# MSDR-D Network Localization Algorithm

Kevin Coogan[1], Varun Khare[1], Stephen G. Kobourov[1], and Bastian Katz[3]

[1] Department of Computer Science
University of Arizona
{kpcoogan, vkhare, kobourov}@cs.arizona.edu
[2] Faculty of Informatics
Universität Karlsruhe (TH)
katz@ira.uka.de

**Abstract.** We present a distributed multi-scale dead-reckoning (MSDR-D) algorithm for network localization that utilizes local distance and angular information for nearby sensors. The algorithm is anchor-free and does not require particular network topology, rigidity of the underlying communication graph, or high average connectivity. The algorithm scales well to large and sparse networks with complex topologies and outperforms previous algorithms when the noise levels are high. The algorithm is simple to implement and is available, along with source code, executables, and experimental results, at http://msdr-d.cs.arizona.edu/.

## 1 Introduction

Wireless sensor networks are widely studied and have numerous applications, such as environmental monitoring and mapping [1]. In most applications, determining the location of the sensors in a sensor network is an important problem. Manual configuration is infeasible in large-scale networks and relying on location hardware such as a GPS in each sensor can be expensive in terms of cost, size, or energy. Relying on a fraction of location-aware sensors (also called *anchors* or *seeds*) offers a good cost/accuracy tradeoff. However, GPS-like devices are ineffective inside buildings, under thick tree canopies and underground, making anchor-based methods unsuitable. With this in mind we consider localizing a network of anchor-free sensors in areas with non-trivial (e.g., non-simple, non-convex) underlying topologies. We assume that the sensors are distributed randomly and have distance (*range*) and angle information about their immediate neighbors, but no initial knowledge of their location. Distance information in sensor networks can be computed by measuring either round trip times or signal strength. Sensors equipped with multiple antennae, such as UCLA's Medusa [19], can be used to compute angle information by providing a counter-clockwise ordering of neighbors, as well as an approximation of the angle between adjacent neighbors. It is typical to assume the values of range and angle measurements approximate a Gaussian distribution centered around the actual value. Our force-directed MSDR-D algorithm uses these two measurements in a distributed fashion to efficiently compute sensor locations without relying on strong assumptions, such as high average degree or rigidity of the underlying graph, use of anchors, simple underlying topologies, etc.

## 1.1 Related Work

Many different algorithms for sensor network localization have been proposed, relying on nearly as many different assumptions. While certainly failing to mention all relevant previous work, here we try to briefly survey earlier work on anchor-free, range-based, force-directed methods. Using the classical multidimensional scaling technique, MDS-MAP [21] is a centralized anchor-free algorithm which depends only on the range data and works well when sensor nodes are uniformly distributed. MDS-MAP does not work well on irregularly-shaped networks where the shortest path distance between pairs of sensor nodes correlates poorly with their actual Euclidean distance. MDS-MAP(P) [20] is a distributed version which stitches together small patches obtained via MDS-MAP. It uses a computationally expensive refinement step to avoid accumulation of errors in the incremental stitching process. Costa *et al.* [3] use local refinements to improve the stitching of local MDS maps but rely on a non-trivial fraction of anchor nodes.

Priyantha *et al.* [17] propose a distributed anchor-free layout technique based on force-directed methods, while Gotsman and Koren [12] utilize a distributed majorization technique. However, neither approach makes use of angular information, and both make the simplifying assumption that sensors are distributed in convex polygonal areas. Of these approaches that do utilize angle information, most assume that some number of the sensors in the graph know their positions *a priori*, either by way of GPS [6, 16, 18, 19] or by manual input [10].

A similar spring-embedder based algorithm is used as part of the VFA sensor deployment algorithm [22]. VFA assumes random placement of some sensors across a region, and then uses a distributed algorithm to determine placement of new sensors for optimal coverage given range information. Fekete *et al.* [8] and Kröller *et al.* [15] use a combination of stochastic, topological, and geometric ideas for determining the topology of the region and its boundary. However, this approach assumes average node degrees in the hundreds to achieve its results, whereas we reply on average node degrees that are an order of magnitude smaller.

Vivaldi [4] and similar systems for predicting network latency [5] use a force-directed style algorithm in a distributed manner to estimate round trip times in a network, without probing all pairs of nodes to measure distances. However, Vivaldi embeds node locations in 2-D space in order to approximate round trip times, without regard for their actual locations. In our setting we would like to embed the nodes so that the embedding approximates the actual locations of the sensors.

