

SIMULATION OF GRAPH LAYOUT ALGORITHMS FOR SENSOR NETWORKS

by

CHRISTOPHER HOGG

Advisors: Dr. Hala ElAarag Dr. Hari Pulapaka

A Senior research paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in the Department of Mathematics and Computer Science in the College of Arts and Science at Stetson University DeLand, Florida

Spring Term 2007

CONTENTS

List of Figures	2
1. Introduction	5
2. Related Work	6
3. Definitions	7
3.1. Essential terms	9
4. The Algorithms	9
4.1. The 2-Dimensional Case	10
4.2. How to Optimize Localized Stress Energy	11
4.3. The 3-Dimensional case	13
4.4. Algorithm Analysis	13
5. The Simulator	14
5.1. Existing simulators	14
5.2. Implementation of the network simulator	14
6. Future Work	15
References	15

LIST OF FIGURES

1	A Connected and a Disconnected graph	9
2	Examples of rigidity in Graphs	9
3	An example of a foldover.	10

ACKNOWLEDGEMENTS

I would like to thank my advisors for this project, Dr. Hala ElAarag and Dr. Hari Pulapaka for their aid and guidance in this project.

Abstract

The purpose of this project is to determine and study a distributed algorithm for locating nodes of a sensor network in 3-dimensional space and testing it to find what restrictions need to be placed on the network before the algorithm ceases to be effective. The project will also implement this algorithm using java, and test the algorithm on a simulated network that will keep track of not just the location of the nodes but will also monitor the energy consumed in running this algorithm and monitor the potential communication problems that would hinder it.

1. INTRODUCTION

A sensor network is a network of, usually, small sensors each of which comprises of a wireless radio transmitter/receiver, a small amount of memory and processing capability, a sensing device, and a power supply which is usually a battery of limited lifetime. These sensors are then networked together to perform certain tasks and are also often networked to one or more node of greater power supply and capabilities. The more advanced nodes act either as a data sink, which collects all relevant data, or as a manager that communicates with a user.

There are many uses for these types of networks because they are usually very cheap, some cost mere pennies per node, and yet they are capable of observing an area quite thoroughly with very little human interference. Indeed, many sensor networks are not 'placed' in the normal sense but are instead dispersed randomly in the area of interest. Networks formed in this manner naturally still need to communicate and thus the networks must construct themselves. This involves understanding the connectivity of the network and an understanding of how to route messages to specified nodes. However due to their energy limitations the nodes should communicate as little as possible during this phase. This is especially true in networks that are either entirely mobile or that have mobile elements in them because in such situations this process will have to be redone more often than other networks, if not be continuously running [3]. Along with needing to construct themselves upon initialization, these networks would then also need efficient routing algorithms to transmit data or queries among the nodes going to or from the sinks. These algorithms can and do use a wide variety of information to help them in the routing process in the hopes of increasing network lifetime. There are several methods that network aware algorithms use to improve performance. *Maximum PA routing* selects the route with the most available power (PA), *Minimum Energy routing* selects the route along which the least amount of energy consumed. There is also *Min hop routing* which naturally selects the route with the fewest number of 'hops', this is to say that it seeks to minimize the number of nodes that a transmission goes through before it reaches its destination. These methods of routing are not algorithms in and of themselves but rather examples of what types of routes an aware algorithm might seek to use. There are additional methods discussed in [3] and [12] that a network aware algorithm could follow. Conversely, there are routing methods that don't take conventional routing needs into consideration and instead disregard everything and

just transmit information using very simple routing algorithms that don't require information about the network. Examples of this minimalistic routing include "*Flooding*" in which when a node receives a message meant for a node other than itself it transmits it to all of its neighbors except the one that sent the message to it, eventually the messages either get where they are going or they die off after a certain number of hops [3]. Another example is "*Gossiping*" which sends the message along to a random neighbor until it gets where its going. In this technique instead of multiple copies of the message flooding the network, there is only one copy of the message that may take a long time with many hops to get where it is going [3]. There are many other methods discussed in [12], each with its own advantages and disadvantages. The best method of routing is usually dependent on the user or application and so no method or algorithm can really be said to be globally better than any other.

