Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 9, no. 3, pp. 327–346 (2005)

Distributed Graph Layout for Sensor Networks

Craig Gotsman

Department of Computer Science
Harvard University, Cambridge, MA 02138
gotsman@cs.technion.ac.il

Yehuda Koren

AT&T Labs – Research
Florham Park, NJ 07932
yehuda@research.att.com

Abstract

Sensor network applications frequently require that the sensors know their physical locations in some global coordinate system. This is usually achieved by equipping each sensor with a location measurement device, such as GPS. However, low-end systems or indoor systems, which cannot use GPS, must locate themselves based only on crude information available locally, such as inter-sensor distances. We show how a collection of sensors, capable *only* of measuring distances to close neighbors, can compute their locations in a purely distributed manner, i.e. where each sensor communicates only with its neighbors. This can be viewed as a distributed graph drawing algorithm. We experimentally show that our algorithm consistently produces good results under a variety of simulated real-world conditions, and is relatively robust to the presence of noise in the distance measurements.

Article Type	Communicated by	Submitted	Revised
Regular paper	E. R. Gansner and J. Pach	November 2004	July 2005

1 Introduction

Sensor networks are a collection of (usually miniature) devices, each with limited computing and (wireless) communication capabilities, distributed over a physical area. The network collects data from its environment and should be able to integrate it and answer queries related to this data. Sensor networks are becoming more and more attractive in environmental, military and ecological applications. See [14] for a survey of this topic.

The advent of sensor networks has presented a number of research challenges to the networking and distributed computation communities. Since each sensor can typically communicate only with a small number of other sensors, information generated at one sensor can reach another sensor only by routing it through the network, whose connectivity is described by a graph. This requires ad hoc routing algorithms, especially if the sensors are dynamic. Traditional routing algorithms relied only on the connectivity graph of the network, but with the introduction of so-called *location-aware* sensors, namely, those who also know what their physical location is, e.g. by being equipped with a GPS receiver, this information can be used to perform more efficient *geographic* routing. See [10] for a survey of these routing techniques.

Beyond routing applications, location-aware sensors are important for information dissemination protocols and query processing. Location awareness is achieved primarily by equipping the sensors with GPS receivers. These, however, may be too expensive, too large, or too power intensive for the desired application. In indoor environments, GPS does not work at all (due to the lack of line-of-sight to the satellites), so alternative solutions must be employed. Luckily, sensors are usually capable of other, more primitive, geometric measurements, which can aid in this process. An example of such a geometric measurement is the distance to neighboring sensors. This is achieved either by Received Signal Strength Indicator (RSSI) or Time of Arrival (ToA) techniques. An important question is then whether it is possible to design a distributed protocol by which each sensor can use this local information to (iteratively) compute its location in some global coordinate system.

This paper solves the following sensor layout problem: *Given a set of sensors distributed in the plane, and a mechanism by which a sensor can estimate its distance to a few nearby sensors, determine the coordinates of every sensor via local sensor-to-sensor communication.* These coordinates are called a *layout* of the sensor network.

As stated, this problem is not well-defined, because it typically will not have a unique solution. A unique solution would mean that the system is *rigid*, in the sense that the location of any individual sensor cannot be changed without changing at least one of the known distances, or by transforming all locations by some rigid transformation. When all $\binom{n}{2}$ inter-sensor distances are known, the solution is indeed unique (up to a rigid transformation), and is traditionally solved using the Classical Multidimensional Scaling (MDS) technique [1]. When only a subset of the distances are known, more sophisticated techniques must be used.

When multiple solutions exist, the main phenomenon observed in the solutions is that of *foldovers*, where entire pieces of the graph fold over on top of others, without violating any of the distance constraints. The main challenge is to generate a solution which is fold-free. Obviously the result will have translation, orientation and reflection degrees of freedom, but either these are not important, or can be resolved by assigning some known coordinates to three sensors.

In real-world sensor networks, noise is inevitable. This manifests itself in the inter-sensor noise measurements being inaccurate. Beyond the obvious complication of the distances possibly no longer being symmetric, thus violating the very essence of the term "distance", there may no longer even exist a solution realizing the measured edge lengths. The best that can be hoped for, in this case, is a layout whose coordinates are, up to some acceptable tolerance, close to the true coordinates of the sensors.

In order to be easily and reliably implemented on a sensor network, the solution to this sensor network layout problem should be fully distributed (decentralized). This means that each sensor should use information available *only* at that sensor and its immediate neighbors. The class of neighbors is typically characterized by a probabilistic variant of the disk graph model: Any sensor within distance R_1 is reachable, any sensor beyond distance R_2 is not reachable, and any sensor at a distance between R_1 and R_2 is reachable with probability p. Of course, information from one sensor may eventually propagate through the network to any other sensor, but this should not be done explicitly.

2 Related Work

The problem of reconstructing a geometric graph given its edges lengths has received some attention in the discrete geometry and computational geometry communities, where it is relevant for molecule construction and protein folding applications [6]. Deciding whether a given graph equipped with edge lengths is rigid in 2D - i.e. admits a unique layout realizing the given edge lengths (up to a rigid transformation) - is possible in polynomial time for the dense class of graphs known as *generic* graphs [7]. However, computing such a layout is in general NP-hard [16]. This does not change even if a layout is known to exist (as in our case).

The problem of distributed layout of a sensor network has received considerable attention in the sensor network community. A recent work of Priyantha *et al.* [12] classifies these into *anchor-based* vs. *anchor-free* algorithms and *incremental* vs. *concurrent* algorithms. Anchor-based algorithms rely on the fact that some of the sensors are already aware of their locations, and the locations of the others are computed based on those. In practice, a large number of anchor sensors is required for the resulting location errors to be acceptable. Incremental algorithms start with a small core of sensors that are assigned coordinates. Other sensors are repeatedly added to this set by local trigonometric calculations. These algorithms accumulate errors and cannot escape local minima once they are entered. Concurrent algorithms work in parallel on all sensors. They are better able to avoid local minima and avoid error accumulation. Priyantha *et al.* [12] review a number of published algorithms and their classifications. All of them, however, are not fully distributed.