The multi-scale dead-reckoning (MSDR) algorithm [7] is a centralized anchor-free force-directed localization algorithm that uses range and angular information. Instead of shortest paths between pairs of nodes, MSDR computes dead-reckoning paths, based on range and angular data, to avoid the accumulation of errors in non-uniform topologies (i.e., topologies where the shortest path in the underlying graph does not correlate with the Euclidean distance). Katz and Wagner [14] describe a force-directed approach that uses angular information similar to [7] in a distributed fashion. Their approach, while distributed, uses a hierarchical grouping of nodes that requires "global" exchange of information prior to the localization of any node, thus incurring high communication and storage overhead. Specifically, a hierarchy with all nodes in the graph must be built in a centralized fashion and stored in all nodes before the actual localization can begin.

### 1.2 Our Contribution

We present MSDR-D, a new sensor network localization algorithm that is fully distributed. Our algorithm employs a force-directed method that computes dead-reckoning distances between nodes in the network, while relying on only local range and angular information. These distance measurements are more accurate than shortest path distances, which becomes crucial in non-convex and non-simple network topologies. MSDR-D also scales well to large and sparse networks and is robust to the accumulation of errors due to its multi-scale nature and low communication overhead. MSDR-D is a truly distributed algorithm, in the sense that instructions are given to each node and result in a localization of the entire graph with no global information (e.g., a representation of the entire underlying network, a communication hierarchy, centralized server) required at any stage. MSDR-D outperforms earlier algorithms especially with noisy input data. Most importantly, the algorithm employs no broadcast messages, and uses a constant number of local messages and constant memory per node.

## 2 Distributed MSDR Algorithm

Our algorithm is a distributed version of the multi-scale dead-reckoning (MSDR) algorithm of Forrester *et al.* [7], which is in turn a force-directed placement algorithm. Therefore, we begin with a brief review of force-directed algorithms in general, and of MSDR in particular.

### 2.1 Force Directed Algorithms

In its full generality, the sensor network localization problem can be thought of as that of calculating a graph layout. Force-directed algorithms, also known as spring-embedders, calculate a graph layout by defining an objective function which maps each layout into a number in $\mathcal{R}^+$ representing the energy of the layout. This function is defined so that low energies correspond to good layouts, that is, layouts in which adjacent nodes are near each other and non-adjacent nodes are well spaced. A force-directed algorithm proceeds by repeatedly calculating forces acting on each node and moving nodes around in search of a (often local) minimum of the energy function. The forces acting on a node can be a combination of attractive forces between neighbors and repulsive forces between non-adjacent nodes as in the Fruchterman-Reingold algorithm [9]. Alternatively the forces can be based on graph theoretic distances, determined by the length of the shortest paths between them, as in the Kamada-Kawai algorithm [13]. The force exerted on node $v$ in graph $G$ in the Kamada-Kawai algorithm is calculated as follows:

$$F(v) = \sum_{\forall u \neq v} \left( \frac{\mathrm{d}(u,v)^2}{\mathrm{dist}_G(u,v)^2 \cdot \mathrm{IdealLength}^2} - 1 \right) (\mathrm{pos}[u] - \mathrm{pos}[v]),$$

where $d(u,v)$ is the current Euclidean distance between nodes $u$ and $v$, $\mathrm{dist}_G(u,v)$ is the length of the shortest path between nodes $u$ and $v$, IdealLength is a constant corresponding to the desired distance between a pair of adjacent vertices (determined by the size of the graph and the drawing area), and $\mathrm{pos}[u]$ and $\mathrm{pos}[v]$ are the positions of nodes $u$ and $v$ in the current layout. The energy of the current layout is obtained by summing the forces acting on all nodes in the graph.

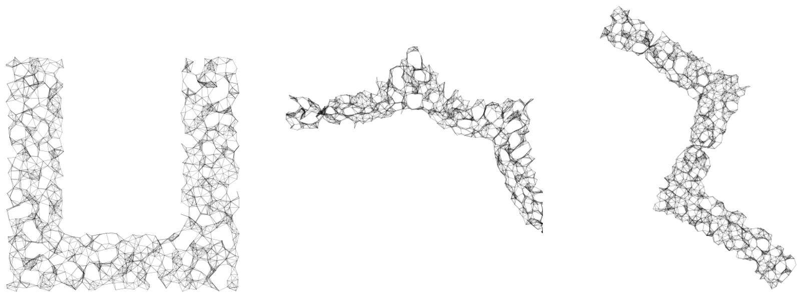


**Fig. 1.** Typical results illustrating localizations of a U-shaped network using only range information (from [7]). The network on the left contains pairs of nodes whose graph distances are not well correlated with their Euclidean distances (especially pairs of nodes at opposite ends of the U) transforming the U into a W or a Z.

