A GEOMETRIC TILING ALGORITHM FOR APPROXIMATING MINIMAL COVERING SETS

A Thesis
Presented to
The Graduate Faculty of The University of Akron

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

Adam Paul Martinez
December, 2011
ABSTRACT

The cover generation problem is relevant to the problem of creating large-scale wireless sensor networks. Wireless sensor networks have short-ranged sensor nodes that may not be capable of transmitting to base station. Quickly and efficiently placing relay nodes allows the sensors to save on battery power and transmit information back to the base station via the relay nodes. Placing a minimal cover of relays is at least an NP-hard problem. We present a geometric tiling algorithm to construct an approximation to a minimal covering set in $O(n)$ time. The algorithm fills the target region with a triangular grid of relays and then culls unnecessary points from the grid. A brief analysis of the algorithm is presented and a comparison to another cover-generation algorithm is performed.
ACKNOWLEDGEMENTS

I would like to thank the following people, without whom I never would have finished this thesis:

Dr. Timothy Norfolk, my thesis advisor, mentor and friend, for endless hours of interesting conversation, helpful advice and believing in me when I did not.

Dr. Curtis Clemons, who convinced me to study mathematics, for providing both direction and criticism when I needed it.

Dr. Kathy Liszka and Dr. Tim O’Neil, whose courses introduced me to the mathematics of computers.

Dr. Gerald Young, for all his assistance in helping me navigate through my undergraduate and graduate studies.

Dr. Michael Ferrara, for teaching me that mathematics is fun, even though graduate school isn’t always so.

Dr. Kevin Kreider, who never turned me away when I stopped by to ask for help.

The rest of the Mathematics faculty at the University of Akron.

My mother, Joyce Martinez, for not allowing me to quit graduate school when I was doubting myself.

And all my friends and family that helped me along the way.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. FORMULATION OF ALGORITHM</td>
<td>11</td>
</tr>
<tr>
<td>III. ANALYSIS OF THE ALGORITHM</td>
<td>16</td>
</tr>
<tr>
<td>IV. CONCLUSION</td>
<td>20</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>22</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>25</td>
</tr>
<tr>
<td>APPENDIX A. TRIANGULAR GRID ALGORITHM CODE</td>
<td>26</td>
</tr>
<tr>
<td>APPENDIX B. EXAMPLE LNB ALGORITHM CODE</td>
<td>29</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>2.1 Table of Variables</td>
<td>13</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>An example of a one-tiered WSN.</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>An example of a two-tiered, multiple-hop WSN.</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>An example of a two-tiered, single-hop WSN.</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>An example of a hexagonally tessellated WSN.</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>A triangular grid and hexagonal cover side-by-side.</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Example output from each algorithm.</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Average relays required to cover sensor networks of various densities.</td>
<td>19</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

While there are a wide variety of polynomial time approximation schemes (PTAS) that can approximate solutions to the minimum geometric disk cover (MGDC) problem, none in current literature can do so in $O(n)$ runtime. This paper presents an algorithm that computes an approximation to the MGDC problem with reasonable disk-to-point efficiency for many instances of the problem in linear runtime. This algorithm was inspired by research into building wireless sensor networks (WSNs) and the algorithms used to compute optimal designs for these networks. Some WSN structure problems can be cast as MGDC problems. While for many applications the long runtimes of the algorithms common in literature are not an issue, for time-sensitive problems or very large regions with large sets of sensor nodes (SNs), computing a covering set of relay nodes (RNs) can take unreasonably long.

For most applications engineers and scientists are willing and able to spend countless hours computing an optimal networking solution that requires as few relays as possible. However, not all networking environments have the luxury of unlimited design and setup time. For time-sensitive applications, computing a fast and reasonably accurate solution to a covering set of a network can achieve a “good enough” solution that will save lives. The network will be more costly, but it can start being
built immediately. This kind of algorithm could be useful for providing real-time logistical and tactical information to moving front-line military units and ensuring that search-and-rescue teams have real-time information. Because these kind of environments do not tolerate time delays, a less efficient network now is far more valuable than a more efficient network later.

WSNs consist of a set of sensor nodes that collect information and wirelessly communicate with one another. There is either a Base Station (BS) that aggregates the information in the network or an outside access point to the network. There may or may not be RNs that act as network gateways for SNs within a small region around them. The presence of RNs determines if a WSN is one-tiered or two-tiered [2].

![Diagram of a one-tiered WSN](image)

**Figure 1.1:** An example of a one-tiered WSN.

*One-tiered* WSNs have only sensor components. As in figure 1.1, SNs are
either deliberately placed or randomly dropped into a region, and an appropriate way to route data through the network must be found. Methods of routing data through the network include constructing connected dominating sets [3] [4], constructing minimum spanning trees via Steinerization schemes [5], and the LEACH and PEGASUS algorithms [6] [7], among others. When representing the network as a graph, a connected dominating set is a network backbone. Steinerization schemes involve constructing a network tree spanning a set of points $P$ with terminal points $E \in P$ such that each edge in the tree has a length no more than some number $r$ and that the number of points in $P$ but not in $E$ is minimized. The PEGASUS algorithm is a development based off of the LEACH algorithm. Both algorithms involve electing cluster heads inside small regions of the network, where the cluster heads act as network relays for their neighbors. SNs typically have limited battery power and transmission range, and so many of these algorithms focus on maximizing network lifetime. An additional fundamental problem with one-tiered WSNs is that, in large networks, the relays nearest the BS are required to relay large volumes of data and will therefore deplete their power very quickly.

