that the best heuristic is in fact just using the connectivity directly: $\Delta\lambda_2$ ("Small Cut" and "Big Cut" configurations of the algorithm).

We noticed that the heuristics seemed to be suboptimal relative to their value; that is, they tended not to perform worse than the next tested value or the second next tested value; we know this is not the consecutive connectivity values in the space of all subgraph connectivity values, but it does recommend that the difference in optimality is bounded by or correlated with some undetermined terms.

C. Discussion

After exploring the heuristics we are left with more questions than answers. However, we do conclude that the size explored heuristics are not optimal in a greedy fashion. We also recognize that when the number of input edges increases, the precision of Fiedler values increases but so does the possibility of reaching the optimal target with a greedy approach. We note that the Randomized algorithm performs best and increases in success as the number of amplification runs increase. It was also discovered by inspection of outputted graphs and reasoning of the algorithms that the graphs outputted by the different algorithms configurations and the randomized algorithm can differ in another quite significant way. Depending on the heuristic of the deterministic algorithm, the outputted graph can result in a tightly connected "component" or subset of vertices and then one vertex with one or very few edges, leading to a "banished"/"alienated" vertex; this also means that the algorithm outputs a very imbalanced graph. The randomized algorithm chooses a random valid edge to remove, this

TABLE I: Number of removed edges by each algorithm to generate the graphs required for some of the figures in round 2. Each row corresponds to an input graph corresponding to one test/figure which involved each algorithm reducing the graph for 7 different desired connectivity values: $t = \lambda_2(G') \rightarrow 0.45, 0.6, 1, 1.25, 1.75, 2, 2.5$; with the exception of $n = 40$ and $\lambda_2 \approx 38$ which set $t = \lambda_2(G') \rightarrow 36.5, 37, 37.5, 38, 38.5, 39, 39.5$

Input Graph	Removed Edges Per Algorithm						
	Randomized	Small Cut	Big Cut	Small Leverage	Big Leverage	Small Gradient	Big Gradient
$n = 40$ and $\lambda_2 \approx 38$	51	7	7	36	36	29	34
$n = 40$ and $\lambda_2 \approx 10$	1361	243	243	1170	1170	1414	1279
$n = 20$ and $\lambda_2 \approx 10$	532	177	177	517	517	545	480
$n = 100$ and $\lambda_2 \approx 10$	2463	170	170	1784	1784	2522	2203
$n = 50$ and $\lambda_2 \approx 10$	1713	210	210	1423	1423	1690	1493
$n = 100$ and $\lambda_2 \approx 10$	2623	222	222	2010	2010	2623	2626
Total	**8743**	**1029**	**1029**	**6940**	**6940**	**8823**	**8115**

means that edges can be removed more uniformly across the graphs and the graph is more "balanced"; the graph also may have significantly fewer edges. To test this hypothesis about fewer edges, we logged how many edges were removed in each of our round 2 tests; note that these numbers correspond to the number of edge actions, in this case removed edges, which means a lot of removed edges results in a final graph with fewer edges. We included some of the logged results in "Tab. I". Here we see that Small Cut (blue bar) and Big Cut (orange bar), which used $\Delta\lambda_2$ directly, removed the least number of edges; this means that the resulting graph had the most remaining edges and thus was the largest; this also means that the least edge actions were performed and thus when compared to other algorithms with the same asymptotic factor, this configuration was the most efficient by a coefficient of $5 - 9$. Conversely, we see that Randomized (red bar), Small Gradient (pink bar), and Big Gradient (purple bar) removed the most number of edges, resulting in a the smallest final returned graph G'. Depending on the application, this information can be useful. If one wants to cut the fewest edges they should use Small Cut or Big Cut; while if someone wants to generate a graph with a target connectivity and the fewest edge (like our use case when we generated the test cases for round 2), one should use Randomized, Small Gradient, or Big Gradient.

VIII. Further Analysis of the Problem

A. Original Decision Problem

Here we show that our the decision problem presented at the beginning of the paper is in fact NP-complete. The decision problem receives an undirected simple graph $G = (V, E)$, a non-negative integer k, a set $E_{cand} \subseteq E$ of edges that are considered for removal, and a threshold range t_{min}, t_{max}. The question is then, if there exists an edge set $A \subseteq E_{cand}$ such that $|A| \geq k$ and the graph $G' = (V, E' = E - A)$ satisfies $t_{min} \leq \lambda_2(G') \leq t_{max}$.

1) NP: We can verify that $t_1 \leq \lambda_2(G') \leq t_2$ for an augmented graph G' and a threshold $t_{min}, t_{max}$by ensuring that the matrix $L(G') - t_{min}(I - U)$ and the matrix $t_{max}(I - U) - L(G')$ are both positive semidefinite, where U is an $n \times n$ matrix with entries equal to $\frac{1}{n}$. This test can be done with Gaussian elimination in polynomial time as shown in [12]. Thus this problem is in NP.

