3-2021

Behavior of Lightning in Developing Storms

Erick A. Tello

Behavior of Lightning in Developing Storms

THESIS

Erick A. Tello, Capt, USAF

AFIT-ENS-MS-21-M-187

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-21-M-187

BEHAVIOR OF LIGHTNING IN DEVELOPING STORMS

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Operations Research

Erick A. Tello, BS

Capt, USAF

March 2021

AFIT-ENS-MS-21-M-187

BEHAVIOR OF LIGHTNING IN DEVELOPING STORMS

THESIS

Erick A. Tello, BS
Capt, USAF

Committee Membership:

Lt Col Timothy W. Holzmann, PhD
Chair

Dr. Edward D. White
Member

Lt Col Andrew J. Geyer
Member

AFIT-ENS-MS-21-M-187

Abstract

Air Force weather squadrons are required by Air Force Instruction 91-203 (AFI 91-203) to issue a warning when lightning activity is observed within 5 nautical miles (NM) of areas under their jurisdiction. Upon receiving this warning, personnel outdoors are expected to pause activities, move indoors, and wait until the warning is cleared. Studies sponsored by the 45th Weather Squadron (45 WS) have concluded that the 5 NM warning radius can be reduced for well-developed storms while maintaining an appropriate level of safety. This thesis investigates whether radii for storms that are in early development can also be reduced. The research presented here develops algorithms to partition lightning sensor data into storms. Next, storms are filtered to their earliest lightning events, and the study calculates distances of early lightning observations both from the origin point of the storm and between successive observations, with the latter forming the basis of our conclusions. Analysis indicates that 4.02% of lightning events during the first 30 seconds of a storm occur more than 4 NM away from the previous event, and 2.11% occur more than 5 NM away. Because these percentages are smaller than the equivalent values for well-developed storms from Sanderson (2019), we conclude that her recommended lightning warning radius of 4 NM is valid for developing storms as well.

Table of Contents

Page

List of Figures

List of Tables

BEHAVIOR OF LIGHTNING IN DEVELOPING STORMS

I. Introduction

1.1 Motivation and Background

Lightning contributes significant challenges to the process of preparing and executing space launches. Safety procedures require that outdoors work cease when a lightning warning is issued, and this work stoppage can cost valuable space launch preparation time in areas prone to storms. For example, Central Florida, home of the Cape Canaveral Space Force Station (CCSFS) and NASA Kennedy Space Center (KSC), has historically received an average of over 2,500 watches and warnings per year, each halting work for a minimum of 20 minutes (Roeder et al., 2017). The goal of this thesis is to examine potential adjustments to the criteria for lightning warnings, with the goal of "buying back" some of this working time while still adhering to safety standards.

The 45th Weather Squadron (45 WS) provides weather forecasting and lightning warnings to several key space launch facilities in the Central Florida region, including the KSC. The KSC must be concerned both with personnel working outdoors in preparation for a launch, and also with the effects of lightning on spacecraft during a planned launch (Tamurian et al., 2012). The 45 WS defines their area of responsibility with lightning warning radii centered on twelve protected facilities. Areas protecting a single facility or closely spaced facilities have a 5 nautical mile (NM) radius, while areas protecting a large facility or a collection of more widely spread facilities have a 6 NM radius (Roeder et al., 2017; Hinkley et al., 2019). If lightning occurs inside any

of these areas, then the 45 WS will issue a warning, and launch preparation must stop as workers seek shelter from potential further strikes. The radii for these areas are derived from Air Force Instruction 91-203, which, in turn, derives the standard 5 NM warning radius from studies performed after the death of an Airman due to lightning in 1996 (Department of the Air Force, 2012; Renner, 1998; Cox, 1999; McNamara, 2002). See Ceschini (2013) for a recent study on potential changes to KSC warning areas.

Sanderson (2019) and Hinkley et al. (2019) investigate lightning behavior in the Central Florida region, and each concludes that the 5 NM and 6 NM warning radii may both be safely decreased to 4 NM. On average, this change saves approximately 180 man-hours and 181 false alarms over five months, measured from May through September each year (Sanderson, 2019), the lightning season in Central Florida. Both studies rely mainly on measuring the distance distribution of lightning beyond a pre-existing area.

In the earliest moments of a storm's life, however, when only a few lightning events have been observed, any "pre-existing area" one might calculate will be subject to rapid, substantial change as the storm moves and expands. During this period, we say we have a developing, or "early", storm. For developing storms, we are uncertain of the degree to which research from Sanderson (2019) or Hinkley et al. (2019) using pre-existing storm areas is applicable. Sanderson (2019) does additionally investigate distance distributions for developing storms; her results are promising, indicating that a 4 NM lightning warning radius remains applicable with regard to developing storms. Since she explores developing storms only as a secondary research topic, however, we seek to study this topic in greater depth. Additionally, Sanderson (2019) and Hinkley et al. (2019) perform their research using data from the Lightning Detection and Ranging System (LDAR); since the completion of their research, data from

the newer Mesoscale Eastern Range Lightning Information System (MERLIN) has become available for study (Roeder and Saul, 2017). MERLIN's improved reliability and precision over LDAR present an opportunity to increase our confidence in the results of our research.

Our primary research question is thus: *how are lightning flashes in a developing storm distributed?* Conclusions from Sanderson (2019) and Hinkley et al. (2019) provide support for a revised weather warning policy that reduces work interruptions while still adhering to lightning safety regulations; we are interested in whether we can further reduce the lightning warning radii for developing storms to achieve even greater gains.

1.2 Approach Summary

We adopt a two-phased approach to our research question. In the first phase, we perform clustering analysis on the raw data to partition the records into storms. Once the storms are identified, we are able to move into the second phase, where we extract and analyze records that represent lightning in developing storms. In particular, we seek to estimate the distribution of distances between successive lightning events in a developing storm, allowing us to study the safety impact of reducing the lightning warning radius from the current 5 NM or 6 NM. If, for example, the distance between successive events is beyond 4 NM with a low enough probability – e.g. a probability at least as low as for pre-existing areas from Sanderson (2019) – this will indicate that 4 NM represents a safe lightning warning radius.

We begin Phase 1 with lightning data provided by the 45 WS from MERLIN. This data set contains the date, time, and location of 55.7 million lightning events, including lightning aloft and cloud-to-ground lightning. This is raw data, however, and does not identify which events belong to what storms. We therefore spend Phase 1 devel-

oping and implementing algorithms to identify storms in the data. Our algorithms include a pairwise comparison algorithm, a grid-based algorithm, and a dynamic clustering algorithm developed by Hyde and Angelov (2015), named Clustering of Online Data Streams (CODAS). The pairwise comparison algorithm is projected from our experiments to require about two weeks to process the data set. This is too slow for our purposes, as the algorithm additionally requires testing parameters over many runs. The grid algorithm is a heuristic; it is capable of processing the MERLIN data much faster, but it sacrifices accuracy in exchange and is prohibitively difficult to test in the time available to us. CODAS falls between the others in terms of run time, is much easier to optimize than either, and avoids the accuracy issues inherent in the grid algorithm. We implement a version of CODAS and use it to create storm identifiers for each lightning event in our data set over four days of computational run time.

We begin Phase 2 of our study by filtering the observational data from each storm to examine only the earliest events of each storm. We can filter by capturing the first n records from each storm or by capturing the first t seconds of lightning data. Next, we calculate each remaining lightning event's geographical distance from its storm "origin," (i.e. the first lightning event in the same storm), as well as the distance between successive events. Using descriptive statistical techniques, we can estimate the probability that early lightning exceeds a threshold of 4 NM from the storm origin or the previous lightning event. We can then compare this with the equivalent probability for 5 NM and 6 NM; if the increase in probability is minimal, then this result supports the conclusion that existing lightning warning radii may be safely reduced. We can also compare the probability of exceeding selected distances with the probability of exceeding the same distances in Sanderson's distribution for well-developed storms. If our probabilities are smaller than Sanderson's equivalents,

then this supports the conclusion that her recommended lightning warning radius of 4 NM is applicable to early lightning.

1.3 Thesis Overview

The remainder of this thesis is organized as follows. Chapter II describes related research to provide context regarding lightning detection equipment and the current understanding of lightning physics and distributions in the scientific community. There is substantial support for a 4 NM lightning warning radius with regard to well-developed storms, but research on developing storms is lacking. Chapter III explains algorithms developed in the pursuit of identifying storms and assigning storm IDs to lightning observations; we select CODAS for its speed and robustness. Chapter IV presents our efforts to filter storm data and analyze the distances of early lightning events; we find that only 5.57% of lightning events occur beyond 4 NM of the previous lightning event in developing storms. Chapter V presents our conclusions, recommendations, and proposals for future research; since the probability of lightning exceeding 4 NM or 5 NM is lower than for distributions developed by Sanderson (2019), we conclude that 4 NM is an acceptable lightning warning radius for developing storms.

II. Literature Review

This chapter introduces lightning detection systems and existing research on lightning distributions, with an aim toward establishing a knowledge base that will provide context and aid in our research.

2.1 Lightning Detection Systems

Systems that have provided lightning data for use in past studies include Weather Surveillance Radar (WSR-88D; Renner, 1998; Cox, 1999); the National Lightning Detection Network (NLDN; Lopez and Holle, 1999; Parsons, 2000; McNamara, 2002; Holle et al., 2016; Roeder et al., 2017), and the Four-Dimensional Lightning Surveillance System (4DLSS; Boyd et al., 2005; Murphy et al., 2008; Tamurian et al., 2012; Hinkley et al., 2019; Sanderson, 2019), consisting of the Cloud-to-Ground Lightning Surveillance System (CGLSS) and the Lightning Detection and Ranging System (LDAR). More recently, the Mesoscale Eastern Range Lightning Information System (MERLIN) has come online and begun producing the lightning data used in our own research.

To provide context for relevant studies, we briefly describe these weather detection systems. WSR-88D is a Doppler radar system; it works by detecting precipitation rather than electrical activity, and gathers data on storm reflectivity, mean radial velocity, and spectrum width (Renner, 1998). These data can be used to identify storms and, by proxy, cloud-to-ground (CG) lightning strikes, enabling calculation of the distances of CG lightning strikes from storm centroids (Renner, 1998).

However, lightning can be observed more precisely and reliably through the direct detection of electrical activity. Streamer theory explains how lightning is formed from paths of least electrical inductance (Horvath, 2006). Essentially, "stepped leaders",

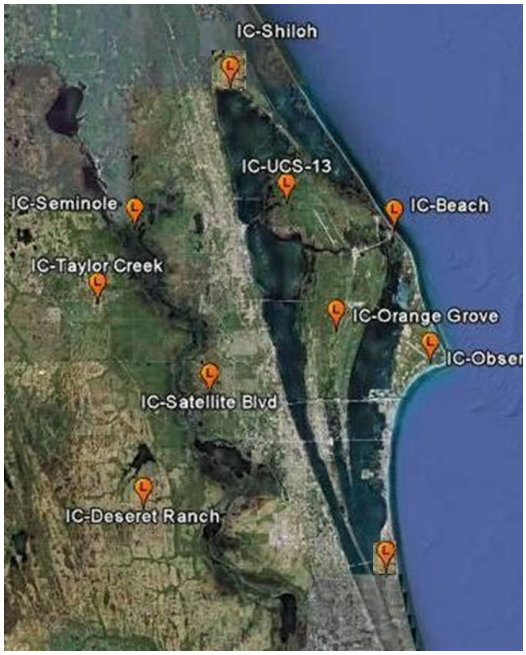
or thin channels of electric current that ionize gases, branch outward and sometimes downward from a cloud. When a stepped leader makes contact with a suitable conductor, it forms a path along which electrical current discharges in the opposite direction, producing the lightning we are accustomed to seeing. This discharge is referred to as a "return stroke" for CG lightning (Horvath, 2006), or a "recoil leader" for lightning aloft (Roeder and Saul, 2017). Modern systems can measure lightning directly by detecting stepped leaders, return strokes, and recoil leaders.

NLDN is owned by Vaisala, and along with the remaining systems described here, it detects electrical activity. Its IMPoved Accuracy from Combined Technology (IMPACT) sensors capture primarily CG lightning return strokes (Parsons, 2000; Zoghzoghy et al., 2014), recording the time, polarity, signal strength, and number of strokes of each observed CG lightning flash (Tamurian et al., 2012). In 2013, NLDN was upgraded to be able to additionally detect the time and location of lightning flashes aloft with about 50% detection efficiency (i.e. true positive rate; Holle et al.; Roeder and Saul, 2016; 2017).

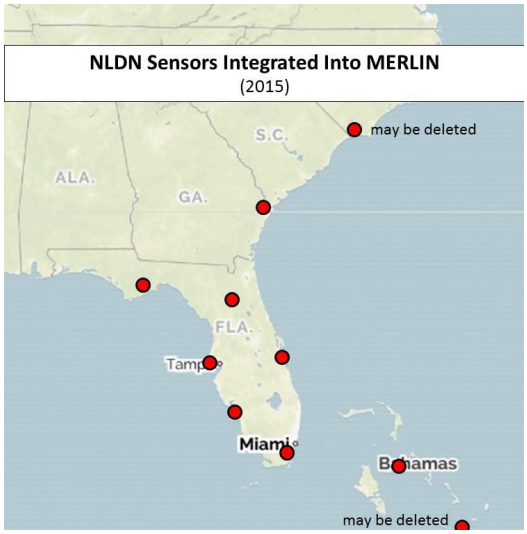
4DLSS uses two sensor suites to detect total lightning. Of these, CGLSS uses the same IMPACT sensors as NLDN, but its sensors are closer together than NLDN, resulting in greater sensitivity and location accuracy (Boyd et al., 2005; Murphy et al., 2008; Tamurian et al., 2012). These are also primarily used for CG lightning detection. LDAR is yet more precise, but uses stepped leaders to detect lightning aloft, and does not directly detect CG lightning (Murphy et al., 2008; Tamurian et al., 2012; Sanderson, 2019).

MERLIN, similar to 4DLSS, is a total lightning detection system serving the Kennedy Space Center and surrounding area (Roeder and Saul, 2017). Its primary sensor, the Total Lightning Sensor Model-200 (TLS-200), serves the combined purpose of both of 4DLSS's sensor suites, detecting both lightning aloft and CG lightning

(Roeder and Saul, 2017). The TLS-200 yields improved location accuracy, recording the location of lightning activity to within 58 meters compared with 350 meters for 4DLSS. TLS-200 sensors additionally detect recoil leaders instead of stepped leaders for lightning aloft, greatly reducing the number of points that may be considered redundant when studying lightning flashes. The MERLIN system contains ten TLS-200 sensors, located as in Figure 2.1a. Ten NLDN sensors (shown in Figure 2.1b) are additionally integrated into the MERLIN system to increase its effective range for CG lightning.



(a) Location of MERLIN's 10 TLS-200 sensors in and around the Kennedy Space Center.



(b) Location of the in-range NLDN sensors integrated into MERLIN for cloud-to-ground return stroke solutions.

Figure 2.1. Locations of sensors comprising the MERLIN network (Roeder and Saul, 2017).

Another key benefit of MERLIN is that it produces many fewer redundant observations of lightning aloft than its predecessor, 4DLSS. The 4DLSS system detects lightning aloft using stepped leaders, which are plentiful for a given flash and have a low signal-to-noise ratio. For these reasons, the 4DLSS system must collect many individual stepped leader signals and aggregate them to detect each actual lightning

flash. Thus, past studies typically aggregate individual data records into flashes (Cox, 1999; Parsons, 2000; Hinkley et al., 2019; Sanderson, 2019). Because MERLIN's TLS-200 sensors instead detect recoil leaders, which are much sparser for a given flash and have a much stronger signal-to-noise ratio, we do not perform this aggregation, but instead treat each record as a lightning flash.

2.2 Lightning Distance From Storm Centroids

In 1996, lightning struck eight minutes after the expiration of a lightning advisory, killing an Airman and injuring 10 others. This event prompted multiple research efforts regarding the safety of lightning warning and advisory practices (Renner, 1998; Cox, 1999; McNamara, 2002).

McNamara and Renner both study the horizontal distance traveled by CG lightning on its way to the ground. Renner (1998), using WSR-88D data, concludes that this distance varies greatly with region and season; McNamara (2002), using LDAR and NLDN data, concludes that 28.6% of CG lightning flashes travel more than 5 nautical miles (NM) from their point of origin.

Renner (1998) studies the distance of CG lightning strikes from storm centroids, and states that 75% of CG flashes occur less than 10 NM away; this is further supported by Bazelyan and Raizer (2000). Cox (1999) expands on Renner's research, studying lightning behavior using data from the same WSR-88D systems as Renner. He calculates storm centroids based on WSR-88D reflectivity measurements and computes lightning centroids by averaging the positions of lightning flashes clustered using an algorithm called Distance Between Successive Flashes (DBSF). He then finds the distances of lightning flashes from associated storm and lightning centroids. Cox (1999) comes to the conclusion that between 32% and 39% of flashes occur beyond 5 NM of associated storm centroids, and between 18% and 34% occur beyond 5 NM

of associated lightning centroids.

Parsons (2000) builds upon Cox's research, performing clustering and analysis similar to Cox's DBSF method, but on a much larger data set procured from NLDN. Parsons builds clusters of flashes that occur within 12 minutes of the first flash in the set, and within 17 kilometers (km) of each other successively. Parsons (2000) finds that 21% to 32% of grouped flashes recorded by NLDN occur more than 5 NM from the centroids of their respective clusters, and 76% to 80% of isolated flashes strike more than 5 NM from the nearest other flash.

Studies up to this point all seem to indicate that 5 NM is an inadequate lightning warning radius; however, these studies all measure the distance of lightning from the center of a storm. In making decisions regarding lightning warnings, weather squadrons measure from the edges of known storm areas, not from storm centers. The 45th Weather Squadron (45 WS) therefore began to investigate the distance of new lightning from a pre-existing storm area's edge, as opposed to its centroid. This research was accomplished via two parallel efforts (Hinkley et al., 2019; Sanderson, 2019).

2.3 Lightning Distance From Pre-Existing Storm Areas

Hinkley et al. (2019) use a convex hull algorithm on lightning flashes aggregated from LDAR data to generate polygons that define pre-existing lightning areas. When a new lightning flash occurs within a distance threshold of one of these polygons, its distance from the polygon is recorded, and then the polygon is expanded to include the new lightning. As new flashes are added, those more than 15 minutes old are phased out. From the data recorded over the duration of this process, Hinkley et al. (2019) then calculate the best fit curves for polygon expansion frequency versus distances, arriving at an exponential decay function. From this function, they calculate an

optimal lightning warning radius. They recommend a warning radius between 3.857 and 3.988 NM, depending on how lightning activity is aggregated into flashes.

Sanderson (2019) performs a similar analysis on LDAR data, basing the pre-existing lightning area on a minimum area ellipse surrounding lightning flashes. She concludes that 4 NM constitutes an optimal lightning warning radius.

Holland (2021) additionally performs an analysis in parallel with our thesis. Her research is conceptually similar to Sanderson (2019), but operates on MERLIN data. Her work further supports a 4 NM warning radius for well-developed storms.

Between Holland (2021), Sanderson (2019), and Hinkley et al. (2019), there is strong support for this new warning radius when applied to well-developed storms. However, this leaves the question of whether the same can be said for early storms beginning their life near a protected area, when there is no pre-existing storm area to measure from. Sanderson (2019) briefly investigates early storms, and using her elliptical storm areas, she finds that the mean distance from the ellipse center is 3.13 NM for the first 5 flashes of a storm, and 3.53 NM for the first 10 flashes. Her preliminary results provide support for using a 4 NM lightning warning radius when the observed lightning is part of a newly developing storm, and not just well-developed storms moving toward a protected area.

2.4 Lightning at Airports

Outside of these studies sponsored by the 45 WS, most studies for optimizing lightning warning radii focus on airport safety. An early study examines the effect of adding lightning aloft to CG data sets (Murphy and Cummins, 2000). Their policy is to issue a warning if any lightning is detected within a large radius R_{Big}, and have everyone stay inside until 15 mins after the last lightning to occur within R_{Big}. They want a high probability that a warning will be issued at least two minutes prior to

the first CG strike within a smaller radius r_{small} = 4.8 km, and no strikes inside r_{small} after releasing the warning. They consider the probability of detection (POD), failure to warn (FTW = 1-POD2), and false alarm rate (FAR). They also consider the percent of time spent under valid warnings versus under false alarms. A larger warning radius yields a higher POD and higher FAR.

Holle et al. (2014) investigate upgrades to NLDN, including upgraded CC lightning detection; they find that the upgrades yield a flash detection efficiency of about 50%, compared with the previous efficiency of between 15% and 25%. In light of this, Holle et al. (2016) seek to determine the value of adding lightning aloft to CG lightning data to improve lightning warnings, and find that doing so improves the 2-minute POD by 13% compared with CG lightning only. Holle et al. (2016) also consider decreasing the inner warning radius (where CG lightning detection is poor) from 4.8 km to 0.5 km; this increases POD to 0.90, but also generates a substantial increase in the FAR.

2.5 Summary and Contributions

Holle et al. (2014) and Holle et al. (2016) support the value of including lightning aloft in lightning distribution research and warning procedures. They further support that a large disparity between the size of lightning warning radii and the size of the encompassed facilities (as modeled by smaller "inner" warning radii) improves the probability of detecting lighting before facilities are struck, but also increases the rate of false alarms. Renner (1998), Cox (1999), and Parsons (2000) each conclude that a significant portion of lightning flashes fall more than 5 NM from their respective storm centroids. Sanderson (2019) and Hinkley et al. (2019) each find much shorter lightning distances by calculating from the edge of a pre-existing storm area, in better keeping with the way weather squadrons make lightning warning decisions. Sanderson

and Hinkley et al. provide strong support for the safety of a 4 NM lightning warning radius when applied to well-developed storms, and Sanderson's work additionally supports this for storms in early development.

Developing storms remain an area of interest, however; Sanderson (2019) is the only known work to differentiate early storms from well-developed storms, and does so as a secondary topic of interest. Additionally, MERLIN's advanced sensors present fresh opportunities for improving our understanding of lightning distance distributions, both for early lightning and in general. Our research uses MERLIN lightning data to explore the distance distributions of lightning events in developing storms, with the aim of investigating the optimal lightning warning radius for developing storms and whether this radius differs from that of well-developed storms. In the following chapters, we will begin to explore the nature of early storms in greater depth.

III. Storm Identification

3.1 Introduction

The 45th Weather Squadron (45 WS) provides the data used for this research, sourced from the Mesoscale Eastern Range Lightning Information System (MERLIN) in Central Florida. MERLIN sensors record the time and location of each observation of lightning activity. The data set contains 55.7 million lightning events observed between May 2019 and September 2019; Table 3.1 provides a sample of the data. Each record includes the time (formatted as seconds since 1 January 1970, or "since epoch"), latitude, and longitude of an observed lightning event. The MERLIN data set includes separate files for lightning aloft and cloud-to-ground lightning. We merge these into a single file, so that our study examines total lightning data. Notably, the provided data does not include storm identification labels. Associating lightning events with storms is essential to the study of lightning in developing storms, so the first part of our study focuses on assigning these storm labels.

Table 3.1. Example of input data formatting. The DateTime column has been formatted to represent seconds elapsed since 1 January 1970.

