

PATCHWORK: Efficient Localization for Sensor Networks by Distributed Global Optimization

Technical Report
August 2005

Yehuda Koren
AT&T Research Labs

Craig Gotsman
Harvard University

Mirela Ben-Chen
Technion – Israel Inst. of Technology

Abstract

We describe a new algorithm for sensor network localization based on short inter-sensor distances and fully decentralized computation. The algorithm computes local coordinates of small “patches” of sensors and glues them together using a distributed global optimization process. Thanks to the global optimization, our method is more robust in the presence of noisy measurements than existing incremental methods. Most notably, and unlike some other global optimization techniques, our method cannot be fooled by local minima that might lead to foldovers in the network layout. The method is anchor-free, so advance knowledge about locations of some sensors is not required. Nonetheless, we provide a way to take advantage of known locations of any number of anchors. An experimental study shows the effectiveness of the new algorithm in noisy environments where the sensors are distributed over regions with a variety of geometric shapes.

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). 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 thru 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 [7] for a survey of these routing techniques.

Beyond routing, localization is important for the sensor network application itself. For most applications, sensed data without spatial and temporal coordinates is of very limited use. Sensor nodes have to be aware of their location to be able to specify where a certain event has occurred. For military, police, or other radio networks, knowing the precise location of each person with a radio can be critical. In warehouses, object location and tracking applications are possible with large-scale ad-hoc networks of wireless tags. Environment monitoring, vehicle tracking and mapping are just a few other application domains where knowledge of sensor location is critical.

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 dis-

tance 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 compute its location in some global coordinate system. This paper solves the following *sensor localization* problem:

*Given a set of n 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.*

The localization problem has a unique solution when 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. When all $n(n-1)/2$ inter-sensor distances are known, the solution is indeed unique (up to a rigid transformation), and is traditionally found efficiently using the classical multidimensional scaling (MDS) technique [1]. When only a subset of the distances is known, more sophisticated techniques must be used. In this case, even when the solution is unique, its recovery is computationally difficult as the problem becomes NP-hard [17]. Obviously, as long as the computation is based only on inter-sensor distances, 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 a small number of *anchor* sensors.

In real-world sensor networks, noise is inevitable. This manifests in the inter-sensor distance 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 the sensor localization problem should be fully distributed (decentralized). This means that each sensor should compute based on information available *only* at that sensor and its immediate neighbors. The class of neighbors is typically characterized by the disk graph model: Any sensor within distance R is reachable. Of course, information from one sensor may eventually propagate thru the network to any other sensor in a number of *hops*, but this should not be done explicitly.

2. Related Work

The sensor localization problem has received considerable attention in the sensor network community. A recent work of Priyantha *et al.* [10] classifies localization algorithms into *anchor-based* vs. *anchor-free* algorithms and *incremental* vs. *concurrent* algorithms. Anchor-based algorithms rely on the fact that a subset 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 are 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 geometric 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 error accumulation. Priyantha *et al.* review and classify a number of published algorithms. All of these, however, are not fully distributed.

Priyantha *et al.* [10] proposed a concurrent, anchor-free algorithm called Anchor-Free Localization (AFL). This algorithm operates in two stages. In the first stage a heuristic is applied to try generate a well-spread fold-free layout which "looks similar" to the desired layout. The second stage applies a more accurate "stress-minimization" optimization procedure that uses all given distances, converging to the final result. The heuristic used in the first stage involves the election of five reference sensors strategically positioned within the network. Initial coordinates are assigned to these sensors, and then to all others, based on those of the reference sensors and the network connectivity. The second stage of the AFL algorithm attempts to minimize the partial stress energy of the layout using a gradient descent technique. This involves updating the coordinates of each sensor iteratively by moving them an infinitesimal distance in the direction of the spring force operating on the sensor.

The localization algorithm of Gotsman and Koren [3] is also concurrent, anchor-free and involves two stages with objectives similar to AFL. The first stage aims to generate a fold-free layout. This is done based on a distributed weighted 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, rather on a more effective *majorization* technique. While both Priyantha *et al.* [10] and Gotsman and Koren [3] demonstrate quality results for the anchor-free case, they do not suggest any means for integrating knowledge about the location of anchors in case they exist. Here we refer mostly to the first phase of both these approaches. This is unfortunate, as it seems that in many practical applications one would have *some* knowledge about the location of a few sensors (e.g., three sensor locations are anyway needed to eliminate rigid transformations). This is valuable information that can greatly improve the accuracy of the result, thus any localization method should have the flexibility to incorporate it. Another drawback of these two approaches is the arbitrariness of their first stages. No significant relation holds between the true layout and the initial layout generated at this stage. As we will show in Section 4, this arbitrariness significantly weakens the results of these concurrent methods on certain geometries, compared to incremental methods. The latter are more flexible in adapting to different geometries.

Moore *et al.* [8] 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 an improvement by running stress minimization on that patch. 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 the rigid component of the network. The algorithm will not produce anything useful for the other sensors.

