Optimizing Sensor Locations to Improve The Worst Case Detection Performance of Sensor Detection Systems

Ryan Taylor Vegrzyn
rvegrzyn@yahoo.com

Follow this and additional works at: https://huskiecommons.lib.niu.edu/allgraduate-thesesdissertations

Part of the Electrical and Computer Engineering Commons

Recommended Citation

This Dissertation/Thesis is brought to you for free and open access by the Graduate Research & Artistry at Huskie Commons. It has been accepted for inclusion in Graduate Research Theses & Dissertations by an authorized administrator of Huskie Commons. For more information, please contact jschumacher@niu.edu.
This thesis discusses the optimization of distributed sensor systems to maximize detection of an emitter at an unknown location. More specifically, this thesis discusses the creation of an algorithm that seeks to optimize the spatial positions of sensors in a distributed sensor array in order to maximize the chance of detecting an emitter even if it were to occur in the worst possible location for detection. Two versions of this algorithm are discussed. The first algorithm considers a fusion rule that detects the emitter based on the sum of all sensor measurements. The second algorithm considers the scan statistic fusion rule. Both versions implement a local search algorithm designed by Torczon.

Both versions of the algorithm showed that it is possible to improve the worst-case detection of sensor detection systems over a grid pattern. In the conditions tested, the worst-case detection increased by up to 23%, or an increase factor of 2.11, under different conditions when compared with a standard grid pattern. Testing proved that as long as the sensor array is not too large, these algorithms are able to improve the worst-case detection of sensor systems within a reasonable amount of time.
OPTIMIZING SENSOR LOCATIONS TO IMPROVE THE WORST-CASE DETECTION PERFORMANCE OF SENSOR DETECTION SYSTEMS

BY
RYAN TAYLOR VEGRZYN

A THESIS SUBMITTED TO THE GRADUATE SCHOOL IN FULL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Benedito Fonseca
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>PREFACE</td>
<td>vii</td>
</tr>
</tbody>
</table>

**Chapter**

1. INTRODUCTION ............................................................................................................. 1

2. RELATED WORK ............................................................................................................. 8

   2.1 Distributed Sensor Detection Systems .............................................................. 8

   2.2 The Scan Statistic ................................................................................................. 10

   2.3 Numerical Methods ............................................................................................... 14

3. LOCAL SEARCH ALGORITHMS ......................................................................................... 16

   3.1 Nelder and Mead’s Algorithm .............................................................................. 16

   3.2 Torczon’s Algorithm ............................................................................................ 20

4. MODEL DESCRIPTION ................................................................................................... 25

5. ALGORITHM FOR THE SUM FUSION RULE .................................................................. 27

   5.1 Methodology .......................................................................................................... 27
LIST OF TABLES

Table 1: $P_D$ for Best and Worst Algorithm Outputs Compared Against a Grid ... 39
Table 2: Scan Statistic Algorithm Results vs. Grid Results................................. 56
Table 3: Adjusted Algorithm for Four Best Clusters for Worst POI...................... 63
Table 4: Adjusted Algorithm for Best Cluster for Four Worst POIs...................... 63
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of clustering</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Example of sensor placement</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Potential steps of Torczon’s algorithm when K=1.</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Subregion demonstration</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Histogram of $P_D(L_{ft}, S^-)$ when K=4.</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Best sensor positions obtained from 100 loops when K=2, as well as $P_D(L^+, S^-)$.</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>Best sensor positions obtained from 100 loops when K=3, as well as $P_D(L^+, S^-)$.</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Best sensor positions obtained from 100 loops when K=4, as well as $P_D(L^+, S^-)$.</td>
<td>34</td>
</tr>
<tr>
<td>9</td>
<td>Best sensor positions obtained from 100 loops when K=5, as well as $P_D(L^+, S^-)$.</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>Best sensor positions obtained from 100 loops when K=6, as well as $P_D(L^+, S^-)$.</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>Best sensor positions obtained from 100 loops when K=7, as well as $P_D(L^+, S^-)$.</td>
<td>35</td>
</tr>
<tr>
<td>12</td>
<td>Best sensor positions obtained from 100 loops when K=8, as well as $P_D(L^+, S^-)$.</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>Best sensor positions obtained from 100 loops when K=9, as well as $P_D(L^+, S^-)$.</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>Worst sensor positions obtained from 100 loops when K=9, as well as $P_D(L_{ft}, S^-)$.</td>
<td>37</td>
</tr>
</tbody>
</table>
Figure 15: Increase in worst-case $P_D$ over a grid pattern with sum fusion rule.... 39

Figure 16: Algorithm flow chart ................................................................. 42

Figure 17: POI diagram (Fonseca, 2020) ....................................................... 46

Figure 18: Best sensor positions and cluster set obtained for K=4 ............ 49

Figure 19: Best sensor positions, worst POI, and corresponding non-intersecting
cluster set obtained for K=4 ................................................................. 50

Figure 20: Best sensor positions and cluster set obtained for K=9 ............... 51

Figure 21: Best sensor positions and non-intersecting cluster set obtained for
K=9, ........................................................................................................ 51

Figure 22: Best sensor positions and cluster set obtained for K=16 .......... 52

Figure 23: Best sensor positions and non-intersecting cluster set obtained for
K=16. ...................................................................................................... 52

Figure 24: Best sensor positions and cluster set obtained for K=25 .......... 53

Figure 25: Best sensor positions and non-intersecting cluster set obtained for
K=25. ...................................................................................................... 53

Figure 26: Best sensor positions and cluster set obtained for K=36 .......... 54

Figure 27: Best sensor positions and non-intersecting cluster set obtained for
K=36. ...................................................................................................... 54

Figure 28: Best sensor positions and cluster set obtained for K=49 .......... 55

Figure 29: Best sensor positions and non-intersecting cluster set obtained for
K=49. ...................................................................................................... 55

Figure 30: Adjusted algorithm and cluster set for K=49. ......................... 62

Figure 31: Adjusted algorithm and non-intersecting cluster set for K=49. .... 62
PREFACE

In this thesis, the problem of sensor placement in a distributed sensor system is discussed. In Chapter 1, the general problem is discussed. This chapter contains information on the goal of the research and the basic research plan. In Chapter 2, related work done by others is discussed in relation to the research discussed in this thesis. In Chapter 3, various local search algorithms are discussed. Local search algorithms are an important part of the algorithm designed in this research, and this section describes them in the general case. In Chapter 4, the general model of the algorithms is discussed. In Chapter 5, the initial algorithm that seeks to optimize sensor positions in a system with a sum fusion rule is discussed. In Chapter 6, the updated algorithm that seeks to optimize sensor positions, as well as the cluster set, in a system with the scan statistic is discussed. In Chapter 7, the results of both algorithms are summarized along with some potential future work.
CHAPTER 1
INTRODUCTION

Consider the detection of a signal emitter at an unknown location within a region. Assume that the power of the emitted signal decays as the distance from the source increases. Also, assume that the received signal is corrupted by noise, motivating the use of statistical procedures (Kay, 1998) for deciding between two hypotheses: $H_0$ (no emitter is present) or $H_1$ (emitter present). In such a statistical approach, the goal is generally to maximize the probability of detection ($P_D$), while also keeping the probability of false alarm ($P_{FA}$) below some set value. Probability of detection is the probability of the system deciding for $H_1$ when $H_1$ is true, or in other words, successfully detecting the presence of an emitter. Probability of false alarm is the probability of the system deciding for $H_1$ when $H_0$ is true, or in other words, detecting an emitter when none is present. It is important to have a high $P_D$ in a system to maximize the odds of detecting whatever anomaly the system is searching for. At the same time, it is important to minimize $P_{FA}$ because in certain cases, false alarms can be expensive. Systems such as this can be used in many applications: from detecting a source of radiation in a populated area to detecting submarines in the ocean (Rao, 2008).

There are many ways to design such systems. One common way is to use an array of sensors. When using a sensor array, the sensors are generally distributed throughout the region of interest to maximize the probability of detection. These sensors then periodically attempt to detect whether there is an emitted signal somewhere within the region. All sensors in the system...
then send their received signal to a fusion center which determines whether an emitter is present within the region. This is done by first fusing the sensor measurements and then comparing the result against some preset threshold. In general, if the result is greater than the threshold, the system decides for H1. Otherwise the system decides for H0. If the number of sensors in the array is sufficiently large, the proposed fusion rule can provide good system performance even without knowledge of the emitter location and under a low signal-to-noise ratio (SNR) (Niu, 2006).

While sensors measuring their received values independently can lead to good detection if the number of sensors is sufficient for the region of interest, a higher number of sensors generally increases the cost of the system. Because of this, alternative methods to improve the detection performance have been considered.

One alternative is the sum fusion rule. In this rule, all sensors combine their received signal value, which is then compared against a threshold (Fonseca, 2019b). This can lower the $P_{FA}$, however the problem with this rule is if an emitter is present somewhere in the region, the values received by sensors both near to and far from the emitter are combined. This adds unnecessary noise to the system and can lower the probability of detection.

To compensate for this unwanted noise, the scan statistic can be considered. In the scan statistic, nearby sensors are combined into clusters. These clusters then combine their received signal values, which is then compared against a threshold (Fonseca, 2019b). The scan statistic does not suffer from a high probability of false alarm like when sensor measurements are not fused and does not combine weak measurements from far away sensors with strong measurements from nearby sensors as in the sum fusion rule.
Figure 1 shows an example of possible clusters. In this case, there would be four clusters of four sensors. It is not necessary for the clusters to be square, for clusters to be nonintersecting, or for all clusters to be identical, this is just the case in this example. Clusters can be any size or shape within reason. In general, a scan statistic system can contain intersecting clusters as well, or clusters that share certain sensors.

![Figure 1: Example of clustering.](image)

One factor that significantly impacts the $P_D$ of an array of sensors is the spatial positions of the sensors. In general, sensors can be placed anywhere within the region of interest; however, there are some ways to place sensors that are better than others. The most common way to place sensors is in a standard grid pattern. Placing sensors in a grid pattern is reasonable because it spreads sensors over the region of interest, ensuring that at least one sensor will be near an emitter wherever it occurs; however, can deployment patterns that improve detection performance upon the grid pattern be found? Answering this question is a main motivation for this thesis.

In this research, the goal will be to place sensors in order to maximize the worst-case $P_D$. The worst-case $P_D$ refers to the probability of detection if the emitter was located at the point in the region that minimizes the odds of successful detection (Fonseca, 2019b). By optimizing the
worst-case detection, it can be guaranteed that the detection at any other location within the region of interest will be at least as high. Also, the emitter location is unknown, so the sensors cannot be specifically placed to optimize the exact emitter location.

Figure 2 demonstrates the difficulty in placing sensors while trying to improve on the worst possible emitter location. In this figure, the blue circles represent sensors and the red circle represents an emitter at a possible location. In the top figure, the emitter is not close to any of the sensors. In this case, the detection would likely be low since $P_D$ depends on the distance from the emitter to the sensors. The middle plot shows a way to improve the $P_D$ by moving a sensor to the emitter. In this case, $P_D$ would be high for the indicated emitter location because there is a sensor very close to the emitter. However, this emitter location is not the worst possible emitter location for the changed sensor location; instead, the worst possible emitter location is at a different point, farther from the new sensor placement. If the emitter were moved, as in the bottom plot, it is now less likely to be detected due to the new sensor location. In practice, the emitter location is often unknown, so it is important to properly distribute sensors throughout the region of interest to maximize the odds of detecting an emitter at its worst possible location.