2) The Maximum Algebraic Connectivity Augmentation Problem: Now to show that the problem is NP-hard, we reference [9]. In [9], the authors show the maximum algebraic connectivity augmentation problem is NP-hard. In their paper, the authors use The Maximum Algebraic Connectivity Augmentation Problem to try to add $3n^2 + 3m$ edges to the input graph to obtain a complete 3-partite graph $K_{n,n,n}$ and with the goal of obtaining a target connectivity of $2n$. They show that $\lambda_2(K_{n,n,n}) = 2n$. They then show that if they are able to add that specified number of edges and reach the target connectivity value, then the resulting graph is isomorphic to $K_{n,n,n}$ if and only if G is 3-colorable. We reference their result to show that our problem is also NP-hard. To avoid confusion, we change some of the names of the variables.

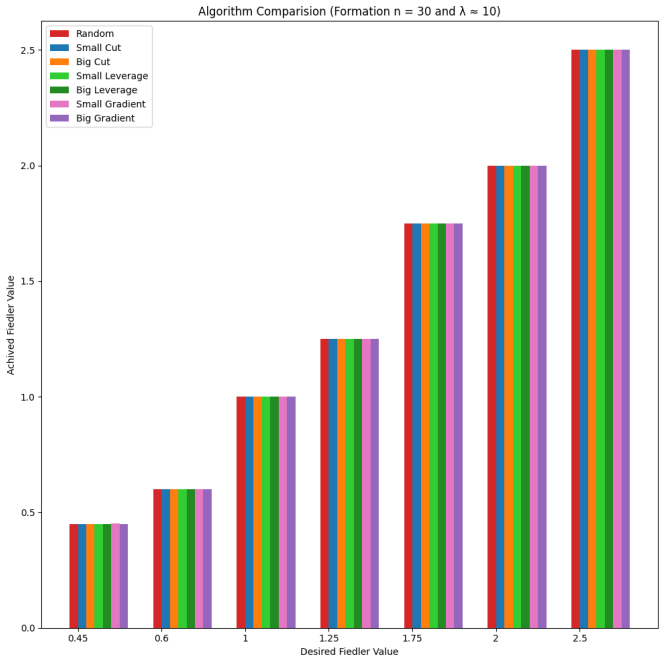
In this problem, an undirected simple graph $G = (V, E)$, a non-negative integer c, and a non-negative threshold t_{thresh} are provided. The question is then: does there exist a subset $B \subseteq E^C$ of size $|B| \leq k$ such that the graph $G' = (V, E \cup B)$ satisfying $\lambda_2(G') \geq t_{thresh}$.

3) NP-hard: We will now show that our original decision problem in NP-hard by reducing from The Maximum Algebraic Connectivity Augmentation Problem to our decision problem.

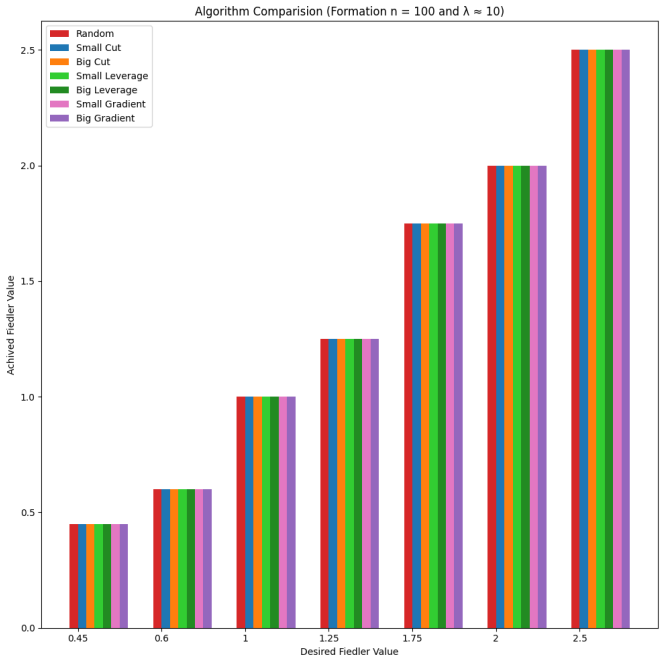
First we take in the input graph G and we create a initialize an empty set for E_{cand}; then for every edge in E^C, we add the edge to G and E_{cand} to create $G_{reduc} = (V, E_{reduc} = E \cup E^C = E \cup E_{cand})$; thus G_{reduc} should be the complete graph and $|E_{cand} = E^C = E_{reduc} - E|$. We then set $k = |E_{cand}| - c$. Lastly, we set $t_{min} = t_{thresh}$ and $t_{max} \geq \lambda_2(G_{reduc})$ (t_{max} can be arbitrarily large).

This transformation can easily be done in polynomial time. Generating the transformation graph G_{reduc} by adding the edges takes $O(m)$. Creating a list of size m takes $O(m)$. Setting k can be done in $O(m)$. Setting t_{min}, t_{max} takes $poly(n)$ if a slow Laplacian eigenvalue solver is used and takes constant time if t_{max} is set to a supported max/infinity value. Thus the entire transformation takes $poly(n)$.

