Leader Election for UAV Swarm with Limited Range Communication

Nicholas Scharan Cysne

December 6, 2022

Unmanned Aerial Vehicles (UAVs) can act in a wide range of scenarios in which sending human personal is of great risk or difficulty, such as exploration and reconnaissance missions in dangerous areas or of extreme conditions for human beings, even in disaster sites or military operations. These missions commonly employ multiple UAVs, forming a decentralized network that allows communication and cooperation between them. For the success of this type of formation, it is necessary to have a robust network topology, that is, capable of maintaining certain characteristics such as connectivity between all agents in the network and resilience to failures. Failures in such scenarios, especially if the failure happens on a Leader node, can lead to fragmentation of the network, jeopardizing the mission. This work proposes a consensus algorithm based on Raft's Leader Election to define the best leader in terms of robustness to failures in the network.

1 Introduction

In this chapter, motivation, and objectives of this work are discussed, as well as a brief literature review of related works.

1.1 Motivation

Multi-agent systems are defined as one of the main challenges in robotics for the next decade [Yang et al. 2018], and most of what is attributed to its eventual success is the existence of a resilient network topology for these agents. Yang *et al.* define resilience as a "property about systems that can

bend without breaking. Resilient systems are self-aware and self-regulating, and can recover from large-scale disruptions". In other words, a resilient network topology has all its members connected, and provides robustness to failures, meaning that even if a node fails, the network will not break into multiple components, and will continue to be able to perform its task.

In Aeronautics, Unmanned Aerial Vehicles (UAVs), a.k.a Drones, can form such network topologies to perform missions of exploration, surveillance and reconnaissance, which can take place in dangerous or extreme condition environments, after disaster events, in inhospitable settings or for military purposes.

Such networks can be understood as a distributed system in which each drone is a automated agent *per se*. In this context, one common approach to organizing the network is to implement a Lead-Follow Formation, in which a leader is elected in the network, and it helds the responsability to give general guidelines of the network's behavior.

Given that the context in which the network operates is of extreme conditions, it is possible that the network suffers from failures or attacks (internal and external disruptions). An issue arises in this situation, that is if the Leader of the network is damaged by these disruptions the network is lost and may face fragmentation without a clear guidance. The use of Leader Election algorithms in distributed scenarios is one of many ways to address this issue, but it needs to take into account that the just a random leader is not enough the sustain the network working properly, the network must elect a leader considering the characteristics of its own dynamics. For so, it needs to address the characteristics of connectivity and robustness to failures.

1.2 Objective

This work proposes a consensus algorithm based on Raft's Leader Election voting mechanism for a decentralized UAV Swarm in military operations that considers the drone's characteristics of connectivity and robustness to failures.

1.3 Related Works

In the context of network topologies for multi-agent missions, Ghedini *et al.* addresses the issue by proposing a combination of control laws derived from metrics of algebraic connectivity and betweenness centrality to identify

vulnerable nodes [Ghedini et al. 2018]. But, contrary to this work, Ghedini *et al.* used a decentralized network without a leader, with each agent being responsible for its actions and for estimating all connectivity and robustness metrics.

Also, in the context of Leader Election for a UAV swarm, Zuo *et al.* determines a framework of Leader Election for a static swarm of drones that act as a grid providing wifi in a determined region [Zuo et al. 2022]. Their work proposes a leader election algorithm based on the location of the drone relative to the network. This framework can be adapted to our purposes, using metrics relative to the dynamics of the network instead of relative position.

2 Methodology

In this chapter, we first cover the theoretical background needed in Network Theory, then how we model the problem, and finally the proposed algorithm.

2.1 Algebraic Connectivity

For all following concepts, consider an undirected graph $\mathcal{G} = (V, E)$, and let $\mathcal{L} \in \mathbb{R}^{N \times N}$ be the Laplacian Matrix of graph $\mathcal{G}$, $\mathcal{L}$ is defined as follows:

$$L_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Calculating the eigenvalues λ_i, $i = 1, ..., N$, of the Laplacian Matrix, we can extract the following properties [Godsil and Royle 2001]:

- For all eigenvalues λ_i, $\lambda_i \in \mathbb{R}$;
- The eigenvalues can be sorted in a way that $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$.