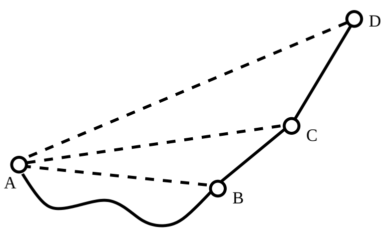
In simple, convex topologies the graph distance, computed as shortest path between two nodes, correlates well with the Euclidean distance between them. However, this is not true for more complicated topologies where a long path between two nodes does not necessarily imply they are physically far from each other (e.g., nodes at opposite ends of a U-shaped network); see Fig. 1. With the help of angular information about neighboring edges along the path between a pair of nodes we can overcome this problem as shown by the MSDR algorithm.

## 2.2 MSDR

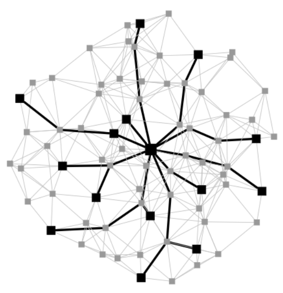
MSDR is a force-directed algorithm which computes the placement of nodes in few phases involving only subsets of nodes. It uses a modified force calculation which takes into account edge lengths and angles between adjacent edges. MSDR consists of two main stages: in the first stage a graph filtration is computed in a bottom-up fashion and in the second stage the filtration is used in the top-down manner to place the nodes in the graph.

The first stage constructs the graph filtration $\mathcal{V} = \{\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_m\}$. The set initial set $\mathcal{V}_0$ contains all the nodes in the graph. Set $\mathcal{V}_1$ is a maximal independent set and contains nodes at $2^1$ hops from each other. In general, set $\mathcal{V}_i$ contains nodes that are at least $2^i$ hops away from each other, and $\mathcal{V}_i \subset \mathcal{V}_{i-1}$. When complete, there are $m+1$ total filtration sets such that $\mathcal{V}_m \subset \mathcal{V}_{m-1} \subset \ldots \mathcal{V}_2 \subset \mathcal{V}_1 \subset \mathcal{V}_0$, with the smallest set containing exactly 3 nodes.

The second stage of the MSDR algorithm consists of $m$ phases of force-directed calculations, used to compute node locations, one filtration set at a time, starting with the smallest set $\mathcal{V}_m$. It is easy to see that the number of phases $m$ is at most logarithmic in the size of the network, or more precisely at most logarithmic in the diameter of the network, as nodes at filtration level $i$ are at distance at least $2^i$ from each other. In order to run the force-directed algorithm on the nodes in the smallest filtration set $\mathcal{V}_m$, distances between the 3 nodes in this set are needed. A breadth first search from each node is used to find a path to the other two nodes. Once a path has been found, the local distance and angle information at each step is used in a "dead-reckoning" fashion to calculate the distance to each of the other nodes.



(a)



(b)

**Fig. 2.** (a) Computing $\text{dist}_{DR}(A, D)$, given a path between them with intermediate nodes $B$ and $C$, where the dead reckoning distance between adjacent nodes is equal to the distance between them (from range data) using the law of cosines: $\text{dist}_{DR}(A, C) = \sqrt{\text{dist}^2_{DR}(A, B) + \text{dist}^2_{DR}(B, C) - 2\text{dist}_{DR}(A, B) * \text{dist}_{DR}(B, C) * cos(ABC)}$ and similarly $\text{dist}_{DR}(A, D)$ once we have $\text{dist}_{DR}(A, C)$. (b) A combination of neighbors and far away nodes are used to fit a locally obtained map in the larger network.

Deduced reckoning (or dead reckoning) is an ancient method for estimating the current position of a moving object, assuming knowledge of the direction and distance traveled from a previously known position. Given a path between two nodes, along with the lengths of the edges and the angles between adjacent edges, we can calculate the dead reckoning location of one node with respect to the other using the law of cosines; see Fig. 2(a). Thus, we can replace the shortest path distance with the dead reckoning distance in the force-directed calculation, by adding only local calculation and communication overhead (only immediate neighbors in the sensor network need to communicate) to propagate the information. The main advantage of dead reckoning distances is that they correlate well with Euclidean distances for both simple and complex topologies (non-simple, non-convex, etc.).