Figure 2: Example of sensor placement.
Consider a two-dimensional region with K sensors. In such a system, the probability of detection will vary depending on the location of the emitter. This is because $P_D$ is dependent on the distance from the emitter to the sensors. For this reason, it is not possible to maximize $P_D$ for every possible emitter location. This is because any change in sensor positions will simultaneously improve the $P_D$ in some parts of the region of interest while also lowering the $P_D$ in other parts of the region. To solve this issue, the worst-case $P_D$ is considered.

To optimize the worst-case $P_D$, a local search method is used. Local search algorithms have a few major advantages that are essential for this application. Local search algorithms do not require a known gradient (Boyd, 2011). This is important because the derivative cannot be calculated without prior knowledge of the emitter location. Also, the function is not differentiable at all locations. Furthermore, this is not a convex problem so convex optimization algorithms cannot be used. Also, if each sensor has two dimensions to optimize, the system becomes 2*K dimensional, where K is the number of sensors, so calculations become increasingly intensive.

There are downsides to local search methods, however. For one, there is no guarantee that the point of convergence will be the true optimum, but could instead be a local maximum (Boyd, 2011). At the same time, local search “algorithms have the ability to probabilistically accept candidate solutions with higher cost than that of the incumbent solution, in an effort to escape local optima” (Johnson, 1996). Generalized hill climbing can accept a random variable and has fairly loose conditions to use, making it useful for this application (Johnson, 1996). In this case, the algorithm will start with the sensors in random initial locations and slowly alter the parameters until a local maximum for the worst-case $P_D$ is found. From here, another random
drop will occur and find another local maximum. This will repeat many times to try to find the true maximum for the worst-case $P_D$ of the system.

For this research, a simplex local search algorithm is used. A D-dimensional simplex is a complex span of $D+1$ linearly independent points (vertices) (Kelly, 2009). For all algorithms considered, the simplex consists of $2*K+1$ points, each of which are $2*K$ dimensional, where $K$ is the number of sensors in the system. The algorithm then seeks to find the point that maximizes the worst-case $P_D$ by moving the various points of the simplex. Specifically, the local search algorithm designed by Torczon was used for the algorithms discussed in this research (Torczon, 1989).

Overall, two versions of the algorithm were created. The first focuses on optimizing sensor locations in a system that uses the sum fusion rule. The second algorithm attempts to optimize sensor locations in a system that uses the scan statistic. This algorithm is based on the original, however it is altered to include a clustering algorithm and optimize a lower bound for $P_D$. The use of a lower bound is necessary because there are no closed form expressions for the $P_D$ of the scan statistic.

Both versions of the algorithm had a few expectations. First, the algorithms should be able to optimize sensor positions to improve the worst-case $P_D$. Second, the algorithms should show improvement over random sensor distribution as well as over a grid pattern. Third, the algorithms should be functional for both the sum fusion rule and the scan statistic. Fourth, the algorithms should ideally work even in systems with a large number of sensors.
There were also a few numerical expectations for the algorithms. First, the algorithms should ideally be able to improve systems by around 20% when compared with a grid pattern. Ideally, the worst-case $P_D$ should always increase due to this algorithm. Second, both algorithms should work for large systems without the run time becoming unreasonable. This is important because in general, increasing $K$ also increases run time. Third, the algorithms should take as little time to run as possible. This can be done through optimization of the algorithms as well as minimal instructions.

Overall, the question that this research attempts to answer is as follows: can the worst-case probability of detection of sensor systems be improved by optimizing sensor positions? The hypothesis is as follows: if a local search method is used, a maximum for the worst-case probability of detection should be obtainable even in the presence of a large number of sensors. Ideally, the system would optimize sensor positions for any number of sensors, $K$. In practice, the run time would become very large if $K$ were too large. Also, it becomes increasingly more difficult to determine whether a true optimum is found. For these reasons, the goal will be to improve the worst-case $P_D$ as much as possible over the standard grid pattern for various values of $K$. 
CHAPTER 2
RELATED WORK

In this chapter, previous work that others have completed that relates to this research will be discussed. This will include research on sensor detection systems in section 2.1, the scan statistic in section 2.2, and numerical methods that were considered for this algorithm in section 2.3.

2.1 Distributed Sensor Detection Systems

In this section, the use of distributed sensor detection systems will be discussed. Distributed sensor detection systems are commonly used to detect anomalies in systems, but there are many variations of such systems that have been considered with a goal of improving detection.

In Niu’s research, a binary sensor network was used. This is a network where the sensors decide for H0 or H1 and transmit a single bit to the fusion center rather than transmitting their received signal. Each sensor transmits a bit to the fusion center which then counts the number of detections based on decaying signal power and derives the detection analytically using the central limit theorem. From here, the fusion rule and numerical methods are used to find the threshold (Niu, 2006). Niu essentially tried to find the Neyman-Pearson detector of an array of sensors to detect an emitter at a specific location, which is the optimal detector for the system. However, when the emitter location or its distribution is unknown, it is not possible, or at least
very difficult, to find the Neyman-Pearson detector.

Katenka’s research involved an alternative method for improving detection in an array of sensors. Local vote decision fusion is when sensors decide for either H0 or H1 independently, after which they compare with the decisions made by neighboring sensors and alter their decision based on what more neighboring sensors chose. When this is done, the sensors send the decision to the network to decide collectively (Katenka, 2008). This was found to be cheaper and have better detection than ordinary decision fusion (Katenka, 2008). While this is not be used in the research discussed in this paper, it is significant to note that there are various ways to improve upon the standard model of sensor array detection.

Some researchers operate under the assumption that there is no signal decay from the emitter, and thus detection of an emitter that is within the range of a sensor is guaranteed. Because sensor detection is guaranteed, the goal of sensor placement in this case is to make sure all areas are covered by a pre-determined minimum number of sensors. Zhu’s paper discussed coverage and connectivity in wireless sensor networks (Zhu, 2012). The focus was on optimizing sensor networks to conserve energy through sensor deployment, sleep mode, and optimized sensing range (Zhu, 2012). While in most cases, the goal is to improve the detection of the system, conservation of energy is often an important factor as well. For this research, it will be assumed that energy consumption is not an issue.

Some research has been done into the optimal placement of sensors within a region. Tu considered systems of sensors using worst-case testing, an OR fusion rule, and binary sensors. An OR fusion rule is when a single sensor deciding for H1 is enough to decide for a signal detection. Worst-case testing is finding the detection if the emitter is in the worst possible
location for detection (Tu, 2009). In his research, he designed an algorithm for the efficient placement of sensors. This was done with an optimization rule using numerical methods (Tu, 2009). The focus of this algorithm was to optimize sensor placement to minimize power consumption within a one-dimensional system. It is significant to note that previous research has been done on sensor placement; however, Tu considered only a single dimension and focused on optimizing power consumption. This research focuses on optimizing sensor placement in two dimensions to maximize the worst-case probability of detection.

2.2 The Scan Statistic

One method that has been found to significantly improve detection in sensor arrays is known as the scan statistic. In the scan statistic, nearby sensors are combined into clusters. The values of the sensors within a cluster are then combined and compared against a threshold (Guerriero, 2009). The scan statistic allows single sensor errors without necessarily causing a false alarm, same as the sum fusion rule. However, it also only combines nearby sensors into a given cluster, avoiding the combination of weak signals from farther away sensors. This can lead to better overall detection for the system and allow for an increase in $P_D$ without increasing the $P_{FA}$.

The scan statistic is a very powerful tool for many applications, not just sensor arrays. Originally, the scan statistic was not used for signal detection. According to Kulldorff, the scan statistic was designed for detection of disease outbreaks (Kulldorff, 1997). For this application, the scan statistic was used to test if something was purely random or if clusters could be detected. In other words, the scan statistic originally focused on finding clusters that were
beyond some threshold. In this case, each individual measurement was considered to be in one of two states, with one state being normal and the other being anomalous (Kulldorff, 1997). Per Kulldorff, the scan statistic could be used to detect trees in forestry, stars in astronomy, geographical clusters in epidemiology, find mineral deposits, or find bird nests in zoological studies (Kulldorff, 1997). As can be seen, the scan statistic can be utilized for many purposes.

Neill discussed using the scan statistic for the detection of disease outbreak and brain activity scanning (Neill, 2006). He also wrote a fast algorithm that can be used both to detect spatial and spatio-temporal clusters (Neill, 2006). In other words, he was able to create faster algorithms for detection using the scan statistic both in space and in space and time. In his proposal, larger parent regions would be tested first, and child regions would be tested only if the parent region contained indication of a region of interest (Neill, 2006). The goal of this scan statistic was to detect areas of space where the test subject was higher than expected, while also avoiding false positives and false negatives. Neill focused on clustering adjacent regions through cluster detection (Neill, 2006). In general, he focused on a scan window of the same shape but varying size to test each potential cluster against a threshold (Neill, 2006).

Eventually, engineers began to consider the scan statistic as a viable solution for improving the detection of sensor arrays. In Arias-Castro’s article, clusters are defined as a subset of nodes, and classes of clusters are defined as being either geometric in shape or made from connected components (Arias-Castro, 2011). Using a geometric grid of sensors, various cluster shapes were compared for detection capabilities (Arias-Castro, 2011). This article demonstrates the power of scan statistic as well as the importance of cluster shapes.
Like Arias-Castro, Guerriero discussed detection in sensor networks using the scan statistic. In this case, the authors discussed using a moving window of a set size and shape that continually scanned against a threshold and attempted to find the optimum window size (Guerriero, 2009). The focus of the research was to be able to detect localized disturbances using scan statistic rather than assuming the entire system decided for either H0 or H1 (Guerriero, 2009). The sensors were initially assumed to be in a Bernoulli grid pattern, and then distributed randomly. Under each of these conditions, various calculations were completed, and Monte Carlo simulations were used to approximate the detection of the systems (Guerriero, 2009). This is another example of the scan statistic where rather than preset clusters, a moving window scans against a threshold. Guerriero’s use of the scan statistic is somewhat similar to that considered in this research, with the primary differences being this research allows for clusters of varying shapes and sizes and also considers static clusters.

Another researcher, Fonseca, considered the issue of optimizing detection using the scan statistic. Fonseca defines the scan statistic as follows: nearby sensors are grouped into clusters and the set of clusters form a cluster set. It is important to have enough clusters to keep detection high, however, too many clusters increase computational requirements as well as necessitates an increase in the threshold to keep the probability of false alarm low (Fonseca, 2019a). In Fonseca’s article, an algorithm to determine the optimal clustering of sensors for a given set of sensor locations is discussed. The procedure is based on maximizing the probability of detection at “points of interest”. Points of interest are the potential emitter locations where detection is the worst. This method accounts for varying cluster shapes, sizes, and locations, while also allowing sensors to belong to multiple clusters (Fonseca, 2019a). This algorithm was found to generally
improve upon other clustering methods, and a version of it is used in this research. Another researcher (Shulgan, 2020) proposed a heuristic procedure to optimize cluster sets for a given set of sensor locations. Shulgan’s procedure uses a local search algorithm that adds or removes cluster sets in an iterative procedure, searching for the best cluster set for a given set of sensor locations.