2. RELATED WORK

Many have studied the initialization problem to find an efficient algorithm to construct the network, or devise a routing protocol that can bypass the need to construct the network. These networks have been modeled or studied as many different mathematical objects. Examples include random graphs and geometric graphs. These models have then been used to study network properties such as graph connectivity, area or surface area coverage, life of the network, and energy consumption, just to name a few.

These mathematical models have also been used to devise algorithms that can construct a good idea of the relative and exact positions of the nodes and the topology of the network as applied to data routing. An important area of study for our purposes is the study of efficiently routing with the least amount of node information. Cone Based Topology Control (CBTC) is a protocol where the nodes vary their transmission power to get good descriptions of how much power is needed to reach certain neighbors and using this information to maintain a fault-tolerant connected network in a distributed manner [10]. Another algorithm derived for this same purpose is the Local Minimum Spanning Tree (LMST) algorithm [9]. It outperforms CBTC but it requires more information, specifically location information which can be gotten through expensive hardware or by means of inter-node communications, which usually requires a large number of messages which consumes a lot of the limited battery life [9].

In this research we are specifically interested in determining locations of nodes in \mathbb{R}^3 , in a distributed manner. A distributed algorithm would lower messaging costs in determining node locations and therefore make using algorithms like LMST, which requires location information, a more feasible option. There are already several algorithms which are capable of doing this in \mathbb{R}^2 and they serve as a basis for extending the solution to \mathbb{R}^3 . [1, 2] This would not be the first time that a 2-dimensional algorithm has been extended into 3-dimensions. In [10], Mohsen Bahramgiri, Mohammadtaghi Hajiaghayi and Vahab s. Mirrokni presented a clear way to extend the 2-dimensional CBTC into a 3-dimensional version of CBTC. Of course their work focuses on maintaining a connected network rather than determining the locations of the nodes as in our focus.

3. DEFINITIONS

In this section we will define the essential terms related to our problem. Most of the notations we use are adopted from [1]. The LAYOUT PROBLEM can be defined as follows:

Given a graph $G(1, \dots, n, E)$, and for each edge $\langle i, j \rangle \in E$ - its length l_{ij} as estimated by the node, find an optimal layout (p_1, \dots, p_n) where $(p_i \in \mathbb{R}^d$ is the location of sensor i), which satisfies for all $i \neq j$:

$$\begin{cases} \|p_i - p_j\| = l_{ij} & \text{if } \langle i, j \rangle \in E \\ \|p_i - p_j\| > R & \text{if } \langle i, j \rangle \notin E \end{cases} \quad \text{where } R = \max_{\langle i, j \rangle \in E} l_{ij}.$$

$$\text{full stress energy} = \sum_{i < j} (d_{ij} - l_{ij})^2 / l_{ij}^2 \quad (3.1)$$

where $d_{ij} = \|p_i - p_j\|$, the Euclidian distance between p_i to p_j . The *localized stress energy* is:

$$\text{Stress}(x, y) = \sum_{i, j \in E} (d_{ij}(x, y) - l_{ij})^2 \quad (3.2)$$

It is important to note that unlike full stress energy, localized stress energy is not normalized. The energy function is

$$E(x) = \sum_{\langle i, j \rangle \in E} w_{ij} \|x_i - x_j\|^2 / \sum_{i < j} \|x_i - x_j\|^2 \quad (3.3)$$

where x represents the coordinate position of the node. Thus for the 2-dimensional case, $x = (x, y)$. Next, w_{ij} is a measure of the similarity of adjacent sensors i and j and is derived from l_{ij} . Gotsman and Koren [1]

suggested

$$w_{ij} = e^{-l_{ij}} \quad (3.4)$$

Note that $E(x)$ is defined also in higher dimensions. Let D be the diagonal matrix where the i th diagonal is the sum of the i th row of $W = (w_{ij})$. Thus

$$D_{ii} = \sum_{j: \langle i, j \rangle \in E} w_{ij} \quad (3.5)$$

In [1] the authors found that the global minimum of $E(x)$ is the eigenvector of the related weighted Laplacian matrix $L^w = D - W$ associated with the smallest positive eigenvalue. However they found it easier to deal with the eigenvectors of the associated transition matrix $D^{-1}W$, because instead of looking for the eigenvectors with the smallest associated eigenvalue, we will be looking for the eigenvectors with the largest eigenvalues, which are easier to find in the manner they employed and that we will be employing. The first eigenvector v_1 , the eigenvector associated with the largest eigenvalue, is $1_n = (1, 1, \dots, 1)$ which is not of much interest to us. However the next one is our initial x , the one after that is our initial y . We say that y is D -orthogonal iff $y^T D v_1 = 0$.