Moore *et al.* [11] describe an incremental algorithm which attempts to localize small "patches" of the network. Each such patch is a set of four sensors forming a rigid quadrilateral, which is localized using a procedure called "trilateration" and then improved by a local optimization process. Given one quad which has been localized, another quad is found which has some sensors in common with the first. This is then laid out relative to the first by applying the best possible rigid transformation. In such a manner, a sequence of quads is laid out in breadth-first order until no more sensors can be localized. This algorithm will localize only those sensors contained in a rigid component of the network, and will not produce anything useful for the other sensors.

A similar "patching" algorithm was described by Shang and Ruml [13]. However, they localize an individual patch by first computing all pairwise shortest paths (in the weighted network connectivity graph) between sensors in the patch. MDS is then applied to these distances to yield an initial layout, which is subsequently improved by stress minimization. The patches are "stitched" together incrementally in a greedy order by finding the best affine transformation between a new patch and the global layout. A post-processing stage of the algorithm further improves layout quality by minimizing the "stress energy" of the complete network.

These incremental algorithms seem to be sound, but, unfortunately, like any other incremental algorithm, may accumulate error indefinitely, especially when the pairwise distances are significantly noisy.

The algorithm we describe in this paper is most similar in spirit to the so-called Anchor-Free Localization (AFL) algorithm proposed by Priyantha *et al.* [12]. The AFL algorithm operates in two stages. In the first stage a heuristic is applied to try to generate a well-distributed fold-free graph layout which "looks similar" to the desired layout. The second stage applies a stress-minimization optimization procedure to correct and balance local distance errors, converging to the final result. The heuristic used in the first stage involves the selection of five reference sensors. Four of these sensors are well-distributed on the periphery of the network, and serve as north, east, south and west poles. A fifth reference sensor is chosen at the center. Coordinates are then assigned to all nodes, using these five sensors, reflecting their assumed positions. Unfortunately, this process does not lend itself easily to distribution. The second stage of the AFL algorithm attempts to minimize the partial stress energy using a gradient descent technique. At each sensor, the coordinates are updated by moving an infinitesimal distance in the direction of the spring force operating on the sensor. This is a fully distributed protocol. It, however, involves a heuristic choice of the infinitesimal step, and can be quite slow.

Our algorithm also involves two stages with similar objectives. The first aims to generate a fold-free layout. This is done based on a distributed Laplacian eigenvector computation which typically spreads the sensors well. The second stage uses the result of the first stage as an initial layout for an iterative stress-minimization algorithm. As opposed to AFL, it is not based on gradient descent, but rather on a more effective *majorization* technique.

Once again we emphasize that the main challenge is to design algorithms which are *fully* distributed. This is a major concern in sensor network applications, and there is an increasing interest in designing such solutions. These turn out sometimes to be quite non-trivial. Probably the simplest example is a distributed algorithm to compute the sum (or average) of values distributed across the network. See [15] for a discussion on this.

3 The Problem

We are given a n-vertex graph $G(V = \{1, \dots, n\}, E)$, and for each edge $\langle i,j \rangle \in E$ - its Euclidean "length" l_{ij}. Denote a 2D layout of the graph by $x, y \in \mathbb{R}^n$, where the coordinates of vertex i are $p_i = (x_i, y_i)$. Denote $d_{ij} = \|p_i - p_j\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

In the non-noisy version of the problem, we know that there exists a layout of the sensors that realizes the given edge lengths (i.e. $d_{ij} = l_{ij}$). Our goal is then to reproduce this layout. This layout is usually not unique. For example, consider a $2n \times 2n$ square grid, where each internal sensor is connected to its four immediate neighbors with an edge of length one. We can realize all lengths using the degenerate 1D layout where half of the sensors are placed on 0 and the other half is placed on 1. Specifically, given a sensor with grid coordinates (r, c), we place it on point 0 if $r + c$ is even; otherwise, we place it on point 1.

Fortunately, there is additional information which we may exploit to eliminate spurious solutions to the layout problem - we know that the graph is a complete description of the close sensors. Consequently, the distance between each two nonadjacent sensors should be greater than some constant r, which is larger than the longest edge. This can further constrain the search space and eliminate most undesired solutions. Formally, we may pose our problem as follows:

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

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

where $R = \max_{\langle i,j \rangle \in E} l_{ij}$. For the rest of this paper we assume that the sensors are embedded in the plane, namely $d = 2$.

It seems that an optimal layout is unique (up to translation, rotation and reflection) in many practical situations. For example, it overcomes the problem in the $2n \times 2n$ grid example described above. However, there are graphs for which even an optimal layout is not unique. For example, consider the 6-sensor graph in Fig. 1, which shows two different optimal layouts.

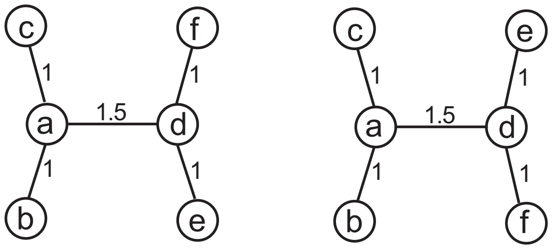

Figure 1: Two different optimal layouts of the same graph.

An optimal layout is similar to that generated by common force-directed graph drawing algorithms that place adjacent nodes closely while separating nonadjacent nodes. Therefore, we may exploit some known graph drawing techniques. For example, separating nonadjacent sensors can be achieved by solving an electric-spring system with repulsive forces between these sensors [2, 3]. Another possibility is to somehow estimate the distances l_{ij} between nonadjacent sensors (e.g., as the graph-theoretic distance) and then to minimize the *full stress energy*: $\sum_{i<j} \frac{(d_{ij}-l_{ij})^2}{l_{ij}^2}$ using an MDS-type technique; see [8].