Let λ be equal to λ_2, the second smallest eigenvalue (counting multiple eigenvalues separately) from the Laplacian Matrix of $\mathcal{G}$, then $\lambda > 0$ if, and only if, $\mathcal{G}$ is a connected graph. The value of λ can serve as indicator of how well connected the overall graph is. In this context, λ is defined as the Algebraic Connectivity of the graph $\mathcal{G}$.

2.2 Betweenness Centrality

The Betweenness Centrality (BC) of a node v is a measure of influence this node has in the graph $\mathcal{G}$ regarding the flow of information through nodes [Godsil and Royle 2001]. This metric is usually used to locate bridge nodes, i.e essential nodes for the flow on information passes from one part of a graph to another. The betweenness centrality of node, $g(v)$, can be calculated as

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2}$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of those paths that pass through v.

Given that $g(v)$ scales with the number of nodes in the graph, we can define a normalized betweenness centrality, $g_{\text{normal}}(v) \in [0, 1]$, given by

$$g_{\text{normal}}(v) = \frac{g(v) - \min(g)}{\max(g) - \min(g)}, \tag{3}$$

where $\min(g)$ and $\max(g)$ are the minimal and maximum values of BC in the graph, respectively.

2.3 Robustness Level

In network topologies, attacks directed to nodes with high ranking of BC are likely to harm the network connectivity, in the worst case breaking the network into smaller components.

We may define *robustness* in network theory as the ability to withstand failures and perturbations. It is an important attribute that helps to evaluate the resilience of networks to attacks or malfunction [Godsil and Royle 2001].

Consider a graph $\mathcal{G}$ with N nodes, and let $[v_1, ..., v_N]$ be the list of nodes ordered by descending value of Betweenness Centrality. Let $\phi < N$ be the minimum index such as removing nodes $[v_1, ..., v_\phi]$ leads to disconnecting the graph. We define the Robustness Level of a graph as

$$\Theta(\mathcal{G}) = \frac{\phi}{N}, \tag{4}$$

being the number of nodes needed to be removed before the graph disconnects, over the total number of nodes in the graph.

2.4 Attack Policy

Our model focus on failures of prominent nodes (attacks) in a way that every episode ends with only 30% of the network remaining.

For the purposes of our work, we deal with 20 agents in the network. This way, for a network of 20 drones, only 6 will remain by the end of each episode. And since the episode is bounded by a 15 second window, and with 14 attacks to occur, we have 1 attack/sec, removing from the network always the node of highest Betweenness Centrality at the time, simulating an attack on that drone.

2.5 Simulator

For the visualization and evaluation of episodes, we have built a simulator using the PyGame library. Figure 1 shows an example of a evaluation episode, in which we have 20 drones in starting position, and 100 obstacles spread throughout the field. In the right we have the target location to which all drones should head.

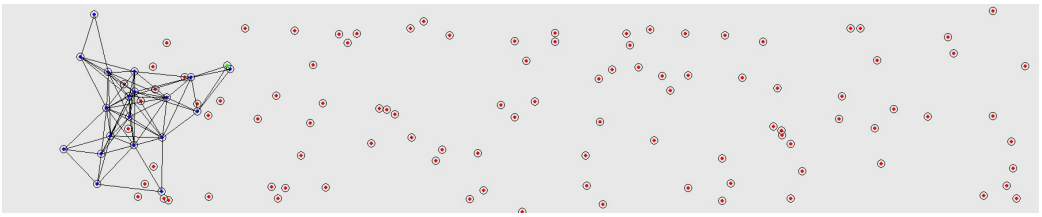


Figure 1: Simulator built with Pygame library for evaluation of episodes.

To create the simulator we utilized the following libraries in Python3:

- numpy;
- pygame;
- networkx;
- logging;
- multiprocessing.

To run the simulator, execute the main file "main.py". The library multiprocessing is responsible for creating 20 separate processes for the 20 drones and handling communication between them using pipes, this way we do not need to instantiate 20 terminals and 20 UDP ports.

The PyGame library instantiates the simulator that can be interacted with the space bar, pausing or resuming the episode.