Lemma 1: *Given two vectors x and y and matrices D and A , the vector Ay is D -orthogonal to x if $A^T D x = 0$.*

Proof: Since $A^T D x = 0$, then $y^T A^T D x = 0$. Equivalently $(Ay)^T D x = 0$ and the lemma follows \square

Let $A = (a_{ij})$ be a matrix defined as follows. For $i, j = 1, \dots, n$:

$$a_{i,j} = \begin{cases} -x_j/D_{ii} & \text{when } \langle i, j \rangle \in E \\ 0 & \text{when } \langle i, j \rangle \notin E, i \neq j \\ -\sum_k A_{i,k} & \text{when } i = j \end{cases} \quad (3.6)$$

Power iteration is a matrix procedure in which a vector b is repeatedly multiplied by a matrix M and then normalized before storing it as b prior to the next iteration. This has the effect of scaling b in the direction of each of M 's eigenvectors in proportion to the associated eigenvalues. If b is D -orthogonal to one of the eigenvectors, then it won't be scaled in that direction. This yields the dominant eigenvector in whose

direction b gets scaled. A distributed form of *Power iteration* is demonstrated as follows:

$$x_i \leftarrow a * \left(x_i + \frac{\sum_{\langle i,j \rangle \in E} w_{ij} x_j}{\sum_{\langle i,j \rangle \in E} w_{ij}} \right) \quad (3.7)$$

where a is a constant, that controls growth. Using the suggestion of [1], we choose $a = 0.51$. The *degree of node i* , denoted as deg_i is the number of nodes that node i is connected to.

3.1. Essential terms. A network/graph is **connected** if for every two nodes in the network, there is a path connecting the two nodes. The algorithm we will be developing only works on connected graphs. Examples of a connected and a disconnected graph are below. A network is said to be **rigid** if the location of any single

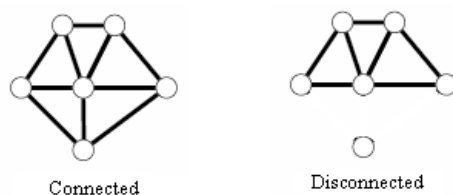


FIGURE 1. A Connected and a Disconnected graph

sensor cannot be changed without changing at least one known distance, or applying a rigid transformation to the entire network. A **foldover** exists when an entire piece of a graph can fold over on top of others

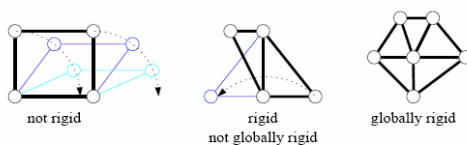


FIGURE 2. Examples of rigidity in Graphs

without violating any of the distance constraints. As can be seen in figure 3 nodes e and f folded over node d without violating any of the constraints. A graph is **fold free** when there cannot be any foldovers. This is important to note because if the graph is capable of folding then we cannot guarantee a unique solution.

4. THE ALGORITHMS

In this section we first present the algorithm for locating nodes in two dimensions. This includes a description of the algorithm and a step-by-step walkthrough of the process. We also present an in-depth

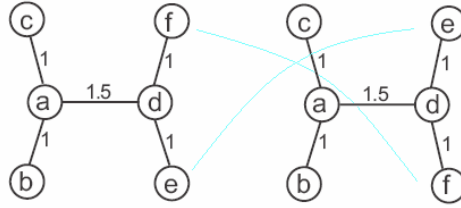


FIGURE 3. An example of a foldover.

description of an algorithm used to optimize localized stress energy followed by a description of how to extend the layout problem into three dimensions.