In another article, Fonseca discussed the design of sensor detection systems using the scan statistic with an unknown emitter location. Because of the unknown signal and unknown source location, the uniformly most powerful (UMP) detector cannot be easily found (Fonseca, 2019b). In this article, least favorable distributions were used to propose an upper bound for the probability of false alarm, as well as two lower bounds for the probability of detection. This is done by assuming the worst-case emitter location (Fonseca, 2019b). These bounds allow for more accurate analysis of the detection of scan statistic systems because there is no known equation to calculate the $P_D$ and $P_{FA}$ for systems that use the scan statistic. These bounds are used in this research as well and are discussed in detail in Section 6.2.

Some researchers have considered combining the scan statistic with other methods of improving detection. Luo did research on combining the scan statistic with Katenka’s local vote (Luo, 2017). Essentially, the system would undergo a local vote as was previously discussed by Katenka in (Katenka, 2008), and then the scan statistic would occur on the adjusted values. Luo considered the sensors to be in a grid pattern with no intersections. This means that each sensor belonged to only one cluster. Luo determined that having no intersections led to similar detection while greatly reducing time requirements when a local vote was also used (Luo, 2017). Luo also determined that this was better than either the scan statistic or local vote alone when used in a
system with a low signal-to-noise ratio (Luo, 2017). This is another example of improvement upon the scan statistic.

2.3 Numerical Methods

To solve the problem of optimizing sensor placement to maximize the worst-case \( P_D \), an optimization problem was considered. Per Boyd, the goal of an optimization problem is usually to minimize or maximize a certain optimizing multivariable function under constraints on the variables (Boyd, 2011). In this research, the goal is to maximize the worst-case probability of detection, the variables are the x and y coordinates of each sensor, and keeping the probability of false alarm below a certain value is the optimization constraint. In the case with the scan statistic, the cluster set is another variable to consider.

There are many forms of numerical methods that exist for various applications. A general case is convex optimization. Using this form, problems with convex optimizing functions and constraints can generally be solved with relative ease and speed (Boyd, 2011). One example of convex optimization is least-square minimization. In least-square minimization, there are no constraints and the goal is to find the sum of squares in a specific form (Boyd, 2011). For this application, convex optimization cannot be used because both the optimizing and the constraint functions are non-convex. It is sometimes possible to approximate a function as convex to allow for the use convex optimization, however that will not be considered in this research.

In this research, a local search algorithm is used. There are various versions of local search algorithms. The first that was be considered was that by Nelder and Mead (Lagarias,
In this algorithm, the point within the simplex that currently has the lowest worst-case $P_D$, i.e. the worst point, is moved with a goal of increasing the worst-case $P_D$ of this point. This is done by moving the point towards the average of all other points in the simplex. There are various potential step sizes which are chosen based on certain criteria. These steps are repeated until all points in the simplex converge to approximately a single point (Lagarias, 1998). This algorithm was found to be ineffective for this specific application and is discussed in section 3.1.

The next algorithm considered, and the algorithm that is used in this research, was that by Torczon. In this algorithm, all points in the simplex, except for the point in the simplex with the highest worst-case $P_D$, i.e. the best point, are moved towards the best point. There are various potential step sizes which are chosen based on the levels of improvement. These steps are repeated until all points in the simplex converge to a single point (Torczon, 1989). A version of this algorithm is used in the final version of both algorithms. Torczon’s algorithm is discussed in section 3.2.
CHAPTER 3
LOCAL SEARCH ALGORITHMS

In this section, various local search algorithms are discussed. First, in section 3.1, the algorithm by Nelder and Mead is analyzed with respect to this research. This algorithm is not the final algorithm used in this research; however, it is the basis for the algorithm that is used. Next, in section 3.2, the algorithm by Torczon is discussed. Torczon’s algorithm is used in all algorithms discussed in this research. This chapter will use the following notation: $L$ refers to a simplex, $L$ refers to a vertex in a simplex, and $l$ refers to an individual sensor location.

3.1 Nelder and Mead’s Algorithm

To solve this complex optimization problem, a local search algorithm was considered. There are many local search algorithms that have various purposes. The first algorithm considered was that by Nelder and Mead. This is a simplex local search algorithm, where the simplex is made up of $2K+1$ points, each of which are $2K$ dimensional, where $K$ is the number of sensors. Each of the $2K$ dimensions correspond to the x- or y-coordinates of a sensor. The goal of the algorithm is to move the points in ways that improve them until all points converge to the same point (Lagarias, 1998).

In this algorithm, the steps are based on the worst vertex (point). The worst vertex is the point, i.e. sensor positions, that result in the lowest worst-case $P_D$. In this section, the algorithm will be considered in a general form, in which an optimizing function $f(L)$ needs to be
maximized, where $L$ is a 2*K dimensional point. In subsequent chapters, $L$ will refer to the K sensor positions being optimized. This algorithm focuses on moving the worst point around the centroid, or average, of all other vertices in the simplex (Lagarias, 1998). However, there are various possible steps.

The first step is known as a reflection. In a reflection, the worst vertex is flipped across the centroid of the other points to be equidistant from the point but on the other side. This is done using equation (3.1)

$$L_r = 2 * L_A - L_w$$

In equation 3.1, $L_r$ is the point after reflection, $L_A$ is the average of the other points, and $L_w$ is the worst point (Lagarias, 1998).

From here, there are four possible steps. If $f(L_{sw}) < f(L_r) < f(L_B)$, where $L_{sw}$ refers to the second worst point in the simplex and $L_B$ refers to the best point in the simplex that maximizes the optimization function, then the reflection is accepted. Accepting a reflection means that the worst point in the simplex, $L_w$, is replaced with $L_r$ (Lagarias, 1998).

If $f(L_r) > f(L_B)$, i.e. a new best vertex is found, then an expansion is attempted. An expansion is similar to a reflection, but the step size is increased. Rather than being equidistant from $L_A$ but in the opposite direction, the expansion point, $L_e$, is double the distance from $L_A$. This is calculated using equation (3.2).

$$L_e = 2 * L_r - L_A$$
If $f(L_e) > f(L_r)$, then the expansion is accepted. Otherwise, the reflection is accepted (Lagarias, 1998).

If $f(L_w) < f(L_r) < f(L_{sw})$, i.e. the reflection is strictly greater than its original position, then an outside contraction is attempted. An outside contraction is similar to a reflection; however, the step size is decreased so that $L_{oc}$ is reflected across $L_A$ but only half the distance from it. This is found using equation (3.3).

$$L_{oc} = L_A + 0.5 \times (L_r - L_A) \tag{3.3}$$

If $f(L_{oc}) > f(L_r)$, then the outside contraction is accepted. If not, then a shrink step occurs (Lagarias, 1998).

If $f(L_w) > f(L_r)$, then an inside contraction occurs. An inside contraction is when $L_w$ moves toward but not past $L_A$, halving the distance between the two. This is done using equation (3.4).

$$L_{ic} = L_A - 0.5 \times (L_A - L_w) \tag{3.4}$$

If $f(L_{ic}) > f(L_w)$, then the inside contraction is accepted. If not, then a shrink step occurs (Lagarias, 1998).

If either an outside or inside contraction failed to improve the optimizing function $f(L)$, then a shrink step occurs. A shrink step is when all points, except for $L_B$, are moved toward $L_B$, halving the distance between them. This is done using equation (3.5) for every point $d$ in the simplex other than $L_B$.

$$L_{ss,d} = 0.5 \times (L_B + L_d) \tag{3.5}$$
In this equation, $L_{ss,d}$ is the shrink step of point $L_d$ (Lagarias, 1998). When a shrink step occurs, the new simplex becomes all $L_{ss,d}$ along with $L_B$.

The above steps are repeated until the stop condition is met. The stop condition is met when the maximum side length of the simplex is below some value. When the side lengths become close to zero, this means that the vertices are essentially converged to the same point. When this occurs, it means that further iterations will not significantly change the simplex vertices. Ideally, this means that a local maximum for the optimizing function has been found and the algorithm is complete (Lagarias, 1998).

The algorithm by Nelder and Mead was found to have a few significant problems that made it ineffective for this application. Due to the nature of local search algorithms, the result can never be guaranteed to be a true maximum for the optimizing function. Ideally, the points of convergence should be stationary points, whether they be points of maximum or saddle points. When using this algorithm for this application, it was found that the points of convergence were often not stationary points. This is likely due to the problem being non-convex. For this reason, even running the algorithm for many random drops was often not sufficient to find a reasonable point of convergence due to the sheer number of potential points of convergence, many of which were not stationary points. The algorithm by Nelder and Mead is most effective when used with a convex problem, so it was determined to be ineffective for this application.
3.2 Torczon’s Algorithm

After determining the algorithm by Nelder and Mead was not suited for this application, a variation on it by Torczon was considered. Like the algorithm by Nelder and Mead, Torczon’s algorithm is a simplex local search algorithm. In the algorithm by Nelder and Mead, the step sizes could vary depending on the steps taken, but not the step directions. The algorithm by Torczon allows for both various step sizes and directions to be considered. This allows for better convergence in many applications (Torczon, 1989).

Like the algorithm by Nelder and Mead, the first step is to create the initial simplex of 2*K+1 points that are 2*K dimensional. For this algorithm, there were some additional specifications for the initial simplex. Rather than a purely random initial simplex, it was desirable to start with an orthogonal simplex, or a simplex where the vertices are all separated by 90 degrees. The center point is still random; however, all other points were formed by altering a single dimension of the center point (Torczon, 1989).

Testing was also done on initial sensor locations that were not purely random, but instead were approximately a standard grid with some slight perturbation. This however was not found to improve the output of the algorithm. Testing was also done on the side length of the initial simplex. In general, larger simplexes require more contractions because the simplex must converge to a single point. Smaller simplexes generally require more reflections and expansions since there may not be a local maximum for the worst-case P_D contained within the initial simplex. Overall, the results were similar for any simplex size.
In Torczon’s algorithm, the goal of every step is to find a new best vertex, $L_B$. To this end, rather than moving the worst point around the average of all other points, instead all points are moved around the current best point. It is reasonable to guess that if $L_B$ is the best point, moving all other points, $L_o$, in the direction of $L_B$ has a chance of improving $f(L)$. This step also allows for the change of multiple vertices at once, as well as multiple search directions (Torczon, 1989). Figure 3 shows the potential steps of Torczon’s algorithm when $K=1$. This figure assumes that the $x$ in the center of the cross is the current best point in the simplex and the circles represent potential steps the algorithm could take.

![Figure 3: Potential steps of Torczon’s algorithm when $K=1$.](image)

The first step of Torczon’s algorithm is also a reflection. In all steps, $L_B$ is removed from the simplex. In a reflection, all points in $L_o$ are reflected across $L_B$ to be equidistant but on the opposite side of $L_B$. This is done with the following equation (3.6): for each point $L_{o,d}$ remaining in the simplex,

$$L_{r,d} = 2 \ast L_B - L_{o,d}$$  \hspace{1cm} (3.6)

From here, there are two possible steps. If $\max_{d=1,\ldots,2K}(f(L_{r,d})) > f(L_B)$, i.e. any of the points of reflection are the new best point, an expansion is attempted. An expansion is like a
reflection but the step size is increased so all points in $L_o$ are reflected across $L_B$ but end up twice the distance from $L_B$ on the other side. This is done with the following equation (3.7): for each point $L_{o,d}$ remaining in the simplex,

$$L_{e,d} = L_B + 2 \times (L_B - L_{o,d})$$  \hspace{1cm} (3.7)

If $\max_{d=1,...,2K} f(L_{e,d}) > \max_{d=1,...,2K} f(L_{r,d})$, i.e. the expansion improved upon the reflection, then the expansion is accepted. Accepting an expansion means that the new simplex consists of all points in $L_e$ as well as $L_B$. If not, the reflection is accepted, and the new simplex consists of all points in $L_r$ as well as $L_B$ (Torczon, 1989).