However, since we aim at a distributed algorithm which should minimize communication between the sensors, dealing with repulsive forces or long-range target distances is not practical, as this will involve excessive inter-sensor interaction, which is very expensive in this scenario. To avoid this, we propose an algorithm which is based only on direct information sharing between adjacent sensors, avoiding all communication between nonadjacent sensors or any centralized supervision. Note that such a restriction rules out all common algorithms for general graph drawing problem; we are not aware of any layout algorithm that satisfies it.

In the real-life noisy version of the problem, the measured distances l_{ij} are contaminated by noise: $l_{ij} = d_{ij} + \epsilon_{ij}$. This means that there might not even exist a solution to the optimal layout problem. In this case we would like to minimize the difference between the true location of the sensors and those computed by the algorithm.

4 Smart Initialization and Eigen-projection

A useful energy function which is minimized by the desired layout is the *localized stress energy*:

$$\text{Stress}(x, y) = \sum_{\langle i,j \rangle \in E} (d_{ij}(x, y) - l_{ij})^2 \qquad (1)$$

Note that this energy is not normalized, as opposed to the full stress energy. This non-convex energy function may have many local minima, which an optimizer may get stuck in. However, since in the non-noisy case, we are guaranteed the existence of a layout where $d_{ij} = l_{ij}$, namely Stress(x, y) achieves the global minimum of zero, it is reasonable to hope that if we start with the optimization process at a "smart" initial layout, the process will converge to this global minimum.

To construct such an initial layout, we exploit the fact that nonadjacent sensors should be placed far apart. This means that we seek a layout that spreads the sensors well. We first deal with the one-dimensional case. We will design an energy function which is minimized by such a layout, and can be optimized in a strictly distributed fashion. The function is defined as follows:

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

Here, w_{ij} is some measure for the similarity of the adjacent sensors i and j. It should be obtained from l_{ij}, e.g., $w_{ij} = 1/(l_{ij} + \alpha)$ or $w_{ij} = \exp(-\alpha l_{ij})$, $\alpha \geqslant 0$; in our experiments we used $w_{ij} = \exp(-l_{ij})$. Minimizing $E(x)$ is useful since it tries to locate adjacent sensors close to each other while separating nonadjacent sensors. It can also be solved fairly easily. Denote by D the diagonal matrix whose i'th diagonal entry is the sum of the i'th row of W: $D_{ii} = \sum_{j:\langle i,j\rangle \in E} w_{ij}$. 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; see [5, 9]. In practice, it is better to work with the closely related eigenvectors of the transition matrix $D^{-1}W$, which have some advantages over the eigenvectors of L^w; see [9]. Note that the top eigenvalue of $D^{-1}W$ is $\lambda_1 = 1$, associated with the constant eigenvector $v_1 = 1_n = (1, 1, \dots, 1)$, so the desired solution is actually the *second* eigenvector v_2.

The vector v_2 can be computed in a distributed manner by iteratively averaging the value at each sensor with the values of its neighbors:

$$x_i \leftarrow a \cdot \left(x_i + \frac{\sum_{\langle i,j\rangle \in E} w_{ij} x_j}{\sum_{\langle i,j\rangle \in E} w_{ij}} \right) \qquad (3)$$

Readers familiar with numerical linear algebra will recognize this process as power iteration of the matrix $I + D^{-1}W$. Power iteration usually converges to the eigenvector of the iterated matrix corresponding to the eigenvalue with highest absolute value. However, here we initialize the process by a vector y which is D-orthogonal to v_1, namely $y^T D v_1 = 0$, using a distributed method that will be described shortly. Hence, the process will converge to v_2 - the next highest eigenvector of $I + D^{-1}W$ (or, equivalently $D^{-1}W$); see [9]. D-orthogonality, rather than simple orthogonality, is required because $D^{-1}W$ is not symmetric. The constant $a > 0$ controls the growth of $\|x\|$; in our implementation we used $a = 0.51$.

4.1 Two dimensional layout

We now turn our attention to the two-dimensional layout problem. $E(x)$ is defined also in higher dimensions (where x is short for (x, y)), and a "smart" initial 2D layout is achieved by taking the x coordinate to be v_2 - the second eigenvector of $D^{-1}W$, and the y coordinate to be v_3 - the third eigenvector of $D^{-1}W$. Unfortunately, the power iteration (3) will not detect v_3, as it is dominated by v_2, unless we start the process (3) with a vector D-orthogonal to $x = v_2$.

Constrained by the distributed computation requirement, it is not easy to initialize the process with a vector D-orthogonal to v_2. We resort to the following lemma:

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

Proof: Since $A^T Dx = 0$, then $y^T A^T Dx = 0$. Equivalently $(Ay)^T Dx = 0$ and the lemma follows. □

Therefore, it suffices to construct a "local matrix" A such that $A^T Dx = 0$. By "local" we mean that $A_{i,j} \neq 0$ only if $\langle i, j\rangle \in E$. This will enable a distributed computation. In our case when D is diagonal, a suitable matrix is the following:

$$A_{i,j} = \begin{cases} -x_j/D_{ii} & \langle i,j\rangle \in E \\ 0 & \langle i,j\rangle \notin E, i \neq j \\ -\sum_k A_{i,k} & i = j \end{cases} \qquad i,j = 1,\dots,n$$

It is easy to verify that $A^T Dx = 0$.

To summarize, to obtain $y = v_3$, we pick some random vector u, and initialize y with Au. Note that the computation of Au involves only local operations, and can be easily distributed. Then, we run the power iteration (3) on the vector y. While the initial vector is D-orthogonal to v_2, it is not necessarily D-orthogonal to $v_1 = 1_n$. Hence, after many iterations, the result will be $y = \alpha v_1 + \epsilon v_3$, for some very small ϵ. While the process ultimately converges to what seems to be an essentially useless vector, its values near the limit is what is interesting. Since v_1 is the constant vector - 1_n, these values are essentially a scaled version of v_3 displaced by some fixed value (α) and they still retain the crucial information we need.