A similar “patching” algorithm was described by Shang and Ruml [12]. 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.

In this paper we present a concurrent sensor localization algorithm. It is reminiscent of the two incremental methods of Moore *et al.* [8] and Shang and Ruml [12] in that it first localizes small patches of sensors. However, instead of then gluing them together incrementally, and accumulating error in the process, it applies a distributed global process which fits the patches together in an optimal manner, trying to preserve the local relationships between the patches. In a nutshell, it involves a distributed least-squares solution to a set of linear equations relating the locations of a sensor to those of its neighbors. The equations are derived from the relationships between the local coordinate systems of the overlapping patches and the global coordinate system. We have been inspired by recent methods which implicitly represent a curved low-dimensional manifold within a higher dimensional space. Commonly known as *dimension reduction*, these techniques are used in learning theory to compactly represent multi-dimensional data. The classical way of doing this, based on inter-point distances, is using the same MDS technique mentioned above. That technique, however, requires global data, relating every sensor to *all* others. In contrast, our methods are similar to the more recent techniques of Local Linear Embedding [11] and Charting a Manifold [2] which build a “global” manifold implicitly by defining a set of local interactions between small pieces of it.

3. The PATCHWORK Algorithm

Our algorithm first localizes small subsets of the sensor network, forming “patches”, and then “stitches” them together, hence the name PATCHWORK. The patches are formed in parallel by subsets of the sensors, with some overlap between neighboring patches. In this sense our algorithm is similar to those of Moore *et al.* [8] and Shang and Ruml [12]. However, while those two algorithms stitch the patches together incrementally, we stitch them together by solving a global optimization problem in a distributed manner. As we will see, this is the key to avoid entirely any error accumulation, and, moreover, significantly reduce the localization error which the noise might otherwise introduce.

Our algorithm bears some similarity to AFL [10] and the algorithm of Gotsman and Koren [3], as we use an iterative stress-minimization algorithm at a post-processing stage. However, our first stage is quite powerful in itself, able to recover the layout accurately. It can also incorporate directly the coordinates of anchor sensors when these exist. We begin by describing the so-called *stress* function, which plays an important role in many localization algorithms.

3.1 The Stress function

Assume the sensor network is modeled by a graph $G(V=\{1,\dots,n\},E)$, such that for every edge $\langle i,j\rangle\in E$ we know the (noisy) distance l_{ij} between sensors i and j . We assume, without loss of generality, that the distances are always symmetric, so $l_{ij}=l_{ji}$. This can be achieved by requiring neighboring sensors to reach an agreement about their distance (e.g. by averaging l_{ij} and l_{ji}). We want to generate a d -dimensional layout $p_1,\dots,p_n\in R^d$ that *realizes* all known distances. This layout can be formally characterized as the minimum of the *stress function*:

$$\text{Stress}(p_1,\dots,p_n) = \sum_{\langle i,j\rangle\in E} (\|p_i - p_j\| - l_{ij})^2$$

The stress function and its variants are widely used in *multidimensional scaling* for recovering the geometry of points from their *complete set* of pairwise distances; see, e.g. [1]. This is a non-convex function, hence, in general, we cannot find its global minimum efficiently. In our scenario, we know only distances between nearby sensors. Because of this partial information, the main phenomenon observed in the (suboptimal) solutions is that of *foldovers*, where entire pieces of the network fold over on top of others, while still realizing well the known distances.

To overcome the tendency of stress minimization to generate folded layouts, it must be augmented by either initializing the process with some adequate, fold-free, layout p_1,\dots,p_n , or introducing some estimates of long-range distances between distant sensors. These will impose the correct global shape on the layout.

The first remedy, smart initialization, was already suggested by Priyantha *et al.* [10] and Gotsman and Koren [3]. The second remedy, incorporating long-range pairwise distances is impractical since their presence will impede a distributed process. 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, even for very simple problems [16].

Luckily, all these concerns are valid only when using stress minimization for computing the complete layout of *all* the sensors. When working with just a small *patch*, the situation becomes much easier.

3.1 Localizing a patch

Consider the neighborhood of a sensor s denoted by $N(s) = \{i \mid \langle i,s\rangle\in E\} \cup \{s\}$. The distances between s and the sensors in $N(s)$ are known, and typically some of the distances between the sensors within $N(s)$ as well. We would like to localize the sensors in $N(s)$ in a common coordinate system, which will

approximately realize the given edge lengths in that neighborhood. These sensors will form a *patch* in the *quilt* that we are *sewing*.

Localizing a patch is significantly easier than localizing a complete network, on two counts. First, when localizing a patch, we are not constrained to a distributed computation that imposes severe restrictions on inter-sensor communications. Instead, all sensors in $N(s)$ may report to s all their inter-distances, and the computation is then done centrally by s . Second, while in the complete network the known distances are considered local and short-range, for a patch the given distances are not local at all (every two sensors are reachable by at most two hops) and we possess a relatively dense set of pairwise distances. As observed in the previous section, having such “long-range” distances makes stress minimization more tractable.