If $\max_{d=1,...,2K} f(L_{r,d}) < f(L_B)$, a contraction is attempted. In a contraction, all points in $L_o$ are moved toward but not past $L_B$ so that the distance between them is halved. This is done using the following equation (3.8): for each point $L_{o,d}$ remaining in the simplex,

$$L_{c,d} = L_B + 0.5 \times (L_{o,d} - L_B)$$  \hspace{1cm} (3.8)

Since this is the last of the steps, a contraction is accepted whether or not a new best point is found. Accepting a contraction means that the new simplex consists of all points in $L_c$ as well as $L_B$ (Torczon, 1989).

These steps are then repeated until the stop condition is met. The stop condition for Torczon’s algorithm is slightly different than that used in the algorithm by Nelder and Mead. In this case, the stop condition is met when

$$\frac{\max_{d=1,...,2K} \|L_{o,d} - L_B\|}{\max(1,\|L_B\|)} < \varepsilon$$  \hspace{1cm} (3.9)
where \( \| \ldots \| \) refers to the Euclidean norm and \( \varepsilon \) is a very small number. This stop condition measures the relative simplex size in relation to the longest side length (Torczon, 1989). This stop condition was found to be more effective than just comparing the distance to a threshold.

For the stop condition of equation (3.9), various values were tested, and it was found that \( \varepsilon = 10^{-9} \) was an effective stop condition. Any value of \( \varepsilon < 10^{-9} \) had increased run time but only very minor changes in the final worst-case \( P_d \). At the same time, any value of \( \varepsilon > 10^{-9} \) reduced the run time, but also greatly reduced the final worst-case \( P_d \) in some cases. For very large \( K \), this was decreased to \( \varepsilon = 10^{-6} \) to have more reasonable run times.

Additionally, testing was done on perturbation of the final point of convergence. Specifically, the final point of convergence would have another orthogonal simplex built around it and the algorithm would start over. This would either happen five times or until the algorithm returned to the same point. The goal of this was to try escaping the local maximum to find the true maximum. The problem with this approach is the very slight difference between certain maxima. This led to the algorithm not always able to escape the local maximum. Additionally, there are many potential points of convergence, so perturbation often just jumped between points without necessarily improving upon the previous point. For these reasons, perturbation was not used in the final algorithm.

Overall, Torczon’s algorithm was found to be much more effective for this application than that by Nelder and Mead. First, it is more simplistic as it only has three possible steps rather than five. This simplicity generally led to better run times. Also, Torczon’s algorithm was found to be much more effective at obtaining stationary points. Torczon’s algorithm can obtain a stationary point at any point that is differentiable (Torczon, 1989). While the function is not
differentiable at all points, it is at most points. Occasionally the point of convergence will not be a true stationary point, however the odds of this are significantly lower than in Nelder and Mead’s algorithm. This allows for less random initial simplexes. For these reasons, Torczon’s algorithm was chosen.

It can be shown that there are certain points at which the function is non-differentiable. This occurs at points where a change in sensor positions leads to a change in the worst emitter location. This change in worst emitter location can lead to a sudden change in the worst-case $P_D$, which in turn leads to a point that is non-differentiable because the function is not continuous. It is possible for the algorithm to converge to points such as this because the algorithm is only effective at points that are differentiable. However, the algorithm is generally still able to obtain stationary points most of the time.
CHAPTER 4

MODEL DESCRIPTION

In this section, a general description of the model that is considered throughout this paper will be discussed. This model applies to both the sum fusion rule algorithm, discussed in Chapter 5, and the scan statistic fusion rule algorithm, discussed in Chapter 6.

Consider that the region of interest, $S_e$, is a two-dimensional space. Recall that the system decides for either: H0, meaning that $S_e$ does not contain an emitter; or H1, meaning that there is an emitter present somewhere within $S_e$. It is assumed that the emitter location is random with an unknown distribution, which motivates the consideration of the worst-case location for the emitter. This research considers a single emitter, which can be considered a worst-case for the detection system.

To detect the emitter, $K$ sensors are distributed throughout the region at locations $\{l_k\}_{k=1}^K$. This set of sensors is referred to as $L$. Each sensor collects a measurement $Z_k$. Under H0, assume that $Z_k \sim N(0, 1)$, where $N(\mu, \sigma^2)$ represents a Gaussian distribution with mean $\mu$ and variance $\sigma^2$. Under H0, $\{Z_k\}_{k=1}^K$ are conditionally independent and identically distributed (i.i.d.). Under H1 and conditioned on a location $l_e$ for the emitter, assume that $Z_k \sim N(A/\|l_k - l_e\|^2, 1)$, where $A$ is the emitter signal amplitude and $l_k$ is a sensor location. Under H1, the $\{Z_k\}_{k=1}^K$ are conditionally dependent since they depend on the distance between $l_k$ and the random location $l_e$; however, when conditioned on any location $l_e$, we assume that $\{Z_k\}_{k=1}^K$ are conditionally
independent. Each sensor, $K$, transmits the received $Z_k$ to the fusion center using a dedicated, error-free, communication channel. The fusion center then decides for either H0 or H1 based on the received $Z_k$ values.

In this research, two fusion rules are considered: the sum fusion rule and the scan statistic fusion rule. The specifics of the sum fusion rule are discussed in Chapter 5, while the scan statistic fusion rule is discussed in Chapter 6. In either case, the goal is to maximize the worst-case probability of detection ($P_D$) while keeping the probability of false alarm ($P_{FA}$) below a maximum value. Because the emitter location and its distribution are unknown, this research takes the conservative approach and designs the system based on the worst-case $P_D$. However, finding the worst-case $P_D$ is difficult because it is not trivial to find the worst-case location for the emitter. To address this, a lower bound for the worst-case probability of detection will instead be considered.
CHAPTER 5
ALGORITHM FOR THE SUM FUSION RULE

This chapter discusses the initial version of the algorithm that considers the sum fusion rule. This algorithm seeks to find the optimal spatial locations for a set of sensors within a region of interest. The goal is to maximize the worst-case probability of detection, which is the probability of detection if the emitter were in the worst possible location.

5.1 Methodology

The goal of this algorithm was to use Torczon’s local search algorithm to find the optimal sensor spatial positions to maximize the worst-case \( P_D \) of the system. To do this, the x- and y-coordinates of each sensor are the variables to be optimized by the algorithm, leading to a 2*K dimensional system, where K is the number of sensors. The algorithm requires the creation of a simplex of 2*K+1 vertices, each of which are 2*K dimensional. The initial simplex is also made to be orthogonal and the center is constrained to be within the region of interest. The constraining variables are the number of sensors, the size of the region of interest, the number of subregions, and the maximum acceptable \( P_{FA} \). The specific values chosen for each of these are discussed in the section 5.3.

Recall from section 3.2 that Torczon’s algorithm is a local search algorithm that attempts to maximize a function by trying multiple points. In this application, Torczon’s algorithm tries
various sets of sensor locations, seeking for the set of sensor locations that maximizes the worst-case probability of detection.

In the system considered in this research, the probability of detection for a given emitter location and set of sensor locations can be found using the following equation (5.1):

\[
P_D = Q \left( \frac{1}{\sqrt{K}} \left( t - \sum_{k=1}^{K} \frac{A}{\|t_k - l_e\|^2} \right) \right)
\]  

(5.1)

where \( Q \) is complimentary cumulative distribution function of the Gaussian random variable with mean 0 and variance 1 and \( t \) is the threshold. This threshold is calculated with the following equation (5.2):

\[
t = \sqrt{K} \cdot Q^{-1}(P_{FA}^{max})
\]  

(5.2)

where \( P_{FA}^{max} \) is the maximum acceptable probability of false alarm.

To determine the worst-case probability of detection, it is necessary to determine the worst-case emitter location for a given set of sensor locations, \( L \). However, it is difficult to find the worst-case emitter location as it depends on the distance from each sensor to the emitter. Instead, a lower bound for the worst-case \( P_D \) is considered. More details on this lower bound can be found in section 5.2.

5.2 Worst-Case \( P_D \) Lower Bound for the Sum Fusion Rule

In this section, the worst-case \( P_D \) lower bound for the sum fusion rule will be discussed. To find this lower bound, the region of interest, \( S_e \), is divided into smaller, equally sized subregions, \( S_r \) (Fonseca, 2017). Calculations are still difficult because \( P_D \) depends on the
distance from each sensor to the emitter location, so finding the maximum total distance is a time-consuming calculation.

To solve this, it will be assumed that each sensor has a separate, independent emitter that are all confined to the same subregion (Fonseca, 2020). In this approximation, each sensor can only detect its own respective emitter. As the subregion size decreases, this becomes approximately the same as having a single emitter within the subregion. For each sensor, the emitter is assumed to be at the maximum possible distance from the sensor while remaining within the subregion. It is worth noting that for any rectangular subregion within a rectangular region of interest, the maximum distance from a sensor will always be a corner of the subregion. This means that it is only necessary to check the total distances for the corners of each subregion. This allows for a simple distance calculation from each sensor to the respective emitter location, greatly simplifying the computational requirements. This is shown in Figure 4 below. In this figure, the blue circles represent sensors, the red square a subregion, and the black arrows the maximum distance to a point in the subregion for each sensor.

Through these approximations, an infinite problem is reduced to a simpler, more solvable problem. All these approximations lead to the calculated worst-case $P_D$ being slightly lower than
the true worst-case $P_D$, i.e. a lower bound for the worst-case $P_D$. This is because each sensor considers an emitter location that is the maximum distance away within the subregion, while in reality the emitter will only have one location which is often not the maximum distance from all sensors. This is important because it means that any $P_D$ within the region of interest will be above this value, thus creating an accurate lower bound.

5.3 Results

In order to test the effectiveness of Torczon’s algorithm in optimizing sensor locations, the maximum $P_{FA}, P_{FA}^{max}$, was set to be 0.05. The region of interest was set to $S_e = [0,10]^2$ and divided into $R=100$ subregions, $\{S_r\}_{r=1}^R$. In other words, the region of interest was a 10-by-10 region divided into 100 equally sized subregions of size 1-by-1. Given that the measurement $Z_k \sim N(0,1)$ from each sensor is conditionally i.i.d. under $H_0$, the $P_D$ can be determined from equation (5.1). In this case, the threshold is only dependent on the number of sensors, not their locations within the region, shown in equation (5.2). Because of this, the threshold only needs to be calculated a single time when running the algorithm for a specific $K$ value. For a given set of sensor locations $L = \{l_k\}_{k=1}^K$, the worst-case $P_D$ lower bound was computed as discussed in Section 5.2.

Using the described setup, the algorithm was run with a goal of maximizing the worst-case $P_D$ of the system for sensors varying from $K=1$ to $K=9$ sensors. For each $K$ value, the algorithm was run for 100 random initial simplexes and the best solution was kept. The goal of multiple random drops was to try finding the global maximum for the worst-case $P_D$. The
emitted signal amplitude (A) for each K value was chosen so that the worst-case $P_D$ when using a standard grid pattern with the same number of sensors was approximately 0.7.