We see that ACA problem and our problem with the reduced input are equivalent. The returned result from our problem is a set A of edges to remove from the graph resulting in $G' = (V, E_{reduc} - A)$; the ACA problem requires a set B of edges to add which results in a graph $G' = (V, E \cup B)$. With the above reduction $E_{reduc} - A = E \cup B$ because $B = E_{reduc} - A \cup E$. This is because our problem restrains the edges that can be removed to E_{cand}. Thus, the ACA problem looks at which edges can be added to maximize connectivity with less than c edges; our problem starts with adding all the edges (resulting



(a) Input Graph: $n = 30$ and $\lambda_2 \approx 10$



(b) Input Graph: $n = 100$ and $\lambda_2 \approx 10$

Fig. 5: (Continued) The comparison between the algorithms size configurations on randomly generated semi-sparse graph. The number of nodes and connectivity of the input graph is noted in the subfigure caption. The x axis is the target Fiedler value, the y axis is the achieved target Fiedler value. The lowest bar is the best performing algorithm.

in the complete graph) and removing $k = |E_{cand}| - c$ or more edges to result in a augmented graph with up to c added edges that maximizes connectivity.

B. The λ_{max} Case

We have presented an algorithm that can operate with different metrics to determine edge actions and when to terminate. As a result, we presented a scenario where our algorithm removed edges from the complement graph and measured λ_{max} of this complement graph. Here we will provide the decision problem formulation of this problem and show that it is NP-complete.

Given a graph $H = (V, E) = G^C$, does there exist a set of edges $A \subseteq E$ such that $|A| \leq \ell$ and the graph $H' = (V, E' = E - A)$ satisfies $\lambda_{max}(G') \leq t$. Notice that in this problem, we use a different ℓ which is the maximum number of edges that can be removed; we also use only one bound on the metric.

1) NP: Again this can be verified in polynomial time by solving for the largest eigenvalue of the Laplacian of H and seeing if it is greater than t. This can be done in $poly(n)$ using some form of a Gaussian solver.

2) NP-hard: Again we reduce from The Maximum Algebraic Connectivity Augmentation Problem to our decision problem.

However first we recall a few properties referenced in [9] and presented in [5]. First, for a graph $G = (V, E)$, $\lambda_2(G) = |V| - \lambda_{max}(G^C)$. Second, let $C_1, \ldots, C_p$ enumerate the connected components of a graph H; for this graph $\lambda_{max}(H) = max_{i=1,\ldots,p}\lambda_{max}(C_i)$. Thus we see that reducing $\lambda_{max}(H)$ is equivalent to increasing $\lambda_2(G)$ an equivalent amount, when $H = G^C$.

Thus to reduce from The Maximum Algebraic Connectivity Augmentation Problem to our decision problem. We take the input graph $G = (V, E)$ and feed our algorithm the complement graph $H = G^C$, we feed our algorithm the same k (the maximum number of edge actions), and then take the input t_{thresh} and feed our algorithm $t = |V| - \lambda_2$; which is to say we want the algorithm to return an H' such that $\lambda_{max}(H') \leq n - \lambda_2$; if this is satisfiable, we return $G' = (H')^C$ and by the properties above, we know that $\lambda_2(G') \geq t_{thresh}$, and by the properties of the algorithm, we know the size of the set of edges removed is less than or equal to k and this set is the same set of edges that were added to G to make G'.

This reduction can be done in polynomial time since the only computation in the conversion is counting the number of vertices in the graph, performing a subtraction, and calculating the graph's complement which can all be done in $O(n^2)$.

C. The Subgraph Case

'The work in [9] has provided us great guidance in classifying the previous problems; it has also shown us some strategies to tackle problems of this nature where we are adding of removing edges and reasoning about eigenvalues. In the time constraint for writing this paper, we were not able

to conclude whether or not the relaxed version of this problem, where $E_{cand} = E$ and $k = 0$, is NP-hard. We hope to further investigate this and encourage other authors to also consider this problem. This will be extremely useful for future works in this area. This was the form explored in the experiments. Even in this relaxed form, this version and the resulting optimization problem has a wide range of applications and is likely to be the most common parameter configuration.

IX. Conclusion

We were inspired to investigate methods of reducing algebraic connectivity in graphs by reducing edges. While we gained much guidance from past works, we were essentially exploring an opposite strategy. Motivated by the applications to multi-robotic systems we presented a simple and flexible algorithm that allows modifying a graph (adding or removing edges) on the basis of any number of metrics and heuristics. We tested this algorithm with a number of fitting heuristics and showed that each resulted in a suboptimal resultwhen used in a greedy approach on small graphs. However, our results showed that our efforts were successful in approximately reducing the connectivity of a graph, especially in medium and potentially large graphs. This has application is networking, robotics, and social networks. We have already begun applying this to the robotic context in [1] where we explore controlling consensus speed and controlling deployed network topology. We hope to explore methods of generating dynamic network topologies and multi-path routing systems, as well as consensus systems resilient to malicious agents.