We propose a three-stage method for localizing a patch. First, we estimate the missing distances within the patch. Second, we compute the coordinates using classical MDS, which is based on the complete set of pairwise distances. Finally, we refine the result using stress minimization based only on the original (more reliable) distances. We emphasize again that all computations are carried out entirely by the sensor s . We now elaborate on these three stages.

I - Estimating missing distances. In this stage we estimate l_{ij} for every $\langle i, j \rangle \notin E$. From the triangle inequality we obtain the upper bound \widehat{l}_{ij} :

$$\widehat{l}_{ij} = \min_{k: \langle i, k \rangle \in E, \langle j, k \rangle \in E} \{l_{ik} + l_{jk}\}$$

Moreover, if the graph is a *disk graph*, we have the following lower bound \widetilde{l}_{ij} :

$$\widetilde{l}_{ij} = \max \left\{ \max_k \{l_{ik}\}, \max_k \{l_{jk}\} \right\}$$

So our estimate for l_{ij} would be $(\widehat{l}_{ij} + \widetilde{l}_{ij})/2$.

II - Classical MDS. After defining all pairwise distances we obtain coordinates using the classical MDS method. This method computes the coordinates by factoring the inner-product matrix, which can be derived from the pairwise distances; see, e.g., [1] for a detailed description. The advantage of working with classical MDS is its ability to generate the optimal solution regardless of the way it is initialized.

III - Stress minimization. We now refine the layout by using only the known original distances, where stress minimization becomes handy. We minimize the stress energy using a “localized” version of the majorization technique, as described by Gotsman and Koren [3]. In essence, it involves iteratively applying the following update rule, for each $i \in N(s)$:

$$p_i \leftarrow \frac{1}{\left| \{j \in N(s) \mid \langle i, j \rangle \in E\} \right|} \sum_{j \in N(s), \langle i, j \rangle \in E} \left[p_j + l_{ij} (p_i - p_j) \operatorname{inv}(\|p_i - p_j\|) \right]$$

where

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

We have observed that performing stress minimization on a small patch is very reliable and far less sensitive to local minima compared with stress minimization at the global level.

3.2 Combining using affine transformations

The n sensors are partitioned into n overlapping patches induced by the neighborhood of each of the individual sensors. We localize each patch individually as described in the previous section. Now we would like to “stitch” all patches together into one coherent layout.

In the ideal case all patches were localized optimally. Then, we can merge them perfectly by using only *rigid transformations* of the patches, i.e., translations, reflections and rotations. These will ensure

that every sensor has the same coordinates in all patches containing it. However, in practice we cannot expect such an ideal case, because of the inaccurate localization of some patches, mostly due to noisy distance measurements. Hence, we seek to transform each patch so that inconsistencies are minimized in a least squares sense. Moreover, we work with the wider family of *affine transformations* that include all rigid transformations with additional operations like scaling and shear. This allows compensating for errors introduced by noise.

We begin with a formal definition of an affine transformation.

Given n points in R^d , their locations are an $n \times (d+1)$ matrix X . The i -th row of X represents the i -th point, where the first d entries are its d coordinates and the last entry is always 1. An affine transformation of the points is a real $(d+1) \times (d+1)$ matrix A , and the transformed coordinates are the $n \times (d+1)$ matrix XA . Note that we use homogeneous coordinates to facilitate translations.

Consider a patch centered at s . Once this patch has been localized, we arrange the *local* coordinates of its sensors in the $n_s \times (d+1)$ matrix $X(s)$ where $n_s = |N(s)|$. Thus, each row of $X(s)$ corresponds to one sensor of $N(s)$ and the last column of $X(s)$ contains only 1's. We also denote by $P(s)$ the corresponding $n_s \times (d+1)$ matrix containing the locations of the same sensors in the *global* coordinate system. Unlike $X(s)$, the matrix $P(s)$ is unknown.

We want to relate the local coordinate system with the global one using an affine transformation. This transformation is described by some $(d+1) \times (d+1)$ matrix $A(s)$ satisfying $X(s)A(s) \approx P(s)$. The best least-squares solution is given by:

$$A(s) = X(s)^+ P(s),$$

where $X(s)^+$ is the pseudo-inverse [9] of $X(s)$. Since $X(s)$ is known, the entries of $A(s)$ are linear expressions of $P(s)$. Now we desire that applying $A(s)$ to $X(s)$ will map it as close as possible to $P(s)$, namely minimize $\|P(s) - X(s)A(s)\|^2$, which is identical to:

$$\min_{P(s)} \left\| P(s) - \left(X(s) X(s)^+ \right) P(s) \right\|^2$$

or, equivalently, find the least-squares solution $P(s)$ to:

$$\left(I - X(s) X(s)^+ \right) P(s) = 0. \quad (1)$$

Of course, a trivial solution is when $P(s) = 0$, which we will later see how to avoid.