Let $L_{f1}, \ldots, L_{f100}$ represent the final set of sensor locations at the end of each of the 100 random initializations. Let $L^+$ be the best set of sensor positions among all $L_{fi}$. Let $S_r^-$ be the subregion with the lowest $P_D$ among all subregions, and let $S^-$ be a list of subregions that contains $S_r^-$ as well as other subregions with poor detection for a given $L$. Figure 5 shows the distribution of $P_D(L_{fi}, S^-)$ when K=4 over R=100 random initial drops. As can be seen, there is a very wide spread of $P_D(L_{fi}, S^-)$ over the different initial drops. This shows that there are many potential points of convergence and demonstrates the importance of multiple random initial starts. It can also be seen that for the case of K=4, many of the initial starts converged to a similar point which is around the best point. For different K values, the spread of $P_D(L_{fi}, S^-)$ is similar. For larger K values, the peak moves to the left as optimizing the sensor locations becomes less common.

![Figure 5: Histogram of $P_D(L_{fi}, S^-)$ when K=4.](image)

To understand why the algorithm stops when it does, it is important to remember that the $P_D$ depends on the distance between the emitter and sensors. Also, moving any sensor in any
direction will simultaneously increase $P_D$ in some subregions, while decreasing it in others. For these reasons, the algorithm tends to stop when the sensors are in a position where moving any sensor in any direction will lower the worst-case $P_D$. This algorithm essentially tries to balance the $P_D$ in $S^-$. This often presents as symmetry in both the sensor positions and worst subregions. Also, it is generally expected that the corners of $S_r$ will be among the worst subregions if the sensors are in an optimal location. This is because the corners are generally near to fewer sensors, thus having lower detection.

In the case where $K=1$, the algorithm always converged to a single sensor in the center of $S_e$, at (5,5), with the worst subregions being the corners of $S_e$. This makes sense because with a single sensor in the center, the worst subregions would be the corners of the region, all of which would be equidistant from the sensor, and thus have the same probability of detection.

The best results obtained through 100 loops for $K=2$ to $K=9$ are shown in Figures 6 through 13. These figures show the sensor locations, the locations of $S^-$, as well as the lower bound for the worst-case $P_D$ in these subregions.

Figure 6 shows the best output for $K=2$. As can be seen, both $L$ and $S^-$ are horizontally and vertically symmetrical. This result is like what was expected for this case. When considering a square region of interest, it is expected that the final sensor positions could be rotated 90 degrees with no change to $P_D(L^+,S^-)$. $S^-$ would similarly rotate 90 degrees in this case. This is due to the symmetry of the problem.
Figure 6: Best sensor positions obtained from 100 loops when K=2, as well as $P_D(L^+, S^-)$.

Figure 7 shows the best output for K=3. In this case, $L$ is symmetrical on the diagonal, while $S^-$ consists of the four corners. Due to the symmetry, this result is reasonable. In this case, the result is very different than a grid pattern.

Figure 7: Best sensor positions obtained from 100 loops when K=3, as well as $P_D(L^+, S^-)$.

Figure 8 shows the best output for K=4. As can be seen, both $L$ and $S^-$ are horizontally and vertically symmetrical. This result is like what was expected for this case. The slight differences in the subregions within $P_D(L^+, S^-)$ show that more random drops could potentially increase $P_D(L^+, S^-)$ further, however this amount would likely be below the precision of the algorithm.
Figure 8: Best sensor positions obtained from 100 loops when $K=4$, as well as $P_D(L^+, S^-)$.

Figure 9 shows the best output for $K=5$. In this case, $L$ is close to horizontally symmetrical, but not quite. While not symmetrical, $S^-$ consists of three of the corners and some edges of the region of interest. This is likely not the best final positions for the sensors. This is because it is expected that the best sensor positions and worst subregions should be symmetrical if they were placed optimally.

Figure 9: Best sensor positions obtained from 100 loops when $K=5$, as well as $P_D(L^+, S^-)$.

Figure 10 shows the best output for $K=6$. In this case, $L$ is not symmetrical, however the positions are near to vertically symmetrical. While not symmetrical, $S^-$ consists of three of the corners and some edges of the region of interest. This is likely not the best final positions for the sensors due to the lack of symmetry.
Figure 10: Best sensor positions obtained from 100 loops when $K=6$, as well as $P_D(L^+, S^-)$.

Figure 11 shows the best output for $K=7$. In this case, $L$ is not symmetrical, however the positions are near to vertically symmetrical. While not symmetrical, $S^-$ consists of three of the corners and some edges of the region of interest. This is likely not the best final positions for the sensors due to the lack of symmetry and the bottom left corner being excluded from the set of worst subregions $S^-$. 

Figure 12 shows the best output for $K=8$. In this case, $L$ is near to vertically and horizontally symmetrical. While not symmetrical, $S^-$ consists of all corners and some edges of the region of interest. This is unlikely to be the best point, due to the lack of symmetry in both the sensor placement and $S^-$ being more heavily concentrated in the right half of the region.
Figure 12: Best sensor positions obtained from 100 loops when K=8, as well as $P_D(L^+, S^-)$.

Figure 13 shows the best output for K=9. In this case, $L$ is not symmetrical, however the positions are somewhat near to vertically and horizontally symmetrical. While not symmetrical, $S^-$ consists of all corners and some edges of the region of interest. There is also one subregion contained within $S^-$ that is located around (6,6). It is very likely that this is not the optimal placement for the sensors due to the lack of symmetry in both the sensor placement and $S^-$. 

Figure 13: Best sensor positions obtained from 100 loops when K=9, as well as $P_D(L^+, S^-)$.

Figure 14 shows the worst $L$ obtained through the 100 loops when K=9, i.e. the $L$ with the lowest $P_D(L_{fl}, S^-)$. As can be seen, $P_D(L_{fl}, S^-)$ is significantly lower than in Figure 13. There is no symmetry in $L$ or in $S^-$. Many of the sensors are very close to one another, leaving large areas with poor coverage. This further demonstrates the importance of multiple random initial drops.
Figure 14: Worst sensor positions obtained from 100 loops when $K=9$, as well as $P_D(L_{f,t}, S^-)$.

As can be seen in Figures 6 to 13, $S^-$ tends to contain the corners of $S_e$, as well as some subregions along the edges of the region. This makes sense because the edges and corners are generally farther from the sensor positions, and thus have lower $P_D$ than elsewhere. In certain cases, primarily when $K=2$ and $K=4$, a greater amount of symmetry can be found in both $L$ and $S^-$. Based on the symmetry of the problem, it is expected that both $L$ and $S^-$ will have horizontal, vertical, or diagonal symmetry in the best sensor positions.

It is worth noting however that, for most cases, the $L$ and $S^-$ are not quite symmetrical. For example, when $K=6$, the upper right corner has a higher $P_D(L^+, S_r)$ than the other corners of $S_e$, where $L^+$ is the best obtained sensor positions. In this case, moving $l_2$ and $l_5$ toward the center of $S_e$ would slightly lower $P_D(L^+, S_r)$ in the upper right corner, but slightly increase $P_D(L^+, S_r)$ in the other corners, thus increasing $P_D(L^+, S_{r}^-)$. However, these sensors are very far from those corners, so the change in $P_D(L^+, S_{r}^-)$ would be minimal. Similarly, when $K=9$, moving $l_5$ toward the center would increase $P_D(L^+, S_r)$ in the subregion near to (6,6). However, the change in $P_D(L^+, S_{r}^-)$ would be negligible. It can be concluded that in most cases, additional random initial starts of the algorithm could slightly improve the symmetry, and thus increase $P_D(L^+, S_{r}^-)$, however the change would most likely minimal.
5.4 Algorithm Results Relative to a Standard Grid Placement

To demonstrate the success of the algorithm, the obtained results were compared with those from a standard grid. When K=2, the sensors were placed in a horizontal line with y=5 and x-coordinates of 2.5 and 7.5. When K=4, K=6, and K=8, a grid containing two rows of sensors and y-coordinates of 2.5 and 7.5 was used. The x-coordinates were spaced by $\frac{10}{K/2}$ from each other and $\frac{10}{K}$ from the edges. When K=3, K=5, and K=7 the sensors were placed the same as in the K-1 case but with the extra sensor placed in the center at (5,5). Lastly, when K=9, the sensors were placed in a 3-by-3 grid with x-coordinates and y-coordinates among 1.66, 5, and 8.33.

Using the above sensor positions, $P_D(L^+,S_r^-)$ obtained through the algorithm was compared with $P_D(L_{\text{Grid}},S_r^-)$, which is the lower bound for the worst-case $P_D$ of the grid pattern with the same number of sensors. The results are shown in Table 1. At a minimum, the algorithm improved the worst-case $P_D$ from about 0.708 to 0.758 when K=2. At a maximum, the algorithm improved the worst-case $P_D$ from about 0.715 to 0.942 when K=8. In all cases, the algorithm produced a higher value for $P_D(L^+,S_r^-)$ than a standard grid pattern as long as enough random drops were run. The increase factor row shows the ratio between $P_D(L^+,S_r^-)$ and $P_D(L_{\text{Grid}},S_r^-)$. Note that this value is relative to the baseline detection for the grid pattern. Figure 15 plots $\Delta P_D$ which is the difference between the best $P_D(L^+,S_r^-)$ obtained by the algorithm and the grid $P_D(L^+,S_r^-)$. This further demonstrates the general increase in $P_D$ that the algorithm produces for K≤ 9.
Table 1

<table>
<thead>
<tr>
<th>K</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
<td>59</td>
<td>36</td>
<td>35</td>
<td>28</td>
<td>27</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>$P_D(L_{grid}, S_r^{-})$</td>
<td>0.708</td>
<td>0.707</td>
<td>0.702</td>
<td>0.712</td>
<td>0.713</td>
<td>0.702</td>
<td>0.715</td>
<td>0.706</td>
</tr>
<tr>
<td>$P_D(L^{-}, S_r^{-})$</td>
<td>0.54</td>
<td>0.505</td>
<td>0.275</td>
<td>0.409</td>
<td>0.405</td>
<td>0.536</td>
<td>0.476</td>
<td>0.474</td>
</tr>
<tr>
<td>$P_D(L^+, S_r^{-})$</td>
<td>0.758</td>
<td>0.818</td>
<td>0.871</td>
<td>0.916</td>
<td>0.887</td>
<td>0.925</td>
<td>0.942</td>
<td>0.899</td>
</tr>
<tr>
<td>$\Delta P_D$</td>
<td>0.05</td>
<td>0.111</td>
<td>0.169</td>
<td>0.204</td>
<td>0.174</td>
<td>0.223</td>
<td>0.227</td>
<td>0.193</td>
</tr>
<tr>
<td>Increase Factor</td>
<td>1.07</td>
<td>1.16</td>
<td>1.24</td>
<td>1.29</td>
<td>1.24</td>
<td>1.32</td>
<td>1.32</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Figure 15: Increase in worst-case $P_D$ over a grid pattern with sum fusion rule.

5.5 Analysis

For the sum fusion rule, the optimization of sensor locations using Torczon’s algorithm provided a significant improvement in the lower bound for the worst-case $P_D$. Compared with a grid pattern, the algorithm was able to improve $P_D(L^+, S_r^{-})$ by up to nearly 23% with the specific scenario that was tested. Even in cases with less improvement, such as $K=2$, $P_D(L^+, S_r^{-})$ still improved by about 5%. For these parameters, it appears that as $K$ increases, so too does the amount that $P_D(L^+, S_r^{-})$ can be improved. It is also reasonable to assume that due to the nature of
local search algorithms, the results obtained are likely not the true optimum. This means that running the algorithm more times could potentially lead to an even greater increase in $P_D(L^+, S^-)$.