2.6 Proposed Algorithm

We based our algorithm in Raft's Leader Election voting mechanism. In this scenario, as soon as a worker understands that the leader is down, it calls for a new election. The new leader is elected as the first worker to receive more than 50% of the votes.

In our scenario, a robust network topology is needed so that the swarm is able to act in its mission, and to consider only the timestamp of the vote to elect a new leader is at most sub-optimal. For this specific context, we need to consider not only the velocity of electing a new leader, but also other metrics important for the mission. For example, we need to elect a leader in a way that minimizes the number of direct attacks on the leader the network will suffer. To do this we propose to select a leader that is less likely to suffer from attacks or failures during the mission. One way to do this is to always select the leader with the minimum value of betweenness centrality in the network, this way we always have the last node to be attacked following the attack policy. We propose to use Raft's 50% rule and the term value being the node's betweenness centrality.

Also, a possible scenario to occur is that the network fragments even with a good leader, all due to a high ranking BC node, so we implement a gossip protocol to allow fast communication of all nodes in the network. All nodes will be communicating their respectful positions in the field, including the leader, this way they can use this information to regroup in case of fragmentation.

We chose to set the regroup position as the leader's last known location to all drones. The pseudo-code of the main loop of this model is presented in Algorithm 1.

Algorithm 1 Proposed Leader Election's Pseudocode

```
# Election Loop
if leader elected then
  Break loop and become follower;
else
  if Network Disconnected then
    Regroup to leader's last seen position;
    Continue;
  else
    Calculate own Qualification (Betweenness Centrality);
    if Qualification ≥ Requests then
      Broadcast vote request;
      if #responses ≥ 50% then
        Broadcast elected claim;
        Accept Leader Role;
      else
        Continue election;
      end if
    else
      Cast vote and broadcast response;
    end if
  end if
end if
```

2.7 Evaluation Metrics

To evaluate the model, besides the visual aid of the simulator, we use 2 main metrics: Algebraic Connectivity λ, and Robustness Level $\Theta(\mathcal{G})$.

To summarize the evaluation of the model over all 200 scenarios, we calculate the Mean Algebraic Connectivity $\overline{\lambda}$, and Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, throughout an episode. These metrics show how each behaves throughout an episode on average for all 200 possible scenarios.

3 Results and Discussions

This chapter presents the simulation results of the proposed model and a few discussions on top of them.

3.1 Raft's Implementation

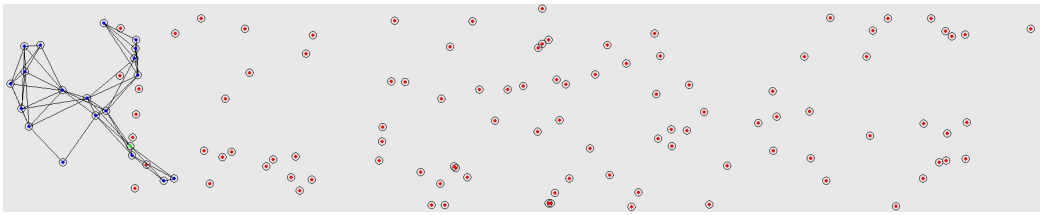
The first experiments were made with the original implementation of Raft's Leader Election algorithm. With it we simulated the network for 200 predetermined scenarios and measured how well the network performed. Figure 2 presents 4 snapshots of the network in one evaluation episode. The green node represent the current leader of the swarm.

By analyzing the network's topology, we can notice that the Raft's implementation, as expected, doesn't select nodes that provide a robust topology, instead the position of the leader in the network seems to be working in the other direction. Since the algorithm doesn't have any input for this characteristic, and it counts on 50% of the votes to elect a new leader, it is plausible to assume that with Raft's algorithm the leader will always have a high BC value, due to be an important node given the flow of information in the network. With higher BC value, a node has a quicker access to the nodes in the network, so it can be elected faster than a node in the border with low BC value.