We now rephrase these equations using the indices of the sensors in the global system. Recall that the locations of the points in the global coordinate system were defined as $p_1, \dots, p_n \in R^d$. For each $i \in N(s)$, denote by $[i]_s$ the index of the row in $X(s)$ and $P(s)$ corresponding to i . Hence, the first d entries in the $[i]_s$ -th row of $P(s)$ are identical to p_i . Consequently, the patch centered at s contributes these n_s equations:

$$p_i - \sum_{j \in N(s)} \left(X(s) X(s)^+ \right)_{[i]_s, [j]_s} p_j = 0, \quad i \in N(s) \quad (2)$$

Note that we omitted the constant last column of $P(s)$ from these equations. As long as $n_s > d+1$, we can safely assume that it is redundant, since the sum of every row of $X(s)X(s)^+$ is always 1. This can be proven by observing that the last column of $X(s)$ contains only 1's.

Alternatively, it would be legitimate to consider at this point only the coordinates of s . This allows us to reduce the number of equations, so the patch centered at s contributes only a *single* equation:

$$p_s - \sum_{j \in N(s)} \left(X(s) X(s)^+ \right)_{[s]_s, [j]_s} p_j = 0$$

or, in other words

$$p_s = \sum_{j \in N(s)} w_{[j]_s} p_j \quad s = 1, \dots, n \quad (3)$$

for some weights $w_{[j]_s}$ summing to 1.

To summarize, in order to relate all patches together using implicit affine transformations relative to a global layout, we generate a single vector equation for each sensor i (or, alternatively, n_i vector equations). Each such equation expresses its coordinates in the global layout as an affine combination (i.e. linear combination with weights having unit sum) of the coordinates of its neighbors. So overall we obtain a system of n (or $\sum n_i$) linear homogeneous vector equations in n vector unknowns. Each equation is constructed internally within the sensor, and no collective distributed effort is required. Later, we will show how to find the (least squares) solution to this global set of equations. However, our system is still under-determined. Due to all the rows summing to unity, there is at least one degree of freedom corresponding to the translation of the origin. We can eliminate this by specifying the position of one sensor. However, this is not sufficient, as the solution can still be degenerate - all sensors will collapse to the position of this anchor (the zero solution to (1), as mentioned above). As we shall see, in order to avoid another degenerate situation, where all sensors are located within a low-dimensional subspace (e.g., along a line), we need to specify the position of at least d additional sensors. We call these anchors.

An obvious question is which of two alternative systems of equations, (2) or (3), is preferred? Certainly, system (3) has an advantage in terms of computational efficiency, as fewer equations are involved. However, our experiments show a small advantage in favor of using (2) in terms of localization quality; see the results of Section 4. Therefore, we cannot make a firm recommendation. To improve readability, we will henceforth refer only to system (3), although system (2) is no less legitimate.

3.3 Anchoring sensors

As explained in the previous section, our system of equations (3) does not uniquely characterize the global coordinates. In particular, it is possible to completely shrink some of the d axes, resulting in degenerate layouts.

One way around this is to require that the sensor coordinate vectors in the global layout be orthonormal. In this way, it can be shown that the optimal solution to (3) is obtained by solving an $n \times n$ lowest eigenvector problem. This would make our method intriguingly similar to the Locally Linear Embedding (LLE) method [11] for dimension reduction. However, such an eigenvector-based approach is problematic. First, it imposes uniform norms on all axes. This means that the resulting layout always has balanced aspect ratio, which clearly does not always reflect reality. A second issue is that fully distributed computation of eigenvectors is non-trivial. The most significant issue is guaranteeing the orthogonality of the eigenvectors; see [3] and [5].

A more appropriate way to avoid degenerate layouts, in which some of the axes have collapsed to nothing, is by fixing locations of $d+1$ affinely-independent *anchor sensors*. Affine independence means that these sensors do not all lie on any hyperplane of dimension less than d . We will explain shortly how we can know the locations of some sensors. For now, assume the existence of a set $S \subseteq V$, s.t. the location of each $i \in S$ is known to be $x_i \in \mathbb{R}^d$. Consequently, we augment our system with additional $|S|$ equations:

$$p_i = x_i, \quad i \in S \tag{4}$$

Note that alternatively, we could impose these as “hard” constraints by replacing every unknown p_i with the value x_i . However, we found in many experiments that introducing “soft” constraints in the form of additional equations (which are then solved together with (3) by least squares) tends to produce better results.

How can we obtain the coordinates of such anchor sensors? First, in many situations some of the sensors in the network are aware of their locations. For example, a few reference sensors might be equipped with a GPS, or the network contains some key sensors that were configured manually. In such a case it would be a pity *not* to integrate the known locations of these sensors as anchors in our system. This will also eliminate the rigid transformations degrees of freedom that are inherent as long

as we have only inter-sensor distance information. We consider this ability to integrate locations of reference sensors as an advantage of our method over previous anchor-free approaches.