In phase $i$, nodes that are in $\mathcal{V}_{i-1}$, but not in $\mathcal{V}_i$ are localized. For each new node, breadth first search is used to find its distance to a small number of nodes that have already been placed. After each phase, a local force-directed refinement to the placements is applied. In this way new nodes receive an initial placement in the graph, and already placed nodes have their positions refined. The process continues in this "multi-scale" fashion until the nodes in $\mathcal{V}_0$ (i.e., all the nodes) have been placed. Note that the dead-reckoning forces used to calculate node displacements are given by:

$$F_{DR}(v) = \sum_{\forall u \neq v} \left( \frac{\text{d}(u, v)^2}{\text{dist}_{DR}(u, v)^2 \cdot \text{idealLength}^2} - 1 \right) (\text{pos}[u] - \text{pos}[v]),$$

where $\text{dist}_{DR}(u, v)$ is the dead-reckoning distance between nodes $u$ and $v$, computed from the edge lengths and angles between adjacent nodes along the $u$-$v$ path.

MSDR creates very good anchor-free localizations, performing much better than earlier algorithms when the underlying network topologies are non-simple, or non-convex. The algorithm scales to large and sparse networks as it is resilient to noise and accumulation of errors. The main disadvantage of the MSDR algorithm is that it is inherently centralized, thus forcing prohibitively high communication overhead if implemented in distributed fashion. Katz and Wagner [14] use a distributed, hierarchical reduction technique in an effort to decentralize the inherently centralized force-directed algorithm. In the first of two phases of this algorithm, a global overlay hierarchy is computed in an inherently centralized way and stored in each node. The second phase utilizes the hierarchy in a distributed localization of the sensor nodes. The algorithm begins with each node exchanging information with its 3-hop neighborhood and running a local force-directed localization. Only a fraction of the nodes, a maximal independent set, then defines an overlay network, connecting pairs of nodes having distance at most 3 in the original graph. Using estimated distances and directions from local solutions as the input, the same step is recursively applied on the overlay network, until the overlay network finally consists of a single node. This node then is localized as the origin and all nodes in the network are assigned positions in a top-down fashion. Given a localization for a node in some level's overlay network, close "dominated" nodes from the next lower level network are positioned using the local solution of the already localized node in the respective overlay level. Unlike iterative approaches that "glue" together local solutions, this approach benefits from the stress minimization on different scales, preventing cumulative localization errors especially in very large networks.

## 2.3 MSDR-D

Both of the above force-directed algorithms [7, 14] result in very accurate localizations but they are far from perfect when implemented in distributed fashion. The main disadvantage of these approaches is the reliance on a centralized server which computes the localization and the corresponding prohibitively high communication overhead. Specifically, the Katz and Wagner algorithm [14] builds and stores a hierarchy for the entire graph before the localization, which leads to the high communication and storage overhead. In our approach we overcome such problems due to the truly distributed nature of our algorithm, albeit at the expense of somewhat reduced quality of the localization.

**MSDR-D Overview:** MSDR-D begins with a single arbitrarily chosen root node and then localizes the network in a "bottom up" manner. The root node calculates a neighborhood of nearby nodes (including itself) that it will localize. We call this neighborhood the root's $k$-neighborhood, where $k$ refers to the maximum number of hops from the root to any node in the neighborhood and is typically a small constant like 2 or 3. To this neighborhood, the root adds a small number of "far-away" nodes that are used to improve the global localization. Using these two types of nodes, the root performs a local MSDR force-directed localization that utilizes distance and angle information, and assigns a position to each node in its $k$-neighborhood. Finally, the root examines its $k$-neighborhood, and selects nodes adjacent to, but outside, the neighborhood to be root nodes for the next iteration.

In the next iteration, all selected nodes become root nodes and build their own neighborhoods. Since these new roots are on the edge of the old neighborhood, it is expected that a fraction of the nodes in the new $k$ neighborhoods will have already been placed. The distance and angle information for these nodes is used to perform the localization, but their positions never change. Once a node is placed, it is considered fixed. It is this overlap with previous iterations along with the far-away nodes that allows for the “stitching together” of local results. The iterations of localizing neighborhoods and selecting new roots continues until all nodes in the graph have been placed.