4.1. The 2-Dimensional Case. In this section we present the algorithm for solving the 2-dimensional layout problem as per the method described in [1]. The algorithm is designed to concurrently run on all of the nodes in the network. It is made to only know information about the direct neighbors - specifically all that is known is a rough estimate of the distance to its neighbors and the estimated position at each of its neighbors. A clear outline of the algorithm is presented below.

- (1) Estimate distance to neighbors. There are many methods to do so. For example, we can use round trip time as an indicator. Other methods include a distance measuring probe, and using power drain per communication with a node to estimate distance. But all of these methods come with a degree of error that if too high could cause the process to fail.
- (2) Store these estimated distances as a matrix L .
- (3) Use the values from L to construct W as per equation 3.4. Note that since nodes don't contain full information, each node only possesses one row of W . So we need not store the entire matrix.
- (4) Construct D as per equation 3.5.
- (5) Construct A as per equation 3.6 using $x = (1, 1, \dots, 1)$. Again each node need only store one row of the matrix.
- (6) Choose a random vector u . It is best if all the nodes used the same u and since u can be any random vector a hardcoded u vector should be acceptable.
- (7) Calculate Au and use it as the initial vector g . Note that each node will only know one of the values of g . Specifically, it will know only its estimated location, each node can however find out its neighbors

values in g . The reason we use Au for g is because it is D -orthogonal to the input vector when making A . The reason we know this is because the A we constructed met the property that $A^T Dv_1 = 0$ and according to Lemma 1, this condition gaurontees D -orthogonality.

- (8) Perform power iteration on g . Here, we have each node perform one iteration of equation 3.7 followed by a sharing of the new g values and then this step is repeated. However, due to upcoming needs for the minimum and maximum values in g to be different by 1 we scale g every several iterations. When the stopping criteria are met g is the initial x values for the layout.
- (9) Reconstruct A , this time use the just derived x as the x in equation 3.6.
- (10) Repeat steps 6 through 8. This time however we need to not only scale but also translate g every several iterations to guarantee that its minimum and maximum are seperated by 1 unit. This results in the initial y .
- (11) Use (x, y) as the initial layout and minimize the *localized stress energy* to get a near optimal layout.

The exact algorithm for this is below.

4.2. How to Optimize Localized Stress Energy. To optimize the stress energy we use a method called *majorization*. We start by using the Cauchy-Schwartz inequality to apply bounds to the stress function yielding:

$$Stress(x, y) \leq x^T Lx + y^T Ly + x^T L^{a,b} a + y^T L^{a,b} b + c \quad (4.1)$$

where $x, y, a, b \in \mathbb{R}^n$, $x = a$ and $y = b$ and c is a constant independent of the others. L is the graph's unweighted ($n \times n$) Laplacian matrix, which is also independent of the others. L is defined as follows for $i, j = 1, \dots, n$:

$$L_{i,j} = \begin{cases} -1 & \text{when } \langle i, j \rangle \in E \\ 0 & \text{when } \langle i, j \rangle \notin E \\ -\sum_{j \neq i} L_{i,j} & \text{when } i = j \end{cases} \quad (4.2)$$

$L^{a,b}$ is defined as follows for $i, j = 1, \dots, n$:

$$L_{i,j}^{a,b} = \begin{cases} -l_{ij} * inv \left(\sqrt{(a_i - a_j)^2 + (b_i - b_j)^2} \right) & \text{when } \langle i, j \rangle \in E \\ 0 & \text{when } \langle i, j \rangle \notin E \\ -\sum_{j \neq i} L_{i,j}^{a,b} & \text{when } i = j \end{cases} \quad (4.3)$$

where $inv(x) = 1/x$ unless $x = 0$ in which case $inv(x) = 0$. Given a layout (a, b) we can find another layout (x, y) which minimizes the right side of the above equation. Since the right side decreases in value we know the *stress* hasn't increased. Thus we can repeatedly find a new right side potentially reducing *stress* each time. We find this new layout by solving the following linear equations:

$$Lx = L^{(a,b)}a \quad (4.4)$$

$$Ly = L^{(a,b)}b. \quad (4.5)$$

These equations can be rewritten in terms of $x(t), y(t), x(t+1), y(t+1)$ in the natural manner to make it clearer that it is an iterative process producing

$$Lx(t+1) = L^{(x(t),y(t))}x(t) \quad (4.6)$$

$$Ly(t+1) = L^{(x(t),y(t))}y(t). \quad (4.7)$$

Without loss of generality fix one of the sensor locations and we can obtain a strictly diagonally dominant matrix which we can safely solve using Jacobian iteration. This yields the following equations:

$$x_i \leftarrow 1/deg_i \sum_{j:\langle i,j \rangle \in E} (x_j + l_{ij}(x_i(t) - x_j(t)) inv(d_{ij}(t))) \quad (4.8)$$

$$y_i \leftarrow 1/deg_i \sum_{j:\langle i,j \rangle \in E} (y_j + l_{ij}(y_i(t) - y_j(t)) inv(d_{ij}(t))) \quad (4.9)$$

After these equations converge, which is to say that each iteration changes (x_i, y_i) by no more than some specified amount, there are similar but faster equations that could be used because scaling is no longer an issue. It is important to note that, as mentioned in [1], the solution will have translation, orientation and reflection degrees of freedom but that these can be solved by fixing several node positions and correcting to them, if you need exact locations of the nodes.

4.3. The 3-Dimensional case. This section presents the extension of the above mentioned algorithm into 3-dimensions. The first part of the extended algorithm will be very similar to the 2-dimensional algorithm, however the procedure to determine y will differ and it will naturally have to find the third coordinate z as well prior to moving on to utilizing a 3-dimensional version of stress optimization. The x and y coordinates should once again come out as the 2nd and 3rd eigenvectors and a good starting z coordinate should be the 4th eigenvector. This is why there is an alteration in how y and z will be found, because it could become tedious to find them using *power iteration* since power iteration will naturally try to find the eigenvectors associated with the larger eigenvalues. Hence we separate the network into disjoint subsets all with cardinality of at least 3, with the possibility of singletons being left over. Then the singletons will set their z value as 0 while the subsets will have a single node determine the z values and assign its neighbors these values. This is because the assigning node should be connected to all of its neighbors and so know everything about its little local network and can thus can find D-orthogonal vectors very easily. Once it has determined these values it will assign them out to its neighbors and after all subsets have done this the network is ready to enter into a stress optimization phase. It is important to note that both y and z will be assigned out by the single node in each subset however x will be found the same way as in the two-dimensional case.

4.4. Algorithm Analysis. This section considers the complexity of the algorithms. An algorithm's complexity is a relationship between the amount of resources, be they time or memory, that the algorithm needs in order to solve a given problem and the size of the problem. For example it would correlate how long it would take to sum up a set of numbers and the size of the set. *Big O* notation is a way of expressing an upper bound on an algorithms resource usage as a function of input size, always in terms of the dominant function. *Worst-Case* complexity refers to the complexity of the algorithm for the least efficient case. *Average-case* refers to complexity given that the average situation occurs at every input size. To determine the complexity of an algorithm you must first determine what its "performance" is a function of, although sometimes you may find that it is a function of more than one variable. For the algorithms we are using it is clear that their complexity is a function of the degree of the node. To determine the complexity we would vary the input and construct a table of complexity vs. input and then attempt to match that to a known pattern,

such as matching it to $f(x) = ax + c$ or $f(x) = ax^2 + bx + c$, of course the second of those would be written as $O(x^2)$. In this project, we will also analyze the computational complexity of all of our algorithms.

5. THE SIMULATOR

5.1. Existing simulators. The increasing number and variety of algorithms and protocols for efficient sensor networks has led to an increasing need for accurate simulators for these networks. There are many network simulators in use. Perhaps, the most popular network simulator is NS-2 [11], however this simulator does not easily lend itself to simulating sensor networks due to the specialized information that is needed on each node and the difficulty of adding in new features to it. There are extensions packages available for it, however all of these packages have their own limitations on what they look at and what they provide for [4]. There are other simulators available such as J-sim, OMNETT++, DiSenS and Sense [2, 3, 4, 5, 6, 7, 8], that are constructed for various intended uses, naturally each has its own strengths and weaknesses. Some of these network simulators are object-oriented in nature while others are component-oriented in nature. These simulators have been written in a wide variety of languages and or language combinations . All this variation has led to a growing basis for testing new protocols and algorithms, however, often many restrictions are placed on the testing. Be these limitations on the number of nodes, or on the type of protocol that can be tested, or on what information can be gathered about each node, or they are limited due to increasing time cost for each added node or connection.