DateTime	Latitude	Longitude
1556808217.833	26.9517	-81.4851
1556808217.857	26.9674	-81.4758
1556808217.988	26.9802	-81.4698
1556808341.512	27.0024	-81.4612
1556810579.614	27.295	-81.1728

For our study, a storm is defined as a group of lightning events that are densely packed in space and time, relative to the surrounding area. Using this definition, identifying the storms from the lightning data is non-trivial. When multiple storms occur at the same time, data points observed from each are intermixed in the data. Further, storms that begin their lives in separate geographical regions may later col-

lide with each other, effectively becoming one storm. In short, storms do not occur uniformly, one after another; thus, data representing the beginnings of storms cannot be identified via simple aggregation, as when grouping the data by date. In addition, the size of the data set precludes identifying storms manually. An algorithmic approach is required.

In this chapter we consider three different approaches for grouping the lightning events into storms: pairwise comparison, a grid-based algorithm, and an online clustering algorithm. Pairwise comparison is straightforward in concept, but we show that the quality of the output is overly sensitive to user-input parameters, and the algorithm is too computationally slow to be practical. A grid-based algorithm has the potential to run much faster, but at the cost of accuracy. Its run time and output quality appear to be inversely related, and achieving the best balance requires optimizing four parameters simultaneously. Time constraints would not allow us to fully implement this grid algorithm and perform the necessary parameter optimization. Clustering of Online Data Streams (CODAS), an algorithm published by Hyde and Angelov (2015), proves effective for our purposes; it runs more quickly than pairwise comparison, and its low sensitivity to input parameters allows us to achieve useful output on the first attempt. An exploration of each algorithm follows.

3.2 Pairwise Comparison

The pairwise comparison approach compares each observation with each other observation occurring within a specified number of records, searching for the most recent record occurring within a specified time and distance. In particular, Algorithm 3.1 accepts as input a set of lightning observation records formatted as in Table 3.1 and appends a predecessor ID and storm ID to each record. Storm IDs are assigned using pairwise comparison, accomplished in two parts. Part 1 (lines 5-18) identifies,

for each row of data, a predecessor lightning event, if one exists. In Part 2 (lines 20-27), events with no predecessors are each assigned new storm IDs, and the remaining events are assigned the storm ID of their predecessor.

Algorithm 3.1 Pairwise Comparison

1: **Input:** List A of lightning observation records (time since epoch t_i, longitude x_i, latitude y_i), sorted by t_i
2: **Parameters:** record count threshold C, distance threshold D, time threshold T
3: **Output:** vector s of storm IDs corresponding to records in A
4:
5: **for** i from 1 to $|A|$ **do**
6: Retrieve set $S = \{\max(i - C, 0),\ \ldots\ ,\ i - 2,\ i - 1\}$
7: **for** each $j \in S$ **do**
8: $d_j = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
9: **if** $d_j > D$ **then**
10: Remove record j from S
11: **if** $S \neq \varnothing$ **then**
12: Find record $k = \arg\max_{j \in S}(t_j)$ ▷ i's "predecessor"
13: **if** $t_i - t_k < T$ **then**
14: $p_i = k$ ▷ assign predecessor ID
15: **else**
16: $p_i = -1$ ▷ indicate no predecessor found
17: **else**
18: $p_i = -1$ ▷ indicate no predecessor found
19:
20: $B = \varnothing$
21: $s_1 = 1$
22: **for** i from 2 to $|A|$ **do**
23: **if** $p_i == -1$ **then**
24: $s_i = \max_{j=1}^{i-1}(s_j) + 1$ ▷ assign new storm ID
25: **else**
26: $s_i = s_{p_i}$ ▷ copy storm ID from predecessor
27: **return** s

3.2.1 Algorithm Walkthrough

Algorithm 3.1 begins in line 5 by iterating through lightning observation records formatted as in Table 3.1 and sorted by time. In line 6, each record i is compared

with the set S of up to C preceding records:

$$S = \begin{cases} \{i - C,\ \ldots\ ,\ i - 2,\ i - 1\} & \text{if } i - C > 0 \\ \{0,\ \ldots\ ,\ i - 2,\ i - 1\} & \text{otherwise.} \end{cases}$$

From this set, lines 7-10 remove any elements j associated with lightning events that occur more than D nautical miles (NM) away, calculated using the Euclidean (ℓ_2) norm to approximate geographical distance. In lines 11-12, if any records remain in S, the algorithm finds the most recent and labels it k. Line 13 gets the time difference between t_k and t_i and checks it against the time threshold T. If the time difference falls within T, record i is given predecessor ID k in line 14. If set S is empty or the time difference $t_i - t_k$ exceeds our time threshold T, this indicates that any candidate predecessors were either too far away or too old; in either case, the algorithm assigns a value indicating that no predecessor was found.

The algorithm can then use predecessor information to assign storm IDs. It iterates through the records a second time. In line 21, the very first record is automatically assigned storm ID $s_1 = 1$. In lines 23-24, if a given record i does not have a predecessor, it is considered to be the beginning of a new storm. In this case, the algorithm assigns it a new storm ID according to the formula

$$s_i = \max_{j=1}^{i-1}(s_j) + 1.$$

Otherwise, it is considered to be a continuation of an existing storm, and in line 26 it is therefore assigned the same storm ID as its predecessor. The algorithm then returns the vector s associating storm IDs with each record in A.

3.2.2 Algorithm Evaluation

The parameter C limits the number of records being compared with record i on each iteration, and thus helps to alleviate run time issues. Without this parameter, the algorithm performs a full pairwise comparison, and the running time grows prohibitively large; in our experiments, it took approximately 2.5 hours to process a file representing less than 0.02% of the complete data set. By limiting the comparison to only the previous $C = 30$ records (a value chosen based on multiple trial runs), the algorithm was able to process the entire data set over the course of approximately two weeks. However, this parameter creates a risk of missing valid predecessors, and therefore of assigning storm IDs incorrectly. We have traded a speed issue for an accuracy issue.

Further, the algorithm's output quality is sensitive to input parameters T and D. By using this algorithm, we effectively define the size of a storm with simple time and distance cutoffs, and small changes to these cutoffs can dramatically alter the assignment of storm IDs. If, for example, two dense groups of lightning activity are 6.5 NM from each other at their closest point, then with $D = 6.4$ we identify them as two separate storms, but with $D = 6.6$ we connect them to form a single storm of a much larger size. It is therefore important to determine the parameter settings that most accurately define a storm. However, the only method we have for accomplishing this is to test a wide variety of parameter settings on large data sets (weeks of data, or millions of records). Even with parameter C limiting comparisons, the run time for a large data set is prohibitive to this kind of experimentation.

Finally, this algorithm requires access to multiple records at once; this necessitates a data handling scheme, as the full data set is too large to load into memory at once. We can divide the data set into batches to be processed separately (e.g. process one day of lightning data at a time). When we do this, however, the algorithm fails to

handle the rollover between batches – due to the break when one batch is dropped from memory and the next is loaded, storms that begin within one batch and continue into the next are incorrectly identified as two separate storms. We can work around this rollover issue by formatting the data as a database that we can query, such that only the records we are actively comparing are loaded in memory. This exacerbates existing run time issues, however, as it takes significantly longer to perform a database query than to retrieve information that is already loaded in memory.

Overall, Algorithm 3.1 has the advantage of being fairly straightforward in concept, but it requires too much balancing of accuracy, run time, and memory requirements to be effective for large data sets, such as the one we use in this study.

3.3 Grid Algorithm

The grid algorithm presented here is a modification of an algorithm proposed by Roeder (2020). The original version from Roeder calculates approximate distances of lightning flashes from pre-existing lightning areas; our modified version in Algorithm 3.2 aggregates lightning observation records into storms.

This algorithm seeks to improve on the run times of existing clustering algorithms by discretizing the data and replacing the typical Euclidean (ℓ_2) distance calculation with the faster rectilinear or "Taxicab" (ℓ_1) norm. In this section, we present Algorithm 3.2, followed by an example, and then we provide an evaluation.

3.3.1 Algorithm Walkthrough

Before running the algorithm, the user must specify conversion factors U and V and a time interval T. Parameters U and V are applied to the longitude and latitude of each point, respectively, to define the dimensions of the spatial grid onto which lightning events are aligned. For a convenient example, 1 degree (°) latitude equals

Algorithm 3.2 Grid Algorithm

Input: list A of lightning observation records (time since epoch t_i, longitude x_i, latitude y_i), sorted by t_i
Parameters: conversion factors U, V; time interval T, neighbor threshold L
Output: vector s of storm IDs corresponding to records in A

DISCRETIZE:
$timeMin = \min_{i \in A}(t_i)$
$longMin = \min_{i \in A}(x_i)$
$latMin = \min_{i \in A}(y_i)$
$J = (J^t,\ J^x,\ J^y,\ J^s)$ ▷ empty table w/ 4 integer cols
$J^n = \varnothing$ ▷ empty list of sets
for each $i \in A$ **do**
    $t = \lfloor (t_i - timeMin)/T \rfloor$
    $x = \lfloor U(x_i - longMin) \rfloor$
    $y = \lfloor V(y_i - latMin) \rfloor$
    $s = -1$ ▷ storm ID; -1 indicates no assignment
    if $(t,\ x,\ y,\ s) \notin J$ **then**
        Append $(t,\ x,\ y,\ s)$ to J
        Append singleton set $\{i\}$ to J^n ▷ map J to A
    else
        for $j \mid (t,\ x,\ y,\ s) == J_j$ **do**
            Append i to set J^n_j ▷ map redundant point

INITIALIZE:
$C = \varnothing$ ▷ empty set of graph edges
$s_{new} = 1$ ▷ initial storm ID
$s = -\mathbf{1}$ ▷ storm ID vector of length $|A|$; -1 indicates no assignment

function IsNeighbor(integer i, integer j)
    $l_i = |J^x_i - J^x_j| + |J^y_i - J^y_j|$
    if $l_i \leq L$ **then**
        return TRUE

33: **for** each $t \in \{J^t\}$ **do** ▷ with distinct, ascending t
34:
35: $N = \{i \mid J_i^t == t\}$
36: $P = \{i \mid J_i^t == t - 1\}$
37: **if** $P \neq \varnothing$ **then** $D = C$ **else** $D = \varnothing$
38: $C = \varnothing$
39:
40: **for** each $\{i, j\}$ such that $i \neq j$ and $i, j \in N$ **do**
41: **if** IsNeighbor(i, j) **then**
42: Add edge $\{i, j\}$ to C if not present
43: **for** each connected subgraph $c(c_V, c_E) \subseteq G(N, C)$ **do**
44: **if** $P \neq \varnothing$ **then**
45: **for** each connected subgraph $d(d_V, d_E) \subseteq G(P, D)$ **do**
46: **for** each $\{i, j\}$ such that $i \in c_V$ and $j \in d_V$ **do**
47: **if** IsNeighbor(i, j) **then**
48: **for** each $k \in c_V$ **do**
49: $J_k^s = J_j^s$ ▷ inherit storm ID
50: **continue** to next c in line 43
51: **for** each $k \in c_V$ **do** ▷ only executes if no storm ID assigned
52: $J_k^s = s_{new}$ ▷ assign new storm ID
53: $s_{new} = s_{new} + 1$
54:
55: **for** each $i \in J$ **do**
56: **for** each $j \in J_i^n$ **do**
57: $s_j = J_i^s$
58: **return** s

60 NM; if, as part of discretizing latitude values, we multiply them by $U = 60$, we effectively create a grid that has vertical (latitudinal) increments of 1 NM. Longitude is more complicated, as the spacing between standard longitudinal lines varies with latitude. For computational ease, we use the average latitude across the entire data set to approximate that 1° longitude ≈ 51.8 NM for our area of concern. If desired, we can make further conversions from there. Meanwhile, T defines a window of time within which lightning events are grouped together. Since we group time windows by discretizing in a similar manner to the spatial data, it can be said that we snap data to a three-dimensional grid, with time acting as a third axis.

Algorithm 3.2 begins in lines 6-14 by discretizing the data according to the formulae

$$t = \lfloor (t_i - \min_{j \in A}(t_j))/T \rfloor,$$
$$x = \lfloor U(x_i - \min_{j \in A}(x_j)) \rfloor,$$
$$y = \lfloor V(y_i - \min_{j \in A}(y_j)) \rfloor.$$

Here, the "origin" of the discrete grid is defined by subtracting the minimum time, longitude, and latitude in the data from each record's coordinates. The grid spacing is defined by applying T, U, and V, and then the data points are "snapped" to the grid by truncating the resulting values to integers.

The algorithm eliminates redundant data by storing unique grid coordinates in J (line 9), and uses J^n (line 10) to map J back to A so that storm IDs can later be assigned to the original data. In lines 15-21, the algorithm creates a placeholder storm ID, then combines this with our discretized coordinates to form a record in J, which tracks cells on the grid that contain data. The algorithm also creates a set within list J^n to which it adds record index i; the sets in J^n keep track of which records from the original data are contained within each grid cell. If, upon attempting to add a record to J, the algorithm finds that a matching record already exists, it simply adds the original record's index to the appropriate set in J^n.

In lines 24-26, the algorithm then initializes an empty set of graph edges $C = \varnothing$, a counter s_{new} for generating storm IDs, and vector s of length $|A|$ to hold all storm ID assignments. Vector s is pre-assigned values of -1 to indicate that no record has yet been assigned a storm ID.

Lines 33-53 contain the main body of the algorithm, which iterates through distinct time values t in J^t. Lines 35-36 create two index sets that reference subsets of J: a "new" set N referencing records that share the current value of t, and a "previous" set P referencing records occurring during the time $t - 1$ (note that P

is empty on the first iteration and after any gap in t values, indicating no preceding lightning). Line 37 checks whether P contains data; if so, the graph edges C from the previous iteration are associated with this data, and the algorithm relabels the set as D. Otherwise, D is emptied of any existing data. In either case, C is reset in line 38. In lines 40-42, records referenced by N are compared to find "neighbors," as defined by the *IsNeighbor* function in lines 28-28, and any neighbors become connected in C, as shown in Figure 3.1.

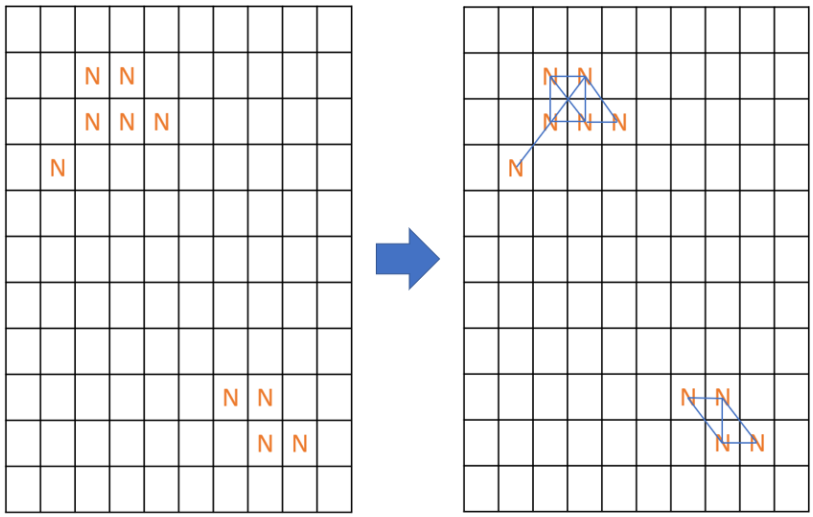


Figure 3.1. Example of the first iteration of the grid algorithm. (b) contains two connected subgraphs. For this example, graph edges are created between filled cells separated by a rectilinear distance of at most 2.

Storm IDs may then be assigned as in lines 43-52 of the algorithm. On the first iteration, set P is empty; each internally connected subgraph c in the graph $G(N, C)$ is associated with a unique storm ID, which is in turn assigned to each associated record in J (note that $G(N, C)$ denotes a graph with vertex set N and edge set C). When both P and N contain data, each subgraph c in $G(N, C)$ is compared with each subgraph d in $G(P, D)$; when c neighbors or overlaps exactly one subgraph d in $G(P, D)$, it is taken to be a continuation of the storm represented by d. Thus, in line 49, the storm ID associated with d is copied to the records identified in c_V. When

c neighbors or overlaps multiple subgraphs in $G(P, D)$, this is taken to represent multiple storms merging into one. In this case, line 49 of the algorithm copies the storm ID of the first subgraph encountered to the records identified in c_V. Finally, when c does not neighbor or overlap any subgraphs in D, records identified in c_V are assigned a new storm ID in lines 51-53. An example of this process is illustrated in Figure 3.2. Note that when we have multiple candidate subgraphs in $G(P, D)$ to connect with c, we deem it acceptable to simply select one because we mainly need to ensure that, in this case, records associated with c are assigned an existing storm ID and not a new one. Additionally, this method allows each storm's beginning (the part we are primarily concerned with) to remain separate.

Once the loop beginning in line 33 completes, each record in J (each occupied grid cell) has been assigned a storm ID. In lines 55-57, the algorithm uses J^n to map these IDs onto the vector s, effectively associating storm IDs with each record in the original data A. The algorithm terminates by returning s.

3.3.2 Algorithm Evaluation

Algorithm 3.2 is capable of achieving fast run times. By aggregating individual events co-located within each grid cell, it dramatically decreases the number of comparisons that must be made. These improvements to the algorithm's run time come at the cost of reduced accuracy, however. Discretizing the data inherently introduces error; when snapping points to a grid, we cause some records to move closer to each other in space and time, and some to move farther apart. Some graph connections may be made that would otherwise not have been, and some may be missed that would otherwise have been captured. We can improve accuracy by increasing the grid's granularity (decreasing its spacing). For example, doubling U doubles the number of grid cells longitudinally. Of course, fitting more grid cells into the same

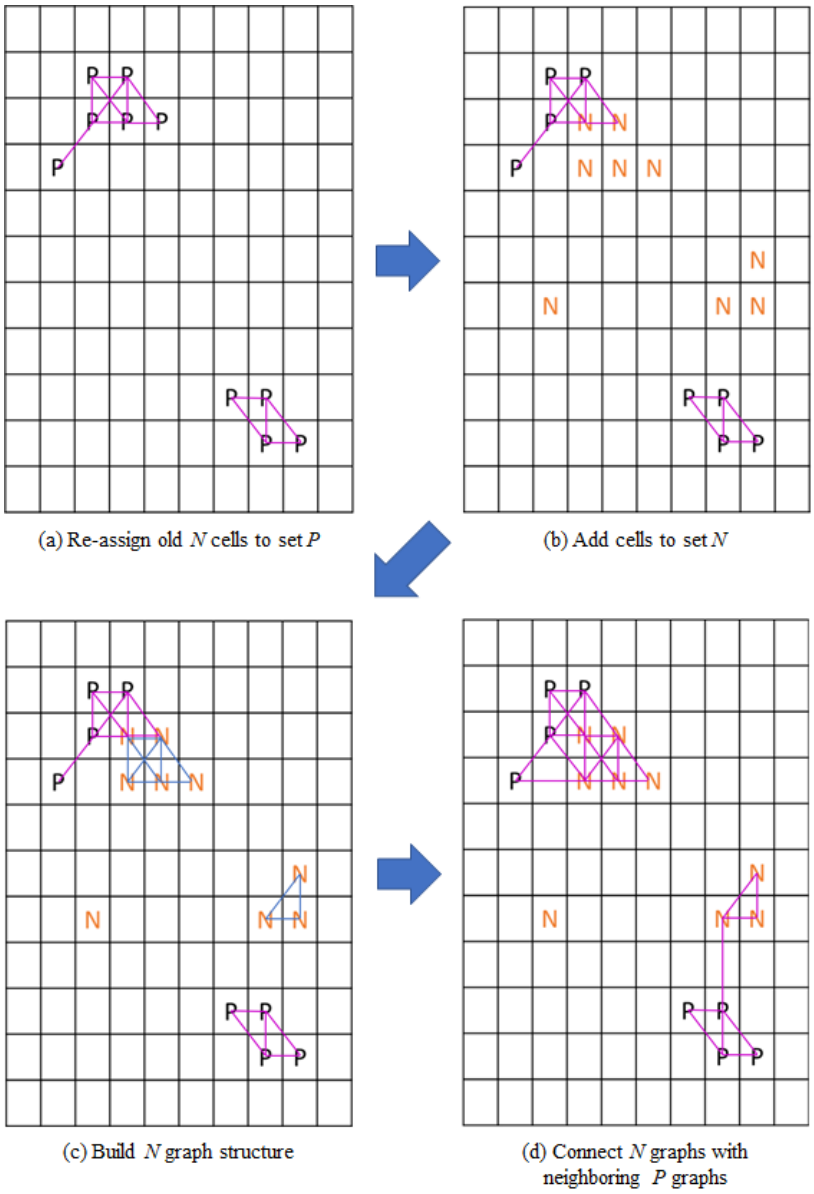


Figure 3.2. Example of a typical iteration of the grid algorithm. This example follows directly from Figure 3.1. As in the previous figure, cells in (c) and (d) are considered to be neighbors when separated by a rectilinear distance of at most 2.

space improves accuracy, but it also means an increase in the comparisons that must be made. Thus, we see that accuracy and run time are inversely correlated, and both are controlled by the parameters U, V, and T.

Extensive experimentation is required to determine the most appropriate parameter settings. This includes not only the time interval T and conversion factors U and

V, but also the definition for when we consider two grid spaces to be neighbors. Moreover, these four factors may interact with each other to varying degrees, depending on the weather patterns inherent in the data set. Thus, we are presented with an opportunity to "slow down to speed up:" we may conduct a parameter-tuning study to optimize Algorithm 3.2 to obtain acceptably accurate storm partitions in a reasonable time frame. However, the complexity of this study is impossible to predict, as are the potential run time improvement and end-state accuracy. Due to these circumstances, we elect to prioritize an algorithm that can be developed and tested more quickly. Nevertheless, Algorithm 3.2 remains a potential avenue for future research.

3.4 Clustering of Online Data Streams (CODAS)

Hyde and Angelov (2015) present the Clustering of Online Data Streams (CODAS) algorithm, which is designed to cluster data quickly, based on local data density, and into arbitrary shapes in an arbitrary number of dimensions. The "online" in the title indicates that the algorithm may be fed one data point at a time, so that the entire data set need not be kept available in memory. This additionally confers the benefit that the rollover from one data file to the next is handled correctly by default. This section first defines the "microcluster," the algorithm's basic building block, then describes how it is used to build "macroclusters," which correspond to storms.

3.4.1 Microclusters

CODAS operates on a two-tiered hierarchy of clusters. The lower-tier clusters are called microclusters (see Figure 3.3). Each microcluster is a hypersphere of the same dimensionality as the data being processed and has a radius (R in Figure 3.3) specified as a parameter input to the CODAS algorithm. The microcluster additionally contains a "kernel"; this is a smaller, concentric hypersphere with radius $R/2$

that CODAS uses to define how the microcluster moves. Because the kernel controls movement, CODAS microclusters are less sensitive to outliers and more limited in how they shift their location to follow a drifting data set. In Section 3.4.3, we describe how the kernel also helps to form the upper-tier macroclusters.