However when the numerical precision is low and the ratio α/ϵ is too high, we might lose the v_3 component. Fortunately, we can work around this by *translating and scaling* y during the power iteration. Specifically, every βn iterations (we use $\beta = 1/2$) compute $\min_i y_i$ and $\max_i y_i$. A distributed computation is straightforward and can be completed with the number of iterations bounded by the diameter of the graph (at most $n - 1$). Then, linearly transform y by setting

$$y_i \leftarrow \frac{y_i - \min_i y_i}{\max_i y_i - \min_i y_i} - \frac{1}{2}, \quad i = 1, \dots, n \tag{4}$$

After this, $\min_i y_i = -0.5$ and $\max_i y_i = 0.5$. Since translation is equivalent to the addition of γv_1 and scaling cannot change direction, we can still express y as $\hat{\alpha} v_1 + \hat{\epsilon} v_3$.

Now assume, without loss of generality, that $\max_i v_3 - \min_i v_3 = 1$, and recall that $v_1 = (1, 1, \ldots, 1)$. The D-orthogonality of v_3 to 1_n implies: $\max_i v_3 > 0$ and $\min_i v_3 < 0$. In turn, $\min_i y_i = -0.5$ and $\max_i y_i = 0.5$ imply that $|\hat{\alpha}| < 0.5$. Moreover, since all the variability of y is due to its v_3 component, we get $\hat{\epsilon} = 1$. Therefore, (4) guarantees that the magnitude of the v_3 component is larger than that of the v_1 component, avoiding potential numerical problems.

4.2 Balancing the axes

Obviously, the process described in Section 4.1 can yield x and y coordinates at very different scales. Usually, we require that $||x|| = ||y||$, but this is difficult to achieve in a distributed manner. An easier alternative that is more suitable for a distributed computation is a balanced aspect ratio, i.e.:

$$\max_i x_i - \min_i x_i = \max_i y_i - \min_i y_i$$

Since the computation of the y-coordinates already achieved $\max_i y_i - \min_i y_i = 1$, it remains to ensure that the x coordinates have the same property. We achieve this by performing:

$$x_i \leftarrow \frac{x_i}{\max_j x_j - \min_j x_j}\,, \quad i = 1, \ldots, n \tag{5}$$

Note that we only scale the x-coordinates and do not translate them, because translation involves the v_1 component that is not part of the x-coordinates.

In fact, it might be beneficial to scale x by (5) a few times during the power iteration (3). This can prevent potential numerical problems when the coordinates are extremely large (overflow) or small (underflow).

5 Optimizing the Localized Stress Energy

At this point we have reasonable initial locations for both the x- and y-coordinates, and are ready to apply a more accurate 2D optimization process for minimizing the localized stress energy (1). A candidate could be simple gradient descent, which is easily distributed, as in [12]. Each sensor would update its x-coordinates as follows:

$$x_i(t+1) = x_i(t) + \delta \sum_{j:\langle i,j\rangle \in E} \frac{(x_j(t) - x_i(t))}{d_{ij}(t)} (d_{ij}(t) - l_{ij})\,, \tag{6}$$

where $d_{ij}(t) = \sqrt{(x_i(t) - x_j(t))^2 + (y_i(t) - y_j(t))^2}$. The y-coordinates are handled similarly. This involves a scalar quantity δ whose optimal value is difficult to estimate. Usually a conservative value is used, but this slows down the convergence significantly.

A more severe problem of this gradient descent approach is its sensitivity to the scale of the initial layout. Obviously the minimum of $E(x)$ is scale-invariant, since $E(cx) = E(x)$ for $c \neq 0$. However, the minimum of Stress(x)

is certainly not scale-invariant as we are given concrete target edge lengths. Therefore, before applying gradient descent we have to scale the minimum of $E(x)$ appropriately.

Fortunately, we can avoid the scale problem by using a different approach called *majorization*. Besides being insensitive to the original scale, it is usually more robust and avoids having to fix a δ for the step size. For a detailed description of this technique, we refer the interested reader to multidimensional scaling textbooks, e.g., [1]. Here we provide just a brief description.

Using the Cauchy-Schwartz inequality we can bound the localized 2D stress of a layout (x, y) by another expression of (x, y) and (a, b), as follows:

$$\text{Stress}(x, y) \leqslant x^T Lx + y^T Ly + x^T L^{(a,b)} a + y^T L^{(a,b)} b + c\,, \quad x, y, a, b \in \mathbb{R}^n\,, \tag{7}$$

with equality when $x = a$ and $y = b$. Here, c is a constant independent of x, y, a, b. L is the graph's unweighted $n \times n$ Laplacian matrix (also independent of x, y, a, b) defined as:

$$L_{i,j} = \begin{cases} -1 & \langle i,j\rangle \in E \\ 0 & \langle i,j\rangle \notin E \\ -\sum_{j \neq i} L_{i,j} & i = j \end{cases} \qquad i, j = 1, \dots, n$$

The weighted Laplacian $n \times n$ matrix $L^{a,b}$ is defined as:

$$L^{(a,b)}_{i,j} = \begin{cases} -l_{ij} \cdot \text{inv}\left(\sqrt{(a_i - a_j)^2 + (b_i - b_j)^2}\right) & \langle i,j\rangle \in E \\ 0 & \langle i,j\rangle \notin E \\ -\sum_{j \neq i} L^{(a,b)}_{i,j} & i = j \end{cases} \qquad i, j = 1, \dots, n$$

where

$$\text{inv}(x) = \begin{cases} 1/x & x \neq 0 \\ 0 & x = 0 \end{cases}$$

Note the special treatment that the inv function gives to the zero value. Given a layout a, b, we can find another layout (x, y) which minimizes the r.h.s. $x^T Lx + y^T Ly + x^T L^{a,b} a + y^T L^{a,b} b + c$ by solving the linear equations:

$$\begin{aligned} Lx &= L^{(a,b)} a \\ Ly &= L^{(a,b)} b \end{aligned}$$

Using inequality (7) we are guaranteed that the stress of the layout has not increased when going from (a, b) to (x, y), i.e., $\text{Stress}(x, y) \leqslant \text{Stress}(a, b)$. This induces an iterative process for minimizing the localized stress. At each iteration, we compute a new layout $(x(t+1), y(t+1))$ by solving the following linear system:

$$\begin{aligned} L \cdot x(t+1) &= L^{(x(t),y(t))} \cdot x(t) \\ L \cdot y(t+1) &= L^{(x(t),y(t))} \cdot y(t) \end{aligned} \tag{8}$$

Without loss of generality, we can fix the location of one of the sensors (utilizing the translation degree of freedom of the localized stress) and obtain a strictly diagonally dominant matrix. Therefore, we can safely use Jacobi iteration [4] for solving (8), which is easily performed in a distributed manner as follows.