When reference sensors do not explicitly exist, we may pick a single set of neighboring sensors, compute their local coordinates, and then adopt this set as the anchor sensors. Certainly, we would like to choose a set for which we can compute the coordinates reliably. One possible way is to choose the patch that contains the largest number of known pairwise distances. An alternative, which we use, is to choose the *largest clique*, as we explain now.

As we mentioned before, the layout computation problem becomes trivial when *all* pairwise distances are known. Hence it makes sense to look for the largest subset of the sensors for which we know all pairwise distances, namely, the largest clique in the sensor network. In general, finding the largest clique is an NP-hard problem. However, the fact that our network is a disk-graph makes solution easier, since a clique means a subset of the sensors lying within a fixed sized circle. Moreover, we may further relax the problem and search only for cliques within a circle *centered* at a sensor. Therefore each sensor may identify the largest clique around it. Towards this end, the sensor ranks all its neighbors by their measured distance from it, starting with the closest one. Then a clique is grown by trying to add each of the neighbors, where a neighbor is added only if it is connected to all the previously added neighbors. Finally, we select the largest clique and localize it using classical-MDS, thereby fixing the sensors in the clique as anchors.

Finally, it is interesting to compare our method to Tutte’s spring embedding method [15] used for planar graph drawing. Tutte’s method fixes coordinates of *boundary nodes* to a convex shape and solves linear equations to locate all interior nodes. Tutte’s equations imply that each interior node is located at some *convex* combination of the locations of its neighbors. Hence all interior nodes will be located within the convex hull of the boundary nodes. Fixing the boundary is essential to avoid collapse of the solution without resorting to eigenvector computations [4,6]. Our method also relates the location of each sensor as some affine (but not convex) combination of the locations of its neighbors, but the fixed (anchor) sensors may be arbitrary. This subtle difference is crucial, as in our case finding a well distributed group of anchors might be impossible, so we must resort to a small condensed group of anchors.

3.4 A distributed solution

At this point, global localization of the sensors is achieved by solving an over-determined system of equations that includes all patch-induced equations (3) and anchor-induced equations (4). Since these are vector equations, we actually solve d independent systems, one for each dimension (axis) of the layout, of the form:

$$Lp^\alpha = b^\alpha, \quad \alpha = 1, \dots, d \quad (5)$$

Here, L is an $m \times n$ matrix, where $m = n + |S|$. The unknown vector p^α is the α -th axis of the layout, so the location of i is $p_i = (p_i^1, \dots, p_i^d)$. The right hand side contains the α -th component of x_i for rows based on equation (3) and zero elsewhere.

The central question is how to solve the linear system (5) in a fully distributed manner. It is well known that the least squares solution of (5) may be computed by solving the $n \times n$ normal equations:

$$(L^T L)p^\alpha = L^T b^\alpha, \quad \alpha = 1, \dots, d$$

Luckily, the sparsity pattern of L facilitates solving these equations using a fully distributed version of the well-known iterative Gauss-Seidel method [9]. As long as the system is over-determined, this process is guaranteed to converge because $L^T L$ is symmetric positive-definite. An alternative would be to employ the Conjugate-Gradient method [9] that guarantees convergence in $O(n)$ iterations.

3.5 Global stress minimization

At this stage we have computed a complete layout of the network. However, since the computations were based on the local layouts of the patches, which in turn were based on noisy distances, the result might not be as accurate as possible. While stress minimization is significantly easier on a single patch,

there is also an advantage to minimizing the stress of the complete network. The entire network contains many more distances, and this helps to eliminate the influence of noisy measurements, who cancel each other out. Consequently, we complete the global localization process by performing global stress minimization on the entire network, by the same majorization process described in Subsection 3.1. It involves iteratively applying the following update rule at each sensor i :

$$p_i \leftarrow \frac{1}{|\{j | \langle i, j \rangle \in E\}|} \sum_{\langle i, j \rangle \in E} \left[p_j + l_{ij} (p_i - p_j) \operatorname{inv}(\|p_i - p_j\|) \right]$$

4. Experimental Results

We have fully implemented our PATCHWORK algorithm in a simulated environment. We experimented with sensor networks modeled as disk graphs, containing 200 sensors uniformly distributed in geometric regions of different shapes. Due to technical difficulties with some of the reference implementations of competing algorithms, we could not run comparisons on larger networks. We varied the disk radius, which affected the average number of sensor neighbors, and the noise level of the inter-sensor edge length measurements. A random configuration of sensors was generated uniformly within the geometric region, and the lengths l_{ij} of the edges in the disk graph computed. Uniform noise was added to these edge lengths, which then served as input to the localization algorithm. The noise level is controlled by setting the maximal allowed relative deviation. The performance of the localization algorithm was measured in terms of the Average Relative Deviation (ARD) of the resulting layout, defined as:

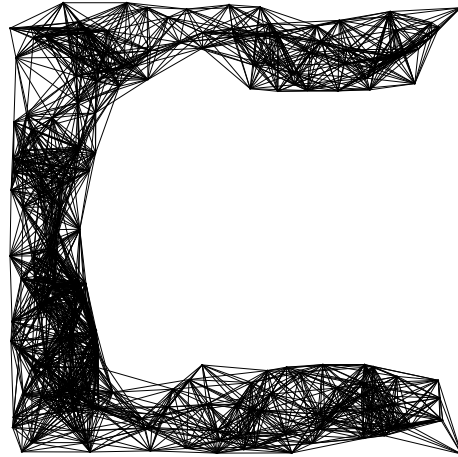
$$ARD = \frac{2}{n(n-1)} \sum_{i < j} \frac{|d_{ij} - l_{ij}|}{\min(d_{ij}, l_{ij})},$$

which is the average ratio of the deviation of the distance between two sensors in reality (l_{ij}) vs. the distances in the layout (d_{ij}). Note that this is averaged over *all possible* sensor pairs, not just sensors adjacent in the network. Thus, the ARD measure takes into account both short-range and long-range errors.

It seems that localization algorithms are sensitive to the global nature of the geometric configuration of the sensor network, as well as the local nature. By global nature we mean the shape of the region in which the sensors are located. Examples of these are square, rectangular, circular and non-convex shapes. Some of the algorithms are particularly sensitive to extreme aspect ratios between the two dimensions of the rectangular region, and certainly to non-convex regions. Hence, we experimented with three different geometric regions. The first is a 4×4 square, the second a 2×8 rectangle, and the third a non-convex C-shape within a 4×4 square, as illustrated in Figure 1. In each region we generated several families of disk graphs characterized by the disk radius and by the noise ratio, as explained above. Then we applied the algorithms to 25 random networks in each family.

Figure 1.

A network of 200 sensors, distributed uniformly within a C-shaped region. Neighboring sensors are connected by edges.



4.1 PATCHWORK variants

In our first experiment, we compared the two variants of PATCHWORK: the one based on equation (2), called PW-Full, and the one based on equation (3), called PW-Single. Moreover, we compared the two ways of choosing anchors, as described in Subsection 3.3: one is the anchor-free version, when the algorithm first localizes the largest clique and then uses it as a set of anchor sensors. The other is when 6 randomly chosen sensors (that is 3% of the 200 sensors) are *given* as anchors that are aware of their true positions. Thus there are four different variants of PATCHWORK. Table 1 shows the results for the 4×4 square. These seem to be representative of all geometries, so for brevity’s sake, we show only the results on the square region. We visually organize the four methods in each cell of the table in the following manner:

| | |
|----------------------|----------------------|
| PW-Full, 0 anchors | PW-Full, 6 anchors |
| PW-Single, 0 anchors | PW-Single, 6 anchors |

| Radius | Error = 1% | | Error = 5% | | Error = 15% | |
|--------|------------|-------|------------|------|-------------|------|
| 0.7 | 1.44 | 1.67 | 5.35 | 5.47 | 19.3 | 20.4 |
| | 2.06 | 9.57 | 5.68 | 7.54 | 22.2 | 27.2 |
| 0.85 | 0.785 | 1.47 | 4.15 | 4.85 | 17.6 | 17.9 |
| | 0.792 | 0.78 | 4.2 | 4.88 | 17.8 | 18.4 |
| 1 | 0.69 | 0.691 | 3.72 | 3.74 | 17.1 | 17.1 |
| | 0.69 | 0.687 | 3.73 | 3.73 | 17.1 | 17.1 |

Table 1. ARD ($\times 1,000$) measured for different variants of PATCHWORK running on 9 different families of networks distributed within a 4×4 square. Each ARD is averaged over 25 random networks. Results are color-coded in gray-scale from worst (black) to best (white).

We begin with a few general observations that are true for all four variants. As expected, performance deteriorates as the noise increases. However, even when the noise causes 15% error in edge length, the average relative deviation of pairwise distances in the resulting layout is much lower than in the input, around 2%. Therefore, to a large extent, errors tend to cancel themselves out. Also, as the radius increases the results are more accurate and there is almost no difference between the four variants. This makes sense, since for the larger radius we have more information about pairwise distances.