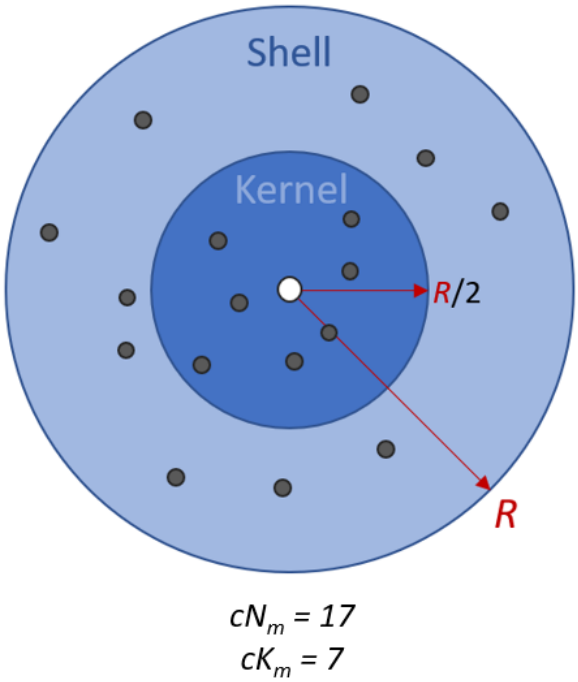


Figure 3.3. Illustration of a CODAS microcluster. The light blue circle is the microcluster m, the white circle is its center cC_m, and the dark blue circle is its kernel. The black circles are points that have been added to the cluster. cN_m counts the number of points in the overall cluster, while cK_m counts the number of points in just the kernel.

CODAS tracks microclusters by recording their center points cC_m, which are initially calculated by averaging the coordinates of all member points inside the microcluster (see Algorithm 3.3, lines 24, 27, 30-31). After microcluster creation, any new points that fall inside the cluster are assigned to it. That is, in H-dimensional space, point P is assigned to the cluster centered at Q if

$$\sqrt{\sum_{h=1}^{H}(P_h - Q_h)^2} \leq R$$

(Euclidean norm; all CODAS distances are calculated this way). Points falling in the intersection of multiple clusters are assigned to the cluster with the nearest center. As points are added to each microcluster m, CODAS tracks the number of points inside the kernel (cK_m) and the number of points assigned to the microcluster ($cN_m \geq cK_m$). The individual points are dropped from memory. Figure 3.3 illustrates this process, further detailed in Algorithm 3.3.

3.4.2 Algorithm to Construct Microclusters

Algorithm 3.3 accepts lightning data formatted as in Table 3.1, in addition to a radius parameter R and an outlier density threshold D. It returns a set of microclusters connected by a graph structure as output. After initializing data structures in lines 6-10 of Algorithm 3.3, lines 12-33 define the function `OutlierHandler`, which dictates what happens when a new point does not fall within any existing microclusters. The point is first added to a list of "outliers". In lines 14-17, CODAS then performs a pairwise comparison of the outliers, checking which points are within one microcluster radius R of each other; we call such pairs "neighbors." In lines 18-21, we find the point p that has the most neighbors N_{max}. In lines 23-33, if p has enough neighbors N_{max} to meet the minimum threshold D, then CODAS forms a new microcluster m. Note that while this differs, strictly speaking, from finding a set of points that fit within a sphere, it is similar and does not detract from CODAS's effectiveness (see Appendix A for more detail).

Once a microcluster is generated, lines 24, 27, and 30 collectively define the center of this new microcluster as the average of the positions of the included points. The microcluster's point count cN_m and kernel point count cK_m are updated in lines 32 and 33 respectively, and the included points are removed from the table of outliers O, and thus from memory. Figure 3.4 illustrates this process.

Algorithm 3.3 CODAS Algorithm

Input: Set A of lightning observation records (time since epoch t_i, longitude x_i, latitude y_i), sorted by t_i
Parameters: radius R, outlier density threshold D
Output: Set cC of microcluster centers and graph G, collectively representing storms

INITIALIZE:
cC (cluster centers; empty table w/ columns DateTime ct_m, Longitude cx_m, Latitude cy_m)
cN (cluster counts; empty integer list)
cK (cluster kernel counts; empty integer list)
O (outliers; empty table w/ columns DateTime ot_j, Longitude ox_j, Latitude oy_j)
E (empty set of graph edges)

function OUTLIERHANDLER(integer i)
    Append record A_i to O
    for each record $j \in O$ **do**
        for each record $k \in O$ **do**
            $d_{jk} = \sqrt{(ot_j - ot_k)^2 + (ox_j - ox_k)^2 + (oy_j - oy_k)^2}$
            if $d_{jk} \leq R$ **then** $n_{jk} = 1$ **else** $n_{jk} = 0$
    $L = \text{length}(O)$
    for each $j \in O$ **do** $N_j = \sum_{k=1}^{L}(n_{jk})$
    $N_{max} = \max_{j=1}^{L}(N_j)$
    $p = \arg\max_{j=1}^{L}(N_j)$

    if $N_{max} \geq D$ **then**
        $C = (0, 0, 0)$
        $K = 0$
        for each $k \mid n_{pk} == 1$ **do**
            $C = C + O_k$ ▷ vector addition is elementwise
            if $d_{pk} \leq R/2$ **then** $K = K + 1$
            Delete O_k
        $C = C/N_{max}$ ▷ elementwise division
        Append C to cC
        Append N_{max} to cN
        Append K to cK

34: **for** each $i \in A$ **do**
35: **if** $cC == \varnothing$ **then**
36: Run OutlierHandler(i)
37: **else**
38: **for** each $j \in cC$ **do**
39: $d_j = \sqrt{(t_i - ct_j)^2 + (x_i - cx_j)^2 + (y_i - cy_j)^2}$
40: $L = \text{length}(cC)$
41: $d_{min} = \min_{j=1}^{L}(d_j)$
42: $m = \arg\min_{j=1}^{L}(d_j)$
43: **if** $d_{min} \leq R$ **then**
44: $cN_m = cN_m + 1$
45: **if** $d_{min} \leq R/2$ **then**
46: $cK_m = cK_m + 1$
47: $cC_m = \frac{cC_m(cK_m - 1) + A_i}{cK_m}$
48: **for** each $j \in cC \mid j \neq m$ **do**
49: $d_j = \sqrt{(ct_m - ct_j)^2 + (cx_m - cx_j)^2 + (cy_m - cy_j)^2}$
50: **if** $d_j \leq 1.5R$ **then**
51: Add edge $\{m, j\}$ to E if not present
52: **else**
53: $m = \text{length}(cC)$
54: Run OutlierHandler(i)
55: **if** $\text{length}(cC) > m$ **then**
56: $m = \text{length}(cC)$
57: **for** each $j \in cC \mid j \neq m$ **do**
58: $d_j = \sqrt{(ct_m - ct_j)^2 + (cx_m - cx_j)^2 + (cy_m - cy_j)^2}$
59: **if** $d_j \leq 1.5R$ **then**
60: Add edge $\{m, j\}$ to E if not present

Lines 34-60 give the main loop of the algorithm. Each iteration reads one record, A_i, from the input data set and processes it according to one of three cases. The first occurs when no microclusters exist yet (line 35); in this case, the new data point A_i of course does not fall within any microclusters. CODAS runs the `OutlierHandler` function and ends the current iteration, moving on to the next data point. At this time, the addition of point A_i may or may not have been enough to cause the first microcluster to form.

In the second case, microclusters exist and A_i falls inside at least one of them. When this happens, lines 38-51 add A_i to the nearest microcluster m (see Figure 3.5).

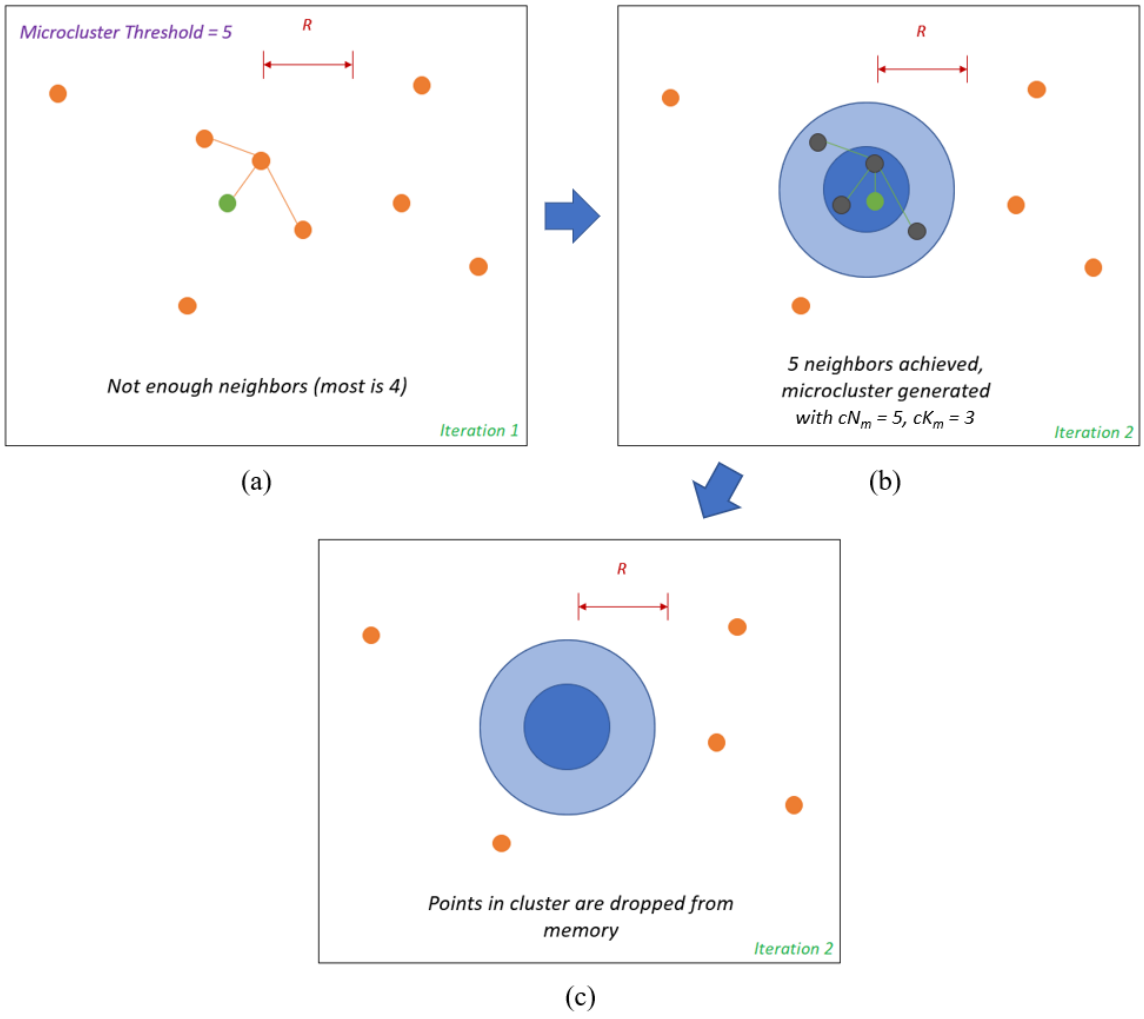


Figure 3.4. Example of a new microcluster being formed. The orange dots are "outliers," and the green dots in (a) and (b) are new data points.

Line 44 increments the appropriate cluster point counter cN_m, denoting that m now contains one additional point. If A_i falls outside m's kernel, then the iteration ends and A_i is dropped from memory. If, however, A_i falls inside the kernel, then CODAS executes lines 46-51. The appropriate kernel point counter cK_m is incremented, and m's center cC_m is updated according to the equation

$$cC_m = \frac{cC_m(cK_m - 1) + A_i}{cK_m},$$

which approximates the new average position of points in the kernel such that the center moves toward A_i. We leave discussion of lines 48-51 for the next section. The iteration ends, and A_i is dropped from memory.

Finally, in the third case, microclusters exist, but A_i does not fall within any of them. If this happens, in lines 53-56 CODAS runs the `OutlierHandler` function and assigns variable m to check whether a new microcluster was formed; if so, m becomes the new cluster's ID. This leads into lines 57-60.

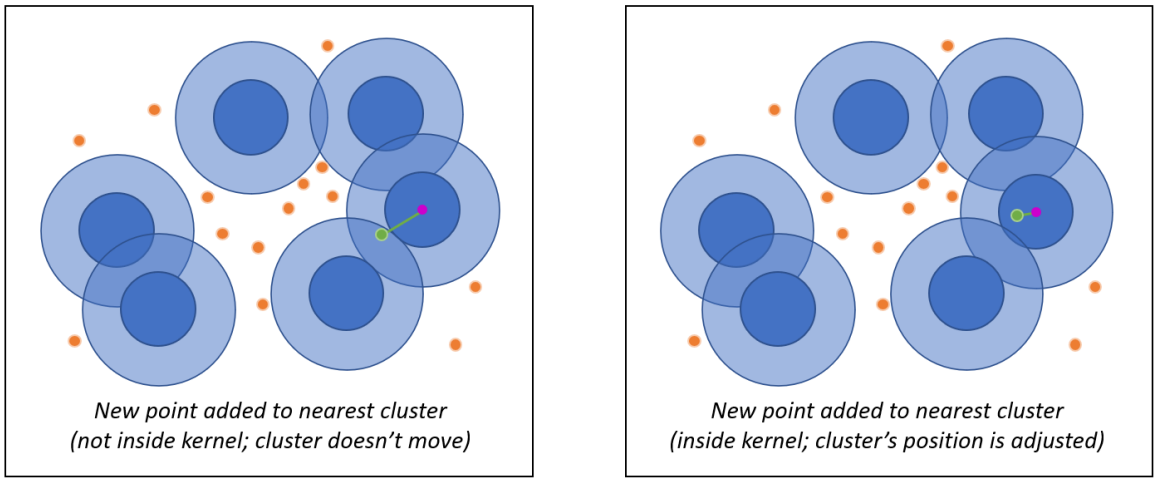
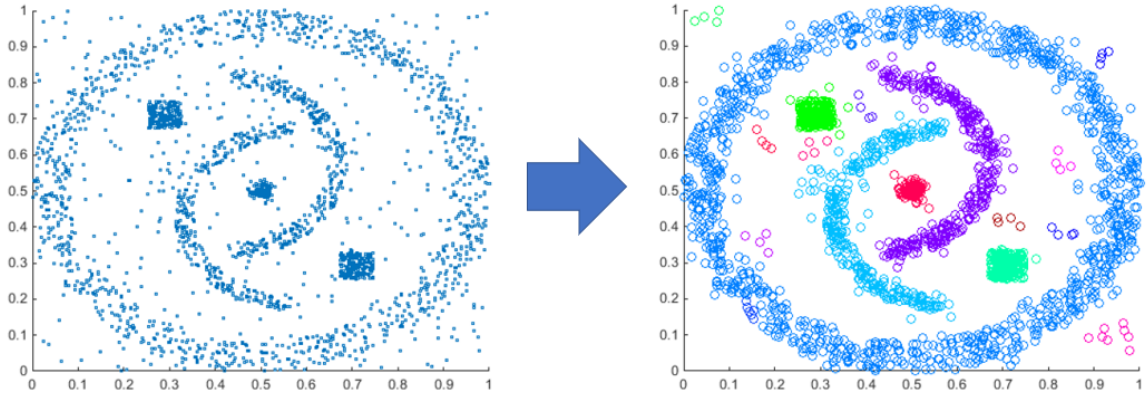


Figure 3.5. Adding a point to an existing microcluster (two cases). The green dots in each case represent a new point being added to a microcluster. The purple dots are the centers of the affected microclusters.

3.4.3 Macroclusters

In lines 48-51 and 57-60, CODAS checks m's distance to each other cluster to see if any are within $1.5R$. If any are, they form graph edges with m. Macroclusters are defined implicitly using the graph structure $G(I, E)$, where $I = \{m \in cC\}$ consists of the indices of all microclusters, representing vertices, and E contains graph edges. Within $G(I, E)$, each connected subgraph represents a separate macrocluster. Figure 3.6b provides a visual example of how macroclusters can represent arbitrarily-shaped

regions of high-density microcluster activity; the figure contains six prominent macroclusters, in addition to a number of other much smaller macroclusters resulting from background noise. This ability to represent arbitrary regions is of particular value in our context, because storms can have unusual shapes. The aim of the CODAS algorithm is to identify high-density clusters in the input data set by fitting macroclusters to them; thus, a macrocluster can be said to represent a region of high data density. In our case, a storm is an area of dense lightning activity, and thus we can identify storms by fitting macroclusters to them.



(a) Raw data (b) Color-coded clusters

Figure 3.6. Macrocluster example (Hyde and Angelov, 2015). In (b), each individual circle represents a microcluster, while each set of circles of the same color collectively represents a macrocluster.

Since graph nodes are represented by microclusters, node creation is already implicit in the algorithm. To construct the graph edges, we simply update edges each time a microcluster is moved (line 47) or added (line 54). Note an important distinction in the rule for generating edges: microclusters do not form edges simply by touching another cluster. They must be closer, within $1.5R$ of each other, to be connected by an edge – in other words, their outer shells must each be in contact with the other's kernel. This helps to minimize the open space that typically forms amidst masses of connected microclusters, as in Figure 3.6. To illustrate this rule, note Fig-

ure 3.7a, where graph edges are shown in purple. The three right-most clusters are connected, forming a macrocluster; however, the cluster in the top center is not close enough to form a connection. It is in contact with another cluster's shell, but not its kernel. Once a new microcluster is generated in Figure 3.7b, it happens to be close enough to connect to both macroclusters, merging them into one.

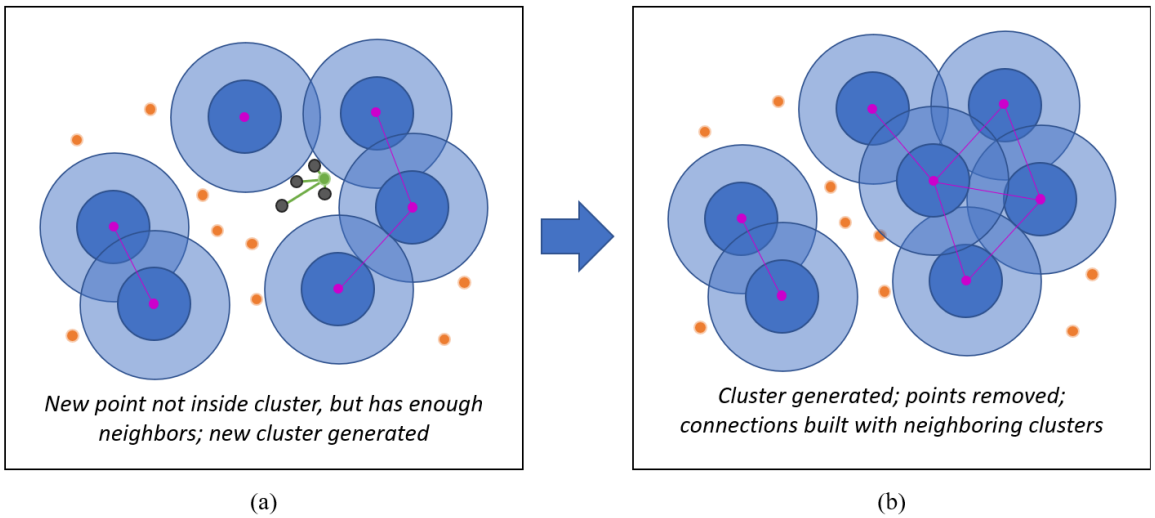


Figure 3.7. Updating macrocluster graph structure. The purple lines represent graph edges; each set of microclusters connected by these graph edges forms a macrocluster.

3.4.4 Configuring CODAS for Lightning Data

To partition our lightning data into storms, we perform clustering in three dimensions: time, latitude, and longitude. In particular, we desire that lightning observations occurring within about 5 NM and 15 minutes of each other should be part of the same storm. Since CODAS operates with hyperspheres, we therefore normalize the data according to

$$\begin{aligned} t_i &= T(t_i), \\ x_i &= X(x_i * 51.8), \\ y_i &= Y(y_i * 60), \end{aligned}$$

where 51.8 and 60 are constants to convert spatial distances to NM, and T, X, and Y ensure that a difference of 15 minutes (in time) or 5 NM (in latitude or longitude) translates to a normalized distance of one. We then set parameter $R = 1$. We additionally set the outlier density threshold $D = 5$, enough to reasonably limit the number of microclusters that will be generated. These are all the preparations necessary to run CODAS on our lightning data. We do so, and after execution, we undo the data normalization on the output. At this point, we have a set of macroclusters that ideally represent every storm in the data.

Storm IDs must still be assigned, however. Algorithm 3.4 iterates through connected subgraphs (macroclusters) g of G; each macrocluster gets a unique ID that is assigned to every member microcluster. Since data points are dropped from memory after being processed, the clusters generated by CODAS are not directly associated with the data at this stage. The algorithm handles this with an approximation: for each data point A_i, it simply finds the nearest microcluster m and checks whether A_i is inside it. If so, we copy the macrocluster ID from m and assign it to $s_i \in s$, where s is a storm ID vector with elements corresponding to records in A. If A_i is not inside any microclusters, we assign a default value of $s_i = 0$ to indicate as such. The algorithm terminates by returning s.

3.4.5 Relationship of Parameters to Lightning Data

The choice of parameter D is trivial in most cases; the user can simply choose a number such that macroclusters collect at least some small number of points. $D = 5$ is sufficient for most use cases. The choice of the parameter R, on the other hand, is fundamentally important to the performance and output of the CODAS algorithm. In application to lightning data, a value of R may be selected that closely correlates to the natural phenomena we want the microcluster to represent. For example, if we

Algorithm 3.4 Associate data with clusters

Input: Set A of lightning observation records (time since epoch t_i, longitude x_i, latitude y_i), sorted by t_i; set cC of microclusters (time since epoch ct_m, longitude cx_m, latitude cy_m), sorted by ct_m; graph G containing vertex set G_V and edge set G_E
Parameters: radius R
Output: vector s of storm IDs corresponding to records in A

INITIALIZE:
$ID = 0$ ▷ storm ID
cS ▷ microcluster storm IDs; empty list
$s = \mathbf{0}$ ▷ lightning event storm IDs; zero vector

for each connected subgraph g in G **do**
    $ID = ID + 1$
    for each m in g_V **do** $cS_m = ID$
for each i in A **do**
    for each j in cC **do**
        $d_j = \sqrt{(t_i - ct_j)^2 + (x_i - cx_j)^2 + (y_i - cy_j)^2}$
    $L = \text{length}(cC)$
    $d_{min} = \min_{j=1}^{L}(d_j)$
    $m = \arg\min_{j=1}^{L}(d_j)$
    if $d_{min} \leq R$ **then** $s_i = cS_m$
return s

make our microclusters cover roughly the size and time span of a typical lightning strike, we can say that our microclusters represent lightning strikes. However, this clustering will provide little information beyond what exists in the raw data set. At the other extreme, we could try to make them large enough to encompass entire storms and say that a microcluster represents a storm, but this would effectively assume that all storms are hyperspheres in time and space. In between these extremes, we might choose to have microclusters represent "lightning warning areas," and size them appropriately. But the lightning warning areas of the 45 WS are spatially located by buildings, not lightning strikes. In summary, we find that these micro-clusters are difficult to define in representational terms, and we instead choose our microcluster

size based on utility. Rather than carrying any specific meaning, a microcluster is simply a hypersphere of an appropriate size to collect a few data points and form macroclusters of good resolution. In our study we choose a microcluster radius similar to lightning warning area dimensions, since the two constructs serve similar purposes: representing an upper bound on the geographic progression of a sequence of lightning events over a specified time interval. Moreover, studies by Hyde and Angelov (2015) indicate that macrocluster structures are fairly robust to changes in the microcluster radius. Nevertheless, since we have not performed tests on a sufficiently wide variety of parameters, it remains an area of future research to examine whether different radii might provide a better representation of storm activity.