**The $k$-neighborhood:** Since our MSDR-D algorithm relies on the notion of localizing nodes that are “close” to the root, we need to define precisely what this means. We use the idea of a $k$-hop neighborhood, where a root node’s $k$-hop neighborhood is all the nodes that can be reached from the root in no more than $k$ hops. Note that here we do not take into consideration actual distances, but count immediate neighbors as nodes that are one hop away. We assume, for the purposes of the localization, that all nodes in the neighborhood know the distances (from the sensor range data) to all other nodes in the neighborhood. Typically, small value of $k$ will result in small neighborhoods that can be localized well with respect to each other, but not with respect to nodes in the graph outside the neighborhood. Conversely, large values of $k$ take into account more graph information, but require more work and communication to calculate the forces for all the nodes in the neighborhood. In our implementation $k$ is an input parameter typically set to $2$ or $3$.

**Far-away nodes:** We would like to use as much graph information as possible to place the nodes at each iteration, but we do not want to suffer the communication, computation, and memory penalties of increasing $k$ too much. Instead, we use a few “far-away” nodes as a substitute of a much larger neighborhood; see Fig. 2(b). The idea is to randomly select some constant number of nodes that are too far away from the root to be in its $k$-neighborhood, and incorporate their information into the localization. The information from these nodes improves the localization steps by aligning the local placements with the overall shape of the network. To select these nodes we use a simple random-walk algorithm starting with a border node (a node with non-neighborhood nodes as immediate neighbors). The number of steps in the walk is chosen at random between 1 and some constant smaller that the diameter of the network. At each step a random neigbor is selected and a message is passed, containing the current path and dead reckoning distance to the root. After the last step, the information is propagated back to the root node along the stored path and the last node is added to the neighborhood as a far-away node. In our implementation the number of far-away nodes is an input parameter typically set to $9$ or $10$.

**Pseudocode:** Algorithm 1 summarizes the MSDR-D algorithm that is executed at each individual sensor node. Each node is assumed to be waiting for a message from elsewhere in the network. Two kinds of messages can arrive: one that wants to assign a location to the node, and another that orders the node to become a root, and perform its own localization. Note that once a node has a location in the graph, it ignores all following messages.

In line 6 of the pseudocode, “myLocation” refers to the location of the node receiving the message. In line 11 of the pseudocode, “$k + 1$ hop neighbors” are the border nodes (nodes at graph distance $k + 1$ from the current root node). These nodes are not assigned locations by the root and are candidates to be root nodes in the next iteration.

## 3 Experiments and Results

In this section, we describe our methodology for experimentation, the generation of our simulated sensor networks, and the metrics we use to compare results to other ex-

**Algorithm 1** MSDR-D Distributed Localization Algorithm

```
1: **Handle Message:: Location loc**
2: if myLocation == null then
3:     myLocation = loc
4: end if
5: Handle Message:: Start Localization
6: if myLocation==null then
7:     Create neighborhood of $k$-hop neighbors and farAway nodes
8:     Calculate angle and distance information from neighbors
9:     Localize
10:    Distribute coordinates to $k$-hop neighbors
11:    Ping k+1 hop neighbors to Start Localization
12: end if
```

periments. Of the previous algorithms that utilize range and angular information, most assume that some number of the sensors in the graph know their positions *a priori*, either by way of GPS [6, 16, 18, 19] or by manual input [10]. Such assumptions make it significantly easier to localize than in our setting, where we have no a priori positions. Further, some of the earlier algorithms assume an underlying network protocol, only report results on small graphs, or do not make source code available. We directly compare MSDR-D to two previous algorithms [7, 14] which make the same assumptions (use of distance and angular information, no reliance on anchors or special network protocols).

## 3.1 Experimental Setup

We use simulated networks of sensors that are distributed across a variety of underlying shapes, leading to different network topologies. Input parameters that control the network type include the number of sensor nodes, the average node degree, shape of the area in which to distribute the nodes, and range and angle errors. We vary the number of sensor nodes from 50 to 1000. The average node degree is typically between 6-12. We use several standard shapes to evaluate the performance of the algorithm, including square, star, square donut (square with a hole in the center), and U-shape; see Fig. 3. Range errors vary from 0 to 50% and angle error varies from $0°$ to $25°$. Note that an angle error of $20°$, which may sound too accurate, means that a neighbor is sensed anywhere $20°$ to the left or $20°$ to the right of its true location (which corresponds to a sector of angle $40°$ or 1/9 of the total 360). Both kinds of errors are drawn from a Gaussian distribution.

Once the shape of the underlying network has been chosen, sensor nodes are distributed at random, so as to achieve the desired average degree. When small average degrees are desired, the network might become disconnected; we ignore these cases and only consider connected networks. This leads to a more regularly connected networks, akin to grid-with-perturbation networks [2].