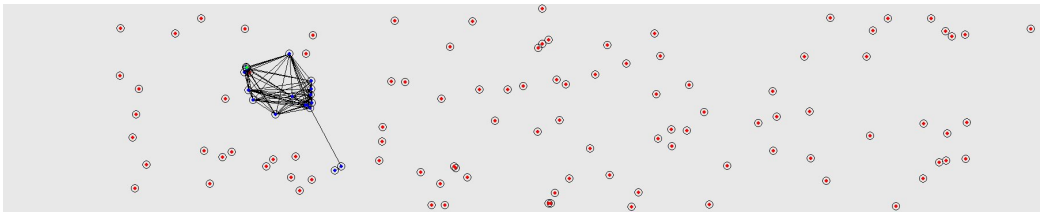
In the snapshots, as well as in other scenarios, we can observe how the leader fails and is elected a new leader in sequence in several times. Although the algorithm constantly provides the network a functional leader, and with this it is able to perform its mission without complications, the exposure of the leader to risk of failures diminishes its performance substantially. Every time an attack occur, and it is in a leader node, the network needs to stop its execution, regroup, and then call a new election, this costs significant time

in a field not free of hazards, putting the whole network in danger. Ideally the leader should always be active and guiding the network in its mission, providing guidelines for its behavior.

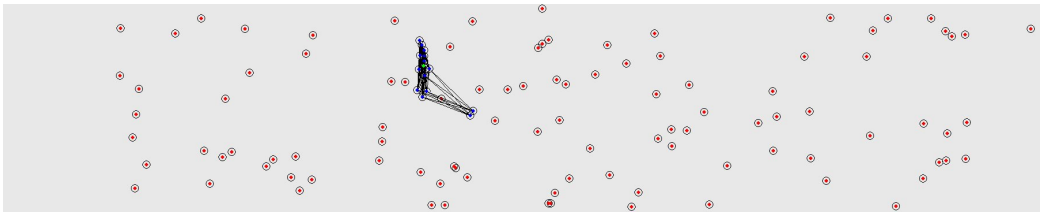
In general, for the objective of the network, this algorithm performs poorly and doesn't allow the mission to be executed in its completion.



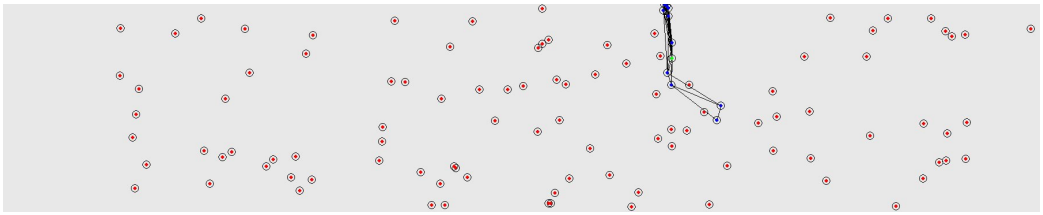
((a)) Raft's Leader Election at $t_1 = 1$.



((b)) Raft's Leader Election at $t_2 = 200$.



((c)) Raft's Leader Election at $t_3 = 400$.



((d)) Raft's Leader Election at $t_4 = 600$.

Figure 2: Behavior of the network with Raft's Leader Election.

3.2 Betweenness Centrality Model's Implementation

To measure how vulnerable a drone is to failures and attacks, we use its betweenness centrality (BC), so it is only natural that to avoid an attack on the leader, we should select that of minimum BC value among its peers.

To include this metric in our algorithm we alter the term value of Raft's to be the node's betweenness centrality instead. Also, with this metric we should always look for the minimum value, so to cast a vote the drone should analyze its BC and the BC from the request it received, and vote in the smaller value.

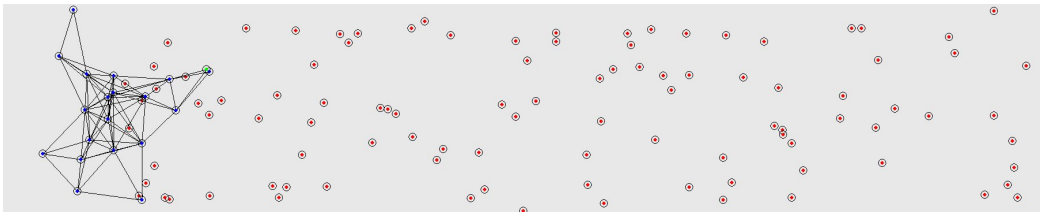
When simulating the network with the altered algorithm we can notice a difference right in the start, the first elected leader is a node in the border of the network instead of the middle, i.e. a node of low BC value, Figure 3(a) presents 4 snapshots of a test scenario in which we can see the leader being elected in the top right.