Assume we are given a layout $(x(t), y(t))$ and want to compute a better layout $(x(t+1), y(t+1))$ by a single iteration of (8). Then we iteratively perform for each $i = 1, \dots, n$:

$$\begin{aligned} x_i &\leftarrow \frac{1}{deg_i} \sum_{j:\langle i,j\rangle \in E} (x_j + l_{ij}(x_i(t) - x_j(t)) \,\mathrm{inv}(d_{ij}(t))) \\ y_i &\leftarrow \frac{1}{deg_i} \sum_{j:\langle i,j\rangle \in E} (y_j + l_{ij}(y_i(t) - y_j(t)) \,\mathrm{inv}(d_{ij}(t))) \end{aligned} \tag{9}$$

Note that $x(t)$, $y(t)$ and $d_{ij}(t)$ are *constants* in this process which converges to $(x(t+1), y(t+1))$. Interestingly, when obtaining $(x(t+1), y(t+1))$ only the angles between sensors in $(x(t), y(t))$ are used. Therefore, this process is independent of the scale of the current layout.

It is possible to simplify the 2D majorization process somewhat. When the iterative process (9) converges the layout scale issue is resolved. Hence, instead of continuing with another application of (8) to obtain a newer layout, it is possible to resort to a faster local process (which, in contrast, *is* scale-dependent). In this process each sensor uses a local version of the energy where all other sensors are fixed. By the same majorization argument the localized stress decreases when applying the following iterative process:

$$\begin{aligned} x_i &\leftarrow \frac{1}{deg_i} \sum_{j:\langle i,j\rangle \in E} (x_j + l_{ij}(x_i - x_j)\mathrm{inv}(d_{ij})) \\ y_i &\leftarrow \frac{1}{deg_i} \sum_{j:\langle i,j\rangle \in E} (y_j + l_{ij}(y_i - y_j)\mathrm{inv}(d_{ij})) \end{aligned} \tag{10}$$

Here, as usual $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. This process is similar to (9), except that x_i, x_j and d_{ij} are no longer constants. We have used this in our implementation, and it seems to accelerate the convergence. Note that this is quite close to the gradient descent (6) when using $\delta = 1/\mathrm{deg}_i$, a different stepsize per sensor.

6 Experimental Results

We have implemented our algorithm and the AFL algorithm [12], and compared their performance on a variety of inputs. In the first experiment, we construct a family of graphs containing 1000 sensors distributed uniformly in a 10×10 square. Each two sensors are connected if they are in range R, where we used $R = 0.5, 0.6, 0.7, 0.8, 0.9, 1$. If the graph is disconnected, the largest connected component was taken. We measure the sensitivity of the algorithms to noise controlled by the fractional range measurement error parameter σ. The distances fed as input to the algorithms are the true distances

l_{ij}, to which uniformly distributed random noise in the range $[-\sigma l_{ij}, \ +\sigma l_{ij}]$ is added; $\sigma = 0, 0.05, 0.1, 0.25, 0.5$. Consequently, each graph in this family is characterized by the values of R and σ. For each pair (R, σ) we generated 250 corresponding random graphs. Some properties of these graphs are displayed in Table 1.

R	size	avg degree	max degree	min degree
0.5	993	7	17.2	1
0.6	999.5	10	22	1.4
0.7	1000	14	27.5	2.4
0.8	1000	18.2	33.5	3.7
0.9	1000	23	40.3	5.2
1.0	1000	28.2	47.5	6.9

Table 1: Average (over 250 experiments) properties of largest connected component of graphs obtained by distributing 1000 sensors in a 10×10 square, using different values of R.

It seems that the key to successful results is a good initial layout from which the stress minimization routine can start. To compare the performance of our algorithm to that of the AFL algorithm and a more naive method, we ran three different initialization methods on each input followed by the same stress minimization algorithm: **(1)** Stress majorization with random initialization (RND). **(2)** Stress majorization with AFL initialization (AFL). **(3)** Stress majorization with eigen-projection initialization (EIGEN). For each method the quality of the final solution is measured by its Average Relative Deviation (ARD), which measures the accuracy of all resulting pairwise distances:

$$ARD = \frac{2}{n(n-1)} \sum_{i<j} \frac{|d_{ij} - l_{ij}|}{\min(l_{ij}, d_{ij})}$$

Note that here we sum over *all* distances between sensors, not just the short range distances, as reflected by the edges of the graph. The results are summarized in Table 2, where each cell shows the average ARD of RND/AFL/EIGEN for 250 different graphs characterized by the same (R, σ) pair. For all graphs, EIGEN and AFL outperformed RND by a significant margin. Also, consistently, EIGEN outperformed AFL by a small margin. As expected, the algorithm performance is improved as the graphs become denser revealing more information about the underlying geometry. Note that the sparser graphs contain nodes of degree smaller than 3, which are inherently non-rigid thereby preventing accurate recovery. We can also see that optimization is quite robust in the presence of noise and performance deteriorates only moderately as σ grows. In Figure 2 we show typical results of EIGEN, before and after stress minimization. For comparison, we also provide the original layout and the AFL initialization for the same graph.