There has been significant research into algorithms for generating two-tiered covers of WSNs within the past decade. Two-tiered WSNs have not only a set of sensors in a region, but also have a set of relay nodes that act as network gateways for sensors in a small region around them. These algorithms usually assume a random distribution of SNs in a given region. A layer of relay nodes is placed such that the relay layer forms a cover over the sensors. The SN’s sole purpose is to gather
information and forward it to a local RN. For many applications the SNs are designed to be built as cheaply as possible, and thus do not have the battery power and design parameters to transmit data long distances. The RNs collect data from the SNs within a small region and relay the data either directly to or through one another back to a BS. Each RN in a single hop two-tier WSN has a direct connection to the BS. An example single-hop two-tiered network is shown in figure 1.3. The RNs in a multiple hop two-tier WSN must relay messages through one another to reach the BS. An example multiple-hop two-tiered network is shown in figure 1.2. In the multiple-hop case, the BS is either inside the network or reached via a piece of specialized equipment such as a satellite uplink or land line attached to a specific RN. The multiple-hop network requires the construction of a connected cover of relay nodes. Mathematically this is best represented as a connected set. In the case of WSNs, there is a path between any two vertices, and a covering set in a WSN is a set of RNs such that every SN is adjacent to at least one RN. The multiple hop problem is more heavily researched than the single hop problem.

Multiple-hop two-tier WSNs require that the RNs form a connected set as well as a covering set. This appears to be the most heavily researched area in WSNs. These networks can be classified into two groups—connected and survivable. A connected WSN guarantees that each relay is covered by an RN and that the RNs form a connected set. Figure 1.2 is a connected WSN. A survivable network requires that each SN be within range of \( k > 1 \) RNs and that there are \( l > 1 \) node-disjoint paths from each RN to the BS. The survivable case guarantees that if any particular RN was
to fail then the network would not lose any coverage [9]. In 2004, Hao, Tang and Xue [10] developed a polynomial time algorithm to generate a “Two-Connected Relay Node Double Cover (2CRNDC)”, which is a survivable two-tier WSN where there are at least two RNs covering each SN \((k = 2)\), and the RNs form a 2-connected set \((l = 2)\). In [11] their algorithm accounted for even larger values of \(k\) and \(l\). They used the shifting lemma [12] to subdivide the original problem to achieve faster polynomial time approximations, as well as make the problem parallelizable. The shifting lemma does this by cutting the region into small squares and computing an optimal solution for each square, then connecting the squares together. This required that \(R\), the transmission range of the RNs, be constrained such that \(R \geq 4r\), where \(r\) is the
transmission range of the SNs. Shams, Chowdhury and Kim [13] have developed a faster algorithm with a numerical complexity of $O(n^2)$. At the cost of survivability and redundancy ($l = k = 1$) many authors were able to create much more efficient algorithms, in terms of the RN to SN ratio, than Hao, Tang and Xue were able to achieve when $l > 1$, $k > 1$ [24], [25], [26], [27]. Additionally, it should be noted that survivable two-tiered WSNs also benefit from load-balanced clustering. Load-balanced clustering guarantees each RN is within range of approximately the same number of SNs. This balances network traffic load in order to extend network lifetime [10].

Figure 1.3: An example of a two-tiered, single-hop WSN.

Single-hop two-tier WSNs only require that the SNs transmit to the RNs,
and do not require that the RNs be able to transmit to one another, as is depicted in figure 1.3. The only requirement on the set of RNs is that it forms a covering set of the SNs. It is assumed that either the data will be consolidated at the RN for later collection or each RN has some capability of transmitting its information back to a BS. In the latter case, the RNs that are more distant from the BS will deplete their power more quickly than those further away [2]. This is the opposite of the problem that one-tier WSNs encounter. Despite its drawbacks over multiple-hop networks, this network architecture is still useful for networking environments where RNs have satellite uplinks, long range directional wireless communication or landline access. There is extensive study on this problem in terms of the minimum geometric disk cover (MGDC) and discrete unit disk cover (DUDC) problems.

**Minimum Geometric Disk Cover:** given a region $D$ containing a set $P$ with $n$ points, generate minimal covering set of unit disks $C$ such that for each $p \in P$, $\exists c \in C$ such that $p \in c$.

**Discrete Unit Disk Cover:** given a region $D$ containing a set $P$ with $n$ points and a set of unit disks $D$, select a minimal covering set of unit disks $C \subseteq D$ such that for each $p \in P$, $\exists c \in C$ such that $p \in c$.