There are some downsides to the algorithm. First, there is a limitation on the number of sensors that can be reasonably optimized. While the algorithm can be run for any number of sensors, each additional sensor increases the run time. This is because each sensor adds two more dimensions to the system, as well as two more points to the simplex. Because of this, each random initial starting point requires more iterations to converge. Additionally, as $K$ increases, the number of potential convergence points within the algorithm also increases. For this reason, more random drops are required to improve the odds of finding a true maximum for $P_D(L^+, S^-)$ for a large number of sensors.

Another downside is the fact that local search algorithms are never guaranteed to find a true maximum. For this reason, it can never be guaranteed that the algorithm has reached the ideal sensor placement. However, with enough random initializations, the algorithm has improved odds of finding a point of convergence that provides improvement over the standard grid pattern.

Overall, the algorithm is very effective at finding good points of convergence for the sum fusion rule in systems up to $K=9$. The algorithm can run for greater than $K=9$, however the run time to obtain good results becomes significantly larger. When $K=9$ or less, 100 random drops was enough to find a point of convergence that is better than the standard grid pattern. With more random drops, it is likely that even more improvement could be achieved.
CHAPTER 6

ALGORITHM WITH THE SCAN STATISTIC

This chapter discusses the second version of the algorithm. This version of the algorithm also uses Torczon’s algorithm to optimize sensor locations, but it considers a system that uses the scan statistic fusion rule. The scan statistic fusion rule decides for $H_1$ if

$$\max_{C \in C_S} (T(Z_C)) > t$$

(6.1)

where $t$ is the threshold that is adjusted to satisfy $P_{FA} < P_{FA}^{max}$. $Z_C$ is a vector of random variables defined as $Z_C := \{Z_k : k \in C\}$, $C_S$ is the set of clusters, and $T(Z_C)$ is the cluster statistic, given by equation (6.2) (Fonseca, 2019b)

$$T(Z_C) = \frac{\sum_{k \in C} Z_k}{\sqrt{|C|}}$$

(6.2)

Since the scan statistic involves grouping sensors into clusters, this algorithm seeks to jointly optimize the spatial locations and the cluster set $C_S$ for a set of sensors within the region of interest. The goal is to maximize the probability of detection lower-bound, which is the probability of detection if the emitter were in the worst possible location, while keeping the probability of false alarm below a maximum value.
6.1 Methodology

When considering the scan statistic, it is important to not only optimize the sensor positions, but to choose a good cluster set as well. To do this, an iterative approach was used. In brief, the procedure iteratively optimizes sensor locations while keeping clusters fixed; and then optimizes the clusters while keeping the sensor locations fixed. The procedure is then repeated until no improvement can be found. The iterative procedure is illustrated in the flowchart of Figure 16. Like the case with the sum fusion rule, this version of the algorithm also considers a lower-bound for the worst-case $P_D$ as the performance metric. This lower bound is discussed in more detail in section 6.2.

Figure 16: Algorithm flow chart.
The scan statistic version of the algorithm focuses on not only optimizing the sensor locations $L$, but also the cluster set $C_s$. The first step of the algorithm is the same as in the sum fusion rule case, where an initial list of random sensor positions, $L$, is chosen. From here, an initial cluster set is formed for this initial $L$, and the initial $P_{DLB}$ is calculated.

Like the sum fusion rule case, the scan statistic algorithm also uses Torczon’s local search algorithm. One limitation of local search algorithms for this application is that when a larger number of sensors is considered, the number of potential solutions and run time also increase. To test the scan statistic in a realistic setting, it was necessary to consider systems with a larger number of sensors. For these reasons, it was not viable to directly run Torczon’s algorithm on the entire system. In order to reduce the number of sensors considered at a time, Torczon’s algorithm was instead run on the sensors within a single cluster, $C^-$, and all other sensors were considered a constant for this iteration. This means that the simplex will not be $2^*K$ dimensional, but $2^*|C^-|$ dimensional, where $|C^-|$ is the number of sensors in the cluster of interest. The cluster to be optimized was chosen to be the best cluster for the worst point of interest (POI). Points of interest are a group of points that are guaranteed to contain the worst-case emitter location among them. Points of interest are discussed in more detail in section 6.2. This specific cluster is chosen because these sensors are likely nearest to the worst possible emitter location, so moving them will have the greatest effect on the optimization function. For this version of Torczon’s algorithm, the optimization function is a lower bound of the worst-case $P_D$ of the scan statistic ($P_{DLB}$) defined in (Fonseca, 2019b). This lower bound is discussed in section 6.2. Note that during an iteration of Torczon’s algorithm, $C_s$ does not change. Also note
that the initial simplex is still made to be orthogonal and the sensors are constrained to be within the region of interest.

After a successful iteration of Torczon’s algorithm, the $P_{DLB}$ for the new sensor positions and the previous $C_s$ are saved. With this done, the clustering algorithm is run on the new sensor positions. The clustering algorithm works as follows. An expanding circle is placed on each POI. Whenever a new sensor is crossed, a cluster set containing all sensors within the circle is added to the set. This is done for each POI, forming $C_s$. Note that this allows for intersecting clusters, but not repeat clusters. Additionally, after each iteration of the clustering algorithm, it is important to recalculate the decision threshold at the fusion rule using equation (6.4). This is because the threshold depends on the number of clusters in $C_s$ (Fonseca, 2019a).

There are three possible next steps. The first potential next step occurs when the cluster set does not change. If this happens, it is assumed a stationary point has been reached and the algorithm is complete. Second, if the cluster set changes but $P_{DLB}$ decreases, then the algorithm is complete. This is because it was found that when the clustering algorithm lowered $P_{DLB}$, the $P_{DLB}$ never went above this value again, so adding this stop condition improved run time without affecting the results. In this case, the current $L$ and previous $C_s$ are kept since the new $C_s$ had a lower $P_{DLB}$. Lastly, if the cluster set changes and this improves $P_{DLB}$, then Torczon’s algorithm is run again on the new best cluster for the worst POI. This is repeated until one of the above stop conditions is met.

Like the sum fusion rule case, there is no guarantee that the global optimum for $P_{DLB}$ will be obtained through a single random drop. This is due to the use of a local search algorithm and
the large number of local minima that are possible. To circumvent this, multiple random drops can be considered, and the best result of the drops kept as the solution.

The probability of detection for a given emitter location \( l_e \) and an individual cluster can be computed using equation (6.3)

\[
P[T(Z_c) > t \mid H1, l_e] = Q\left(\frac{t - \sum_{k=0}^{A} \frac{A}{\sqrt{|c|}}}{{\|k-l_e\|}^{2}}\right)
\]  

(6.3)

where \( Q \) is the complimentary cumulative distribution function of the Gaussian random variable with mean 0 and variance 1 (Fonseca, 2019b). Note that adding more sensors to a cluster can potentially decrease \( P[T(Z_c) > t \mid H1, l_e] \), so it is important that clusters only contain nearby sensors with strong measurements. Specifics on how to compute the probability of detection lower bound for a cluster set will be discussed in section 6.2.

6.2 Probability of Detection Lower Bound

Like the case with the sum fusion rule, this version of the algorithm also considered a lower-bound for \( P_D \), which is referred to as \( P_{DLB} \), representing a lower bound for the \( P_D \) when the emitter is at the worst possible location. However, there are two important differences when considering the scan statistic:

1) As in the sum fusion rule, the emitter location is unknown, and the desire is to find the worst possible emitter location for a set of sensor locations. It is possible to find a lower bound for the worst-case \( P_D \) for the scan statistic fusion rule by using subregions (Fonseca, 2019b); however, the procedure proposed in (Fonseca, 2019b)
requires an iterative procedure on its own to improve on the lower bound. Such an iterative procedure would have to be used whenever evaluating a point in the Torczon’s algorithm, which would result in a significant increase in the processing time required for optimization. To keep the optimization time low, this research considered the alternative lower bound for the worst-case $P_D$ presented in (Fonseca 2020). In this alternative lower bound, we consider that the emitter can only occur within a finite set of points of interest (POIs). This set of POIs represent a set of possible emitter locations that are the hardest to detect. Given the fact that the emitted signal decays by the distance to each sensor, such locations would be the farthest locations from the sensor locations. To obtain this set of locations, a Voronoi diagram is obtained from the current sensor locations. The Voronoi diagram produces $K$ polygons, and each polygon has a sensor location in its center. The set of POIs is then formed by the corners of each of the polygons formed. This reduces the problem to a finite number of points when computing the $P_{DLB}$. Figure 17 illustrates the set of POIs obtained for a set of sensor locations. Note that the set of POIs change for each iteration of the algorithm since the sensor locations change.

Figure 17: POI diagram (Fonseca, 2020).
While the sum fusion rule allows a closed-form expression to compute the \( P_D \) for a given emitter location, there are no closed-form expressions to compute the \( P_D \) for a given emitter location when using the scan statistic. This is because the clusters are overlapping, meaning that \( T(Z_C) \) are dependent random variables. To obtain a closed-form expression that can be evaluated at each step of the optimization procedure, this research considers the lower bound for \( P_D \) proposed in (Fonseca, 2019b): instead of considering all clusters, this research considers non-intersecting cluster sets, one for each POI, when computing the \( P_D \) for a given emitter location. Let \( C_{NI} \) be a non-intersecting cluster set for a POI. The lower bound for the \( P_D \) when the emitter location is at the POI is given by

\[
P_D = 1 - \prod_{C \in C_{NI}} P[T(Z_C) \leq t | H1, l_e = \text{POI}]
\]  

(6.4)

It is important to note that this value depends on the emitter location, sensor locations, threshold, and cluster set (Fonseca, 2019b).

To form these non-intersecting cluster sets (\( C_{NI} \)), the POIs are again considered. For each POI, \( P[T(Z_C) > t] \) is calculated for each cluster individually. These are then sorted from highest to lowest \( P_{DLB} \). From here, clusters are added to the cluster set going from best to worst and not allowing the addition of intersecting clusters (Fonseca, 2020). This results in a \( C_{NI} \) that should have a high \( P_D \).

In summary, to find the lower bound for the worst-case \( P_D \), the POIs and these non-intersecting cluster sets are considered. For each POI, there is an associated \( C_{NI} \). The equation (6.3) for the \( P_D \) of a cluster set and specific emitter location is then used with each POI and their
associated $C_{NI}$. The $P_{DLB}$ is then found by selecting the minimum $P_D$ from this list (Fonseca, 2019b).

6.3 Results

For the case that uses the scan statistic, the $P_{FA}^{\max}$ was set to 0.05. The emitter signal amplitude, $A$, was chosen so that if the same number of sensors were in a square grid, then the $P_{DLB}$ were approximately 0.2. For all tests, the region of interest was chosen to be a square with the same density of nodes. More precisely, the region was considered to be a square of side $5 \times \sqrt{K}$, where $K$ is the number of sensors in the system. Note that the algorithm can be used with any rectangular region, this region was just chosen for testing purposes. For all cases tested, the algorithm was run for 50 random initial sensor distributions and the best results were chosen.

One value to consider when using the scan statistic is the threshold, $t$. Unlike with the sum fusion rule, the threshold is not constant throughout the algorithm. In this case, the number of clusters also affects the threshold. Specifically, as suggested in (Fonseca, 2019b),

$$t = Q^{-1}(1 - (1 - P_{FA}^{\max})^{1/|c_3|})$$  \hspace{1cm} (6.5)

For this reason, it is necessary to recalculate the threshold whenever the cluster set changes as the number of sensors in said cluster set may change as well. Not changing the threshold can result in the constraint of $P_{FA}^{\max}$ not being satisfied.