In another experiment, we worked with 350 sensors distributed uniformly on a ring, with external radius 5 and internal radius 4. Again, the graphs are

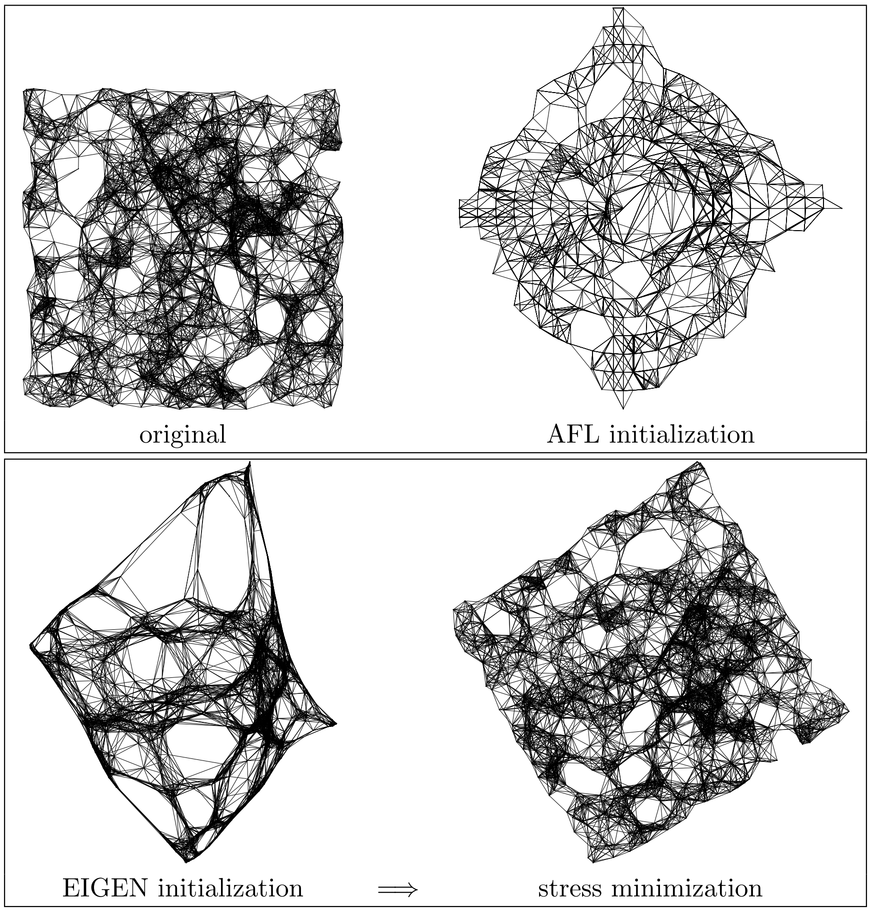

Figure 2: Reconstructing a 1000-sensor proximity graph using EIGEN; here $R = 0.8, \sigma = 0$. Original layout and alternative AFL initialization are also shown.

	$\sigma = \mathbf{0}$			$\sigma = \mathbf{0.05}$			$\sigma = \mathbf{0.1}$		
	RND	AFL	EIGEN	RND	AFL	EIGEN	RND	AFL	EIGEN
$\mathbf{R = 0.5}$	12.6	0.099	0.079	12.6	0.10	0.079	12.4	0.10	0.092
$\mathbf{R = 0.6}$	11.2	0.026	0.0093	11.0	0.028	0.013	10.8	0.031	0.019
$\mathbf{R = 0.7}$	9.70	0.013	0.0031	9.79	0.015	0.0048	9.77	0.017	0.0076
$\mathbf{R = 0.8}$	8.51	0.0086	0.0016	8.52	0.0097	0.0033	8.42	0.012	0.0059
$\mathbf{R = 0.9}$	7.29	0.0064	0.0011	7.37	0.0082	0.0028	7.28	0.011	0.0051
$\mathbf{R = 1.0}$	6.31	0.0054	0.0008	6.40	0.0068	0.0025	6.51	0.0079	0.0047

	$\sigma = \mathbf{0.25}$			$\sigma = \mathbf{0.5}$		
	RND	AFL	EIGEN	RND	AFL	EIGEN
$\mathbf{R = 0.5}$	12.3	0.12	0.091	11.6	0.26	0.22
$\mathbf{R = 0.6}$	11.0	0.046	0.031	10.4	0.12	0.10
$\mathbf{R = 0.7}$	9.71	0.026	0.018	9.53	0.060	0.050
$\mathbf{R = 0.8}$	8.58	0.020	0.014	8.49	0.041	0.034
$\mathbf{R = 0.9}$	7.37	0.017	0.013	7.50	0.033	0.028
$\mathbf{R = 1.0}$	6.33	0.016	0.012	6.52	0.030	0.026

Table 2: Average relative deviation (ARD) of square-based proximity graphs with varying (R, σ) generated by RND / AFL / EIGEN. Each result is averaged over 250 graphs.

characterized by the range and noise parameters (R, σ), and for each such a pair we generated 250 corresponding random graphs. The properties of these graphs are shown in Table 3. Here we worked with a different range of R, producing average degrees similar to those of the previous experiment. Note that we avoided working with $R \leqslant 0.6$ as for these values the largest connected component broke the ring topology with high probability, making recovery impossible. In Figure 3 we show a typical result of EIGEN, before and after the stress majorization. We ran RND, AFL and EIGEN on these graphs, the results summarized in Table 4. The topology of the ring is different than that of the square, and resulted in a lower quality results. However, all the observations from the square-based experiment still hold here. Note that in a ring there is no natural central node. Therefore, the AFL initialization that identifies one node as the center is less appropriate here. A surprising finding is that the performance of AFL seems to deteriorate when increasing R from 1.1 to 1.2, instead of improving, as would be expected. We observed this also with other types of graphs we experimented with. We believe that this is due to the fact that the first phase of AFL models the network as an unweighted graph. Thus, as the variance of the true edge lengths becomes larger, this model is less accurate.

As far as computational complexity is concerned, since we deal with a distributed environment, the most interesting quantity is the communication complexity (i.e. the number of messages passed). In each iteration, each sensor communicates only with its neighbors. The number of neighbors is of course limited by the communication range and is fairly independent of the overall number of sensors. In fact, a larger number of neighbors will definitely accelerate

R	size	avg degree	max degree	min degree
0.7	349	10.4	22	2
0.8	349.6	13.2	26.3	3.2
0.9	350	16.2	30.2	4.8
1.0	350	19.3	33.8	6.8
1.1	350	22.5	37.3	9.0
1.2	350	25.8	40.5	11.5

Table 3: Properties of largest connected component of graphs obtained by distributing 350 sensors on a RING with radii 4 and 5, using different values of R.