For this project we hope to create a netowrok simulator that will provide for an ad hoc sensor network placed in $3D$ space with mobile nodes. It will simulate battery life, realistic power consumption, noise and allow for the running of procedures on the nodes. Specifically it will run our derived distributed procedure for locating of nodes in $3D$. This procedure will then be tested through various rules determined through the mathematical analysis of the algorithm in order to determine its adaptability and robustness.

5.2. Implementation of the network simulator. The simulation is written in java. The node class stores only the information that its been given, so has no actual knowledge of where it truly is, and provides procedures vital to the implementation of the 2-dimensional algorithm. The network class is responsible for knowing where things actually are and for calling the procedures within the node class, as well as passing

information from one node to the next. It doesn't simulate any timesteps or the like so it merely keeps the correct order of events. It currently doesn't implement possible error in finding the distances between nodes or message collisions.

Both *power iteration* and *2-d stress optimization* have been implemented in java. They are procedures within the node class which perform one iteration of the local function and therefore rely on the node to get all relevant information from its neighbors before calling the procedure again.

6. FUTURE WORK

The next step is to begin designing the actual network, with the full functionality. It will need to keep track of when events occur, the power each node has, the energy usage of each node for each activity, the transmission and possible collision of messages, the noise in the network, the mobility of the nodes and the demands of the network to name a few. Then this must be implemented in as efficient a manner as possible to conserve the limited resources of the nodes and provide for dense network testing. While this is happening we will also be exploring the 3-dimensional algorithm and testing it to find what restrictions need to be placed on the network such as the connectivity demands, and how many singletons can be allowed before the algorithm falls through.

REFERENCES

- [1] Craig Gotsman, Yehuda Koren, "Distributed Graph Layout for Sensor Networks", Journal of Graph Algorithms and Applications, Vol. 9 2005.
- [2] Nissanka B. Priyantha, Hari Balakrishnan, Erik Demaine, and Seth Teller, "Anchor-Free Distributed Localization in Sensor Networks", MIT Laboratory for Computer Science Tech Report #892, April 15, 2003.
- [3] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci, "A Survey on Sensor Networks", IEEE Communications Magazine, Aug, 2002
- [4] Boleslaw K. Szymanski and Gilbert Gang Chen, "Sensor Network Component Based Simulator", Handbook of Dynamic System Modeling, chap. 35, 2007
- [5] Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, and Honghai Zhang, "J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks"
- [6] <http://www.ita.cs.rpi.edu/sense/index.html>, "SENSE: Sensor Network Simulator and Emulator"

- [7] C. Mallanda, A. Suri, V. Kunchakarra, S.S. Iyengar, R. Kannan, and A. Durrezi, "Simulating Wireless Sensor Networks With OMNeT++"
- [8] Ye Wen, Rich Wolski, Gregory Moore, "DiSenS: Scalable Distributed Sensor Network Simulation", ACM, 2007
- [9] Paoli Santi, "Topology Control in Wireless Ad Hoc and Sensor Networks", ACM Computing Surveys, Vol. 37, No. 2, June 2005, pp. 164-194
- [10] Mohsen Bahramgiri, Mohammadtaghi Hajiaghayi and Vahab s. Mirrokni, "Fault-Tolerant and 3-Dimensional Distributed Topology Control Algorithms in Wireless Multi-hop Networks", Wireless Networks 12, pg 179-188, 2006
- [11] <http://www.isi.edu/nsnam/ns/> the NS-2 website
- [12] Jamal N. Al-Karaki, Ahmed E. Kamal, "Routing Techniques In Wireless Sensor Networks: A Survey"