To test the algorithm, it was run for square numbers of sensors, i.e. $K=4, 9, \ldots$ This was initially done for up to 49 sensors. The algorithm works with non-square values too; however, it
is easier to compare the results with a grid pattern when considering square values. For these K values, the $P_{DLB}$ was compared with a standard square grid pattern. Specific results are discussed below. The results are compared with a grid pattern in section 6.4.

Figure 18 shows the final sensor positions when the algorithm was run for K=4 using the above parameters. Also shown are the clusters, represented by the dashed lines. As can be seen, $L$ is not symmetrical as it was in the case with the sum fusion rule. It is likely that this is due to the final positions not being the global maximum for $P_{DLB}$. Figure 19 shows the same $L$, but also shows the worst POI and its corresponding $C_{NI}$. This position is a reasonable location for the worst POI since it is not close to any sensors. The cluster set is reasonable as the best cluster since the region is somewhat small, so each sensor had a decent chance of detecting the emitter at this POI.

![Figure 18: Best sensor positions and cluster set obtained for K=4.](image)

*Figure 18: Best sensor positions and cluster set obtained for K=4.*
Figure 19: Best sensor positions, worst POI, and corresponding non-intersecting cluster set obtained for K=4.

Figure 20 shows the final $L$ and $C_S$ when the algorithm was run for K=9 using the above parameters. This result is also not symmetrical, but $L$ is fairly similar to the result for K=9 from the sum fusion rule case. This is unlikely to be the global optimum for the system. As can be seen, as K increases, so too does the number of clusters in $C_S$. Figure 21 shows the same $L$, but also shows the worst POI and the corresponding $C_{NI}$. The large cluster on the right makes sense as the best cluster because all of these sensors are relatively close to the worst POI. The other clusters were contained within $C_S$ and thus included in $C_{NI}$ to increase $P_{DLB}$. Note that while this non-intersecting cluster set is a good cluster set, there is no guarantee that it is the best possible $C_{NI}$. The clustering algorithm does not necessarily choose the best $C_{NI}$, just a reasonably good one. Also, note that in Figure 21, while the sensor around position (7,2) appears to be in multiple
clusters, this is just due to the plotting. The clusters are represented by rectangles, which caused this sensor to appear to be in both clusters.

**Figure 20:** Best sensor positions and cluster set obtained for K=9.

**Figure 21:** Best sensor positions and non-intersecting cluster set obtained for K=9.

Figure 22 shows the final $L$ and $C_S$ when the algorithm was run for K=16 using the above parameters. This result is far from symmetrical and the number of clusters is rapidly increasing with K. This is highly unlikely to be the global optimum for the system. Figure 23 shows the same $L$, but also shows the worst POI and the corresponding $C_{NI}$. The cluster around the POI is reasonable as measurements received by each of these sensors would be reasonably high due to the small distances. The other clusters were contained within $C_S$ and thus included in $C_{NI}$ to increase $P_{DLB}$. Note that there is one sensor that did not appear in any cluster in $C_{NI}$. This sensor is included in clusters in $C_S$, as can be seen in Figure 22, however when choosing non-
intersecting clusters, the clusters it is a part of must all intersect with the other clusters that have better detection, thus making it impossible to include this sensor in a non-intersecting cluster.

**Figure 22:** Best sensor positions and cluster set obtained for K=16.

**Figure 23:** Best sensor positions and non-intersecting cluster set obtained for K=16.

Figure 24 shows the final $L$ and $C_S$ when the algorithm was run for K=25 using the above parameters. This result is also far from symmetrical and contains some sensors that are very close together. The number of clusters continues to increase with K. This is unlikely to be the global optimum for the system. Figure 25 shows the same $L$, but also shows the worst POI and its corresponding $C_{NI}$. In this case, $C_{NI}$ contains 6 clusters of varying size. It is likely that the
single sensor cluster directly above the POI is the best. Again, there are some sensors without a cluster in $C_{NI}$.

![Figure 24: Best sensor positions and cluster set obtained for K=25.](image)

![Figure 25: Best sensor positions and non-intersecting cluster set obtained for K=25.](image)

Figure 24 shows the final $L$ and $C_S$ when the algorithm was run for $K=36$ using the above parameters. This result is also far from symmetrical and contains some sensors that are very close together. The number of clusters continues to increase with $K$. This is unlikely to be the global optimum for the system. Figure 27 shows the same $L$, but also shows the worst POI and its corresponding $C_{NI}$. The worst POI makes sense in this case because the sensors are more
concentrated to the upper left side of the region. Again, there are some sensors without a cluster in $C_{NI}$.

*Figure 26: Best sensor positions and cluster set obtained for $K=36$.*

*Figure 27: Best sensor positions and non-intersecting cluster set obtained for $K=36$.*

Figure 28 shows the final $L$ and $C_S$ when the algorithm was run for $K=49$ using the above parameters. This result is also far from symmetrical and contains some sensors that are very close together. The number of clusters continues to increase with $K$. This is unlikely to be the global optimum for the system. Figure 29 shows the same $L$, but also shows the worst POI and its corresponding $C_{NI}$. The worst POI makes sense because the corners of the region often have less sensor coverage. Again, there are some sensors without a cluster in $C_{NI}$. 

6.4 Algorithm Results Relative to a Square Grid Pattern

In this section, the algorithm results are compared with a standard grid pattern. For the standard grid case, the clustering algorithm was used as before. The results can be seen compared in Table 2. As can be seen, the $P_{DLB}$ obtained through the algorithm was an improvement over a standard grid pattern in all cases except when $K=49$. The maximum improvement can be seen when $K=9$, where $P_{DLB}$ increased by from 0.207 to 0.436, over doubling the probability of detection. The minimum improvement can be seen when $K=36$, where $P_{DLB}$ increased from 0.219 to 0.245. When $K=49$, the algorithm resulted in the $P_{DLB}$
decreasing from 0.197 to 0.166. The increase factor column shows the ratio between $P_{DLB \text{ Alg}}$ and $P_{DLB \text{ Grid}}$. This value is dependent on the baseline detection of the grid.

Table 2
Scan Statistic Algorithm Results vs. Grid Results

<table>
<thead>
<tr>
<th>K</th>
<th>W/H</th>
<th>A</th>
<th>Loop Number</th>
<th>$\varepsilon$</th>
<th>Max Radius</th>
<th>$P_{DLB \text{ Grid}}$</th>
<th>$P_{DLB \text{ Alg}}$</th>
<th>Change in $P_{DLB}$</th>
<th>Increase Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>20</td>
<td>50</td>
<td>1.00E-9</td>
<td>10</td>
<td>0.199</td>
<td>0.381</td>
<td>0.182</td>
<td>1.91</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>23</td>
<td>50</td>
<td>1.00E-9</td>
<td>15</td>
<td>0.207</td>
<td>0.436</td>
<td>0.229</td>
<td>2.11</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>24</td>
<td>50</td>
<td>1.00E-9</td>
<td>20</td>
<td>0.191</td>
<td>0.292</td>
<td>0.100</td>
<td>1.53</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>26</td>
<td>50</td>
<td>1.00E-9</td>
<td>25</td>
<td>0.201</td>
<td>0.302</td>
<td>0.100</td>
<td>1.50</td>
</tr>
<tr>
<td>36</td>
<td>30</td>
<td>27</td>
<td>50</td>
<td>1.00E-6</td>
<td>30</td>
<td>0.219</td>
<td>0.245</td>
<td>0.026</td>
<td>1.12</td>
</tr>
<tr>
<td>49</td>
<td>35</td>
<td>28</td>
<td>50</td>
<td>1.00E-6</td>
<td>35</td>
<td>0.197</td>
<td>0.166</td>
<td>-0.031</td>
<td>0.84</td>
</tr>
</tbody>
</table>

It is possible to conjecture why the K=36 and K=49 cases were less effective than the others. This is mostly because as K increases, so too does the number of potential points of convergence. This results in significantly decreased odds of obtaining a good point of convergence. Also, these K values were run with a more lenient stop condition. It is also possible that as the number of sensors increases, the true optimal may become closer to a grid pattern. Lastly, as K increases, the size of the clusters often increases too. This means that Torczon’s algorithm is run on larger dimensions, which can lead to less accuracy. In section 6.6, further analysis on systems with K $\geq$49 sensors is discussed.

For all these cases, running the algorithm for more random initial starts could potentially find a better point of convergence. This is due to the use of a local search algorithm. Each random drop has the potential to find a different final $L$. In this algorithm, the local search algorithm is generally run multiple times and on different clusters. This could potentially lead to an even greater number of potential points of convergence. It seems that for more than 25
sensors, the algorithm results begin to rapidly fall off and the run time greatly increases. For these values, it may be reasonable to use a grid pattern if there are strict time constraints. Otherwise, the algorithm discussed in section 6.6 could be considered. It is likely that the current algorithm could improve $P_{DLB}$ for any number of sensors and any region, however, the run time would become very high.

While the majority of testing was done with a region of interest with side dimensions of $H = W = 5 \times \sqrt{K}$, this is not a requirement of the algorithm. To demonstrate this, the algorithm was run for 10 loops when $K=16$ with side dimensions of $H=W=10$ and $H=W=30$. The A values were adjusted to still provide a $P_{DLB}$ of around 0.2 when using a grid pattern. When the side dimensions were 10, the algorithm was able to improve $P_{DLB}$ from about 0.191 to 0.240. This is less than was shown in the original tests, but this is likely just due to running less random drops and the nature of local search algorithms. Also, of these 10 random drops, 2 converged to positions that were better than a grid. When the side dimensions were 30, the algorithm was able to improve $P_{DLB}$ from about 0.2 to 0.265. While this is less than was previously found, this is just due to running less random drops. Also, of these 10 random drops, 3 converged to positions that were better than a grid pattern.

6.5 Analysis

For the case with the scan statistic, the algorithm was generally able to improve over a standard square grid pattern. Compared with a grid pattern, the algorithm was able to improve $P_{DLB}$ by up to nearly 23% under certain conditions. In general, once $K$ became too large, the
algorithm became less likely to find a global optimum, thus lowering the final $P_{DLB}$. It is reasonable to assume that due to the nature of local search algorithms, the results obtained are likely not the true optimum. In the $K=49$ case, this is evident because the obtained $P_{DLB}$ is less than the grid pattern. Also, it is easy to show that a grid pattern is not the global optimum either, as moving the corner sensors towards the corner of the region can improve detection. This means that running the algorithm more times could lead to an even greater increase in $P_{DLB}$ in all cases; however, the probability of finding the global optimum decreases with the number of sensors.