3.4.6 Algorithm Evaluation

CODAS makes pairwise comparisons at many different stages; however, in keeping with its online nature, it maintains and operates on much smaller data sets than Algorithm 3.1 at any given time. Thus, CODAS is faster than Algorithm 3.1, processing the MERLIN data set over approximately three to five days of continuous computation. This is still likely to be significantly slower than Algorithm 3.2 in most cases. It has a major advantage over both, however: it is far more robust to changes in the input parameters. The user specifies only a microcluster radius R and an outlier density threshold D, neither of which requires optimization to achieve good results. R only needs to be within a reasonable range: large enough to ensure microclusters will connect to form macroclusters, small enough to prevent macroclusters from connecting to each other over unreasonably large distances. Likewise, D only needs to be large enough to help keep the microcluster count from growing too rapidly. Under these conditions, a reasonable guess is sufficient to select parameters that will accurately identify the vast majority of storms. Thus, we eliminate the need to process

the data many times in order to optimize the parameters, and achieve our greatest speed increase via ease of use.

When we seek to visually validate the storm data output of our algorithm, the results are promising. When the resulting clusters are plotted on a map, we see that known weather phenomena can be identified visually. Figure 3.8 presents an example, where anvil lightning is represented by the shape of the clusters circled in red (Horvath, 2006; Roeder, 2021).

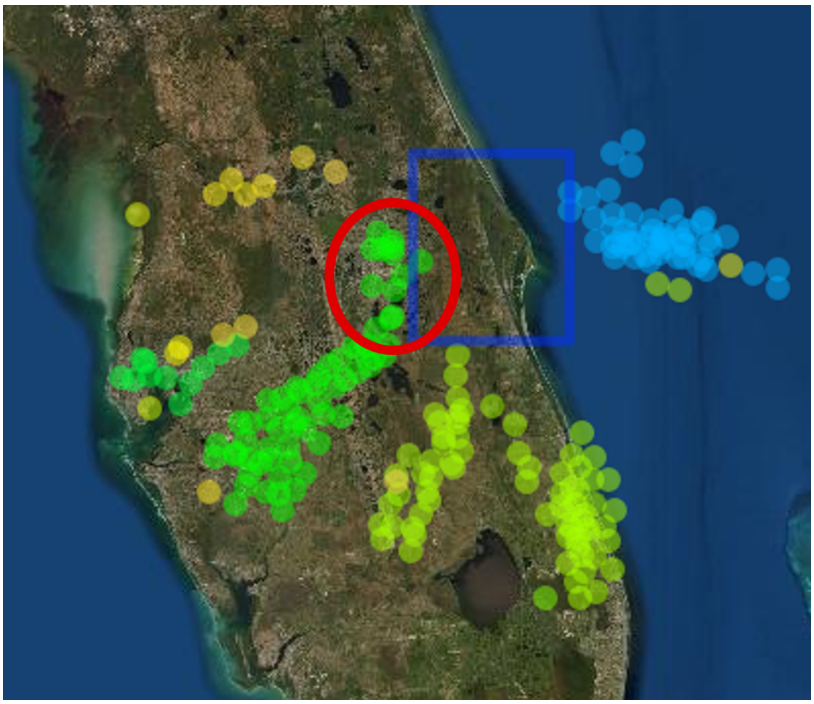


Figure 3.8. Florida storms represented by CODAS clusters. The green microclusters inside the red ellipse collectively represent an instance of anvil lightning. The blue rectangle represents the region containing MERLIN sensors.

3.4.7 Microcluster Benefits

Since the shape of all microclusters is completely defined by a single, uniform value, it is easy to track and store them by simply recording the locations of their centers. They readily scale to any dimensionality by increasing the number of values that define their center coordinates. Parameter selection is straightforward; for most

use cases, one need only select a radius large enough to encompass a few data points, and a density threshold large enough to keep the number of clusters manageable. This enables us to achieve useful results using parameters selected via intuition. Further, microcluster kernels provide a straightforward way to have clusters follow drifting data, improving the robustness of the algorithm as a whole – meanwhile, the size of the kernel relative to the cluster also limits its motion, preventing it from moving so quickly or erratically that it risks pulling away from the bulk of the data. It also is simple to have microclusters each collect a few data points at a time, then connect the microclusters together to form much larger clusters of arbitrary shape. Finally, key to our study, microclusters are excellent at data reduction; because lightning records are dropped from memory upon being added to a microcluster, we can make comparisons over a pair of small data sets representing hundreds or thousands of points (microclusters and outliers), rather than one large set representing millions.

3.5 Summary

The 55.7 million lightning events detected by MERLIN between May 2019 and September 2019 must be organized into storms before they can be properly analyzed. The pairwise comparison algorithm presented in Section 3.2 is capable of processing this data given extensive run time, but the results are suspect, as the output is overly sensitive to the input parameters. The grid algorithm in Section 3.3 is projected to have a much faster typical run time than the pairwise comparison algorithm, but it is yet more sensitive, and to a larger number of parameters. The CODAS algorithm in Section 3.4 runs faster than the pairwise comparison algorithm and is much more robust, allowing us to select its parameters intuitively and achieve useful results on the first attempt. We therefore use CODAS to identify storms in our data set.

IV. Lightning Behavior Analysis

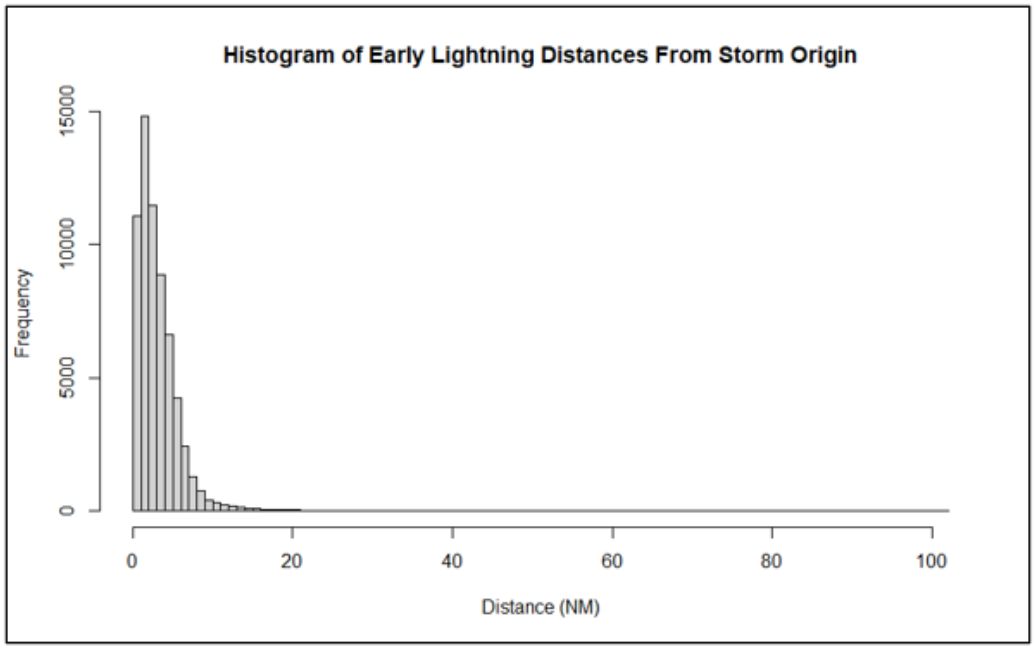
In the previous chapter, we reviewed how to take a data set of lightning events and cluster them into storms. In this chapter we consider the spatial distances between lightning events in developing storms. The goal is to ascertain whether we might safely reduce the current lightning warning radius from the current standard of 5 nautical miles (NM). Sanderson (2019) finds that the distances of lightning flashes from a pre-existing lightning area follow a Weibull(0.833,2.142) distribution, with distances measured in kilometers (km). Thus, by converting NM to km and checking against the distribution, we find that the probability of a lightning strike beyond 5 NM of a lightning area is approximately 3.39%, whereas the probability of a lightning strike beyond 4 NM of a lightning area is approximately 6.01%. Based on empirical validation and a "what-if" analysis exploring how historical failure rates would have changed given a 4 NM warning radius, Sanderson argues that this increase in risk is outweighed by the $1 - (4 \text{ NM})^2\pi/(5 \text{ NM})^2\pi = 36\%$ decrease in overall number of lightning warnings issued. Moreover, her preliminary analysis suggests that similar results may be true for developing storms (where no pre-existing area is available). Based on the observations of Sanderson (2019), we expect that a reasonable lightning warning radius will be approximately 4 NM. In particular, we expect that the probability that lightning strikes in a developing storm are more than 4 NM away from the first lightning strike is no more than 6.01%. This result would strongly support a policy change to reduce the lightning warning radius for developing storms.

4.1 Distance Calculations and Histograms

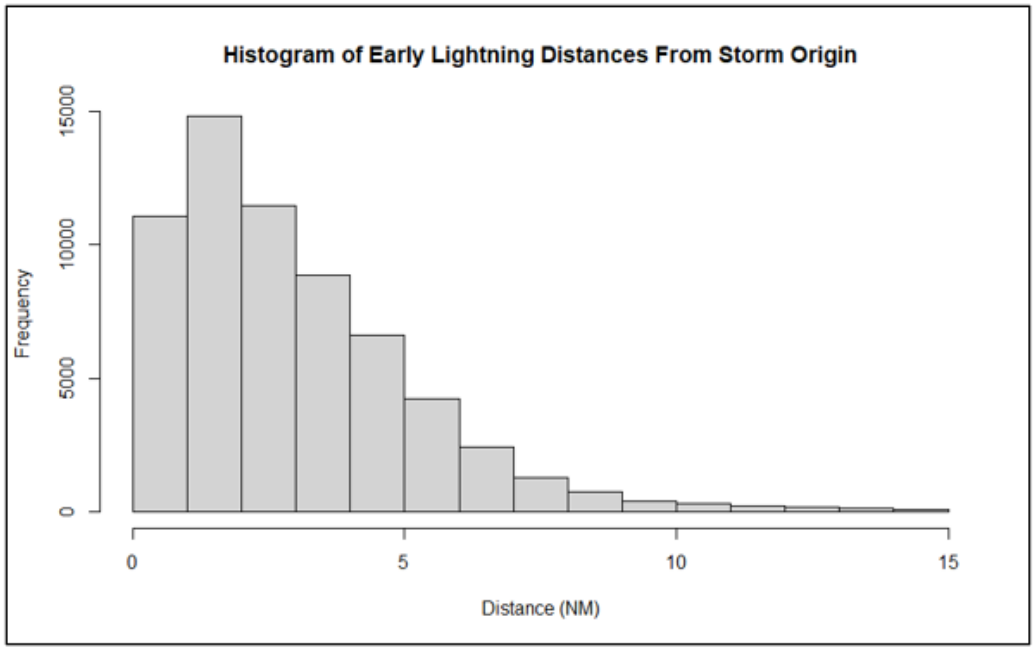
The next stage is to inspect developing, or "early," storm data and calculate distances between lightning events. After grouping the data set by storm ID – using

Clustering of Online Data Streams (CODAS), in our case – we want to filter out all but the first moments of each storm. To this end, we have the option to select either the first n records or those that occur before the first t amount of time. Parameter n allows us finer control over our sample size, ensuring we always have a few points to look at for each storm; however, we run the risk of capturing more extreme time differences than we desire, as a macrocluster may potentially begin with a very small number of points separated by several minutes of time. Parameter t runs the risk of biasing the sample toward storms that have a large point count in the first few seconds; however, we are by definition guaranteed to only capture the early moments of a macrocluster, and thus results should be more representative of the true distribution of lightning in early storms.

We choose to test both approaches in our study. For the time-cutoff approach we use $t = 30$ seconds, and for the other approach we select the first $n = 10$ records from each storm. Figure 4.1 presents histograms of distance to first lightning (a.k.a. "storm origin"). The results are surprising; 27.1% of lightning events are occurring beyond 4 NM. Further, the results include distances greater than 20 NM, with a maximum encountered distance of 101.4 NM. Though they are vanishingly rare (approximately 0.3% beyond 20 NM), the presence of such large values nevertheless appears indicative of an issue with the data or the process.



(a) Full histogram



(b) Zoomed in

Figure 4.1. Example histogram of early lightning distances from storm origin. This is the histogram produced when examining the first 10 records from each storm. There are 211 observations larger than 20 NM. The largest distance encountered is 101.4 NM.

4.2 The Merging Storms Problem

Further experimentation reveals that some macroclusters in the data represent merged storms. When storms collide with each other, CODAS merges them into a single macrocluster. Figure 4.2 provides examples of clusters merging in the lightning data. All microclusters in Figure 4.2a are considered by CODAS to be part of the same macrocluster; the same is true for Figures 4.2b and 4.2c. To explain how the CODAS algorithm handles these events, we take Figure 4.2a as an example. In the first image, we clearly have two small clusters separated by a large distance. At this early stage, the lightning activity in the north and south are represented by separate macroclusters, and their independence persists through the second and the third images. By the fourth image, however, the storms move close enough that the microclusters they generate connect to each other; thus, we no longer have two subgraphs, but one. Because storm IDs are assigned only after all the data have been processed and the macroclusters have fully formed, the lightning events from the first three images in Figure 4.2a retroactively become part of this single larger macrocluster.

When two (or more) such merged storms begin at a similar enough time (i.e. the difference between their start times is smaller than our early-storm cutoff n or t), we end up calculating distances from the origin of one storm to each point of *each* storm. Figure 4.3 provides a notional example stripped down to two dimensions (1 in space and 1 in time) for illustration purposes; Figure 4.4 is provided from MERLIN data for comparison. In Figure 4.3, the two storms merge into one macrocluster at about time 17. When we retrieve this macrocluster and filter to the first 10 points, however, we only see the portion to the left of the red line. If we then simply calculate the distance of each lightning event from the first event, we get the behavior in Figure 4.3a; we prefer the behavior in Figure 4.3b, where each storm is handled separately.

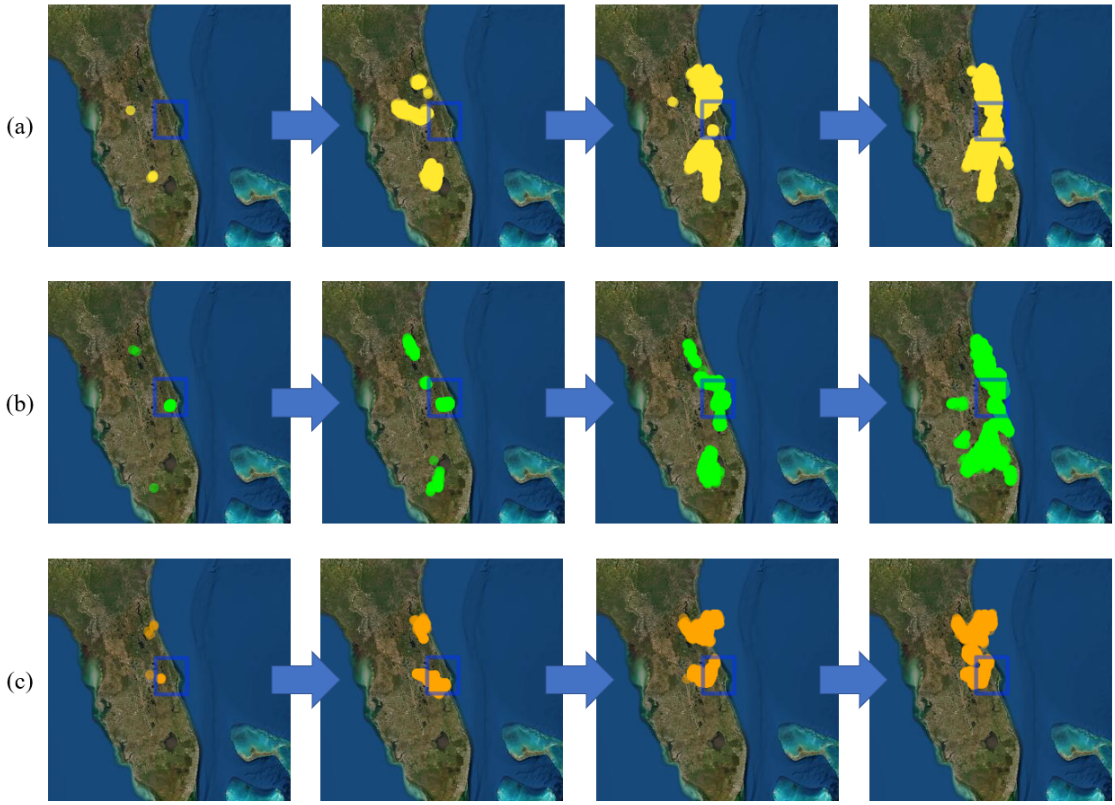
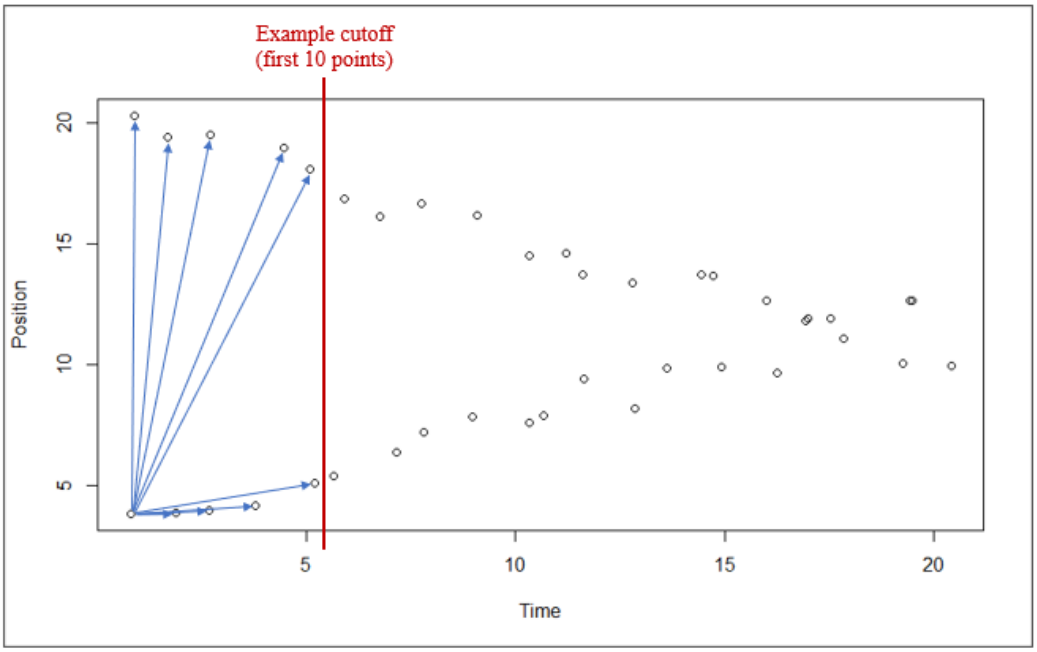


Figure 4.2. Examples of storms merging over time. For each subfigure (each row of images), CODAS has assigned all visible data to the same macrocluster. The blue box indicating MERLIN sensors, included for reference, measures 45.1 NM by 52.9 NM.

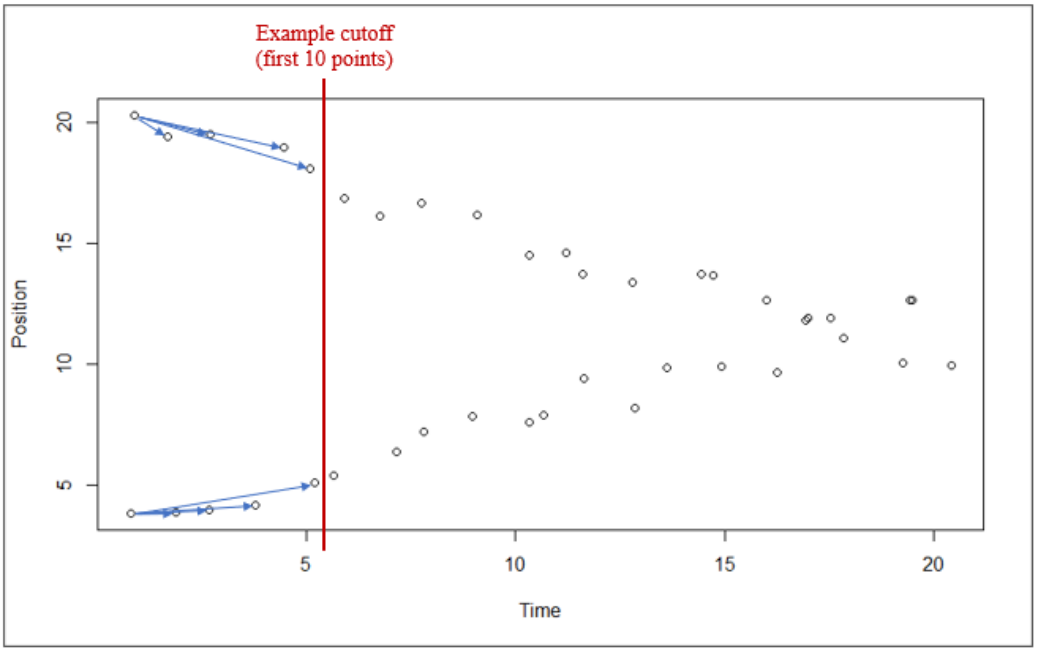
To separate merged storms and achieve the behavior in Figure 4.3b, we implement a second stage of clustering, or "subclustering." Normally, it is difficult to subcluster merged macroclusters without simply recreating the problem; here, however, it works to our advantage that we are studying only early storms. By filtering to the first few records or seconds of a storm, we create a data space with multiple visibly separate clusters, as in the leftmost images in Figure 4.2 or the left side of the red lines in Figure 4.3. Typical clustering algorithms can readily separate clusters such as these.

4.2.1 Hierarchical Subclustering

We begin by applying hierarchical clustering (described in greater depth by Murtagh and Contreras, 2012). This method initially treats each data point as its own cluster,



(a) Problem behavior



(b) Desired behavior

Figure 4.3. Notional 2D illustration of merged storms. Axes are unitless. The blue arrows indicate distances being calculated. The behavior in (a) occurs by default. By applying clustering to only the points on the left of the red line, we can achieve the desired behavior in (b).