However there is still much work left to be done to understand connectivity in general and in understanding how to construct or disconnect graphs to have a specific connectivity. There is also potential work in exploring effectiveness of different heuristics, as well as probability analysis of such heuristics. Further analysis of the problem presented in the paper revealed that the problem of removing edges and adding edges can actually be quite related. This revelation led us to show that the problem presented in this paper is in fact NP-complete. This should inspire work to further analyze this relationship. We also present that reducing λ_{max} is NP-complete by reasoning with the complement graph. Future work should investigate this and more properties of λ_2 because it is now clear that they are closely related.

X. Acknowledgements

The authors would like Harvard Professor Salil Vadhan and Harvard Professor Stephanie Gil for the guidance and support.

XI. Appendix

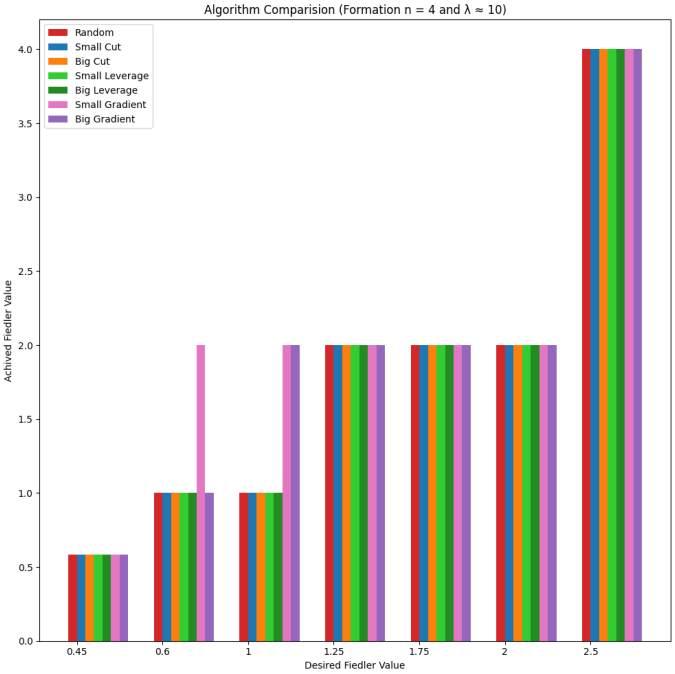


Fig. 6: Input Graph: $n = 4$ and $\lambda_2 \approx 10$

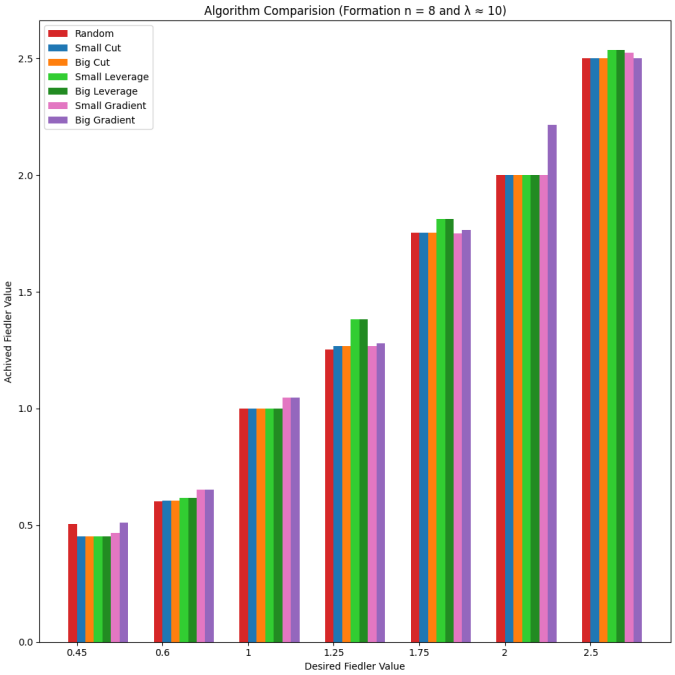


Fig. 7: Input Graph: $n = 8$ and $\lambda_2 \approx 10$

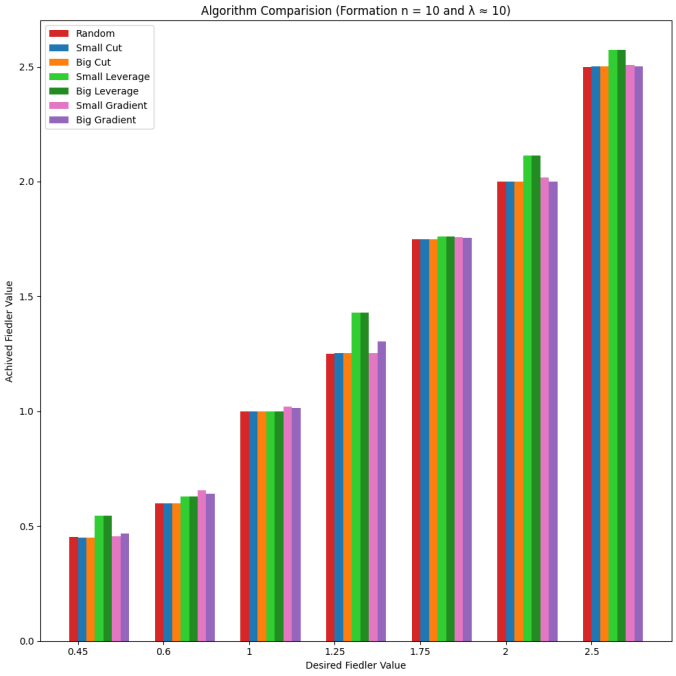


Fig. 8: Input Graph: $n = 10$ and $\lambda_2 \approx 10$

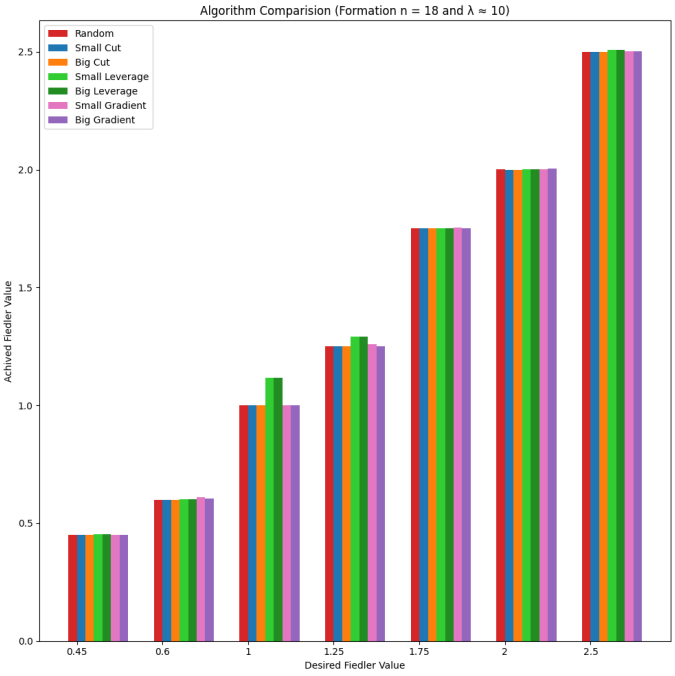


Fig. 10: Input Graph: $n = 18$ and $\lambda_2 \approx 10$

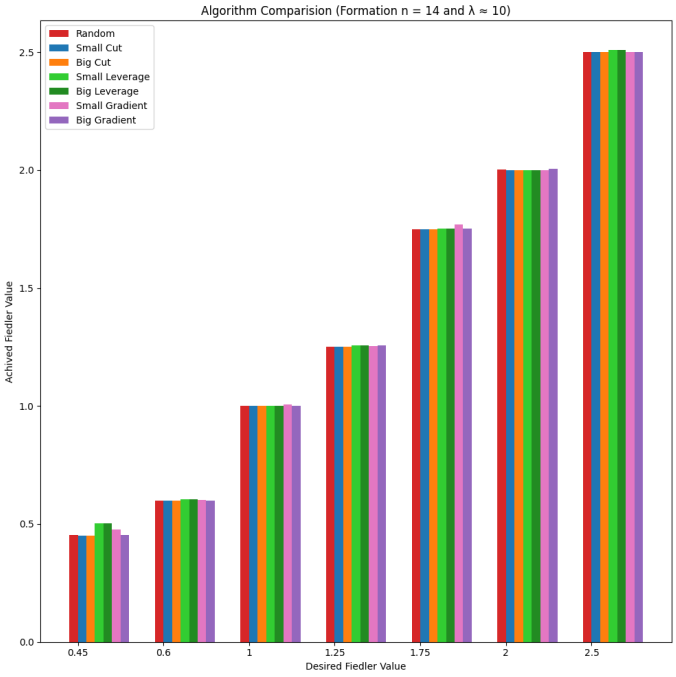


Fig. 9: Input Graph: $n = 14$ and $\lambda_2 \approx 10$

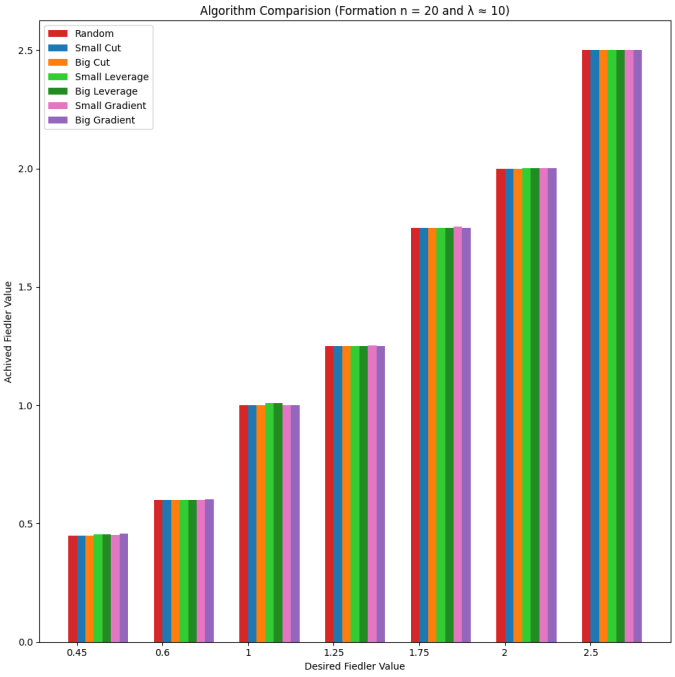


Fig. 11: Input Graph: $n = 20$ and $\lambda_2 \approx 10$

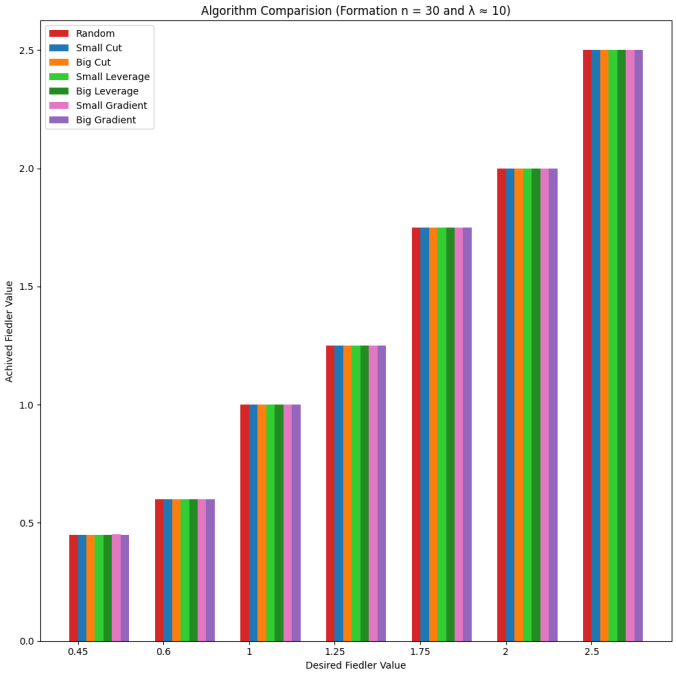


Fig. 12: Input Graph: $n = 30$ and $\lambda_2 \approx 10$