This algorithm has similar downfalls to those seen with the sum fusion rule. First, there is a limitation on the number of sensors that can be optimized efficiently. With the tested parameters, it was found that $K=36$ was near to the maximum $K$ that can be efficiently optimized by this algorithm. It is still possible to use this algorithm for larger systems, however the run times to find a good result become increasingly high. This version of the algorithm also shares the downfalls of a local search algorithm, where there is no guarantee of finding the global optimum. This is countered by considering multiple random starting positions to improve the odds of obtaining a good result. Note that the algorithm only moves sensors in one cluster at a time. In some cases, this can also affect the algorithm results. Lastly, while this clustering algorithm is generally effective at finding a good cluster set, there is no guarantee that it finds the true optimal cluster set or non-intersecting cluster set for the current sensor positions. This can be seen when the clustering algorithm occasionally lowers $P_{DLB}$. 
This section discusses various tests that were done to try improving the odds of finding an improvement over a grid placement for large systems. The first test was to determine whether the use of a more lenient stop condition could improve the problem. To do this, the initial conditions that led to the best results for the K=49 case were run again but with $\varepsilon = 10^{-9}$. Due to the large number of potential points of convergence, this resulted in an entirely different final point of convergence and still did not improve the $P_{DLB}$. Additionally, the algorithm was run with the best point for K=49 as the initial sensor positions but a stop condition of $\varepsilon = 10^{-9}$. While the algorithm did run for additional loops under the new stop condition, the overall result still did not improve over a grid pattern. These tests show that the stop condition for Torczon’s algorithm is unlikely to be the cause of its poor performance in large systems.

Next, tests were conducted to determine whether a grid pattern was near to the true optimal when K=49. First, when the initial sensor positions were placed in the grid pattern, the algorithm lowered the $P_{DLB}$ and thus accepted the grid as the best point. While this implies that a grid is a local maximum, it can easily be shown that moving the corner sensors of the grid toward the region corners is better than using a grid. This is because the corners of the region are the worst points of interest and the center of the region has many sensors distributed throughout. For initial sensor positions with as much symmetry as a grid, the algorithm seems to struggle to find an improved cluster set.

Next, if a grid with a slight perturbation of $\pm 1$ for each dimension was used for the initial sensor positions, the algorithm again accepted the initial sensor positions as the best. This
implies that there are many potential points of convergence that are possible, some of which are very close together. When close together, the change in $P_{DLB}$ is sometimes too small for the precision of the algorithm to perceive, thus ending the algorithm prematurely. Also, the algorithm requires a good starting position to find a good point of convergence. These tests show that it is difficult to choose a good initial starting position, so this is not enough to improve the $P_{DLB}$.

Next, it was considered that the simplex could be the problem. Perhaps the orthogonal simplex was limiting the possible search directions of the algorithm when starting near a grid. The simplex was first adjusted so that rather than being orthogonal and changing just one-dimension at a time, the algorithm also randomly adjusted the $x$ and $y$ coordinates of that sensor. Ideally, this would allow for different search directions. In this case, the algorithm still chose the grid or perturbated grid as the best point.

The same approach was tried with random perturbation of all dimensions of the current simplex. In other words, the points in the simplex would have one dimension changed as usual, but also allow for slight perturbation in all other dimensions. This however still accepted the grid or perturbated grid as the best point. This further demonstrates that when $K$ becomes large, the algorithm struggles to find an optimal point of convergence. Simply choosing initial sensor positions near to a grid pattern is not enough to solve this problem. These tests show that the initial simplex may not be the leading cause of the poor performance of the algorithm in large systems.

At this point, the algorithm itself was considered. As discussed previously, the algorithm only moves sensors in the best cluster for the worst point of interest. In most cases, this is not an
issue. However, when starting near a grid, this can cause some problems. To improve the
detection of a grid, it is necessary to adjust all corner sensors. Only adjusting a single corner
sensor decreases the $P_{DLB}$ because it makes three of the four corners slightly worse. It is never
the case that the best cluster being considered includes all corners, thus making it very difficult
for the algorithm to improve over an initial grid placement.

To address this limitation of the algorithm, a slight variation of the algorithm was created.
This algorithm is the same except that instead of moving sensors for only the best cluster for the
worst point of interest, it instead moved the sensors in the four best clusters for the worst point of
interest. This allowed the algorithm to move sensors in a wider range. When running this
algorithm with a perturbated grid and $K=49$, the algorithm found a point of convergence that
improved over a grid by about 3.9%. This point of convergence is shown in Figures 30 and 31. It
is important to note that the algorithm still has no guarantee of improvement over a grid
placement, but the odds appear to be significantly higher than with the standard algorithm. It is
also worth noting that the run times on this version of the algorithm tend to be significantly
lower. This is likely because starting near to a grid pattern is generally closer to a good point of
convergence than placing the sensors purely randomly. Figure 30 shows the final sensor
positions and cluster set obtained for this algorithm, while Figure 31 shows the non-intersecting
cluster set and worst POI.
In order to further test the limits of this version of the algorithm, it was run 10 times for $K=49$ and $K=64$ and 5 times for $K=81$ and $K=100$. Table 3 shows the results obtained compared with a square grid pattern. As can be seen, this algorithm was able to improve the $P_{DLB}$ over a grid even with a small number of random initializations. At a maximum when $K=100$, the algorithm improved over a grid by about 9.2%, i.e. by a ratio of 1.44. At a minimum, when $K=81$, the algorithm only improved over a grid by 0.33%, i.e. by a ratio of 1.01, which is not statistically significant. In all cases, more random drops could likely improve the detection further. In Table 3, the “Ratio Improved” column signifies the number of random drops that
resulted in a higher $P_{DLB}$ than the grid pattern. Recall that the standard scan statistic algorithm was run 50 times for $K=49$ and never showed improved performance, while this adjusted algorithm was able to obtain improved performance with only 10 drops. This is a clear demonstration that this algorithm is more effective for larger systems. The algorithm also had greatly improved run times when compared with the standard algorithm.

Table 3

<table>
<thead>
<tr>
<th>$K$</th>
<th>$P_{DLB}$ Grid</th>
<th>$P_{DLB}$ Algorithm</th>
<th>$\Delta P_{DLB}$</th>
<th>Increase Factor</th>
<th>Ratio Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>0.197</td>
<td>0.235</td>
<td>0.039</td>
<td>1.19</td>
<td>4/10</td>
</tr>
<tr>
<td>64</td>
<td>0.199</td>
<td>0.210</td>
<td>0.012</td>
<td>1.06</td>
<td>2/10</td>
</tr>
<tr>
<td>81</td>
<td>0.204</td>
<td>0.207</td>
<td>0.003</td>
<td>1.01</td>
<td>2/5</td>
</tr>
<tr>
<td>100</td>
<td>0.210</td>
<td>0.302</td>
<td>0.092</td>
<td>1.44</td>
<td>2/5</td>
</tr>
</tbody>
</table>

Alternatively, the adjusted algorithm was run on the best cluster for each of the four worst POIs. The goal of this was to allow all four corner sensors to move simultaneously, thus allowing the algorithm to escape the grid. The downside of this is that it requires a larger input into Torczon’s algorithm. These results are shown in Table 4. As can be seen, it was also generally able to improve over a standard grid pattern. More random drops of both versions of this algorithm are required to definitively say which version is more effective.

Table 4

<table>
<thead>
<tr>
<th>$K$</th>
<th>$P_{DLB}$ Grid</th>
<th>$P_{DLB}$ Algorithm</th>
<th>$\Delta P_{DLB}$</th>
<th>Increase Factor</th>
<th>Ratio Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>0.197</td>
<td>0.211</td>
<td>0.014</td>
<td>1.07</td>
<td>2/5</td>
</tr>
<tr>
<td>64</td>
<td>0.199</td>
<td>0.273</td>
<td>0.075</td>
<td>1.37</td>
<td>3/5</td>
</tr>
<tr>
<td>81</td>
<td>0.204</td>
<td>0.261</td>
<td>0.057</td>
<td>1.28</td>
<td>1/5</td>
</tr>
<tr>
<td>100</td>
<td>0.210</td>
<td>0.349</td>
<td>0.139</td>
<td>1.66</td>
<td>2/5</td>
</tr>
</tbody>
</table>
Note that the four-cluster version of the algorithm is only necessary for larger systems. Part of what this version of the algorithm does is decrease the search space by starting near a grid. This is important in very large systems where the search space is too large to feasibly explore with the standard algorithm. Limiting the search space however may also limit the possible results, meaning this version of the algorithm could potentially make it impossible to find the true optimum. For this reason, it is better to use the standard algorithm for smaller systems.

Also, note that this version of the algorithm was set to consider four clusters at a time instead of just one, like the standard algorithm. This number was specifically chosen to allow all corners of the perturbated grid to move simultaneously, thus allowing the algorithm to progress. If desired, the algorithm could be adjusted to consider more than four clusters at once. Recall that the reason a single cluster was considered initially was that Torczon’s algorithm struggles when the input is too large. By allowing more clusters to be moved at once, this requires Torczon’s algorithm to consider larger systems. This can lead to very high run times and often poor convergence. For the same reason, for extremely large systems, it is likely that the adjusted algorithm will require continually more initial drops to improve over a grid, as larger sensor systems often have a larger number of clusters, as well as more sensors per cluster in some cases.
CHAPTER 7

CONCLUSION

Overall, the results of this research indicate that it is possible to improve the performance of sensor detection systems over a standard grid pattern. This research also showed that such optimization can be performed with Torczon’s local search algorithm when the system size is not too large. This is true for both systems that use the sum fusion rule, and systems that use the scan statistic.

When considering the sum fusion rule, the algorithm was found to be most viable when the number of sensors was kept below 10. The algorithm is still useable for $K>9$, however the run times increase and the odds of improvement decrease. In the considered scenarios, the worst-case probability of detection was found to improve by up to nearly 23% or an increase factor of 1.32 over a standard grid pattern when $K=8$. This amount varied with $K$ as well as with the initial sensor positions, but the algorithm in general always improved over a grid.

When considering the scan statistic, the initial algorithm was found to be viable for up to 36 sensors, significantly higher than the sum fusion rule. The algorithm was able to improve the probability of detection lower bound by up to nearly 23% or an increase factor of 2.11 over a standard grid pattern when $K=9$. This was similarly dependent on both $K$ and the initial sensor positions. However, increasing the number of sensors beyond 36 often led to a worst-case probability of detection that was worse than that of a standard grid pattern. Also, the run time became increasingly large for systems of this size and increasingly more random initial drops.
were required to try finding the global optimum for the system. It is important to mention that there is room for improvement in the algorithm proposed here. Specifically, the proposed algorithm struggles when the number of sensors becomes too large. One possible improvement is to consider a variation of the algorithm in which the sensor positions of multiple clusters are optimized simultaneously. As described in section 6.6, initial experiments have shown that this variant can generally find sensor placements with improved performance over the standard grid, even in systems with more than 49 sensors. Such a variant may be a fruitful avenue for further research.

There are some potential improvements that could be made on this algorithm. The first would be to implement genetic algorithms on the results of the current algorithm. This would essentially involve combining the best parts of different iterations of the algorithm in an attempt to find a new best overall result. This could include using sensor positions from one iteration and a cluster set from another, or even combining subsets of sensor positions and cluster sets from different iterations.

Another potential improvement could be to update the algorithm to work for various region shapes. Currently, the algorithm can optimize sensor positions in either a square or rectangular region. The main difficulty in this advancement is that the calculation of the probability of detection lower bound for the sum fusion rule depends on square subregions. In many cases, this region could be approximated by a union of square subregions that contains the original region; however, this would likely lead to a slight decrease in the detection lower bound. This is because part of the subregions will be outside of the region of interest, allowing for the
emitter to be considered even if slightly outside of the region. Nevertheless, such a loss can be controlled by using small enough square subregions to approximate the region.

Another potential consideration could be to write an algorithm that can optimize the sensor positions and clusters at the same time, rather than iteratively. However, this would likely be a very complex optimization problem as the clusters depend on sensor positions.