The MGDC algorithm allows disks to be placed anywhere within the region, while the DUDC problem only allows disks to be placed in specific locations. Both the MGDC and DUDC problems have been proven to be NP-complete [14], but both also allow polynomial time approximation schemes. A PTAS generates a solution to an NP-Hard problem in polynomial time that is no more than some constant mul-
tiple of the optimal answer. A wide variety of PTAS have arisen for both of these problems. Many algorithms for DUDC have been proposed that generate solutions of no more than some constant multiple greater than one of the optimal solution in reasonable time [15], [16], [17], [18]. By comparison, algorithms for the MGDC problem generally require much longer runtimes, but can guarantee an arbitrary \((1+\epsilon)\) level of accuracy to the optimal solution of disks placed anywhere in the region. Depending on the accuracy required and the algorithm used, MGDC and DUDC PTAS can be as fast as \(O(n^2)\) or slower than \(O(n^{100})\). Some connected cover algorithms from the multiple-hop two-tiered problem, such as the 2CRNDC algorithm, are very similar to MGDC algorithms, only as a last step they guarantee connectivity [10]. Some of the more recent work in this problem includes research by Liao and Hu [19], of which a modified algorithm is featured later in this work as a point of reference for the algorithm we present. Liao and Hu’s algorithm build off of general set-based PTAS for approximating the MGDC [20].

Regularly tessellated WSNs are also relevant to the algorithm presented. In the field of underwater acoustic WSNs, Pompili, Melodia and Akyildiz [21] have determined the precise relationship between coverage and sensing range of a regular triangularly tessellated region similar to figure 1.4. They found a precise relation between coverage, sensing range and inter-sensor distance. They also discussed the advantages and disadvantages of such a network compared with other methods for creating connected covers. Other authors have found additional properties of regularly tesselated WSNs on rectangular grids in two dimensions [22] and three dimensions
This paper presents a geometric tiling algorithm for approximating a minimal covering set in the context of a two-tiered, single-hop WSN. This can alternately be described as an approximation scheme for the MGDC problem. We have explored the background material for the cover generation problem, including an extensive review of related research into finding covers in WSNs. A formal description of the geometric tiling algorithm and an analysis of its performance will be given. We will then perform a comparison of the geometric tiling algorithm with another algorithm presented in Section 3. The geometric tiling algorithm generates similar types of solutions to a Local-Neighborhood Based (LNB) algorithm by Liao and Hu[19], provided that the
LNB algorithm is given a particular set of relays to choose from. The LNB algorithm computes optimal solutions for progressively larger regions until a stopping criterion is met, then the algorithm glues these sub-solutions together. By comparison, the geometric tiling algorithm simply looks at the set of relays and the set of points and selects the nearest relay in $O(1)$ time.
CHAPTER II
FORMULATION OF ALGORITHM

We present a new algorithm for generating a reasonably small unit disk cover of a set of points in $O(n)$ time. The approach for this algorithm relies on the uniformity of a triangular grid. Consider the problem of finding the most efficient cover of a large but finite plane using disks of radius 1. Pompili et al. showed that the most efficient regular cover is a triangular grid of disks as in figure 2.1A, with a point-to-point transmission distance of $\sqrt{3}$ [21]. However, our problem formulation does not require that we cover the entire region. We only need to provide a covering set for a set of $n$ points in the region, representing SNs. We will abstractly generate a cover of the region by overlaying a tessellation of hexagons of circumradius 1 with centers at each point on a triangular grid of edge length $\sqrt{3}$. The hexagons in the region will appear as in figure 2.1B. A hexagon of circumradius 1 is a regular hexagon inscribed in a circle of radius 1. Potential RN locations are only at the points on the triangular grid. Each RN will only receive messages from SNs within the RN’s corresponding hexagon. The algorithm will iterate through the $n$ SNs and add the nearest point on the triangular grid to a solution set. By placing a unit disk at each point in the solution set, we produce an approximation to the minimal unit disk cover of the $n$ points.
2.1A: A triangular grid of RNs.

2.1B: A hexagonal cover of RNs.

Figure 2.1: A triangular grid and hexagonal cover side-by-side.

**Formal Problem Statement:** given a region \( R \) containing a set \( P \) with \( n \) points, generate an approximation \( C \) to the minimal covering set of unit disks such that for each \( p \in P \), \( \exists c \in C \) such that \( p \in c \).

Table 2.1 provides a description of the notation used. Given a region filled with \( n \) SNs, we will approximate a minimal covering set. For the purpose of this formulation we will assume we are given a square region containing the SNs. In practical problems, the region would be defined as the minimal square that contains the set of SNs. Label this square region \( R \) with side length \( s \). Our objective is to approximate the minimal cover of the SNs using a triangular grid of RNs.