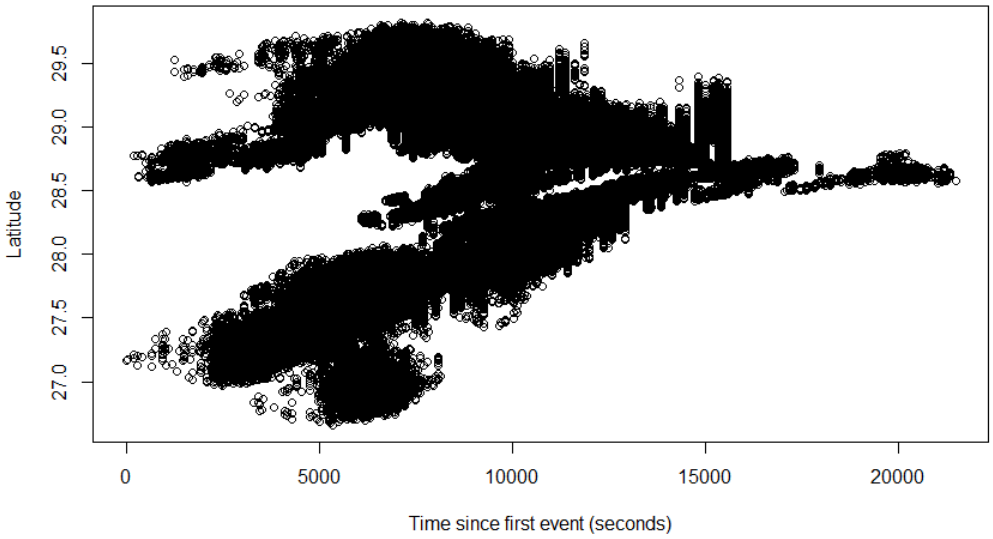


Figure 4.4. 2D example of merged storms from MERLIN data. This CODAS macrocluster, plotted in two dimensions with longitude data omitted, occurs on 11 June 2019 from 16:21:23 to 22:19:27 EDT and spans a maximum latitude difference of 3.16 degrees, or 189 NM.

then iteratively merges the two nearest clusters into one, maintaining knowledge of the members of each merge operation. The final product is a dendrogram showing how each data point and subcluster is related, as in Figure 4.5, where data points that are more similar are connected by smaller subtrees. We can "cut" this tree at any "height" h, such that we re-obtain subclusters that are separated by a distance greater than h. To handle merging storms, we generally want to cut the tree at a height that splits long vertical gaps in the dendrogram. For instance, in Figure 4.5, $h = 20$ is a sensible cut, as it separates three vertical leaders that are significantly longer than any below them, representing a sudden increase in the distances between subclusters. If we cut too high, we merge storms that should remain separate; if we cut too low, we separate lightning events belonging to the same storm.

We desire to find a value for h that works well for the lightning data set. The value must be sufficiently large to avoid truncating the distribution of lightning distances,

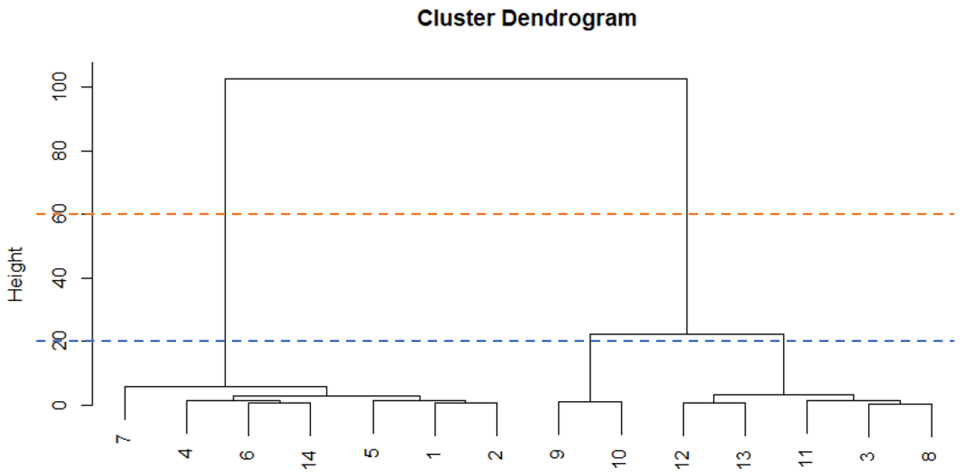
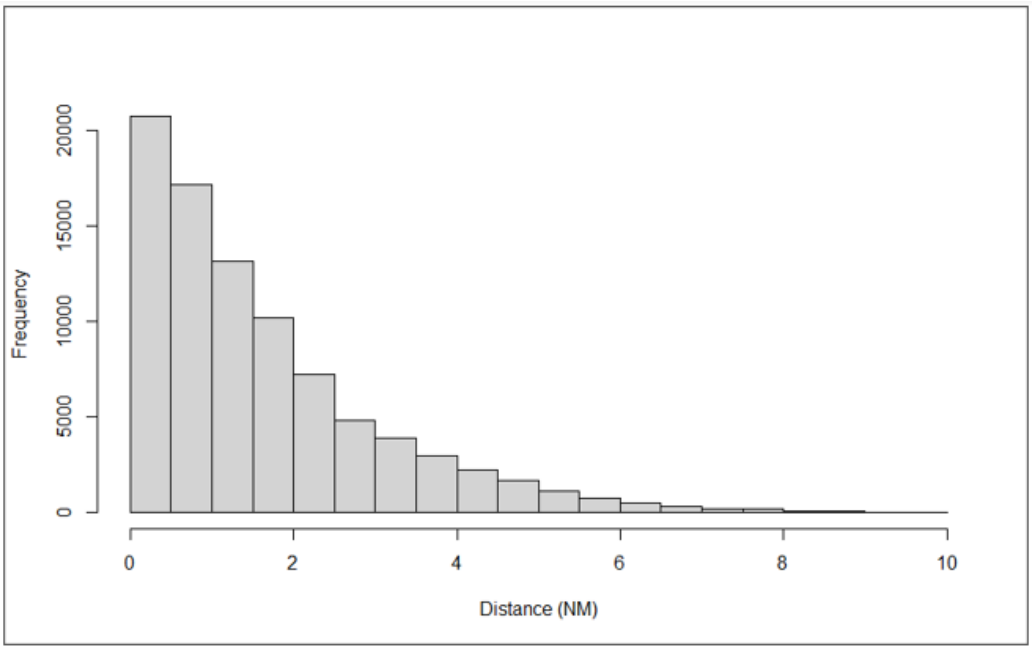
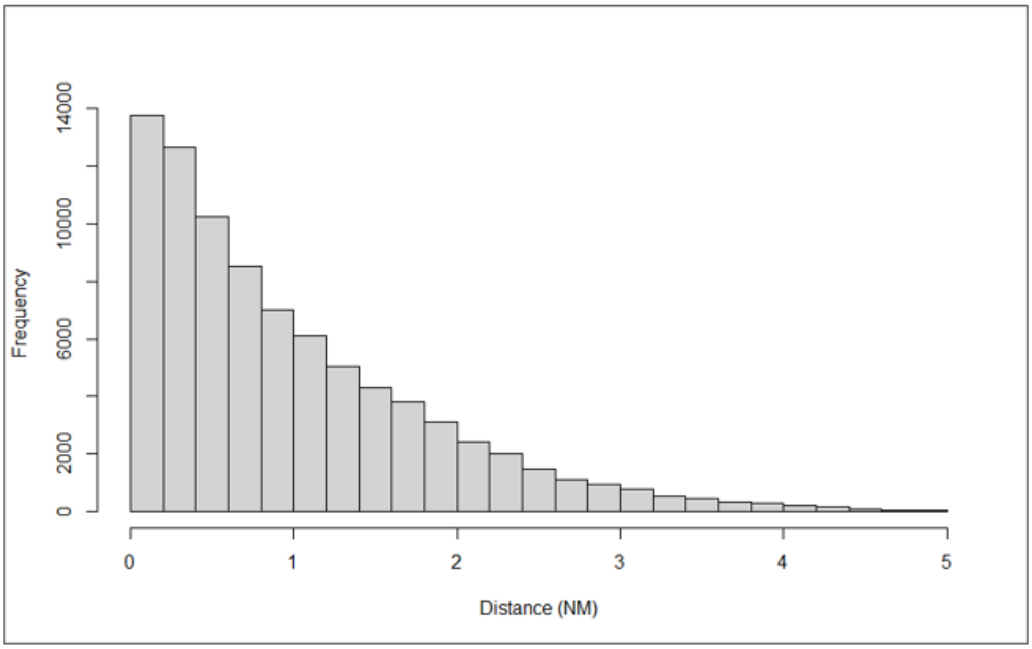


Figure 4.5. Example of a dendrogram resulting from hierarchical subclustering of lightning data. Tree height is measured in NM. The numbered "leaves" along the bottom represent each of 14 lightning events. With the blue cutoff $h = 20$, we obtain three subclusters, two of which are separated by just over 20 NM. With the orange cutoff $h = 60$, we obtain two subclusters separated by just over 100 NM.

but it must be small enough to distinguish between distinct storms that later merge. Figure 4.6 shows two histograms over the distance to the storm origin, where storms are identified using CODAS clustering and hierarchical subclustering. In Figure 4.6a ($h = 10$) the distances range over [0 NM, 9.99 NM], and the the percentage of lightning beyond 4 NM is 8.17%. By contrast, in 4.6b ($h = 5$) the range of distances is [0 NM, 5.00 NM], and the percentage beyond 4 NM is 0.684%. Nevertheless, Figures 4.6a and 4.6b appear to have similarly shaped distributions. Our preliminary experiments reveal that hierarchical clustering generally forms this shape within the bounds of any cutoff h we apply. This indicates that rather than allowing the data to show us the distribution of early lightning events, we are instead controlling the distribution with our selection of h. These preliminary results allow us to conclude that no single cutoff parameter will work for all storms in the data set.



(a) $t = 30$ sec, $h = 10$



(b) $t = 30$ sec, $h = 5$

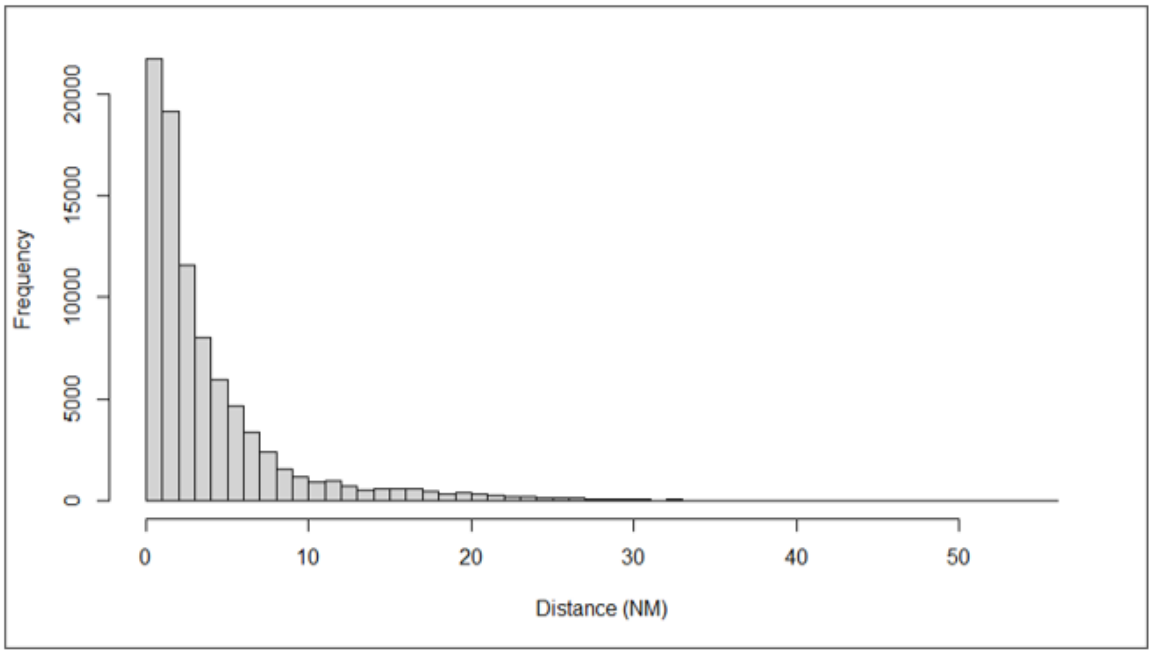
Figure 4.6. Histograms of early lightning distances from storm origin, using time cutoffs and hierarchical subclustering. Parameters include time cutoff t and cluster height cutoff h.

4.2.2 CODAS Subclustering

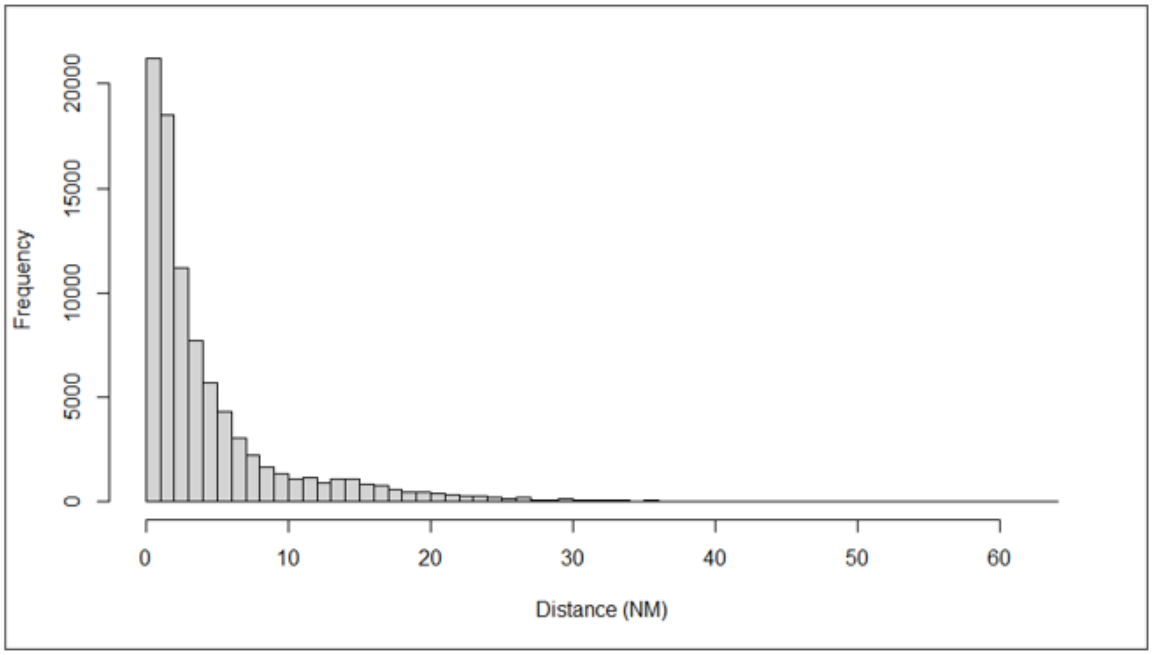
We next consider whether CODAS might perform the desired subclustering. For this application, the time differences involved are small enough as to be irrelevant; thus, we apply CODAS in two dimensions, examining location only. We re-use our existing implementation of CODAS; however, as this implementation is designed for three dimensions, we accomplish two-dimensional clustering by producing a copy of the early storm data with the time values flattened (set equal to 1), and then applying CODAS to this set. We use outlier density threshold $D = 5$ and vary our CODAS radius R and our early storm cutoff to produce Figures 4.7 and 4.8. These appear better than Figure 4.1, but they are still heavier-tailed than expected; estimates for the percentage of lightning beyond 4 NM range from 24.3% - 33.4%, depending on whether we use the first 10 events of the storm or first 30 seconds of data, and (to a lesser effect) the microcluster radius.

Visual inspection of the most extreme results reveals macroclusters such as those in Figure 4.9. Though the storm merging issue appears to have been resolved, these filtered storms are still too large to be representative of the early moments of a new storm. Rather, Figure 4.9b is what one would expect a storm to look like in its final moments coming from the western coast of Florida (Roeder, 2021). Similarly, Figure 4.10 presents an example of a large "early storm" and the surrounding raw data, plotted with a time axis; the events in question occur over a time interval of only 0.411 seconds and are neither preceded nor immediately succeeded by any significant lightning activity. Clusters such as in Figures 4.9 and 4.10 are most likely the result of a well-developed storm entering from outside of MERLIN's "local" range – i.e., the effective range of its network of Total Lightning Sensor Model-200 (TLS-200) units.

For the purposes of this analysis, we apply additional filters to remove situations likely to result in data that are clearly not representative of early storms. We first

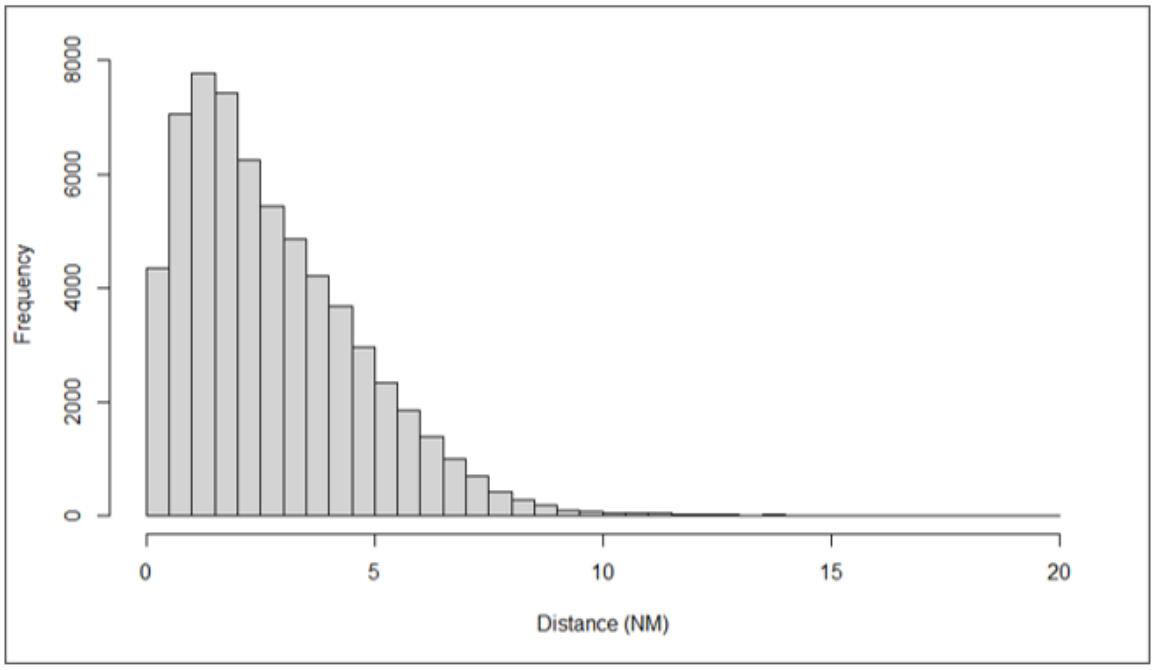


(a) $t = 30$ sec, $R = 5$ NM

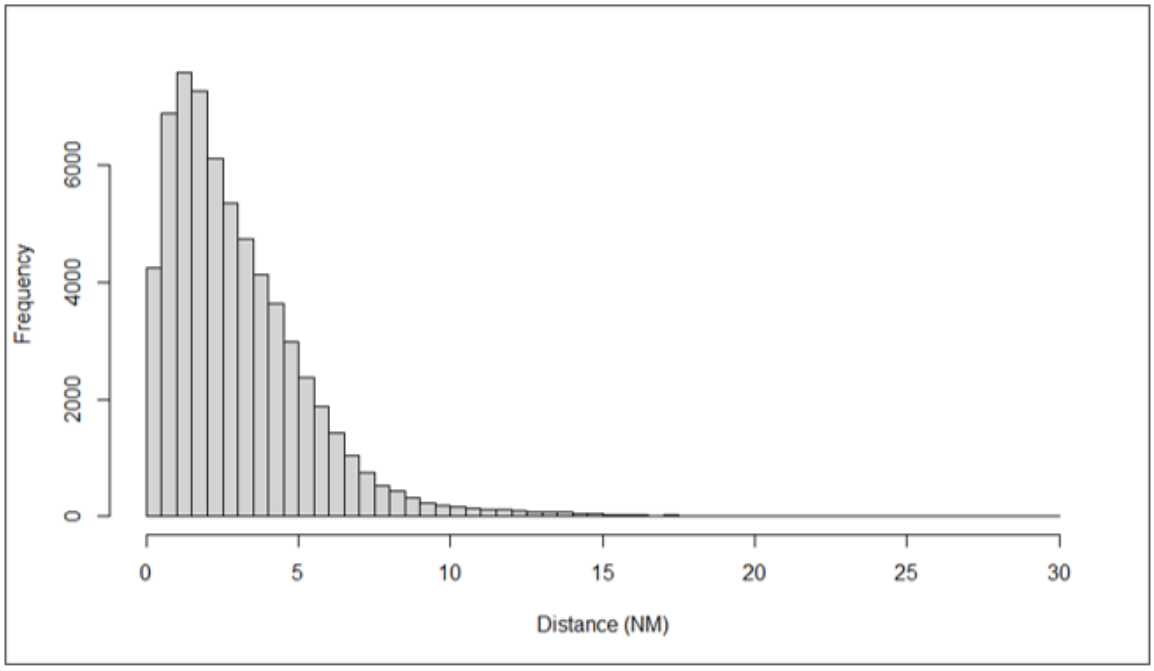


(b) $t = 30$ sec, $R = 10$ NM

Figure 4.7. Histograms of early lightning distances from storm origin, using time cutoffs and CODAS subclustering. Parameters include time cutoff t and CODAS microcluster radius R.

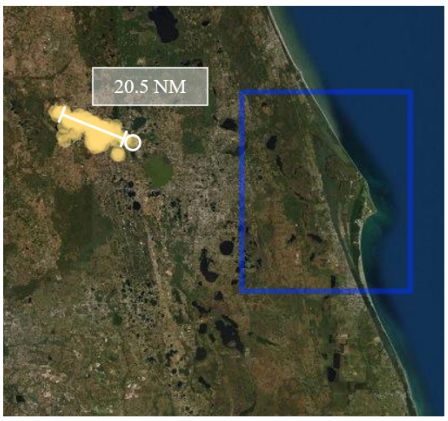


(a) $n = 10$, $R = 5$ NM

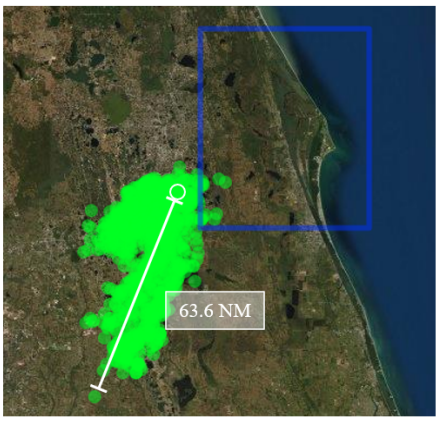


(b) $n = 10$, $R = 10$ NM

Figure 4.8. Histograms of early lightning distances from storm origin, using point-count cutoffs and CODAS subclustering. Parameters include point-count cutoff n and CODAS microcluster radius R.