## 3.2 Evaluation Metrics

There is a large number of different metrics used to evaluate the performance of sensor localization algorithms. Different assumptions (anchored or anchor-free, range-based

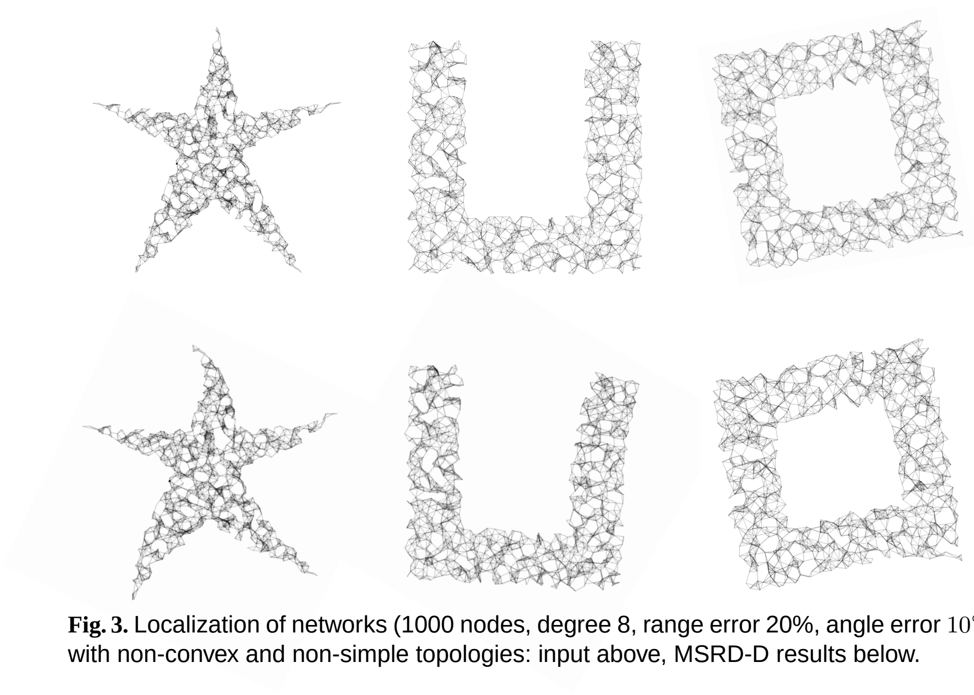


**Fig. 3.** Localization of networks (1000 nodes, degree 8, range error 20%, angle error $10°$) with non-convex and non-simple topologies: input above, MSRD-D results below.

or range-free, static or mobile) and different goals (resilience to errors, scalability in the number of sensor nodes, applicability in non-simple topologies) lead to different metrics that best capture the particular setup. For anchored networks it is often enough to measure the average distance between the location of a node, $loc[u]$, and its true position, $pos[u]$, over all nodes in the network: Error $= (\sum_{\forall u} |pos[u] - loc[u]|)/n$. However, such a metric assumes that some of the nodes are anchors, and the final layout has been aligned so as to match the true network using affine transformations (rotation, translation, scaling) [20, 21].

The *global energy ratio* used by Priyantha *et al.* [17] and the *average relative deviation* (ARD) used by Gotsman and Koren [12] are employed in an anchor-free setting and consider the distances between the $O(n^2)$ pairs of nodes in the network. The two metrics are similar and appropriate for comparing layouts obtained by different anchor-free algorithms for graphs of the same size. However, both metrics fail to compare the quality of the localizations across different graph sizes, as they are total rather than scaled sum-of-squares type measurements.

The Frobenius metric [11] has been used to evaluate the performance of anchor-free localization algorithms [7, 14] and is well-suited to compare the quality of the localizations across different graph sizes. The Frobenius metric is a scaled sum-of-squares and can be used to measure the global quality of the localization, by considering all pairs of distances between nodes in the network:

$$FROB = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\hat{d}_{ij} - d_{ij})^2}.$$

## 4 Results

Here we compare the MSDR-D algorithm to the MSDR and the Katz-Wagner algorithms. The results in Figure 4 show the performance of the three algorithms across four shapes when varying the size of the network from 50 to 1000 nodes and using range and angle data with little noise. All three algorithms scale well as the number of nodes in the sensor network increases. When the noise level is low MSDR-D produces localizations with higher errors that the other two algorithms. However, even though the errors are higher, the localizations obtained by MSDR-D are still very good as seen in Fig. 3. More importantly, MSDR-D is truly distributed algorithm and does not incur the high communication and memory overhead of the other two, which require $O(n \log n)$ broadcast messages and $O(n)$ size memory per sensor node. The MSDR-D algorithm limits the memory needs at each sensor node since it gathers information for only $k$-neighborhood and far away nodes. As communication at a global scale is avoided, no routing information needs to be maintained at the nodes. The dramatic reduction in communication and memory usage for MSDR-D do come with a price, namely a loss in localization accuracy, when compared to the other two algorithms.