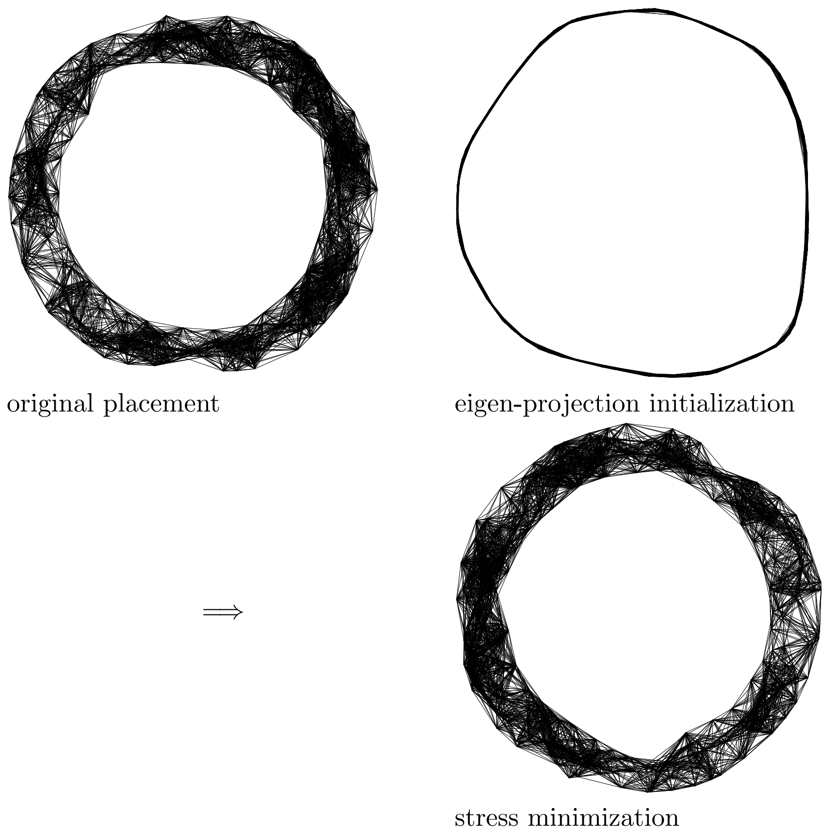

Figure 3: Reconstructing a 350-sensor ring-based proximity graph by localized stress majorization preceded by eigen-projection. Here $R = 0.7, \sigma = 0$.

	$\sigma = \mathbf{0}$			$\sigma = \mathbf{0.05}$			$\sigma = \mathbf{0.1}$			$\sigma = \mathbf{0.25}$			$\sigma = \mathbf{0.5}$		
	RND	AFL	EIGEN	RND	AFL	EIGEN	RND	AFL	EIGEN	RND	AFL	EIGEN	RND	AFL	EIGEN
$\mathbf{R = 0.7}$	4.96	0.34	0.14	5.16	0.26	0.13	4.94	0.26	0.13	4.66	0.33	0.15	4.88	0.39	0.21
$\mathbf{R = 0.8}$	7.69	0.19	0.091	7.53	0.23	0.091	7.54	0.020	0.090	7.81	0.19	0.10	7.41	0.29	0.16
$\mathbf{R = 0.9}$	7.52	0.14	0.064	7.35	0.16	0.065	7.56	0.14	0.065	7.27	0.18	0.080	7.14	0.22	0.13
$\mathbf{R = 1.0}$	6.61	0.10	0.041	6.62	0.11	0.045	6.41	0.11	0.046	6.54	0.13	0.055	6.40	0.15	0.091
$\mathbf{R = 1.1}$	5.77	0.10	0.029	5.72	0.098	0.031	5.69	0.10	0.035	5.62	0.12	0.044	5.69	0.14	0.070
$\mathbf{R = 1.2}$	4.97	0.11	0.021	4.98	0.11	0.021	4.88	0.11	0.026	5.08	0.13	0.032	4.97	0.16	0.058

Table 4: Average relative deviation (ARD) of disk-based proximity graphs with varying (R, σ) constructed using RND / AFL / EIGEN. Each result is averaged over 250 graphs.

convergence (by enabling faster information transfer through the network) and also improve the quality of the results (as clearly shown in our experimental results). However, the key factor in determining the communication complexity is the number of iterations. The iterative nature of the algorithm suggests a trade-off between the number of iterations (and hence, communication complexity) and the accuracy of the resulting solution. Typical performance is demonstrated in Fig. 4. This figure shows the decrease in the ARD (the Average Relative Error) as a function of the number of iterations. The results were averaged over 100 runs on a 1000-sensor network. The sensors were distributed on a 10×10 square with communication range of 0.7, with and without transmission noise. As evident in the figure, the quality improves rapidly at the beginning of the process, dropping the ARD to 5% after about 120 iterations. This then slows down, reaching an ARD of 1% only after about 600 additional iterations. Soon after this the process converges as the ARD stabilizes. A somewhat surprising result was that the algorithm converged faster for noisier networks. After experimenting with quite a few networks, we speculate that about n iterations are required for convergence.