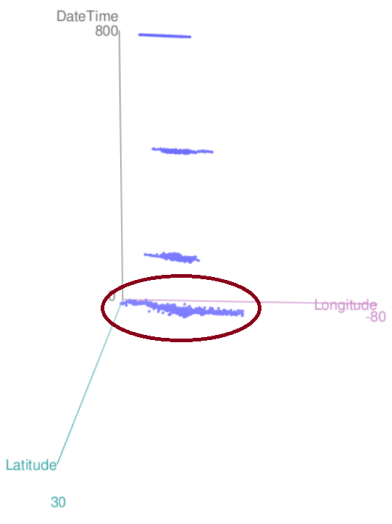


(a) Lightning events on 12 Aug 2019, 21:45:39.1 – 21:45:39.5

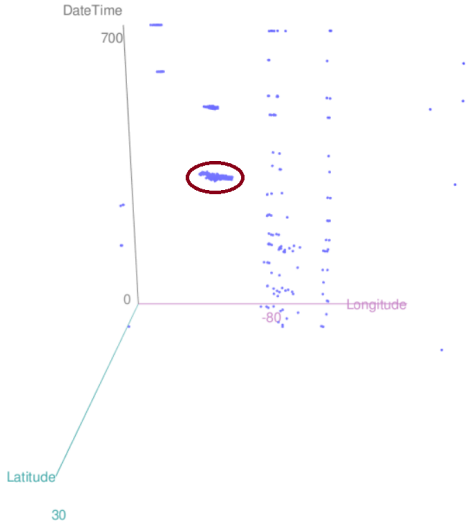


(b) Lightning events on 30 Jul 2019, 00:00:02 – 00:00:31

Figure 4.9. Examples of early storms with subclustering. The origin of each storm is marked with a white circle. (a) contains 89 lightning events occurring within 0.4 seconds. (b) contains 1,305 points occurring within 29 seconds. The blue box indicating MERLIN sensors is included for reference, and measures 45.1 NM by 52.9 NM.



(a) A single microcluster.



(b) Contextualized within surrounding data.

Figure 4.10. Three-dimensional plot of a large macrocluster and surrounding raw data. Figure (a) represents the raw lightning events contained within a single macrocluster, with the first 30 seconds circled in red. The circled portion contains 1,874 points occurring over a total time interval of 0.411 seconds. Figure (b) displays a time interval centered on the same data and includes all surrounding data from the raw set; the cluster in question is not preceded by any significant lightning activity within 5 NM.

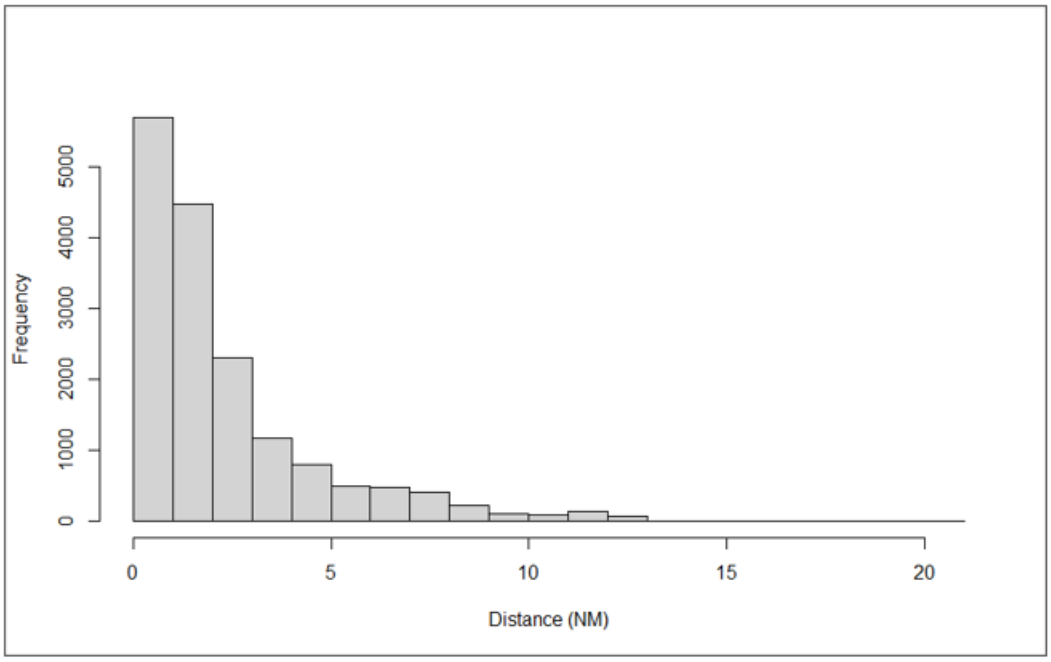
reduce the early storm data to consider only macroclusters that are not preceded by any lightning activity within the past 15 minutes. We additionally filter to macroclusters beginning between 1400 and 1800 UTC, as this is the period of the day during which storms are most likely to form within MERLIN's local sensor range and least likely to encroach from outside of it (Roeder, 2021). With these additional filters in place, we obtain the histogram in Figure 4.11a.

The tail in Figure 4.11a is still too heavy; 12.0% of distances are beyond 5 NM. However, we note that the percentiles we have obtained so far, particularly in Figures 4.7, 4.8, and 4.11a, are reminiscent of those from Cox (1999) and Parsons (2000). Their technique of measuring from storm centroids yields large distances and does not match with how warnings are issued, and we posit that we may be committing a similar error by measuring from storm origins. Roeder (2021) confirms this, and suggests that we instead calculate the distance between successive events. Weather squadrons are concerned primarily with the most recent lightning in a storm; the origin can quickly become irrelevant as the storm moves, even during the first few seconds. The distance between successive events tracks recent changes, and therefore better aligns with warning procedures. Note that this serves a similar purpose to the concept of measuring the distance from a pre-existing lightning area for a well-developed storm.

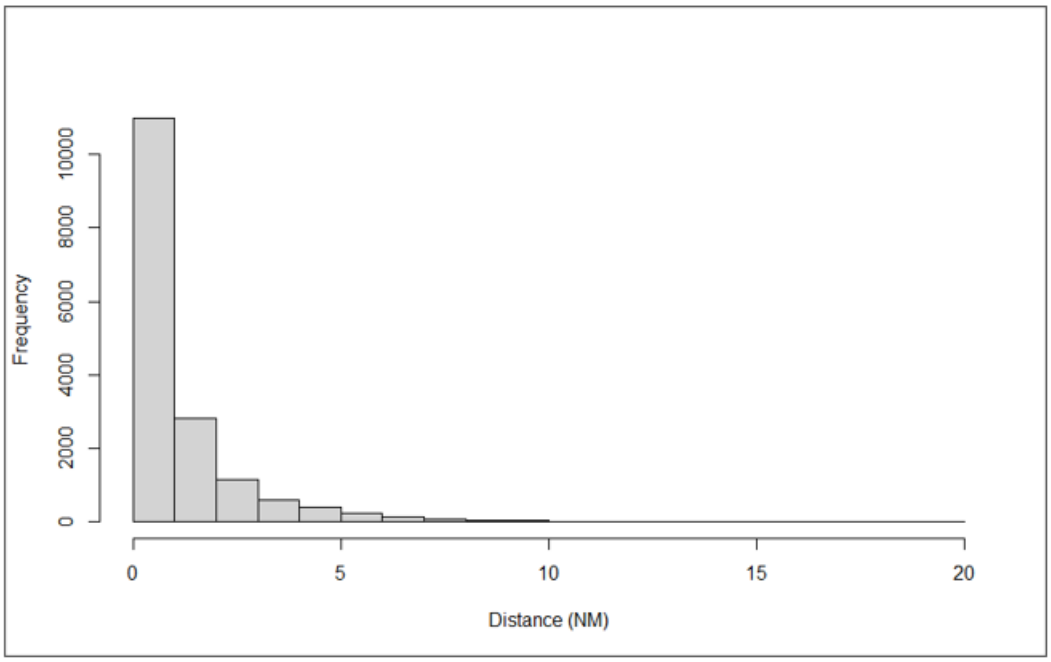
We therefore re-use our parameters and filters from Figure 4.11a, but change our calculations to measure the distance between successive events, producing Figure 4.11b. This figure represents 63,819 distance values calculated from 71,843 lightning events (no distance is calculated for the first event in each of 8,024 storms). The median distance in 4.11b is 0.529 NM; 5.57% of values exceed 4 NM, and 3.24% exceed 5 NM. For comparison, recall that data from Sanderson (2019, p. 67) follows a Weibull distribution that indicates that 6.01% of lightning events occur beyond

4 NM of pre-existing lightning areas and 3.39% occur beyond 5 NM. Holland (2021) also finds a Weibull distribution for lightning distances from a pre-existing area; her distribution indicates 7.32% of lightning falls beyond 4 NM and 3.86% falls beyond 5 NM. As our percentiles fall within those of both Sanderson (2019) and Holland (2021), we can conclude that their recommended warning radius of 4 NM for well-developed storms is applicable to developing storms as well.

The distribution that best fits our early lightning data is also a Weibull, shown in Figure 4.12. 4.02% of this distribution exceeds 4 NM, and 2.11% exceeds 5 NM, further supporting our conclusions. Table 4.1 presents a comparison of the statistics for our distribution with those of Sanderson (2019) and Holland (2021). In making this comparison, we must keep in mind that we are representing qualitatively different phenomena. While their distributions measure the probability of lightning outside of a pre-existing lightning area throughout the entire storm life cycle, our distribution focuses exclusively on early lightning. Nevertheless, while there may be many behavioral similarities and differences compared to lightning later in the storm, we believe that our analysis provides sufficient evidence to conclude that the probability of early lightning striking more than 4 NM from the previous lightning event does not exceed the total probability of lightning striking more than 4 NM beyond a pre-existing lightning area. However, this conclusion comes with the caveat that early storm behavior, like mature storms, can deviate greatly from the storm origin. It is therefore important to base warnings on the most recent lightning information available. In this way, our use of the previous lightning event may be seen as something akin to a proxy for pre-existing lightning areas. Additional research might be able to show an even tighter radius for developing storms using a better proxy measure.



(a) $t = 30$ sec, $R = 5$ NM, distances calculated from first lightning event



(b) $t = 30$ sec, $R = 5$ NM, distances calculated between successive lightning events

Figure 4.11. Histograms of early lightning distances, using time cutoffs, CODAS subclustering, and additional filters. As compared with Figure 4.7a, these figures result from filtering to include only macroclusters that begin between 1400 and 1800 UTC and are not preceded by any lightning activity within 15 minutes. In (a), 17.1% of distances exceed 4 NM and 12.2% exceed 5 NM, with a median of 1.51 NM. In (b), 5.57% of distances exceed 4 NM and 3.24% exceed 5 NM, with a median of 0.529 NM.

Table 4.1. Comparison of Weibull distributions of lightning events.

	Early Storms	Mature Storms	
		Sanderson	Holland
Shape	0.818	0.833	0.98
Scale	0.961	1.16	1.50
Mean	1.08	1.28	1.48
Median	0.613	0.782	0.993
% >4 NM	4.02	6.01	7.32
% >5 NM	2.11	3.39	3.86

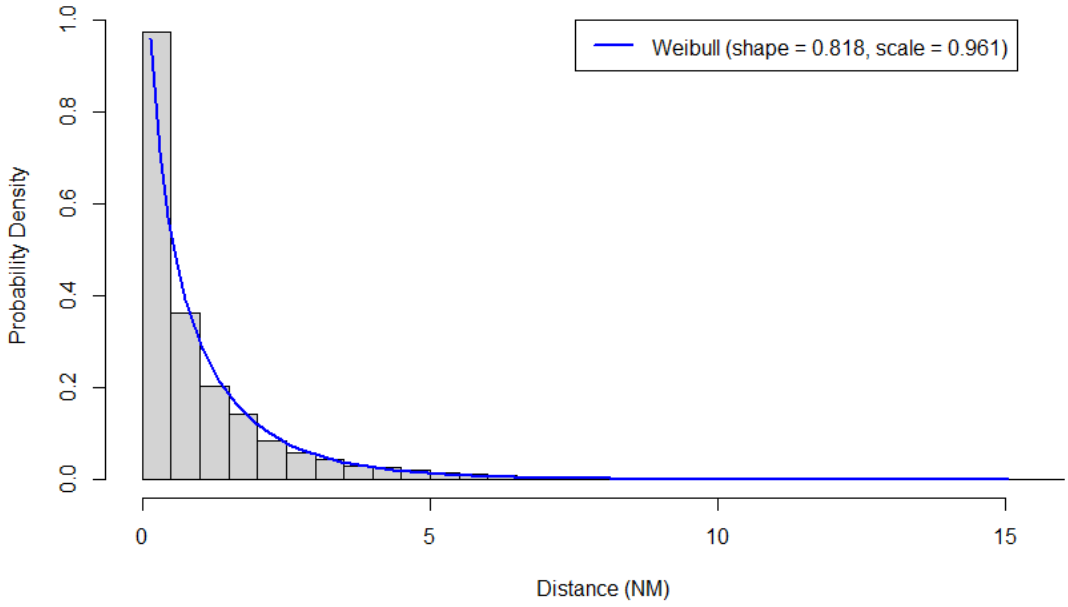


Figure 4.12. Weibull distribution for early lightning distances, using time cutoffs, CODAS subclustering, and additional filters. This distribution has mean = 1.08 NM and median = 0.613 NM. 4.02% of distances exceed 4 NM, and 2.11% exeed 5 NM.

4.3 Summary

Initial calculations of the distances of early lightning events from their storm origins yield distances as large as 101.4 NM. Some of these distances, including the largest extremes, result from initially distant storms that collide and merge into one storm later in their lifespans. Hierarchical clustering on early storm data fails to handle this properly, as it controls the shape of the resulting histograms rather than indicating the true distribution of the data. CODAS appears to perform better based on a cursory visual inspection of the resulting clusters. It still yields much heavier-tailed distributions than anticipated, however; in the worst case, 33.4% of events are more than 4 NM from their storm origins, and the maximum distance encountered is 63.6 NM. We hypothesize that the extreme values result from well-developed storms entering MERLIN's TLS-200 sensor range. To eliminate situations where data are less likely to accurately represent early storms, we filter to include only macroclusters that begin between 1400 and 1800 UTC and are not preceded by any lightning activity within the past 15 minutes. We also change our calculations to measure the distance between successive flashes, more closely aligning with the way lightning warnings are issued. With these changes in place, we filter to the first 30 seconds of each storm and perform CODAS subclustering with $R = 5$ NM. This results in percentiles that are not only smaller than anything else we have obtained, but also smaller than the percentiles calculated by Sanderson (2019, p. 67) for lightning distances from pre-existing storm areas; only 5.57% of our event distances exceed 4 NM, and only 3.24% exceed 5 NM.

V. Conclusions and Recommendations

5.1 Conclusions

5.1.1 Storm Identification

It is complex to build a practical algorithm for identifying storms within a data set containing millions of records, such as that obtained from the Mesoscale Eastern Range Lightning Information System (MERLIN). Most algorithms for this task require balancing speed against accuracy, and the balancing process can require prohibitive amounts of time. The pairwise comparison algorithm described in Section 3.2 is the most conceptually straightforward for this application. However, its input parameters require us to tell it beforehand how far apart lightning events should be in time and space; this is information we want the algorithm to tell us, not the other way around. It is conceptually possible to work around this challenge with sensitivity analysis, but this requires testing the algorithm with a wide variety of parameter values, visually inspecting the resulting storm clusters for accuracy each time. This is not practical for a large data set, as each combination of parameters we test requires weeks of processing time followed by manual validation.

The grid algorithm proposed in Section 3.3 can alleviate the run time issue, but it introduces error and exacerbates the challenge of handling parameter sensitivity. Where the pairwise comparison algorithm is sensitive to at least two of its input parameters, the grid algorithm is sensitive to four. This represents an exponential increase in the amount of testing required to achieve quality results, which again renders the algorithm impractical for this study. Note, however, that once appropriate parameters are found, they should be applicable to other studies on similar MERLIN datasets, at which point the algorithm's expected run time benefits may be more fully realized. Development and testing of this algorithm is left for future work.

The Clustering of Online Data Streams (CODAS) algorithm developed by Hyde and Angelov (2015) and presented in Section 3.4 resolves the issue of parameter sensitivity. It is robust: rather than requiring precise optimization of its two parameters, it produces quality results as long as said parameters are each within a reasonable range. CODAS's run time of three to five days on the MERLIN data set is middling compared with the other algorithms in this study – faster than the pairwise comparison algorithm, and significantly slower than the grid algorithm. Because we can run it once with parameters selected by intuition, this run time becomes practical for our purposes. This is the algorithm we have used to process our MERLIN data and identify storms.

5.1.2 Lightning Behavior

Upon filtering MERLIN data to study early storms, we calculate distances of lightning events from storm origins and find that some are over an order of magnitude larger than anticipated. In the worst case, we encounter a maximum distance of 101.4 NM occurring within the first 30 seconds of a storm. We find many of the most extreme cases to be due to storm merging, where storms that begin in separate locations later collide, causing CODAS to identify them collectively as one storm (see Figures 4.3 and 4.2). When we calculate distances for a merged storm, they are then all in reference to the starting point of only one of the original constituent storms. We solve this by performing a second round of clustering, this time on the filtered data representing early storms.

Hierarchical clustering with a height cutoff of $h = 10$ yields a histogram that looks promising, but experimentation reveals an issue. When we vary h, the resulting histogram changes such that approximately the same distribution shape fits within the range 0 to h. This indicates that the algorithm is controlling the distribution

we see, rather than showing us the true shape of the data. This could be corrected with sensitivity analysis, but that is a time-intensive process. We set hierarchical clustering aside.

We next revisit CODAS, applying it this time to the filtered early storm data. The resulting histograms represent an improvement over Figure 4.1, but are still heavier-tailed than expected (see Figures 4.7 and 4.8 for examples). Visuals of some of the clusters, particularly those related to Figure 4.7b, are surprising; the early storm in Figure 4.9b is 30 seconds old, contains 1,305 lightning events, and spans a maximum distance from origin of 63.6 NM. This extreme is again much larger than anticipated, and is likely due to mature storms that enter from outside the range of MERLIN sensors. We apply filters to exclude such situations, filtering to only storms that begin between 1400 and 1800 UTC and have no preceding lightning activity within the past 15 minutes.

We additionally switch to measuring distances between successive lightning events, which more closely aligns with the way lightning warnings are issued. These changes result in a distribution for which only 5.57% of lightning distances in developing storms exceed 4 NM, and only 3.24% exceed 5 NM. These percentages differ by only 2.33% and are smaller than those produced from Sanderson's distribution for well-developed storms; thus, we conclude that 4 NM represents a reasonable reduction of the standard lightning warning area.

5.2 Recommendations and Future Work

Despite the large size of the data set used in this thesis, it represents only May-September 2019 in Central Florida. This is lightning season in the region of greatest lightning activity in the United States (Holle et al., 2016), and is therefore akin to a worst-case scenario, but it is still only one summer in one region of the country.

We recommend expanding the sample to include all seasons and a greater number of years for locations across the country. The winter season is of particular interest, as its weather patterns are a substantial departure from those of the summer months (Roeder, 2021). To support research on this scale, it may be necessary to refine CODAS or to find or develop a faster clustering technique applicable to enterprise level computing.

Regarding results for our data set: our first application of CODAS yields macroclusters that are, as we desire, closely packed in space and time. However, we do not perform a sensitivity analysis or a complete validation of the algorithm's output. Thus, though our output is useful, it is unlikely to be truly optimized; it may be possible to further improve our parameter settings.

It may also be possible to improve CODAS's `OutlierHandler` function, which deviates slightly from the definition of a microcluster. This issue, described in Appendix A, does not appear to detract from the algorithm's performance, but resolving it may make CODAS friendlier to the user.

Further, in studying early storms, we encounter situations that clearly do not represent early lightning activity but are erroneously identified as such, likely as a result of well-developed storms encroaching from outside of sensor range. In the interest of time, we simply filter out situations likely to produce such issues. For future study, we recommend a more thorough investigation, so as to determine the root cause and handle the problem more directly.

Finally, the grid algorithm (Algorithm 3.2) currently exists only as a concept. This algorithm appears promising as a heuristic; though it is expected to be difficult to find appropriate parameters for most use cases, it is projected to be significantly faster than most other comparable algorithms once said parameters are optimized. Further, the grid algorithm has an interesting advantage over CODAS for the case

of studying developing storms: since it assigns storm IDs "on the fly" as it iterates through time, it avoids CODAS's storm merging problem entirely. If fully developed, it may be worth running the grid algorithm at a high "resolution" to see how its results compare with our two-stage application of CODAS.

Appendix A. CODAS `OutlierHandler` Considerations

In Algorithm 3.3, when looking for points to add to a microcluster, `OutlierHandler` finds the number of "neighbors" within distance R of each outlier. An outlier must have at least D neighbors for a microcluster to form. This method comes with the caveat that there is an edge case that it cannot handle properly. When a group of D or more outliers is close enough to fit in a microcluster, but they are all positioned far from the center as in Figure A.1, they may be up to $2R$ away from each other. Thus, they have fewer neighbors than fit in the sphere, and a microcluster might not form. This means that CODAS's clustering is slightly more restrictive in practice than indicated by the microcluster definition from the previous section. Despite this, however, the existing CODAS algorithm is no less effective at performing dynamic clustering of data streams based on point density. The behavior obtained is different than expected, but not inherently worse, and therefore we leave the resolution of this issue for future work.

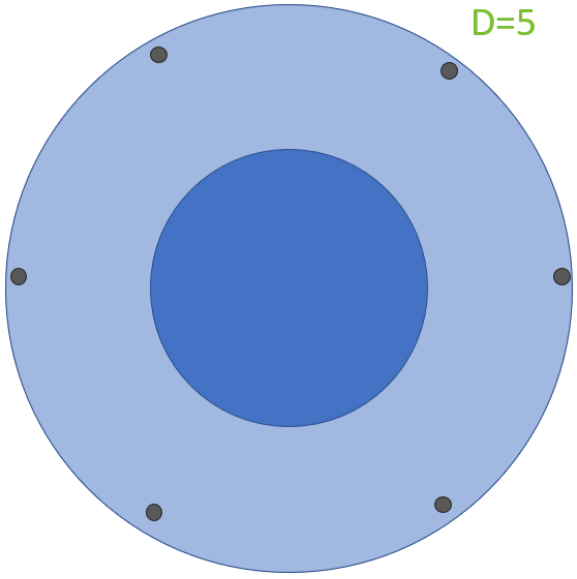


Figure A.1. Edge case not handled by CODAS. Each black point has exactly two "neighbors" for clustering purposes, despite that all six fit in the circle. No points have at least five neighbors, so no microcluster forms.