The RN-SN transmission range \( r \) forms a convenient nondimensional scaling for this problem. We will call the nondimensionalized region \( R_n \) with side length \( s_n \). In this region, the RN-SN transmission range is 1. By scaling all distances involved by \( r \), the algorithm generates a cover for any size region efficiently.
Table 2.1: Table of Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>The 2D region.</td>
</tr>
<tr>
<td>$R_n$</td>
<td>The nondimensionalized 2D region.</td>
</tr>
<tr>
<td>$s$</td>
<td>The side length of $R$.</td>
</tr>
<tr>
<td>$s_n$</td>
<td>The side length of $R_n$.</td>
</tr>
<tr>
<td>$r$</td>
<td>RN-SN transmission range.</td>
</tr>
<tr>
<td>$C$</td>
<td>An approximation to the minimal disk cover.</td>
</tr>
<tr>
<td>$P$</td>
<td>The set of sensor nodes.</td>
</tr>
<tr>
<td>$n$</td>
<td>The number of SNs.</td>
</tr>
<tr>
<td>$x_p$</td>
<td>The x-coordinate of the sensor $p$.</td>
</tr>
<tr>
<td>$y_p$</td>
<td>The y-coordinate of the sensor $p$.</td>
</tr>
<tr>
<td>$x_{(n,p)}$</td>
<td>The nondimensionalized x-coordinate of the sensor $p$.</td>
</tr>
<tr>
<td>$y_{(n,p)}$</td>
<td>The nondimensionalized y-coordinate of the sensor $p$.</td>
</tr>
</tbody>
</table>

We will now abstractly tessellate the nondimensionalized region $R_n$ with hexagons of circumradius 1. The RNs at the centers of the hexagons form a triangular grid. Each SN in $R$ will be mapped to points in $D_n$ via the transformation

$$
(x_{(n,p)}, y_{(n,p)}) = \left(\frac{x_p}{r}, \frac{y_p}{r}\right),
$$

(2.1)

where $(x_p, y_p)$ is the location of the SN $p$.

Given any square or rectangular region, we may orient the hexagons as in figure 2.1B, such that the hexagons fit neatly in the upper left corner of the region with minimal waste. We can then compute the coordinates of any RN. Tessellating the square region $R_n$ this way will require no more than $\left\lceil \frac{s_n}{\sqrt{3}} \right\rceil + 1$ columns and $\left\lceil \frac{s_n}{1.5} \right\rceil + 1$ rows of hexagons of circumradius 1. As we will show, implementing the
algorithm does not actually require the generation and storage in memory of the entire set of hexagons, merely the conceptual knowledge that we have overlaid it on the region.

We will now iterate through each SN and select the nearest hexagon in the grid. The regular nature of the tessellation makes finding the nearest RN a constant time arithmetic process. These RNs locations do not need to be precomputed and then selected, as they can be computed on the fly using arithmetic and rounding.

The RNs, as they are set up in this problem, form a triangular grid. Each row on the triangular grid has the same y-coordinate. Every second row has an offset x-coordinate. We can immediately eliminate all but two potential candidates for the nearest RN to a SN by simply looking at the coordinates of the sensor. The sensor will fall between two rows of relays. Counting from the top, odd numbered rows will have a horizontal offset of $\frac{\sqrt{3}}{2}$ from the even rows. Within each row, the x-coordinate of the SN will be closer to either the RN on its left or on its right. Each row then has a closest RN to the SN, and the closer of these two RNs is chosen to cover that SN.

We will create a “selected relays” matrix $M$ of booleans that stores whether or not the $j^{th}$ RN in the $i^{th}$ row of RNs must be selected to form a cover. This has the disadvantage of taking up a large block of memory by requiring a matrix of $\left(\left\lceil \frac{s_n}{\sqrt{3}} \right\rceil + 1 \right) \left(\left\lceil \frac{s_n}{\sqrt{1.5}} \right\rceil + 1 \right)$ booleans, but avoids writing duplicate RNs to the solution set $C$. We will then compute the location of the $j^{th}$ RN in the $i^{th}$ row for each true in $M$. These RNs make up $C$.

To map the locations of the RNs in $R_n$ back to the region $R$, the corresponding
location in $R$ for an RN at $x_n, y_n$ in $R_n$ is

$$(x, y) = (rx_n, ry_n).$$  \hspace{1cm} (2.2)

By applying this transformation to every disk in $C$, we now have a set of points such that when we place an RN at each of these points, we have a cover of the SNs in $R$. Each SN will be within $r$ units of distance of the nearest RN. This is an approximation to the minimal cover of disks of radius $r$.

Due to the properties of MGDC problem and the formulation of the triangular grid algorithm, there is no simple way to compare precisely how accurate of a solution the algorithm provides to the optimal solution. The MGDC problem is NP-complete, and so finding the optimal disk cover takes an unreasonable amount of time to compute for any random simulation. Additionally, the only mathematical bound the algorithm gives to the efficiency of its cover is that it is the most globally efficient layout of relays on the plane, as was explained in Section 1.