7 Extensions

There are a number of ways our basic algorithm can be extended:

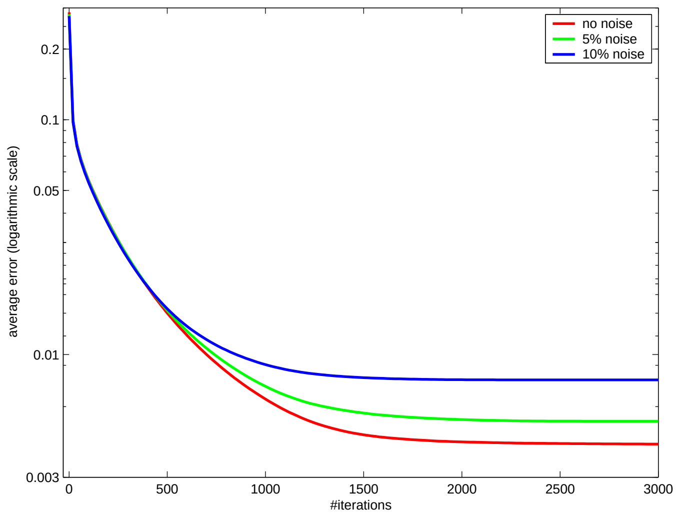

Figure 4: Layout accuracy (represented by ARD) vs. number of iterations. Results were averaged on 100 runs on a network with 1000 sensors distributed on a 10×10 square and $R = 0.7$.

7.1 Hybrid method

It is possible to couple the stress optimization together with the eigen-projection in a single process. In such a process, we continuously perform a few local majorization iterations, where each sensor is relocated according to process (10), followed by a single barycentric placement, where each sensor is placed at the 2D centroid of its neighbors.

The introduction of a few barycentric iterations during the majorization goes a long way towards preventing folding and convergence to local minima. Our recommendation is to start the process as a pure eigen-projection, then to continue with this hybrid method and to finish with a pure localized stress majorization.

7.2 Termination

A central issue in distributed systems is reaching agreement. In our application, this is relevant in reaching agreement when to terminate any particular iterative stage of the algorithm. It is easy for each sensor to check whether it has converged, so each sensor can terminate that way. However, transition to another phase of the algorithm that involves a different type of computation requires some sort of collective agreement on convergence. Currently, we just limit the maximal number of iterations (as a function of the number of sensors). In some settings, it would be reasonable to allow the sensors to perform an eternal process of stress minimization. When asked for their coordinates they should deliver the freshest result. This is especially suitable when sensor locations are dynamic, so they should be continuously updating their estimated locations.

7.3 Numerical stability

When computing the y-coordinates, the power iteration process (3) may occasionally lose its D-orthogonality to $x = v_2$, due to limited numerical precision. This can lead to high correlation between the x- and y-coordinates. Currently, we are using double precision arithmetic and our application will suffer from this problem only when the graphs are quite dense (average degree $\geqslant 30$). For such dense graphs the performance of the hybrid method is excellent and compensates for this deficiency of the power iteration. We believe that if the algorithm is implemented with extended numerical precision, one should not encounter such problems.

7.4 Working in three dimensions

When applying our eigen-projection method to 3D layouts, the z vector should be v_4 - the fourth eigenvector of $I + D^{-1}W$. This means we must compute a vector z which is D-orthogonal to both x and y already computed. To achieve this, we partition the sensors into disjoint sets, each of cardinality 3 at least. Possibly, some sensors are left as singletons. In each set there should be a sensor that is adjacent to all other sensors of its set; call it the "center". This is a randomized max "star-matching" that can be performed in a distributed manner, using a few sweeps. Consider a set $\{i, j, k\}$, where i is the center. Now, i should know x_i,y_i,D_{ii},x_j,y_j,D_{jj},x_k,y_k,D_{kk}, which is possible since i can communicate with both j and k. Using this information, sensor i computes a vector (z_i, z_j, z_k) which is "D-orthogonal" to (x_i, x_j, x_k), and (y_i, y_j, y_k). By this we mean that $D_{ii}z_ix_i + D_{jj}z_jx_j + D_{kk}z_kx_k = 0$, and also $D_{ii}z_iy_i + D_{jj}z_jy_j + D_{kk}z_ky_k = 0$. This is done simply by a standard Gram-Schmidt process. Similarly, each center assigns the sensors of its set their z-coordinates. Also, each sensor i that was not assigned to a set takes $z_i = 0$. This way we get an initial z which is D-orthogonal to x and y. Before computing this z, we should use the same technique to compute an exact $y = v_3$, which is D-orthogonal to both 1_n and x.

8 Conclusion

We have presented an algorithm to generate fold-free sensor network layouts based on short-range inter-sensor distances. This algorithm is fully distributed (decentralized), and relies on no explicit communication other than that between immediate neighbors. The fully distributed nature of the algorithm is crucial for a practical implementation which avoids excessive communication. To the best of our knowledge, this is the first fully distributed algorithm for graph drawing. Beyond this important feature, our experiments indicate that our algorithm seems to be superior to the state-of-the-art in the sensor network literature.

Future work includes extension of our methods to dynamic sensor networks and sensor networks where more geometric information (such as angles) is available.

References

[1] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer-Verlag, 1997.

[2] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[3] T. M. G. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21:1129–1164, 1991.

[4] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[5] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219–229, 1970.

[6] B. Hendrickson. The molecule problem: Exploiting structure in global optimization. *SIAM Journal on Optimization*, 5:835–857, 1985.

[7] B. Hendrickson. Conditions for unique graph realizations. *SIAM Journal on Computing*, 21:6–84, 1992.

[8] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[9] Y. Koren. Drawing graphs by eigenvectors: Theory and practice. *Computers and Mathematics with Applications*, 49:1867–1888, 2005.

[10] J. W. M. Mauve and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15:30–39, 2001.

[11] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *SenSys 2004: Proc. 2nd Inter. Conf. on Embedded Networked Sensor Systems*, pages 50–61, 2004.

[12] N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. In *SenSys 2003: Proc. 1st Inter. Conf. on Embedded Networked Sensor Systems*, pages 340–341, 2003.

[13] Y. Shang and W. Ruml. Improved mds-based localization. In *Infocom '04: Proc. 23rd Conf. of the IEEE Communicatons Society*, pages 340–341, 2004.

[14] M. Tubaishat and S. Madria. Sensor networks : An overview. *IEEE Potentials*, 22:20–23, 2003.

[15] L. Xiao and S. Boyd. ast linear iterations for distributed averaging. *Systems and Control Letters*, 53:65–78, 2004.

[16] Y. Yemini. Some theoretical aspects of location-location problems. In *FOCS '79: Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–8, 1979.