It is also evident that PW-Full (top row of each cell) outperforms PW-Single (bottom rows) almost always, but not very significantly. As noted earlier, PW-Full is based on more equations and hence tends to have some quality advantages. When comparing the anchor-free runs (left columns) to the runs where 6 anchors were given (right columns), we see a small difference with a small consistent advantage to the anchor-free results. How can an anchor-free algorithm outperform an anchored algorithm? The reason is that in the anchor-free case we pick the largest clique as anchors, and these cliques appear to contain between 15 and 25 sensors. Thus, it gives the algorithm an advantage over the case where only 6 anchors are explicitly given. However, note that in the anchor-free case the anchors’ coordinates are not accurate but computed by our algorithm and thus subject to noise. Moreover, the clique is a concentrated set of sensors, while, when given sensor locations from an external source, they are probably better distributed within the region. Hence, as the number of anchors increases, the anchor-based variant will certainly outperform the anchor-free variant by utilizing the extra external information. For larger networks, we can still expect the largest clique to contain just 15-25 sensors, as this is a local property of the network. Thus, if the number of anchors grows as a fraction of the overall

network size, it will ultimately exceed the largest clique size and the anchor-free method will be outperformed.

4.2 Comparison to the competition

In our second experiment, we compared anchor-free PATCHWORK to three previously published anchor-free localization algorithms of Priyantha *et al.* [10], Gotsman and Koren [3] and Shang and Ruml [12]. Implementations of the two latter algorithms were obtained from their respective authors. We would have liked to provide also a full comparison to the very interesting anchor-free algorithm of Moore *et al.* [8]. However, we could not use the reference implementation to generate layouts when the noise was significant. In fact, at the 15% error rate, the reference implementation succeeded in localizing less than 5% of all sensors. It is unclear to us at present whether this is an intrinsic property of their algorithm, or due to them employing a (Gaussian) noise model different from ours (uniform).

The results are presented in Table 2. Once again, each cell of the table is formed by averaging the ARD of the four methods over 25 networks of the same family. The internal order within each cell is as follows:

| | |
|---------------------|------------------------------|
| PW-Full, 0 anchors | Gotsman and Koren [3] |
| Shang and Ruml [12] | Priyantha <i>et al.</i> [10] |

Table 2. Comparison of ARD ($\times 1,000$) of four different methods, on various families of disk graphs characterized by their geometric shape, disk radius, and noise level. Each ARD is averaged over 25 random networks. Results are color-coded in gray-scale from worst (black) to best (white).

| Square of size 4x4 | | | | | | |
|-----------------------|------------|-------|------------|------|-------------|------|
| Radius | Error = 1% | | Error = 5% | | Error = 15% | |
| 0.7 | 1.44 | 5.14 | 5.35 | 8.41 | 19.3 | 22.9 |
| | 1.46 | 3.57 | 8.99 | 6.95 | 40.3 | 20.3 |
| 0.85 | 0.785 | 1.7 | 4.15 | 4.94 | 17.6 | 18 |
| | 1.34 | 1.12 | 10.8 | 4.38 | 50.5 | 17.6 |
| 1 | 0.69 | 0.957 | 3.72 | 3.93 | 17.1 | 17.1 |
| | 1.63 | 1.07 | 13.6 | 4.03 | 61.1 | 16.7 |
| Rectangle of size 8x2 | | | | | | |
| Radius | Error = 1% | | Error = 5% | | Error = 15% | |
| 0.7 | 3.06 | 152 | 6.83 | 154 | 21.3 | 163 |
| | 1.38 | 97.7 | 8.64 | 97.5 | 42.6 | 101 |
| 0.85 | 0.858 | 132 | 4.26 | 142 | 18.2 | 163 |
| | 1.34 | 51.9 | 10.7 | 53.6 | 49.9 | 66.6 |
| 1 | 0.681 | 96.3 | 3.56 | 95.6 | 16.6 | 91 |
| | 1.5 | 137 | 12.8 | 111 | 59.3 | 102 |
| C-Shape of size 4x20 | | | | | | |
| Radius | Error = 1% | | Error = 5% | | Error = 15% | |
| 0.7 | 5.5 | 214 | 14.7 | 215 | 37.7 | 209 |
| | 2.79 | 259 | 17.6 | 271 | 69.6 | 268 |
| 0.85 | 2.79 | 192 | 8.34 | 195 | 28.2 | 193 |
| | 2.53 | 214 | 17.6 | 213 | 73.3 | 212 |

We can learn a lot about the various algorithms from Table 2. First, we can see that the concurrent global methods, i.e., Priyantha *et al.* [10] and Gotsman and Koren [3] shine when applied to the 4×4 square. However, they significantly and consistently under-perform when operating on less regular geometries, the elongated rectangle and the non-convex C-shape. This is probably due to the fact that both methods depend in their first phase on some assumptions on the network structure, which are in fact incorrect for the two latter geometries. On the other hand, these two concurrent methods have an advantage over the incremental method of Shang and Ruml [12] by tolerating noise better. This is to be expected, since their global optimization is better able to reduce noise. The method of Shang and Ruml, which consistently delivers good results for low noise levels for all three geometries, tends to deteriorate rapidly as the noise level increase, becoming the worst performer in some cases.

As far as PATCHWORK is concerned, these tables illustrate its main advantage: It is able to reasonably handle all three geometries, while still delivering good results even as the noise increases. In fact, in 19 network families of the 24 tested, it outperformed all the other algorithms.