For more details of the algorithm refer to Appendix A.
CHAPTER III

ANALYSIS OF THE ALGORITHM

Due to the similarity to the algorithm presented in Section 2 to the discrete unit disk cover problem, we have chosen a Local-Neighborhood Based (LNB) algorithm by Liao and Hu as a comparison [19]. Liao and Hu wrote a polynomial time approximation scheme for computing an $\epsilon$-approximation to the minimal discrete unit disk cover. The LNB algorithm requires that it be given a set of disks $D$ that cover a set of points $P$. It guarantees that it is within $(1+\epsilon)$ of the minimal number of disks $C \subseteq D$ to cover $P$ and has a runtime of $O\left(\frac{1}{\epsilon^2 \log_2 \epsilon} \frac{1}{\epsilon}\right)$. Hence, on a large region computing an approximation with small $\epsilon$ requires a tremendously long time. More modest approximations (larger $\epsilon$) to the solution can be computed in a reasonable time. The code for this particular implementation of the LNB algorithm can be found in appendix B.

These algorithms were run on a desktop machine using MATLAB R2011b. A set of points was generated using a uniform distribution in a square region of side length $s = 10$. This side length was chosen as it was large enough to see noticeable differences in solutions and computational times, as well as being small enough to compute in a reasonable time due to the growth rate of the LNB algorithm. Each algorithm computed the locations of a triangular grid of unit disks that covered...
the region, then found approximations to the minimal cover of unit disks on that grid. Sensor densities of \( \left( \frac{n}{s^2} = 1, .25, .5, .75, 1, 2 \right) \) were considered to determine each algorithm’s response to networks of varying densities. An example of the output for \( \frac{n}{s^2} = .5 \) is shown in figure 3.1A and 3.1B. The algorithms generate somewhat different approximations to the minimal cover. For this particular instance, the LNB algorithm had one less relay than the triangular grid algorithm.

![3.1A: The LNB algorithm.](image1)

![3.1B: The triangular grid algorithm.](image2)

Figure 3.1: Example output from each algorithm.

Results of simulations showed that the triangular grid algorithm performed in \( O(n) \) time for simulations of all sizes of \( n \). Memory usage was not a concern until the input data size grew to a very large \( n \). A precise comparison with the time usage of the LNB algorithm would be unfair, as the LNB algorithm implemented in the appendix could be optimized for better time and memory performance. The LNB algorithm does not have to run to completion with every step and can often skip
large volumes of computation once it has met the stopping criterion. The theoretical runtime of the LNB algorithm with $\epsilon = .25$ was $O(ns^{64})$. Practically, the runtime varied greatly with respect to the data randomly generated for each instance of the simulation. Most instances completed within a time that was reasonable, but still significantly longer than the triangular algorithm by orders between $10^2$ and $10^6$. Due to the runtime properties of the LNB algorithm and corresponding computational limitations, a comparison on larger regions ($s > 10$) was attempted, but a significant data set could not be generated with the current implementation of the LNB algorithm and computing equipment available. Even with $s = 10$, some instances of the LNB algorithm either took unreasonably long to compute or simply ran out of memory. While this was a property of the LNB algorithm, different implementation procedures may have minimized this occurrence.

For the instances of the LNB algorithm that did produce results, the same randomly generated set of sensors was given to the triangular grid algorithm and it then computed its approximation to the minimal cover. Trials that did not produce results due to the LNB algorithm running out of memory were discarded. While this may have influenced the results, the random scatterings of relays in these trials did not appear to differ from trials that did run correctly. Figure 3.2 shows, averaged across all trials, the triangular grid algorithm has only a few more relays than the LNB algorithm across all sensor densities. For sensor densities less than .25 and greater than 1, the median difference of sensors was actually zero. At most for these
simulations, the triangular grid algorithm had about 10% more relays in its solution than the LNB algorithm. Given that the LNB algorithm with $\epsilon = .25$ has at most 125% of the optimal solution, this puts the triangular grid algorithm at approximately 137.5% of the optimal number of relays on the triangular grid required to cover the points. It is important to note here that the LNB algorithm actually computes an approximation to the DUDC problem, not the MGDC. A solution to the MGDC over these points may have many fewer disks.
CHAPTER IV
CONCLUSION

In this paper we presented an $O(n)$ triangular grid algorithm for approximating the minimum geometric disk cover of a set of points in a region. While the algorithm presented is not an $\epsilon$-approximation, its speed and practical performance for generating approximations to an NP-Complete problem in linear time makes it suitable for some applications including guaranteeing coverage in dense or rapidly changing wireless sensor networks. For emergency situations and military applications, time is the primary issue for building an effective network, not cost. This algorithm provides a method for quickly generating covering sets of almost any size network.

The triangular grid algorithm could be improved. A better approximation could be found by culling relays from the grid using some simple enumerative techniques to identify unnecessary relays. Additionally, search techniques could be used to generate connected covers by finding unconnected spaces and connecting them with a shortest path of relays. While this would increase the runtime of the algorithm, it could provide fast solutions to the multiple-hop WSN problem. The runtime increase would likely be dominated by the runtime of the search algorithm. Common search algorithms, such as breadth-first and depth-first searches, are $O(n^2)$. The algorithm would also generate an acceptable starting point for iterative methods for calculating
a minimal cover.