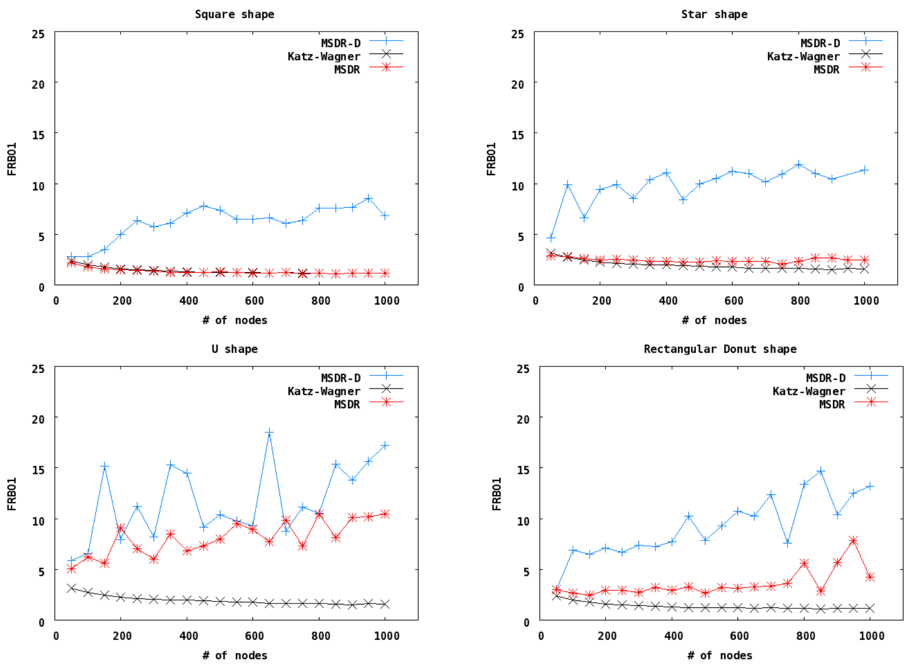


**Fig. 4.** Comparison between MSDR, Katz-Wagner, and MSDR-D measured by Frobenius error across square-shape, star-shape, U-shape and donut-shape with 50 to 1000 nodes. There were ten trials per shape, using networks with average node degree 8, range error of 20%, angle error of 10°, $k$-neighbor value of 3, and far away value of 10.

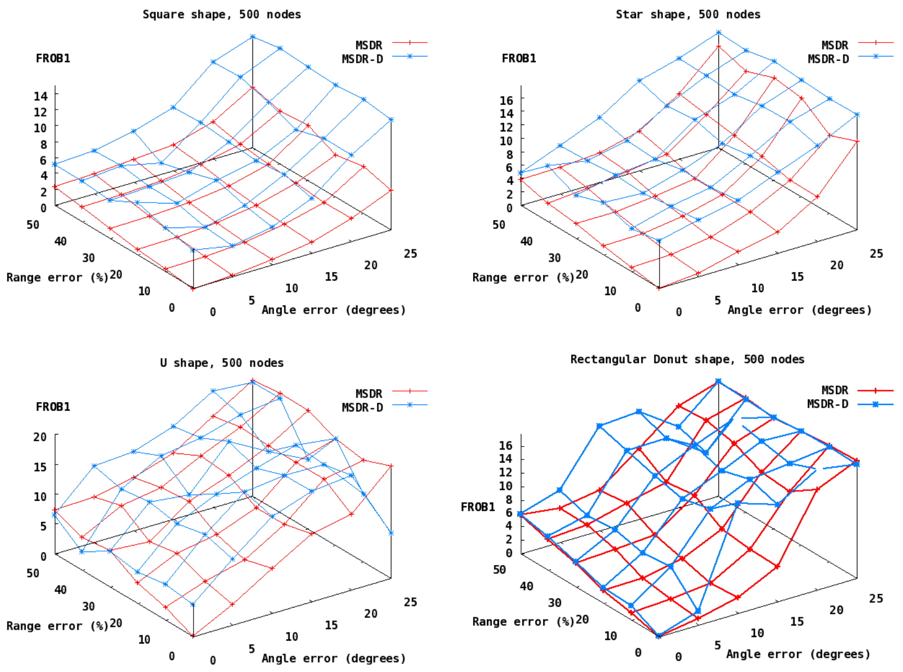


**Fig. 5.** Comparison between MSDR and MSDR-D measured by Frobenius error across square-shape, star-shape, U-shape and donut-shape with 500 nodes. There were ten trials per shape, using networks with average node degree 8, range error of 0-50%, angle error of $0^\circ$-$25^\circ$, $k$-neighbor value of 3 and far away value of 10.