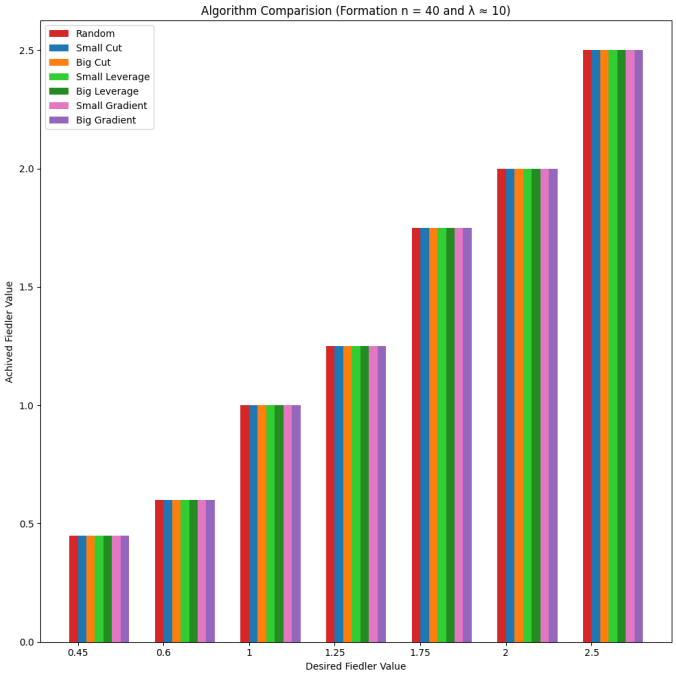


Fig. 13: Input Graph: $n = 40$ and $\lambda_2 \approx 10$

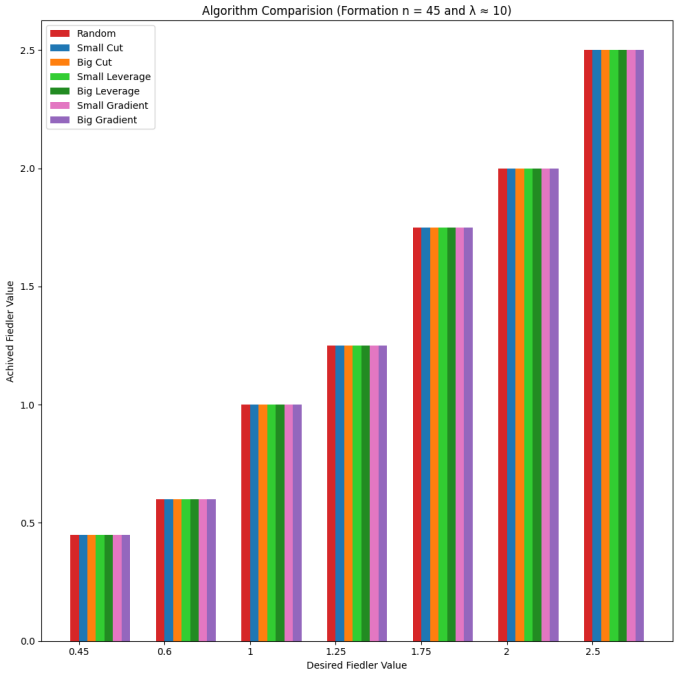


Fig. 14: Input Graph: $n = 45$ and $\lambda_2 \approx 10$

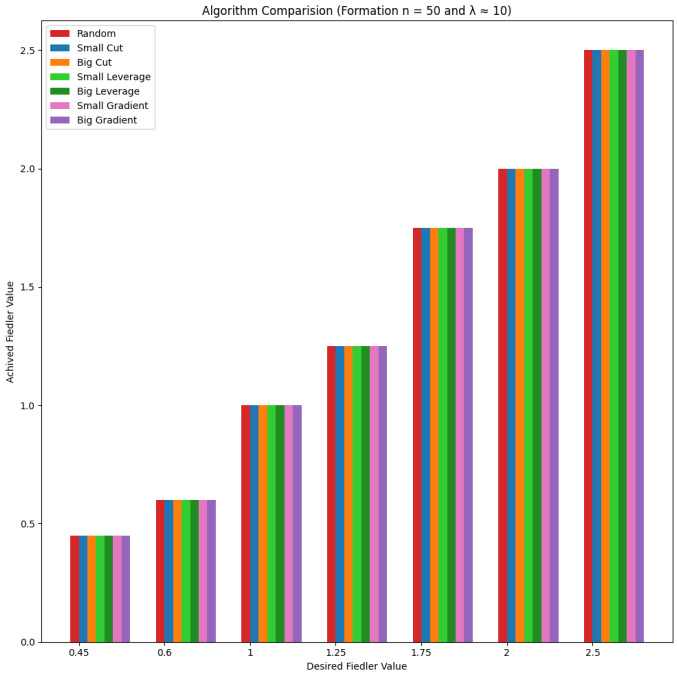


Fig. 15: Input Graph: $n = 50$ and $\lambda_2 \approx 10$

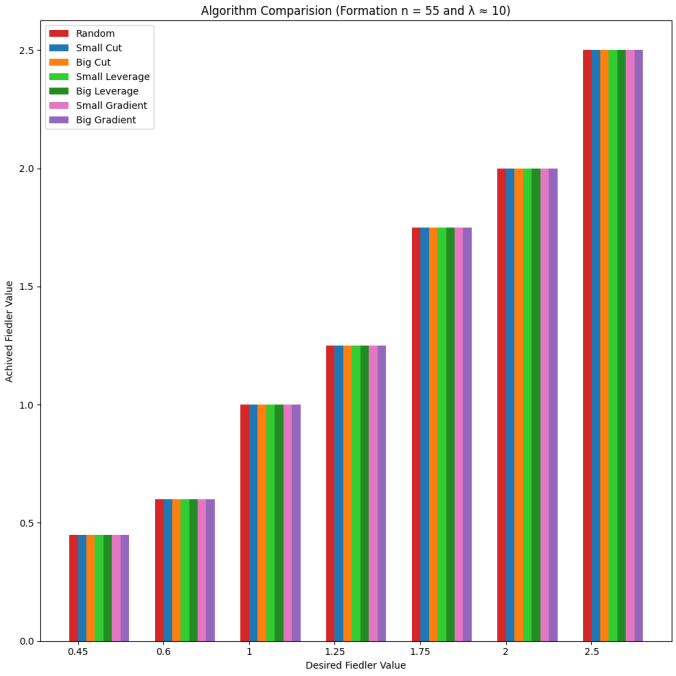


Fig. 16: Input Graph: $n = 55$ and $\lambda_2 \approx 10$

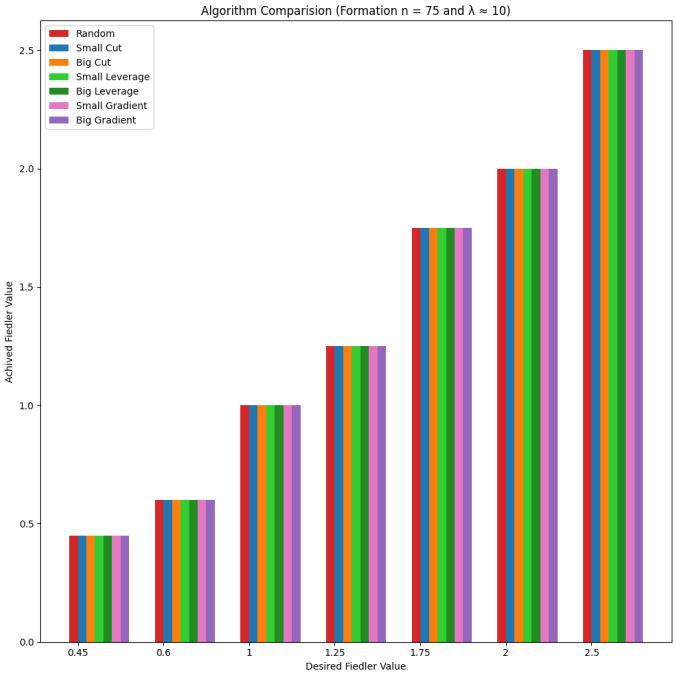


Fig. 17: Input Graph: $n = 75$ and $\lambda_2 \approx 10$

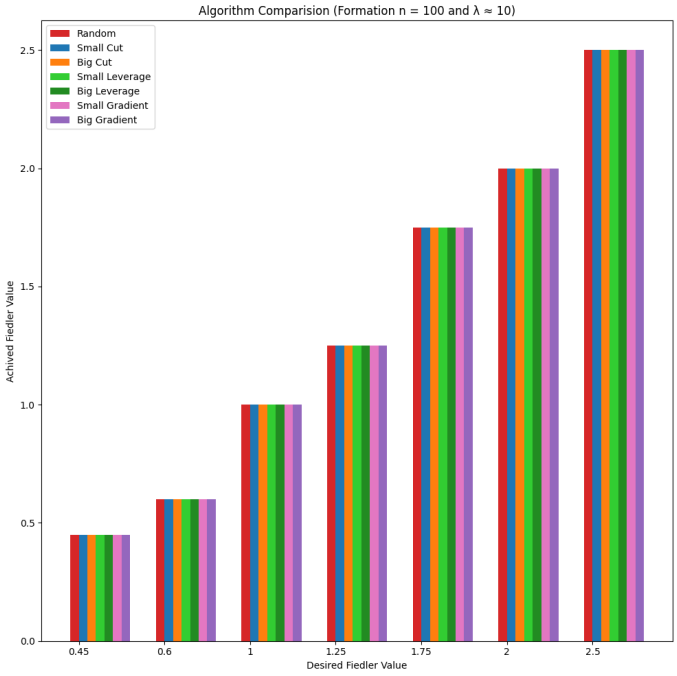


Fig. 18: Input Graph: $n = 100$ and $\lambda_2 \approx 10$

REFERENCES

[1] D. Rodrigues and E. Sun, "Dictating algebraic connectivity as a topology in networked multi-agent systems."