Additionally, the algorithm deserves a comparison to a true MGDC algorithm that is not restricted to a triangular grid. While the triangular grid algorithm provides adequate solutions on the grid, at this time we are unsure how the algorithm compares to a true MGDC algorithm in terms of RN-SN ratio.
BIBLIOGRAPHY


APPENDICES
APPENDIX A

TRIANGULAR GRID ALGORITHM CODE

s = 10;  \% square edge length
ns = 50;  \% numsensors

sensors = s*rand(2,ns);

numRelayCols = ceil(s/sqrt(3)) + 1;
numRelayRows = ceil(s/1.5) + 1;

markedrelays = false(numRelayCols, numRelayRows);

\% We will now iterate through each sensor and place a 1 in
\% relays(numRelayCols, numRelayRows) for the sensor closest to that
\% particular relay.
for i = 1:ns
  x = sensors(1,i);
y = sensors(2,i);

  \% "+ 1" is due to MATLAB not allowing index of zero.
  upperRow = ceil((y-.5)/1.5) + 1;
  lowerRow = floor((y-.5)/1.5) + 1;

  \% Case 1: upper row is even, lower row is odd
  if mod(lowerRow,2) == 1
    upperCol = round(x/sqrt(3)) + 1;
    lowerCol = round(x/sqrt(3) - 1/2) + 1;
  \% Case 2: upper row is odd, lower row is even
  else
    upperCol = round(x/sqrt(3) - 1/2) + 1;
    lowerCol = round(x/sqrt(3)) + 1;
  end

  if mod(lowerRow,2) == 0
upperx = upperCol*sqrt(3) - sqrt(3)/2;
uppers = upperRow*1.5-1;

lowerx = lowerCol*sqrt(3) - sqrt(3);
lowery = lowerRow*1.5-1;

upperdist = sqrt(( x-upperx )^2 + ( y-uppers )^2);
lowerdist = sqrt(( x-lowerx )^2 + ( y-lowery )^2);

if upperdist > 1 && lowerdist > 1
  error = true
end

else

  upperx = upperCol*sqrt(3) - sqrt(3);
  uppers = upperRow*1.5-1;

  lowerx = lowerCol*sqrt(3) - sqrt(3)/2;
  lowery = lowerRow*1.5-1;

  upperdist = sqrt(( x-upperx )^2 + ( y-uppers )^2);
  lowerdist = sqrt(( x-lowerx )^2 + ( y-lowery )^2);

  if upperdist > 1 && lowerdist > 1
    error = true
  end
end

if lowerRow < 1 || lowerCol < 1 || lowerCol > numRelayCols
  markedrelays(upperCol,upperRow) = true;
elseif upperRow > numRelayRows || lowerCol < 1
  || upperCol > numRelayCols
  markedrelays(lowerCol,lowerRow) = true;
elseif upperdist <= lowerdist
  markedrelays(upperCol,upperRow) = true;
else
  markedrelays(lowerCol,lowerRow) = true;
end

sensorsfile = fopen('sensors.txt','wt');
fprintf(sensorsfile,'%(06.4f)\t%(06.4f)\n',sensors);
fclose(sensorsfile);

markedrelaysfile = fopen('markedrelays.txt','wt');
allrelaysfile = fopen('allrelays.txt','wt');

for m = 1:numRelayCols
    for n = 1:numRelayRows
        if mod(n,2) == 1
            fprintf(allrelaysfile,'%(06.4f)\t',m*sqrt(3) - sqrt(3)/2);
            fprintf(allrelaysfile,'%(06.4f)\n',n*1.5-1);
            if markedrelays(m,n) == true
                fprintf(markedrelaysfile,'%(06.4f)\t',m*sqrt(3) - sqrt(3)/2);
                fprintf(markedrelaysfile,'%(06.4f)\n',n*1.5-1);
            end
        else
            fprintf(allrelaysfile,'%(06.4f)\t',m*sqrt(3) - sqrt(3));
            fprintf(allrelaysfile,'%(06.4f)\n',n*1.5-1);
            if markedrelays(m,n) == true
                fprintf(markedrelaysfile,'%(06.4f)\t',m*sqrt(3) - sqrt(3));
                fprintf(markedrelaysfile,'%(06.4f)\n',n*1.5-1);
            end
        end
    end
end

fclose(markedrelaysfile);
fclose(allrelaysfile);
APPENDIX B
EXAMPLE LNB ALGORITHM CODE

clear
clc

s = 10; % square edge length
ns = 50; % numsensors
eps = .25; % approximation factor

% Randomly generate sensors.
Pset = s*rand(2,ns);

sensorsfile = fopen('HLsensors.txt','wt');
fprintf(sensorsfile,'%06.4f	%06.4f
',Pset);
fclose(sensorsfile);

% Generate triangular array of relays.
numRelayCols = ceil(s/sqrt(3)) + 1;
numRelayRows = ceil(s/1.5) + 1;