Appendix B. Pairwise Comparison Algorithm Code

B.1 Wrapper (pairwise_wrapper.R)

```
### Wrapper for Pairwise Comparison Algorithm ###
### Capt Erick Tello, 2021                    ###

library(dplyr) #Used to select/aggregate data
library(Rcpp)  #Used to run pairwise.cpp

# Set numerical format for POSIXct DateTime conversion
options(digits = 13)
options(digits.secs = 3)

# Get file list
fList = list.files(pattern = "*.TL.csv")

# Get algorithm source code
sourceCpp("pairwise.cpp")

# Iterate through files, run pairwise algorithm on each
for (i in fList) {
  t1 = read.csv(i, colClasses=c(DateTime="character"))
  temp = mutate(t1, DateTime = as.numeric(
    as.POSIXct(DateTime, format = "%Y-%m-%d %H:%M:%OS", tz="UTC")))

  start_time <- Sys.time()
  tempGap = round(timeGap(temp),3)
  end_time <- Sys.time()

  print(end_time - start_time)

  t1$TimeGap = tempGap
  write.csv(t1, paste0("WithTGs/g",i), row.names=FALSE)
}
```

B.2 Algorithm (pairwise.cpp)

```
/// Pairwise Comparison Algorithm ///
/// Capt Erick Tello, 2021        ///

// Set up Rcpp for compatibility with R
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
DataFrame timeGap(DataFrame df){
  // INITIALIZE
  int C = 30;
  int D = 5;
  NumericVector dateTime = df[0];
  NumericVector latitude = df[1];
  NumericVector longitude = df[2];
  IntegerVector outIdx(dateTime.size());
  NumericVector outTime(dateTime.size());
  NumericVector outDist(dateTime.size());
  DataFrame out;
  IntegerVector r(dateTime.size());
  std::iota(r.begin(),r.end(),0);
  IntegerVector shortIdx(C);
  LogicalVector l1(dateTime.size());
  LogicalVector l2(C);
  NumericVector x(C);
  NumericVector y(C);
  NumericVector z(C);
  NumericVector yGap(C);
  NumericVector zGap(C);
  NumericVector gap(C);
  double time = 0;
  int idx = 0;

  // Main body
  for (int i = 0; i < dateTime.size(); i++) {
    // Get a collection of C previous records and their indices
    l1 = (r < i) * (r >= i - C);
    x = dateTime[l1];
    y = latitude[l1];
    z = longitude[l1];
    shortIdx = r[l1];

    // Calculate Euclidean distance between each point and point i
    yGap = (y - latitude[i]) / 0.01667;
    zGap = (z - longitude[i]) / 0.0193;
    gap = sqrt(yGap*yGap + zGap*zGap);

    // Filter to points within D nautical miles
    l2 = (gap <= D);
```

```
    x = x[l2];
    y = y[l2];
    z = z[l2];
    gap = gap[l2];
    shortIdx = shortIdx[l2];

    // Find the most recent remaining record
    for (int j = 0; j < x.size(); j++) {
      if(x[j] > time) {
        time = x[j];
        idx = j;
      }
    }

    // Check whether any records were found
    if(time == 0) { //If none, indicate as such in the output
      outIdx[i] = -1;
      outTime[i] = NAN;
      outDist[i] = NAN;
    } else { //If found, append index, time, and time gap
             //  from most recent record to output
      outIdx[i]  = shortIdx[idx] + 1; //Add 1 to account for zero-indexing
      outTime[i] = dateTime[i] - time;
      outDist[i] = gap[idx];
    }

    time = 0; //Reset time for next loop
  }

  out["PredIndex"] = outIdx;
  out["TimeGap"] = outTime;
  out["DistGap"] = outDist;

  return out;
}
```

Appendix C. CODAS Algorithm Code

C.1 Wrapper (CODAS_Wrapper.R)

```
### Wrapper for CODAS algorithm ###
### Capt Erick Tello, 2021      ###

library(dplyr)  #Used to select/aggregate data
library(Rcpp)   #Used to run CODAS.cpp
library(igraph) #Used to find graph components
library(beepr)  #Used to play alert sound

# NOTES:
# - The "fList1" and "fList2" variables must be able to read input data files;
#   change filepaths as necessary.
#   - The "path" argument in list.files() defaults to the current working
#     directory, which you, the user, must set in RStudio before running this
#     script. Use the RStudio file explorer or the setwd() command.
#   - The data file(s) beginning with "nd" have had their dates converted to a
#     numerical format (seconds since January 1, 1970) for processing in Rcpp.
#     This saves time over pre-processing them each time the script is run.
#     However, we then want to write outputs to files with text dates, hence
#     the input to fList2.

overallStartTime = Sys.time()
#CEDAS parameters
#Notes on coordinates:
# - Avg seq diffs for time, lat, long: [0.1743, 0.0617, 0.0537]
# - (1 nmi = 0.01667, lat = 0.0193 long)
# - Min Lat: 25.9034
# - Min Long: -84.071
radius = 1           # CEDAS microC radius (do not adjust this)
minThreshold = 5     # Min microC threshold
timeThreshold = 900  # Time threshold in seconds (adjust this)
distThreshold = 5    # Distance threshold in nmi (adjust this)
outputFolder = "CEDAS 5nmi 30min (all data)"

# Set DateTime precision
options(digits = 13)
options(digits.secs = 3)

# Get file lists
fList1 = list.files(path = "./POSIXct/", pattern = "*.TL.csv")
fList2 = list.files(pattern = "*.TL.csv")

# # Subset files if necessary
# fList1 = fList1[112:135]
# fList2 = fList2[112:135]

# Get CODAS and associated functions
sourceCpp("CODAS.cpp")
```

```
# Initialize CODAS data structure
output = list(data.frame(DateTime = numeric(0),
                         Latitude = numeric(0),
                         Longitude = numeric(0)),
              integer(0),
              integer(0),
              data.frame(DateTime = numeric(0),
                         Latitude = numeric(0),
                         Longitude = numeric(0)),
              integer(0),
              integer(0))
names(output) = c("clusterCenters","clusterCounts","clusterKernels",
                  "outliers","graphL","graphR")
fileCounter = 1L #Progress message helper

# Iterate through lightning data files
for (i in fList1) {
  # Read file and clean up NAs, any formatting issues
  temp = read.csv(paste0("POSIXct/",i))
  temp = na.omit(temp)
  temp$DateTime = as.numeric(temp$DateTime)
  temp$Latitude = as.numeric(temp$Latitude)
  temp$Longitude = as.numeric(temp$Longitude)
  temp = na.omit(temp)

  # Normalize data such that a difference of 1 meets our thresholds:
  temp[,1] = temp[,1] / timeThreshold           # secs -> sec increments
  temp[,2] = temp[,2] / 0.01667 / distThreshold # lat  -> NM -> NM increments
  temp[,3] = temp[,3] / 0.0193  / distThreshold # long -> NM -> NM increments

  # Start timer, run CODAS algorithm
  startTime <- Sys.time()
  output = CODAS(temp,
                 fileCounter,
                 output$clusterCenters,
                 output$clusterCounts,
                 output$clusterKernels,
                 output$outliers,
                 radius,
                 minThreshold,
                 output$graphL,
                 output$graphR)
  endTime <- Sys.time() #End timer

  # Progress message
  message(paste0("CEDAS run time (file ",i,"):"))
  print(endTime - startTime)
  fileCounter = fileCounter + 1L #Progress message helper
}

# Allow for shorter variable references:
```

```
attach(output)

# Fix graph node IDs (off due to Rcpp zero-indexing):
graphL = graphL + 1
graphR = graphR + 1

# Find graph components
clusterGraph = make_undirected_graph(c(rbind(graphL,graphR)),
                                     nrow(clusterCenters))
macroClusters = components(clusterGraph)

# Undo normalization, write clusters to file
ClusterData = data.frame()[1:nrow(clusterCenters),]
ClusterData$DateTime      = clusterCenters$DateTime * timeThreshold
ClusterData$Latitude      = clusterCenters$Latitude * 0.01667 * distThreshold
ClusterData$Longitude     = clusterCenters$Longitude * 0.0193 * distThreshold
ClusterData$ClusterLife     = clusterLives
ClusterData$ClusterCount    = clusterCounts
ClusterData$ClusterKernelCt = clusterKernels
ClusterData$MacroClusterID         = macroClusters$membership
write.csv(ClusterData, paste0(outputFolder,"/ClusterData.csv"), row.names=FALSE)

# Undo normalization, write outliers to file
outliers[,1] = outliers[,1] * timeThreshold
outliers[,2] = outliers[,2] * 0.01667 * distThreshold
outliers[,3] = outliers[,3] * 0.0193  * distThreshold
write.csv(outliers, paste0(outputFolder,"/Outliers.csv"), row.names=FALSE)

# Write graph data to file
GraphData = data.frame()[1:length(graphL),]
GraphData$Left = graphL
GraphData$Right = graphR
write.csv(GraphData, paste0(outputFolder,"/GraphData.csv"), row.names=FALSE)

for (i in 1:length(fList1)) {
# for (i in 1:24) { #Manual looping control for handling reduced file list
  startTime <- Sys.time() #Start timer

  # Read files, clean NAs and any formatting issues
  temp = read.csv(paste0("POSIXct/",fList1[i]))
  temp = na.omit(temp)
  temp$DateTime = as.numeric(temp$DateTime)
  temp$Latitude = as.numeric(temp$Latitude)
  temp$Longitude = as.numeric(temp$Longitude)
  temp = na.omit(temp)
  out = read.csv(fList2[i])
  out = out[complete.cases(out[,1:3]),]
  out$Latitude = as.numeric(out$Latitude)
  out$Longitude = as.numeric(out$Longitude)
  out = out[complete.cases(out[,1:3]),]

  # Normalize data to compare with clusters:
```

```
  temp[,1] = temp[,1] / timeThreshold           # secs -> sec increments
  temp[,2] = temp[,2] / 0.01667 / distThreshold # lat  -> nmi -> nmi increments
  temp[,3] = temp[,3] / 0.0193  / distThreshold # long -> nmi -> nmi increments

  # Assign cluster (storm) IDs
  out$MicroClusterID = MicroMembers(temp,clusterCenters,radius)
  out$MacroClusterID = MacroMembers(out$MicroClusterID,macroClusters$membership)
  temp$MicroClusterID = out$MicroClusterID
  temp$MacroClusterID = out$MacroClusterID

  # Un-normalize data:
  temp[,1] = temp[,1] * timeThreshold           # secs -> sec increments
  temp[,2] = temp[,2] * 0.01667 * distThreshold # lat  -> nmi -> nmi increments
  temp[,3] = temp[,3] * 0.0193  * distThreshold # long -> nmi -> nmi increments

  # Write data with storm IDs to files
  write.csv(out, paste0(outputFolder,"/cd",fList2[i]), row.names=FALSE)
  write.csv(temp, paste0(outputFolder,"/cdpx",fList2[i]), row.names=FALSE)

  endTime <- Sys.time() #End timer

  #Progress message
  message(paste0("Writing to file (cd",fList2[i],"):"))
  print(endTime - startTime)
}

detach(output) #Undo variable attachment
beep(2) #Play alert sound

# Print run time
overallEndTime = Sys.time()
messsage("Overall run time:")
print(overallEndTime - overallStartTime)

# Write run time to file
timeDiff = overallEndTime - overallStartTime
write(paste(date(),"-",timeDiff, attr(timeDiff, "units")),
      file = "LOG.txt", append = T)
```

C.2 Algorithm (CODAS.cpp)

```
/// CODAS Algorithm                                              ///
/// Original by Dr. Richard Hyde: https://github.com/RHyde67     ///
/// This implementation by Capt Erick Tello, 2021                ///

// Set up Rcpp for compatibility with R
#include <Rcpp.h>
using namespace Rcpp;

// Helper function: append rows to dataframe
DataFrame dfAppend(DataFrame df, NumericVector newRow) {
  if (df.length() != newRow.length()) {
    stop("Input mismatch: dataframe column count does not match vector length");
  }

  DataFrame out;
  CharacterVector tempNameV = df.names();
  for (int i = 0; i < df.length(); i++) {
    NumericVector temp = df[i];
    temp.push_back(newRow[i]);

    String tempName = tempNameV[i];

    out[tempName] = temp;
  }
  return out;
}

// Helper function: keep only selected rows in dataframe
DataFrame dfKeep(DataFrame df, LogicalVector lv) {
  if (df.nrows() != lv.length()) {
    stop("Length mismatch");
  }

  DataFrame out;
  CharacterVector tempNameV = df.names();
  for (int i = 0; i < df.length(); i++) {
    NumericVector temp = df[i];
    temp = temp[lv];

    String tempName = tempNameV[i];

    out[tempName] = temp;
  }
  return out;
}

// Helper function: remove selected rows from dataframe
DataFrame dfRemove(DataFrame df, IntegerVector iv) {
  if (df.nrows() != iv.length()) {
```

```
    stop("Length mismatch");
  }

  DataFrame out;
  CharacterVector tempNameV = df.names();
  for (int i = 0; i < df.length(); i++) {
    NumericVector temp = df[i];
    temp = temp[iv == 0];

    String tempName = tempNameV[i];

    out[tempName] = temp;
  }
  return out;
}

// Helper function: copy row from dataframe
NumericVector dfRow(DataFrame df, int rowID) {
  NumericVector row(df.length());
  for (int i = 0; i < df.length(); i++) {
    NumericVector temp = df[i];
    row[i] = temp[rowID];
  }
  return row;
}

// Helper function: edit row in dataframe
void dfUpdate(DataFrame& df, int rowID, NumericVector data) {
  for (int i = 0; i < df.length(); i++) {
    NumericVector temp = df[i];
    temp[rowID] = data[i];
  }
  return;
}

// Helper function: concatenate vectors together
IntegerVector vConcat(IntegerVector a, IntegerVector b) {
  int n = a.length() + b.length();
  IntegerVector out(n);
  for (int i = 0; i < a.length(); i++) {
    out[i] = a[i];
  }
  for (int i = 0; i < b.length(); i++) {
    out[i+a.length()] = b[i];
  }
  return out;
}

// Helper function: calculate Euclidean distance between dataframe rows, pairwise
NumericMatrix pDfDist(DataFrame df1, DataFrame df2) {
  NumericMatrix out(df1.nrows(),df2.nrows());
  NumericVector x1 = df1[0];
```

```
  NumericVector y1 = df1[1];
  NumericVector z1 = df1[2];
  NumericVector x2 = df2[0];
  NumericVector y2 = df2[1];
  NumericVector z2 = df2[2];

  for(int i = 0; i < df1.nrows(); i++){
    double xi=x1[i];
    double yi=y1[i];
    double zi=z1[i];
    for(int j = 0; j < df2.nrows(); j++){
      double xDiff = xi - x2[j];
      double yDiff = yi - y2[j];
      double zDiff = zi - z2[j];
      out(i,j) = sqrt(xDiff*xDiff + yDiff*yDiff + zDiff*zDiff);
    }
  }

  return out;
}

// Helper function: calculate Euclidean distance between point and dataframe rows
NumericVector pDist(NumericVector v, DataFrame df) {
  NumericVector out(df.nrows());
  NumericVector x2 = df[0];
  NumericVector y2 = df[1];
  NumericVector z2 = df[2];

  double xi=v[0];
  double yi=v[1];
  double zi=v[2];
  for(int i = 0; i < df.nrows(); i++){
    double xDiff = xi - x2[i];
    double yDiff = yi - y2[i];
    double zDiff = zi - z2[i];
    out[i] = sqrt(xDiff*xDiff + yDiff*yDiff + zDiff*zDiff);
  }

  return out;
}

// Helper function: compare numeric matrix against treshold, make boolean matrix
IntegerMatrix MatrixCompare(NumericMatrix fmat, double t) {
  IntegerVector dims = fmat.attr("dim");
  NumericVector v(fmat);
  IntegerVector lv = wrap(v < t);
  return IntegerMatrix(dims[0], dims[1], lv.begin());
}

// CODAS function: check outliers, generate first microcluster (mC) if appropriate
void StartCluster(DataFrame&     clusterCenters,
                  IntegerVector& clusterCounts,
```

```
                  IntegerVector& clusterKernels,
                  DataFrame&     outliers,
                  double         radius,
                  int            minThreshold){
  // Note: graph update not necessary for first mC, excluded from function
  NumericMatrix distances = pDfDist(outliers,outliers); //Pairwise comparison
  IntegerMatrix neighbors = MatrixCompare(distances, radius); //Get neighbors
  IntegerVector neighborSums = colSums(neighbors); //Get neighbor counts
  int maxID = which_max(neighborSums); //Get point with most neighbors
  int maxN  = neighborSums[maxID]; //Get max neighbor count
  NumericVector distanceCol = distances(_,maxID); //Max neighbor distances
  IntegerVector neighborCol = neighbors(_,maxID); //Max neighbor IDs

  // If enough neighbors, generate mC
  if (maxN >= minThreshold) {
    NumericVector newRow(outliers.length()); //Container for new mC
    for (int i = 0; i < outliers.length(); i++) { //Get neighbor points, avg them
      NumericVector temp = clone(as<NumericVector>(outliers[i]));
      temp = temp[neighborCol==1];
      newRow[i] = mean(temp); //Avgs become mC coordinates
    }
    IntegerVector kern = wrap(distanceCol < radius*0.5); //Count points in kernel
    int kernSum = sum(kern); //Sum count
    clusterCenters = dfAppend(clusterCenters,newRow); //Assign mC coordinates
    clusterCounts.push_back(maxN); //Write point count
    clusterKernels.push_back(kernSum); //Write kernel point count
    outliers = dfRemove(outliers,neighborCol); //Remove points from outliers
  }
  return;
}

// CODAS function: build graph edges
void Graph(IntegerVector& graphL,
           IntegerVector& graphR,
           DataFrame clusterCenters,
           int nID, double radius) {
  NumericVector CCRow = dfRow(clusterCenters,nID); //Get ID'd microcluster (mC)
  NumericVector distances = pDist(CCRow,clusterCenters); //Get distances to mCs

  // Get mCs within 1.5R
  IntegerVector edgeClusters;
  for (int i = 0; i < distances.length(); i++) {
    if (distances[i] < radius*1.5) {
      if (i != nID) {
        edgeClusters.push_back(i);
      }
    }
  }

  // Build edges
  if (edgeClusters.length() > 0) {
    IntegerVector CCRep(edgeClusters.length(),nID);
```

```
    // Ensure the smaller of each ID pair is considered the left end of the edge:
    IntegerVector edgeL = pmin(CCRep,edgeClusters);
    IntegerVector edgeR = pmax(CCRep,edgeClusters);

    LogicalVector flags(edgeL.length(),1); //Used to subset candidate edges

    // Check each candidate for matching edges in the graph:
    for (int i = 0; i < edgeL.length(); i++) {
      // For a given candidate edge, find any matching left ends in the graph,
      // then get their corresponding right ends:
      IntegerVector temp1 = graphR[graphL==edgeL[i]];
      IntegerVector temp2 = clone(temp1);

      // Loop through right ends to see if any match the candidate, unflag if so:
      for (int j = 0; j < temp2.length(); j++) {
        if (temp2[j] == edgeR[i]) {
          flags[i] = 0;
          break;
        }
      }
    }

    // Remove edges already in graph:
    edgeL = edgeL[flags];
    edgeR = edgeR[flags];

    // Put everything together and update graph
    if (edgeL.length() > 0) {
      graphL = vConcat(graphL,edgeL);
      graphR = vConcat(graphR,edgeR);
    }
  }
  return;
}

// CODAS function: move/generate microclusters (mCs)
void Assign(NumericVector  sample,
            DataFrame&     clusterCenters,
            IntegerVector& clusterCounts,
            IntegerVector& clusterKernels,
            DataFrame&     outliers,
            double         radius,
            int            minThreshold,
            IntegerVector& graphL,
            IntegerVector& graphR) {
  // Find distance to nearest mC
  NumericVector distances = pDist(sample,clusterCenters);
  int nID = which_min(distances);
  double nearestDist = distances[nID];

  // If inside a microcluster, update microcluster:
```

```
  if (nearestDist < radius) {
    clusterCounts[nID]++; // Increment sample count
    if (nearestDist < radius*0.5) {
      clusterKernels[nID]++; // Increment kernel sample count

      // Update cluster location:
      NumericVector CCRow = dfRow(clusterCenters,nID);
      CCRow = (CCRow*(clusterKernels[nID]-1) + sample) / clusterKernels[nID];
      dfUpdate(clusterCenters,nID,CCRow);

      // Update graph with any new edges:
      Graph(graphL,graphR,clusterCenters,nID,radius);
         // (Note: cluster movement never results in deletion of edges)
    }
  } else { // If not inside a microcluster, try to create new microcluster:
    outliers = dfAppend(outliers,sample); //Add record to outliers
    NumericMatrix distances = pDfDist(outliers,outliers); //Pairwise comparison
    IntegerMatrix neighbors = MatrixCompare(distances, radius); //Get neighbors
    IntegerVector neighborSums = colSums(neighbors); //Get neighbor counts
    int maxID = which_max(neighborSums); //Get point with most neighbors
    int maxN  = neighborSums[maxID]; //Get max neighbor count
    NumericVector distanceCol = distances(_,maxID); //Max neighbor distances
    IntegerVector neighborCol = neighbors(_,maxID); //Max neighbor IDs

    // If enough neighbors, generate mC
    if (maxN >= minThreshold) {
      int n = clusterCenters.nrows(); //Get number of existing mCs
      NumericVector newRow(outliers.length()); //Container for new mC
      for (int i = 0; i < outliers.length(); i++) { //Get avg of neighbor points
        NumericVector temp = clone(as<NumericVector>(outliers[i]));
        temp = temp[neighborCol==1];
        newRow[i] = mean(temp); //Avgs become mC coordinates
      }
      IntegerVector kern = wrap(distanceCol < radius*0.5); //Count kernel points
      int kernSum = sum(kern); //Sum count
      clusterCenters = dfAppend(clusterCenters,newRow); //Assign mC coordinates
      clusterCounts.push_back(maxN); //Write point count
      clusterKernels.push_back(kernSum); //Write kernel point count
      outliers = dfRemove(outliers,neighborCol); //Remove points from outliers

      // Update graph with any new edges:
      Graph(graphL,graphR,clusterCenters,n,radius);
    }
  }

  return;
}

// Main CODAS function (accessible from R script)
// [[Rcpp::export]]
List CODAS(DataFrame     data,
           int           fileCounter,
```

```
            DataFrame     clusterCenters,
            IntegerVector clusterCounts,
            IntegerVector clusterKernels,
            DataFrame     outliers,
            double        radius,
            int           minThreshold,
            IntegerVector graphL,
            IntegerVector graphR) {
  // Iterate through records
  for (int i = 0; i < data.nrows(); i++) {
    NumericVector sample = dfRow(data,i); //Select record from dataframe

    // Remove outliers older than twice the time threshold
    LogicalVector outliersSelect = as<NumericVector>(outliers[0]) >= sample[0] - 2;
    dfKeep(outliers,outliersSelect);

    if (clusterCenters.nrows() == 0) { //If no microclusters (mCs) exist
      outliers = dfAppend(outliers,sample); //Add record to outliers
      StartCluster(clusterCenters,clusterCounts,clusterKernels,
                   outliers,radius,minThreshold); //Generate mC if appropriate
    } else {
      Assign(sample,clusterCenters,clusterCounts,clusterKernels,
             outliers,radius,minThreshold,graphL,graphR); //Assign to or generate mC
    }
    // Print progress message every 1,000 records
    if ((i%1000 == 0)*(i>0)) {
      Rcout << "File "<< fileCounter << ": Row " << i << ", " << outliers.nrows()
            << " outliers, " << clusterCenters.nrows() << " clusters" << "\n";
    }
  }
  // Assemble & output cluster information
  List output = List::create(_["clusterCenters"] = clusterCenters,
                             _["clusterCounts"] = clusterCounts,
                             _["clusterKernels"] = clusterKernels,
                             _["outliers"] = outliers,
                             _["graphL"] = graphL,
                             _["graphR"] = graphR);
  return output;
}

// Post-CODAS function: associate data records with nearest microcluster (mC)
// [[Rcpp::export]]
IntegerVector MicroMembers(DataFrame data, DataFrame clusterCenters, double radius) {
  IntegerVector microIDs(data.nrows()); //microID values default to zero

  // Iterate through records
  for (int i = 0; i < data.nrows(); i++) {
    NumericVector sample = dfRow(data,i); //Get data record

    // Find nearest mC, associate with record if appropriate
    NumericVector distances = pDist(sample,clusterCenters);
    int minID = which_min(distances);
```

```
    double minDist = distances[minID];
    if (minDist < radius) {
      microIDs[i] = minID + 1; //(account for zero-indexing)
    }
  }
  return microIDs;
}

// Post-CODAS function: assign macrocluster (MC) IDs to data records
// [[Rcpp::export]]
IntegerVector MacroMembers(IntegerVector microIDs, IntegerVector macros) {
  IntegerVector macroIDs(microIDs.length()); //macroID values default to zero

  // Iterate through records, assign MC IDs based on association w/ mCs
  for (int i = 0; i < microIDs.length(); i++) {
    if (microIDs[i] != 0) {
      macroIDs[i] = macros[microIDs[i]-1]; //(account for zero-indexing)
    }
  }
  return macroIDs;
}
```