[2] Yoonsoo Kim and M. Mesbahi, "On maximizing the second smallest eigenvalue of a state-dependent graph laplacian," *IEEE Transactions on Automatic Control*, vol. 51, no. 1, pp. 116–120, 2006.

[3] Y.-T. Chow, W. Shi, T. Wu, and W. Yin, "Expander graph and communication-efficient decentralized optimization," 2016.

[4] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[5] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, pp. 298–305, 01 1973.

[6] G. Jiang, G.-D. Yu, and J. Cao, "The least algebraic connectivity of graphs," *Discrete Dynamics in Nature and Society*, vol. 2015, pp. 1–9, 10 2015.

[7] J.-B. Liu, M. Javaid, M. Raza, and N. Saleem, "On minimum algebraic connectivity of graphs whose complements are bicyclic," *Open Mathematics*, vol. 17, no. 1, pp. 1490 – 1502, 01 Jan. 2019. [Online]. Available: https://www.degruyter.com/view/journals/math/17/1/article-p1490.xml

[8] M. Javaid, M. Raza, M. U. Rehman, W. C. Teh, and J. Cao, "Minimum algebraic connectivity of graphs whose complements are bicyclic with two cycles," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 0, no. 0, pp. 1–20, 2020. [Online]. Available: https://doi.org/10.1080/09720529.2019.1668144

[9] D. Mosk-Aoyama, "Maximum algebraic connectivity augmentation is np-hard," *Oper. Res. Lett.*, vol. 36, no. 6, p. 677–679, Nov. 2008. [Online]. Available: https://doi.org/10.1016/j.orl.2008.09.001

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

[11] D. Spielman, *Spectral and Algebraic Graph Theory*, 2019. [Online]. Available: http://cs-www.cs.yale.edu/homes/spielman/sagt/

[12] L. Lovász, *Semidefinite Programs and Combinatorial Optimization*. New York, NY: Springer New York, 2003, pp. 137–194. [Online]. Available: https://doi.org/10.1007/0-387-22444-0_6