In Figure 5 we consider the quality of localizations produced by MSDR and MSDR-D under varying angle and range errors. Recall that MSDR considers the angle and distance information between all pairs of sensor nodes. The large number of forces used to localize individual nodes results in an large error in the final localized position. MSDR-D only considers small neighborhood graphs of fixed sizes and hence limits the number of forces used in the localization. This explains the higher quality localizations obtained by MSDR-D when the noise levels increase. For high noise levels MSDR-D outperforms the centralized MSDR algorithm. This is especially pronounced for non-convex network topologies such as the U-shape and the star.

## 5 Conclusion

We presented a new distributed algorithm, MSDR-D, that localizes sensor networks with non-trivial topologies, using only noisy range and angular information. MSDR-D uses a combination of nearby nodes and randomly selected far-away nodes from outside the neighborhood to place local nodes in a way that is accurate locally and fits well with the overall shape of the network. The algorithm scales well as the number of nodes in the network increases. With increasing data noise, MSDR-D outperforms its predecessors. Most importantly, unlike earlier force-directed, anchor-free algorithms, MSDR-D is a truly distributed algorithm that employs no broadcast messages. The algorithm is simple to implement and is available, along with source code, executables, and experimental results, at `http://msdr-d.cs.arizona.edu/`.

## References

1. I. F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.
2. J. Bruck, J. Gao, and A. A. Jiang. Localization and routing in sensor networks by local angle information. In *MobiHoc'05*, pages 181–192, 2005.
3. J. A. Costa, N. Patwari, and A. O. Hero III. Distributed weighted-multidimensional scaling for node localization in sensor networks. *ACM Trans. on Sensor Networks*, 2(1):39–64, 2006.
4. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *ACM Conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.
5. C. de Launois, S. Uhlig, and O. Bonaventure. Scalable route selection for ipv6 multihomed sites. In *Networking'05*, pages 1357–1361, 2005.
6. L. Doherty, K. Pister, and L. E. Ghaoui. Convex optimization methods for sensor node position estimation. In *INFOCOM-01*, pages 1655–1663, 2001.
7. A. Efrat, C. Erten, D. Forrester, A. Iyer, and S. Kobourov. Force-directed approaches to sensor localization. *ALENEX-06*, pages 108–118, 2006.
8. S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. In *ALGOSENSORS-04*, pages 123–136, 2004.
9. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991.
10. A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem. Distributed online localization in sensor networks using a moving target. In *IPSN-04*, pages 61–70, 2004.
11. G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Press, Baltimore, MD, 1996.
12. C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. In *12th Symposium on Graph Drawing (GD)*, pages 273–284, 2004.
13. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
14. B. Katz and D. Wagner. Multi-scale Anchor-free Distributed Positioning in Sensor Networks. In *WTASA-07*, 2007.
15. A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *SODA-06*, pages 1000–1009, 2006.
16. D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AOA. In *INFOCOM-03*, pages 1734–1743, 2003.
17. N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. In *SenSys-03*, pages 340–341, 2003.
18. F. Reichenbach, R. Salomon, and D. Timmermann. Distributed obstacle localization in large wireless sensor networks. In *IWCMC-06*, pages 1317–1322, 2006.
19. A. Savvides, C. Han, and M. Srivastava. Dynamic Fine-Grained localization in Ad-Hoc networks of sensors. In *MOBICOM-01*, pages 166–179, 2001.
20. Y. Shang and W. Ruml. Improved mds-based localization. In *In Proceedings of IEEE INFOCOM 04*, pages 2640–2651, 2004.
21. Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from mere connectivity. In *In Proceedings of MobiHoc '03*, pages 201–212, 2003.
22. Y. Zou and K. Chakrabarty. Sensor deployment and target localization in distributed sensor networks. *Trans. on Embedded Computing Sys.*, 3(1):61–91, 2004.