Dset = zeros(2,numRelayCols*numRelayRows);
relayx = zeros(numRelayCols, numRelayRows);
relayy = zeros(numRelayCols, numRelayRows);

for m = 1:numRelayCols
    for n = 1:numRelayRows
        relayy(m,n) = n*1.5-1;
        Dset(2,n+(m-1)*numRelayRows) = n*1.5-1;
        if mod(n,2) == 1
            relayx(m,n) = (m-1)*sqrt(3) + sqrt(3)/2;
            Dset(1,n+(m-1)*numRelayRows) = (m-1)*sqrt(3) + sqrt(3)/2;
        else
            relayx(m,n) = (m-1)*sqrt(3);
            Dset(1,n+(m-1)*numRelayRows) = (m-1)*sqrt(3);
        end
    end
end
relaysfile = fopen('HLrelays.txt','wt');
fprintf(relaysfile,'%06.4f\t%06.4f\n',Dset);
fclose(relaysfile);

% We now have a set of all points (Pset) and a set of all disks (Dset)
SolArray = [-1;-1];

% Iterate the following process until Pi is empty.
while not(isempty(Pset))
    % Pick a random point.
    randpoint = ceil(size(Pset,2)*rand(1));

    % Pick a disk that covers the point. Modified technique from cover
    % generation algorithm will be used.
    x = Pset(1,randpoint);
    y = Pset(2,randpoint);

    % "+ 1" is due to MATLAB not allowing index of zero.
    upperRow = ceil((y-.5)/1.5) + 1;
    lowerRow = floor((y-.5)/1.5) + 1;

    % Case 1: upper row is even, lower row is odd
    if mod(lowerRow,2) == 1
        upperCol = round(x/sqrt(3)) + 1;
        lowerCol = round(x/sqrt(3) - 1/2) + 1;
        % Case 2: upper row is odd, lower row is even
    else
        upperCol = round(x/sqrt(3) - 1/2) + 1;
        lowerCol = round(x/sqrt(3)) + 1;
    end

    % Mark the relay closest to the sensor, considering if a col is
    % outside a boundary

    if mod(lowerRow,2) == 0

upperx = upperCol*sqrt(3) - sqrt(3)/2;
upery = upperRow*1.5-1;

lowerx = lowerCol*sqrt(3) - sqrt(3);
lowery = lowerRow*1.5-1;

upperdist = sqrt(( x-upperx )^2 + ( y-uppery )^2);
lowerdist = sqrt(( x-lowerx )^2 + ( y-lowery )^2);

if upperdist > 1 && lowerdist > 1
    error('No disc could be found to cover point.');
end

else
    upperx = upperCol*sqrt(3) - sqrt(3);
    upery = upperRow*1.5-1;

    lowerx = lowerCol*sqrt(3) - sqrt(3)/2;
    lowery = lowerRow*1.5-1;

    upperdist = sqrt(( x-upperx )^2 + ( y-uppery )^2);
    lowerdist = sqrt(( x-lowerx )^2 + ( y-lowery )^2);

    if upperdist > 1 && lowerdist > 1
        error('No disc could be found to cover point.');
    end
end

if lowerRow < 1 || lowerCol < 1 || lowerCol > numRelayCols
    Dcov0 = [upperx;upery];
elseif upperRow > numRelayRows || lowerCol < 1 || upperCol > numRelayCols
    Dcov0 = [lowerx;lowery];
elseif upperdist <= lowerdist
    Dcov0 = [upperx;upery];
else
    Dcov0 = [lowerx;lowery];
end
% We have now found a base case for a minimal cover: the nearest unit
% disk on a triangular grid that covers the randomly selected point.
% Now we call a recursive function that finds minimal covers for
% progressively larger regions until the (1 + eps) criterion is
% reached.

% Pset is the set of all SNs. 2D array, Pset(1,i) is x-coordinate of
% pi, Pset(2,i) is y-coordinate of pi.
% Dset is the set of all disks. 2D array, Dset(1,i) is x-coordinate of
% di, Dset(2,i) is y-coordinate of di.
% pcen is an origin point.
% i is the ith time this function has been called.

Dcovi = Dcov0;
Dcovim1 = Dcov0;
i = 0;
pcen = [x,y];
maxi = 10;

% This loop will grow a minimal set of covering points (Dcovi) for
% larger and larger concentric circles centered at pcen until the
% stopping criterion is met.
while size(Dcovi,2) <= (1+eps)*size(Dcovim1,2) && i < maxi
    i = i+1;
    Dcovim1 = Dcovi;

    if mod(i,100) == 0
        str = ['Iterations = ', num2str(i)];
        disp(str);
    end

    % We abstractly create a LargeDisk Dpcen
    %center = pcen
    %radius = 2i

    % Find all points p in P st |p-pcen|<2i (p is in Dpcen). Put
    % them in Pi.
    Pi = [-1;-1];
    for k = 1:size(Pset,2)
        if sqrt(((Pset(1,k)-pcen(1))^2 + (Pset(2,k)-pcen(2))^2) <=
            2*i
            Pi = [Pi Pset(k,:);

            end

    Dcovi = Dcovim1;
\begin{verbatim}

Pi = horzcat(Pi, [Pset(1,k); Pset(2,k)]);
end
end
Pi(:,1) = [];
\%
 Pi now contains all p in Pset st p is in Dpcenti.