Also, by analyzing the episode, it is clear that the leader is not attacked during its duration, maintaining its role and guiding the network until the end, all because we avoid choosing a leader likely to be attacked.

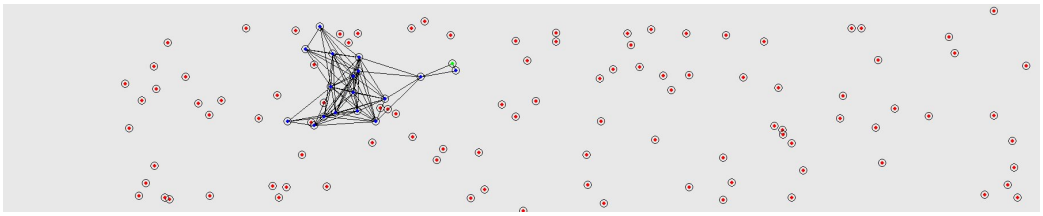
This does not mean that the network does not disconnect now and then, Figure 3(b) presents a configuration with a node with high BC, being attacked in the next frame disconnecting the leader from the rest of the network. In this scenario our regrouping behavior takes action, making the drones head towards the leader's last known location and the leader to await its peers. After regrouping, since the leader is still active, there is no need of a new election and the network can continue its mission.

When voting, an important thing to notice, is that since we are not taking into account the timestamp of the vote, as long there is a smaller value of BC in the network the voting will continue to take place, guaranteeing the global optimum in every scenario, even if the election takes a little longer.

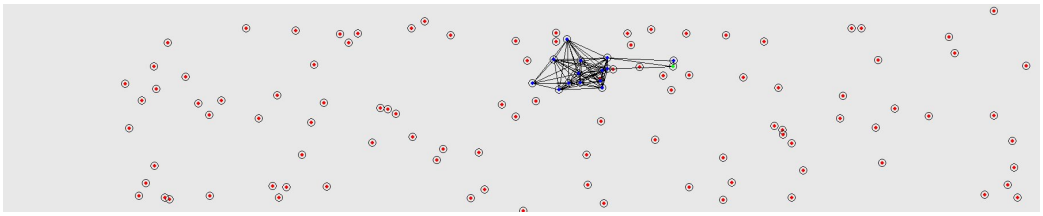
To measure in terms of Algebraic Connectivity λ and Robustness Level $\Theta(\mathcal{G})$, we plot the values of Mean Algebraic Connectivity $\overline{\lambda}$, Figure 4, and Mean Robustness Level $\overline{\Theta}(\mathcal{G})$, Figure 5, calculated throughout an episode over 200 scenarios.



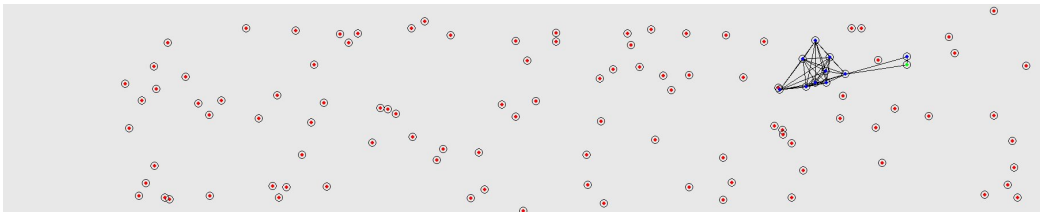
((a)) Proposed Model's Leader Election at $t_1 = 1$.



((b)) Proposed Model's Leader Election at $t_2 = 200$.



((c)) Proposed Model's Leader Election at $t_3 = 400$.



((d)) Proposed Model's Leader Election at $t_4 = 600$.

Figure 3: Behavior of the network with Proposed Model's Leader Election.

In both plots it is possible the verify the result of the attacks in the network, in which both $\overline{\lambda}$ and $\overline{\Theta}(\mathcal{G})$ present a periodical behavior. Note that these values return to the previous level before the attacks, showing that it provides the network a stable behavior that does not fragment easily.