5. Discussion

Localization is a fundamental technique for sensor networks, which is essential for the basic operation of the network. It facilitates location-stamped information and more advanced capabilities like location-based routing. Hence, not surprisingly, many algorithms have been recently proposed to address the localization problem. Many criteria can be used to classify these methods. We have found the classification given by Priyantha *et al.* [10] very useful. They use two dichotomies: *anchor-based* vs. *anchor-free* algorithms and *incremental* vs. *concurrent* algorithms.

Interestingly, our algorithm "spoils" these dichotomies, combining the best of both worlds. We can incorporate the valuable information about locations of designated sensors ("anchors") as in an anchor-based method. However, we do not need many such anchors, and, in fact, can operate well even without explicitly given anchors, becoming an anchor-free method. Likewise, we follow the bottom-up methodology employed by many incremental approaches, by performing local coordinate computations. This allows us to treat local patterns present in the network accurately, without masking them by assumptions on the global nature of the network. Consequently, we can deal with regions having a wide variety of geometric shapes. However, all the pieces of local information are merged together by a concurrent global optimization process, which is not sensitive to local noise as a greedy incremental process is.

For technical reasons we were not able to compare the performance of PATCHWORK to that of the algorithm of Moore *et al.* [8], and we hope to complete this in the near future. While we believe that PATCHWORK has many advantages over that algorithm, their idea of first localizing the rigid component of the network seems to be a powerful one. It allows reaching an accurate localization for the component of the network which is most robust to noise, without interference by other sensors which are more sensitive.

PATCHWORK performs only one-hop computations, namely, involving only a sensor and its immediate neighbors. This may be improved significantly, at the cost of additional communication, by interacting also with two-hop neighbors. A variant of the algorithm of Shang and Ruml [12] does this, and we would like to test its effect on PATCHWORK.

We are quite excited at the similarity between PATCHWORK's global algorithm for reconciling between multiple pieces of local information and existing methods for low-dimension manifold representations developed in the learning community. We would like to believe that the connection is deeper than just what we have presented here. This will be a subject of future investigation, and we hope to

find additional ways that sensor networks may benefit from techniques developed in the learning community.

References

- [1] Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, Springer-Verlag, 1997.
- [2] M. Brand. *Charting a Manifold*. Proc. Neural Information Processing Systems 15 (NIPS), 2002.
- [3] C. Gotsman and Y. Koren, *Distributed Graph Layout for Sensor Networks*, Proc. Intl. Symp. Graph Drawing, LNCS 3383, Springer Verlag, pp. 273--284, 2004.
- [4] K. M. Hall, *An r -dimensional Quadratic Placement Algorithm*, Management Science 17:219-229, 1970.
- [5] D. Kempe and F. McSherry. *A Decentralized Algorithm for Spectral Analysis*. Proc. Symp. Theory of Comp. (STOC), 2004.
- [6] Y. Koren, *On Spectral Graph Drawing*, Proc. Intl. Computing and Combinatorics Conf. (COCOON), LNCS 2697, Springer-Verlag, pp. 496-508, 2003.
- [7] M. Mauve, J. Widmer and H. Hartenstein, *A Survey on Position-Based Routing in Mobile Ad-Hoc Networks*, IEEE Network 15:30-39, 2001.
- [8] D. Moore, J. Leonard, D. Rus and S. Teller. *Robust Distributed Network Localization with Noisy Range Measurements*, Proc. ACM Conf. on Embedded Networked Sensor Systems (SenSys), 2004.
- [9] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing* (2nd Ed.), Cambridge University Press, 2002.
- [10] N. B. Priyantha, H. Balakrishnan, E. Demaine and S. Teller, *Anchor-Free Distributed Localization in Sensor Networks*, Proc. ACM Conf. on Embedded Networked Sensor Systems (SenSys), pp. 340-341, 2003. Also TR #892, MIT LCS, 2003.
- [11] S. Roweis and L. Saul. *Nonlinear Dimensionality Reduction by Locally Linear Embedding*. Science, 290(5500):2323-2326, 2000.
- [12] Y. Shang and W. Ruml. *Improved MDS-Based Localization*, Proc. IEEE Infocom, 2004.
- [13] J. B. Tenenbaum, V. de Silva and J. C. Langford, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science, 290(5500):2319-2323, 2000.
- [14] M. Tubaishat and S. Madria, *Sensor Networks: An Overview*, IEEE Potentials 22:20-23, 2003.
- [15] W. T. Tutte, *How to Draw a Graph*, Proc. London Math. Soc. 13:743-768, 1963.
- [16] L. Xiao and S. Boyd, *Fast Linear Iterations for Distributed Averaging*, Systems and Control Letters 53:65-78, 2004.
- [17] Y. Yemini, *Some Theoretical Aspects of Location-Location Problems*, Proc. IEEE Sympos. Found. Comp. Sci., (FOCS), pp. 1-8, 1979.