\%
 Find all disks d st |dcent-pcent|<2i (dcent is in Dpcenti). Put
\%
 them in Di.
Di = [-1;-1];
for k = 1:size(Dset,2)
    if sqrt((Dset(1,k)-pcent(1))^2 + (Dset(2,k)-pcent(2))^2) <=
        2*i + 1
        Di = horzcat(Di, [Dset(1,k); Dset(2,k)]);
    end
end
Di(:,1) = [];

\%
 Di now contains all d in Dset dcent is in Dpcenti.

\%
 Exhaustively search all subsets of Di to find a minimal covering
\%
 set. Call it Dcovi.
\%
 How do I do this?
\%
 Generate all subsets of size Pi or smaller

if(size(Pi,2) < size(Di,2))
    maxsize = size(Pi,2);
else
    maxsize = size(Di,2);
end

maxchoose = 0;
for k = 1:maxsize
    maxchoose = maxchoose + nchoosek(size(Di,2),k);
end

disp(['Number of combinations = ', num2str(maxchoose), ',
    size(Di,2) = ', num2str(size(Di,2)), ',
    maxsize = ', num2str(maxsize), ',
    and i = ', num2str(i)]);
if maxsize > 15
    disp(['maxsize = ', num2str(maxsize), ',
          and i = ', num2str(i)]);
end
\end{verbatim}
stringset = dec2bin(LHAlgRecV2(maxsize,size(Di,2)),size(Di,2));

disp(['Time elapsed: ', num2str(recursivetime), ' seconds.']);

% stringset contains (size(Di,2) choose maxsize) - 1 elements.

% stringset is a set of all possible combinations of the Di disks,
% where the i-th element of Di is included in the k-th combination
% if stringset(k,i) = '1'.

% Eliminate all subsets that do not provide a cover.

garbageset = dec2bin(zeros(size(stringset,1),1),size(stringset,2));
gspointer = 1;

% Iterate through each combination.
tic
for j = 1:size(stringset,1)
    if size(stringset,1) > 1000
        if mod(j,round(size(stringset,1)/10)) == 0
            disp(['Iterations are ', num2str(round(100*j/size(stringset,1))), '% complete.']);
        end
    end
end

% Iterate through each point in Dpcenti
for k = 1:size(Pi,2)
    % Iterate through each disk in the combination
    pkcovered = false;
    for l = 1:size(Di,2)
        % If |dl-pk|<1, the point is covered, break.
        % dl must be in the subset (j-th combination, l-th disk
        % of stringset = 1).
        if stringset(j,l) == '1' && sqrt((Di(1,l)-Pi(1,k))^2 + (Di(2,l)-Pi(2,k))^2) < 1
            pkcovered = true;
            break
        end
    end
    % If there is a point that is not covered, we add the
    % subset to a garbage set, to be removed from stringset
    % later.
    if pkcovered == false

34
garbageset(gspointer,:) = stringset(j,:);  
gspointer = gspointer + 1;  
break  
end  
end  
end  
garbageset = garbageset(1:gspointer-1,:);  
stringset = setdiff(stringset,garbageset,'rows');  

% Choose the smallest remaining subset as Dcovi.  
sumsarr = zeros(size(stringset,1),1);  
sumsarr(1) = sum(stringset(1,:)-48);  
for k = 2:size(stringset,1)  
    sumsarr(k) = sum(stringset(k,:)-48);  
end  
smallestpointer = 0;  
smallestvalue = realmax;  
for k = 1:size(stringset,1)  
    if sumsarr(k) < smallestvalue  
        smallestvalue = sumsarr(k);  
        smallestpointer = k;  
    end  
end  
if smallestvalue == realmax  
    error('No smallest value was found in sumsarr.');  
end  

% We now have a pointer that points a row in stringset st it is of  
% minimal cardinality. We must extract those points from Di. This  
% is Dcovi.  

diskbools = stringset(smallestpointer,:);  

% Note Di must be a kx2 matrix for this computation.  
Di = Di';  
tempset = [-1,-1];  
for k=1:size(diskbools)  
    if diskbools(k) == '1'  
        tempset = vertcat(tempset,Di(k,:));  
    end  
end
end
Dcovi = tempset'; % Assign Dcovi
% We now have Dcovi, the minimal covering set of unit disks of the
% points in Dpcenti.
end

% Remove Pi from Pset.
Pset = setdiff((Pset'),(Pi'),'rows');
str = [num2str(size(Pi,2)), ' points were removed.'];
disp(str);
str = ['There are ', num2str(size(Pset,2)), ' points remaining.'];
disp(str);

% Place the points in Dcovi in a solution array.

% Dcovi
SolArray = union((SolArray'),(Dcovi'),'rows');

% Start over again.
end

% Print out the solution: SolArray
SolArray(:,1) = [];

markedrelaysfile = fopen('HLmarkedrelays.txt','wt');
fprintf(markedrelaysfile,'%06.4f %06.4f
',SolArray);
fclose(markedrelaysfile);