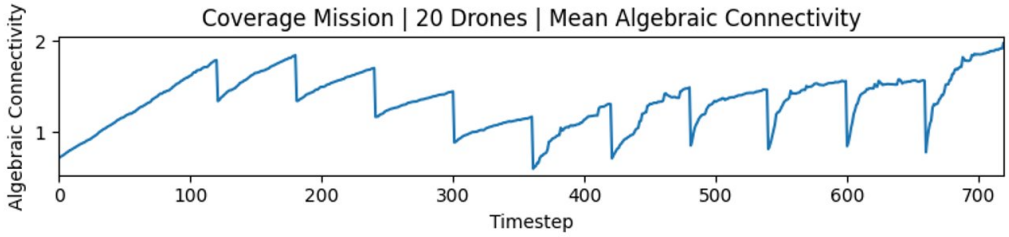


Figure 4: Mean Algebraic Connectivity $\overline{\lambda}$ throughout an episode over 200 scenarios.

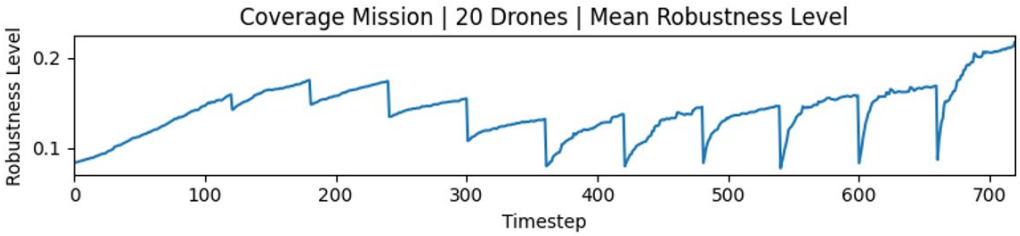


Figure 5: Mean Robustness Level $\overline{\Theta}(\mathcal{G})$ throughout an episode over 200 scenarios.

4 Conclusion

This chapter contains the final thoughts on this work, presenting a final conclusion and analysis on what we achieved, problems that we found and possible ideas of improvements.

4.1 Final Thoughts

By analyzing the resulting behavior of the network, it is possible to conclude that by taking into account the dynamics of the network, as betweenness centrality and algebraic connectivity, the algorithm showed better results than the original implementation of Raft.

In terms of speed to elect a new leader, the algorithm is slower than the original implementation given that a worker can change its vote mid election, but in practical terms the difference is imperceptible. In terms of stability and robustness to failures during the execution of the mission, the algorithm

presented a much more stable configuration, while almost zero attacks in the leader during evaluation. With fewer elections happening we can compensate for the slower algorithm, since it will be executed far less frequently.

A limitation of this model is the extended time required to finish the vote, and also once a leader is elected, it only grants its place after it is attacked, this may cause the network to enter vulnerable configurations.

4.2 Future Works

The following presents a few ideas of possible enhancements and research paths based on this work.

- Increase the complexity of the model by introducing errors in communication or by actively calling new votes even without failures;
- Adapt and test the model to a real scenario in multicriteria missions, such as coverage missions;
- Consider other parameters in the election not related to robustness, but that also contributes to the mission: battery level, connectivity, position.

References

[Ghedini et al. 2018] Ghedini, C., H.C.Ribeiro, C., and Sabattini, L. (2018). Toward efficient adaptive ad-hoc multi-robot network topologies. *Ad Hoc Networks*, 74:57–70.

[Godsil and Royle 2001] Godsil, C. and Royle, G. (2001). *Algebraic Graph Theory*. MIT Springer.

[Yang et al. 2018] Yang, G.-Z., BELLINGHAM, J., and DUPONT., P. E. (2018). The grand challenges of science robotics. *Science Robotics*, 3(14).

[Zuo et al. 2022] Zuo, Y., Yao, W., Chang, Q., Zhu, X., Gui, J., and Qin, J. (2022). Voting-based scheme for leader election in lead-follow uav swarm with constrained communication. *Edge Computing for Urban Internet of Things*, pages 11–14.