Appendix D. Distance Calculations Code

D.1 With Hierarchical Subclustering (ltngDistHC.R)

```
### Lightning Distance Calc with Hierarchical Subclustering ###
### Capt Erick Tello, 2021                                   ###

library(geosphere) #Used for calculating geographic distances
library(beepr)     #Used to play alert sound
library(dplyr)     #Used to select/aggregate data
library(Rcpp)      #Used to run CODAS.cpp
library(igraph)    #Used to find graph components
library(sp)        #geosphere helper
library(rgdal)     #geosphere helper

# Set DateTime precision
options(digits = 13)
options(digits.secs = 3)

# Initialize ID and distance containers
folderID = c()
fileID = c()
clusterID = c()
dists = c()

# Iterate through 11 output folders
for (m in 1:11) {
  outputFolder = paste0("CEDAS 5nmi 30min (all data) Final ",m)
  fList = list.files(path = paste0("./",outputFolder,"/"), pattern = "cdpx*")

  # Iterate through files
  for (i in fList) {
    ltngData = read.csv(paste0(outputFolder,"/",i))

    # Iterate through macrocluster IDs
    for (j in min(ltngData$MacroClusterID):max(ltngData$MacroClusterID)) {
      tempData = ltngData[ltngData$MacroClusterID == j,]
      # # Option: get 1st 10 records
      # tempData = tempData[1:10,]
      # Option: get 1st 30 seconds
      tempData = tempData[tempData$DateTime <= min(tempData$DateTime) + 30,]

      tempData = tempData[,c(3,2)] #Arrange columns for distance calculation
      tempData = na.omit(tempData) #Omit any NAs

      # If any data, apply hierarchical subclustering
      if (nrow(tempData) > 1) {
        # Detect any separation of electrical activity w/ heirarchical clustering
        distMatr = distm(tempData) * 0.000539957 #Get distances, convert to NM
        hc = hclust(as.dist(distMatr), method="complete") #Apply clustering
        localClust = cutree(hc, h=5) #Set cutoff distance, assign cluster IDs
```

```
        if (max(localClust) > 1) { #max(localClust) > 1
          # Process subclusters
          for (k in 1:max(localClust)) {
            tempCData = tempData[localClust == k,]
            if (nrow(tempCData) > 1) {
              # Geographic distance calculation (storm origin)
              tempDists = sapply(2:nrow(tempCData),function(i){
                distm(tempCData[1,],tempCData[i,])}) * 0.000539957
              dists = c(dists,tempDists) #Append to distance list

              # Append IDs
              folderID = c(folderID,rep(m,length(tempDists)))
              fileID = c(fileID,rep(i,length(tempDists)))
              clusterID = c(clusterID,rep(j,length(tempDists)))
            }
          }
        } else {
          # Geographic distance calculation (storm origin)
          tempDists = sapply(2:nrow(tempData),function(i){
            distm(tempData[1,],tempData[i,])}) * 0.000539957
          dists = c(dists,tempDists)

          # Append IDs
          folderID = c(folderID,rep(m,length(tempDists)))
          fileID = c(fileID,rep(i,length(tempDists)))
          clusterID = c(clusterID,rep(j,length(tempDists)))
        }
      }
      print(paste0("Points: ",i,", ",j)) #Progress message
    }
  }
}
beep(2) #Play alert sound
ltngDistsRaw = data.frame(Folder   = folderID, #Build dataframe
                          File     = fileID,
                          Cluster  = clusterID,
                          Distance = dists)
ltngDists = data.frame(ltngDistsRaw) #Copy dataframe
ltngDists = ltngDists[ltngDists$Cluster != 0,] #Remove outliers
ltngDists = na.omit(ltngDists) #Remove NAs

#---- Analytics ----#

# Basic histogram
hist(ltngDists$Distance,
     main = NULL, #"Histogram of Early Lightning Distances From Storm Origin",
     xlab = "Distance (NM)",
     breaks = seq(0,ceiling(max(ltngDists$Distance)),1))

# Percent of values beyond 4 NM
length(ltngDists$Distance[ltngDists$Distance > 4])/length(ltngDists$Distance)
```

D.2 With CODAS Subclustering (ltngDistCODAS.R)

```
### Lightning Distance Calc with CODAS Subclustering ###
### Capt Erick Tello, 2021                           ###

library(geosphere)    #Used for calculating geographic distances
library(beepr)        #Used to play alert sound
library(dplyr)        #Used to select/aggregate data
library(Rcpp)         #Used to run CODAS.cpp
library(igraph)       #Used to find graph components
library(sp)           #geosphere helper
library(rgdal)        #geosphere helper
library(fitdistrplus) #Used to fit distributions

# Set DateTime precision
options(digits = 13)
options(digits.secs = 3)

# Initialize ID and distance containers
folderID = c()
clusterID = c()
dists = c()

# Get CODAS and associated functions
sourceCpp("CODAS.cpp")
radius = 5           # CEDAS microC radius
minThreshold = 1     # Min microC threshold

# Iterate through 11 output folders
for (m in 1:11) {
  outputFolder = paste0("CEDAS 5nmi 30min (all data) Final ",m)
  fList = list.files(path = paste0("./",outputFolder,"/"), pattern = "cdpx*")

  # Initialize lightning data container
  ltngData = data.frame(DateTime = numeric(0),
                        Latitude = numeric(0),
                        Longitude = numeric(0),
                        MicroClusterID = integer(0),
                        MacroClusterID = integer(0))
  for (i in fList) { #Iterate through files
    ltngData = rbind(ltngData,read.csv(paste0(outputFolder,"/",i)))
  }

  # Iterate through macrocluster IDs
  for (j in min(ltngData$MacroClusterID):max(ltngData$MacroClusterID)) {
    tempDataInit = ltngData[ltngData$MacroClusterID == j,]
    tempDataInit = na.omit(tempDataInit)

    # Filter to times between 1400 and 1800 UTC
    if ( (nrow(tempDataInit) > 1) &
         (tempDataInit$DateTime[1] %% 86400 >= 50400) &
```

```
   (tempDataInit$DateTime[1] %% 86400 <= 64800) ) {

# Filter to data not preceded by anything in past 15 minutes
chckWinStart = tempDataInit$DateTime[1] - 1
chckWinEnd   = tempDataInit$DateTime[1] - 900
chckWin = ltngData[(ltngData$DateTime >= chckWinStart) &
                   (ltngData$DateTime <= chckWinEnd),]

if(nrow(chckWin) == 0){
  # # Option: get 1st 10 records
  # tempDataInit = tempDataInit[1:min(c(10,nrow(tempDataInit))),]
  # Option: get 1st 30 seconds
  tempDataInit = tempDataInit[tempDataInit$DateTime <=
                              min(tempDataInit$DateTime) + 30,]

  # Arrange columns for distance calculation
  tempData = tempDataInit[,c(3,2)]

  # If any data, apply CODAS subclustering
  if (nrow(tempData) > 1) {
    #----
    clusteredData = data.frame(tempDataInit[,1:3])
    clusteredData$DateTime = rep(1,nrow(clusteredData)) #Flatten times
    clusteredData$Latitude = clusteredData$Latitude / 0.01667 #lat -> NM
    clusteredData$Longitude = clusteredData$Longitude / 0.0193 #long -> NM

    # Initialize CODAS data structure
    output = list(data.frame(DateTime = numeric(0),
                             Latitude = numeric(0),
                             Longitude = numeric(0)),
                  integer(0),
                  integer(0),
                  data.frame(DateTime = numeric(0),
                             Latitude = numeric(0),
                             Longitude = numeric(0)),
                  integer(0),
                  integer(0))
    names(output) = c("clusterCenters","clusterCounts","clusterKernels",
                      "outliers","graphL","graphR")
    fileCounter = 1L #Progress message helper

    # Run CODAS algorithm
    output = CODAS(clusteredData,
                   fileCounter,
                   output$clusterCenters,
                   output$clusterCounts,
                   output$clusterKernels,
                   output$outliers,
                   radius,
                   minThreshold,
                   output$graphL,
                   output$graphR)
```

```
            # Fix graph node IDs (off due to Rcpp zero-indexing):
            output$graphL = output$graphL + 1
            output$graphR = output$graphR + 1

            # Find graph components
            clusterGraph = make_undirected_graph(c(rbind(output$graphL,
                           output$graphR)), nrow(output$clusterCenters))
            subMacroClusters = components(clusterGraph)

            # Assign subcluster IDs
            subMicroID = MicroMembers(clusteredData,output$clusterCenters,radius)
            subMacroID = MacroMembers(subMicroID,subMacroClusters$membership)

            #----

            # Process subclusters
            for (k in 1:max(subMacroID)) {
              tempCData = tempData[subMacroID == k,]
              if (nrow(tempCData) > 1) {
                # Geographic distance calculation
                tempDists = sapply(2:nrow(tempCData),function(i){
                  distm(tempCData[i-1,],tempCData[i,])}) * 0.000539957
                dists = c(dists,tempDists) #Append to distance list

                # Append IDs
                folderID = c(folderID,rep(m,length(tempDists)))
                clusterID = c(clusterID,rep(j,length(tempDists)))
              }
            }
          }
        }
      }
    }
    print(paste0("Folder: ",m,"; Max Macrocluster: ",j)) #Progress message
}
beep(2) #Play alert sound
ltngDistsRaw = data.frame(Folder   = folderID, #Build dataframe
                          Cluster  = clusterID,
                          Distance = dists)
ltngDists = data.frame(ltngDistsRaw) #Copy dataframe
ltngDists = ltngDists[ltngDists$Cluster != 0,] #Remove outliers
ltngDists = na.omit(ltngDists) #Remove NAs

#---- Analytics ----#

# Basic histogram
hist(ltngDists$Distance,
     main = NULL, #"Histogram of Distances Between Successive Lightning Events",
     xlab = "Distance (NM)",
     breaks = seq(0,ceiling(max(ltngDists$Distance)),1))
```

```
# Summary statistics
cGroup = group_by(ltngDists,Folder,Cluster)
cSummary = summarize(cGroup, length(Distance), max(Distance), .groups="keep")

# Percent of values beyond 4 NM
length(ltngDists$Distance[ltngDists$Distance > 4])/length(ltngDists$Distance)

# Distribution plotting
hist(ltngDists$Distance,
     main = NULL, #"Histogram of Distances Between Successive Lightning Events",
     xlab = "Distance (NM)", ylab = "Probability Density", freq = FALSE,
     breaks = seq(0,ceiling(max(ltngDists$Distance)),0.5))
weib = fitdist(ltngDists$Distance, "weibull",
               start = list(shape = 0.833, scale = 2.142*0.539957), method = "mse")
curve(dweibull(x, shape=weib$estimate[1], scale=weib$estimate[2]),
      from=0, to=max(ltngDists$Distance), add=TRUE,
      col="blue", lwd=2)
legend(x=max(ltngDists$Distance)/2, y=1, col="blue", lty=1, lwd=2,
       legend=c(paste0("Weibull (shape = ",round(weib$estimate[1],3),
                       ", scale = ",round(weib$estimate[2],3),")")))

# Percent of distribution beyond 4 NM
1-pweibull(4, shape = weib$estimate[1], scale = weib$estimate[2])

# Sanderson distribution comparison
1-pweibull(4,0.833,2.142*0.539957)
mean(rweibull(1000, shape = 0.833, scale = 2.142*0.539957))
median(rweibull(1000, shape = 0.833, scale = 2.142*0.539957))
```

Appendix E. Data Visualisation Code (ltngVis.R)

```
### Data Visualisation App ###
### Capt Erick Tello, 2021 ###

library(dplyr)      #Used to select/aggregate data
library(DBI)        #Used for data manipulation
library(shiny)      #Used for data visualisation
library(shinyTime) #Used for improved time controls
library(leaflet)    #Used to overlay data on satellite maps

# Set DateTime precision
options(digits = 13)
options(digits.secs = 3)

# # Default data file selection (user should edit filepath as desired)
# ltng = read.csv("CEDAS 5nmi 30min May2/cd2019.05.02.TL.csv",
#                 colClasses=c(DateTime="character"))
# dateTime = as.POSIXct(ltng$DateTime)

# Commented lines represent clusters of interest

# ltng = read.csv("CEDAS 5nmi 30min (all data) Final 5/cdpx2019.07.02.TL.csv")
# ltng = ltng[ltng$MacroClusterID == 228,]

# ltng = read.csv("CEDAS 5nmi 30min (all data) Final 8/cd2019.07.30.TL.csv")
# ltng = ltng[ltng$MacroClusterID == 170,]

# ltng = read.csv("CEDAS 5nmi 30min (all data) Final 4/cdpx2019.06.26.TL.csv")
# ltng = ltng[ltng$MacroClusterID == 423,]

# ltng = read.csv("CEDAS 5nmi 30min (all data) Final 3/cdpx2019.06.11.TL.csv")
# ltng = ltng[ltng$MacroClusterID == 338,]


# Get data across CODAS output files, build data structures
outputFolder = paste0("CEDAS 5nmi 30min (all data) Final ",8) #Select folder
fList = list.files(path = paste0("./",outputFolder,"/"), pattern = "cdpx*")

ltng = data.frame(DateTime = numeric(0),
                  Latitude = numeric(0),
                  Longitude = numeric(0),
                  MicroClusterID = integer(0),
                  MacroClusterID = integer(0))
for (i in fList) {
  ltng = rbind(ltng,read.csv(paste0(outputFolder,"/",i)))
}
# ltng = ltng[ltng$MacroClusterID == 589,] #Optional filter to macrocluster



dateTime = as.POSIXct(ltng$DateTime, origin="1970-01-01")
```

```
# Build color palette for data points:
colr = topo.colors(50)
colr2 = grDevices::colors()[grep('gr(a|e)y', grDevices::colors(), invert = T)]
pal = colorFactor(grDevices::colors()[grep('gr(a|e)y',
                    grDevices::colors(), invert = T)], 1:433)
pal2 = colorFactor(topo.colors(50), 1:50)
factpal <- colorFactor(topo.colors(5), 1:5)

# Build UI:
ui <- fluidPage(
    align = "center",
    sliderInput(inputId = "timeStart", label = "Time start",
                min = as.POSIXct(dateTime[1]),
                max = as.POSIXct(tail(dateTime, n=1)),
                value = as.POSIXct(dateTime[1])),
    numericInput(inputId = "timeOffset", label = "Time window (seconds):",
                 value = 30, min = 1, step = 1),
    sliderInput(inputId = "pointCount", label = "Point Count",
                min = 1, max = 10, value = 10),
    leafletOutput("map")
)

# Translate inputs to outputs:
server <- function(input, output) {
  output$map <- renderLeaflet({
    # # Alternative filtering:
    # timeStart = as.POSIXct(input£dateStart) + input£timeStart*60
    # timeEnd = as.POSIXct(input£dateStart) + input£timeStart*60 + input£timeOffset
    # output£timeStart <- renderText({ format(timeStart, tz = "GMT") })
    # output£timeEnd <- renderText({ format(timeEnd, tz = "GMT") })
    ltngSubset = ltng[(ltng$DateTime >= as.numeric(timeStart)) &
                      (ltng$DateTime <= as.numeric(timeEnd)),
                      c("Latitude","Longitude","MacroClusterID")]

    # Option: filter by time
    ltngSubset = ltng[(dateTime >= input$timeStart) &
                      (dateTime <= input$timeStart + input$timeOffset),
                      c("Latitude","Longitude","MacroClusterID")]
    # # Option: filter by point count
    # ltngSubset = ltng[1:input£pointCount,c("Latitude","Longitude","MacroClusterID")]

    # Overlay points on satellite map
    leaflet(ltngSubset) %>%
      setView(lat = 28.424, lng = -80.825, zoom = 8) %>%
      addProviderTiles(providers$Esri.WorldImagery,
                       options = providerTileOptions(noWrap = TRUE)) %>%
      addCircleMarkers(lat = ~Latitude, lng = ~Longitude, radius = 6,
                       color = ~pal2(MacroClusterID %% 50 + 1),
                       stroke = FALSE, fillOpacity = 0.5, group = "Cloud-to-Cloud") %>%
      addRectangles(
        lat1 = 28.1016, lng1 = -81.2038,
```

```
        lat2 = 28.9861, lng2 = -80.3546,
        fillColor = "transparent"
      )
  })
}

# Required for Shiny app
shinyApp(ui = ui, server = server)
```

Bibliography

Bazelyan, E. M. and Raizer, Y. P. (2000), *Lightning physics and lightning protection*, CRC Press.

Boyd, B. F., Roeder, W. P., Hajek, D. L. and Wilson, M. B. (2005), 'Installation, upgrade, and use of a short baseline cloud-to-ground lightning surveillance system in support of space launch operations', *Conference on Meteorological Applications of Lightning Data* .

Ceschini, E. M. (2013), Optimization of lightning warning areas at Cape Canaveral/Kennedy Space Center, Master's thesis, Naval Postgraduate School.
URL: *http://hdl.handle.net/10945/34640*

Cox, C. C. (1999), A comparison of horizontal cloud-to-ground lightning flash distance using weather surveillance radar and the distance between successive flashes method, Master's thesis, Air Force Institute of Technology.

Department of the Air Force (2012), *Safety: Air Force Consolidated Occupational Safety Instruction*, HQ USAF.

Hinkley, J. J., Huddleston, L. L. and Roeder, W. P. (2019), Lightning strike distance distribution beyond a preexisting lightning area, Technical report, National Aeronautics and Space Administration.

Holland, K. G. (2021), CCSFS/KSC total lightning warning radii optimization for MERLIN using preexisting lightning areas, Master's thesis, Air Force Institute of Technology.

Holle, R. L., Demetriades, N. W. S. and Nag, A. (2014), 'Lightning warnings with nldn cloud and cloud-to-ground lightning data', *2014 International Conference on Lightning Protection (ICLP)* pp. 315–324.

Holle, R. L., Demetriades, N. W. S. and Nag, A. (2016), 'Objective airport warnings over small areas using NLDN cloud and cloud-to-ground lightning data', *Weather and Forecasting* **31**, 1061–1069.

Horvath, T. (2006), *Understanding Lightning and Lightning Protection*, Wiley.

Hyde, R. and Angelov, P. (2015), *A New Online Clustering Approach for Data in Arbitrary Shaped Clusters*, IEEE 2nd International Conference on Cybernetics (CYBCONF).
URL: *https://github.com/RHyde67*

Lopez, R. E. and Holle, R. L. (1999), 'The distance between successive lightning flashes', *Bulletin of the American Meteorological Society* **80**, 2035–2041.

McNamara, T. M. (2002), The horizontal extent of cloud-to-ground lightning over the Kennedy Space Center, Master's thesis, Air Force Institute of Technology.

Murphy, M. J. and Cummins, K. L. (2000), 'Early detection and warning of cloud-to-ground lightning at a point of interest', *Preprints, 2nd Symposium on Environmental Applications* .

Murphy, M. J., Cummins, K. L., Demetriades, N. W. S. and Roeder, W. P. (2008), 'Performance of the new four-dimensional lightning surveillance system (4DLSS) at the Kennedy Space Center/Cape Canaveral Air Force Station complex', *13th Conference on Aviation, Range, and Aerospace Meteorology* pp. 20–24.

Murtagh, F. and Contreras, P. (2012), 'Algorithms for hierarchical clustering: an overview', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* pp. 86–97.

Parsons, T. L. (2000), Determining the horizontal distance distribution of cloud-to-ground lightning, Master's thesis, Air Force Institute of Technology.

Renner, S. L. (1998), Analyzing horizontal distances between WSR-88D thunderstorm centroids and cloud-to-ground lightning strikes, Master's thesis, Air Force Institute of Technology.

Roeder, W. P. (2020), 'Personal communication presenting a new grid-based algorithm for grouping lightning observation data into lightning flashes'.

Roeder, W. P. (2021), 'Personal communication on size of data intended to represent early storms'.

Roeder, W. P., McNamara, T. M., McAleenan, M., Winters, K. A., Maier, L. M. and Huddleston, L. L. (2017), 'The 2014 upgrade to the lightning warning areas used by the 45th Weather Squadron', *18th Conference on Aviation, Range, and Aerospace Meteorology* .

Roeder, W. P. and Saul, J. M. (2017), 'The new Mesoscale Eastern Range Lightning Information System', *Sensors* **10**.

Sanderson, D. L. (2019), Modeling the distribution of lightning strike distances outside a preexisting lightning area, Master's thesis, Air Force Institute of Technology.

Tamurian, Z. N., Fuelberg, H. E. and Roeder, W. P. (2012), 'Determining the characteristics of anvil and thunderstorm lightning for use in the lightning Launch Commit Criteria at Cape Canaveral Air Force Station and Kennedy Space Center', *4th International Lightning Meteorology Conference* .

Zoghzoghy, F. G., Cohen, M. B., Said, R. K., Basilico, S. S., Blakeslee, R. J. and Inan, U. S. (2014), 'Lightning activity following the return stroke', *Journal of Geophysical Research: Atmospheres* **119**, 8329–8339.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	August 2019 — March 2021

4. TITLE AND SUBTITLE

Behavior of Lightning in Developing Storms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Tello, Erick A., Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-21-M-187

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

45th Weather Squadron
William P. Roeder, GS-12
10400 Phillips Pkwy
Cape Canaveral SFS, FL 32925-2618
william.roeder@spaceforce.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

45 WS/WXT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Air Force weather squadrons issue a warning when lightning activity is observed within 5 nautical miles (NM) of protected areas. Upon receiving this warning, personnel outdoors are expected to pause work and move inside. Studies sponsored by the 45th Weather Squadron (45 WS) have concluded that the 5 NM warning radius can be safely reduced for well-developed storms. This thesis investigates whether radii for storms in early development can also be reduced. Our research develops algorithms to partition lightning sensor data into storms. Next, storms are filtered to their earliest lightning events, and the study calculates distances between successive early lightning observations. Analysis indicates that 4.02% of lightning events during the first 30 seconds of a storm occur more than 4 NM away from the previous event, and 2.11% occur more than 5 NM away. Because these percentages are smaller than the equivalent values for well-developed storms from Sanderson (2019), we conclude that her recommended lightning warning radius of 4 NM is valid for developing storms as well.

15. SUBJECT TERMS

lightning, meteorology, dynamic clustering, distribution fitting

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col Timothy W. Holzmann, AFIT/ENS
U	U	U	UU	100	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4337; timothy.holzmann@afit.edu