Sensor Network Localization for Moving Sensors

Arvind Agarwal*, Hal Daumé III*, Jeff M. Phillips† and Suresh Venkatasubramanian†
*Department of Computer Science
University of Maryland, College Park, Maryland 20743
Email: {arvinda, hal}@cs.umd.edu
†School of Computing
University of Utah, Salt Lake City, Utah 84112
Email: {jeffp, suresh}@cs.utah.edu

***Abstract*—Sensor network localization (SNL) is the problem of determining the locations of the sensors given sparse and usually noisy inter-communication distances among them. In this work we propose an iterative algorithm named PLACEMENT to solve the SNL problem. This iterative algorithm requires an initial estimation of the locations and in each iteration, is guaranteed to reduce the cost function. The proposed algorithm is able to take advantage of the good initial estimation of sensor locations making it suitable for localizing moving sensors, and also suitable for the refinement of the results produced by other algorithms. Our algorithm is very scalable. We have experimented with a variety of sensor networks and have shown that the proposed algorithm outperforms existing algorithms both in terms of speed and accuracy in almost all experiments. Our algorithm can embed 120,000 sensors in less than 20 minutes.**

***Keywords*-Embedding, sensor network localization;**

I. Introduction

With the advancement of computer and wireless technology, wireless sensors are becoming smaller, more computationally capable, and more inexpensive causing wireless sensor networks to be commonplace. Wireless sensors can be effectively used to measure temperature, vibrations, sound, pressure, etc, and find applications in many areas such as military applications, environment or industrial control and monitoring, wildlife monitoring, and security monitoring; however, in most applications, if the networks are to achieve their purpose, locations of the sensors must be known. Yet, in many networks, locations are given for a few of the sensors (called anchors), and the locations of the remaining sensors must be determined. Although Global Positioning System (GPS) can be used to determine the locations, GPS systems are usually bulky, consume too much power, usually expensive compared to the sensors, and can only be used outdoor. In such scenarios when GPS cannot be used to find the locations, finding locations by other means become important.

One way to find the locations of these sensors is to use the intercommunication distances (sensor-sensor and sensor-anchor), and treat the localization problem as a distance embedding problem. Distances between these sensors are usually measured by communicating with only nearby sensors since further communication is limited by sensor power. Such distance measurements provide a sparse distance matrix. Sensor network localization (SNL) [1], [2], [3], [4], [5], [6], [7] is the problem of determining sensor locations using known anchor locations and (usually noisy) distance measurements (i.e. sparse noisy distance matrix). This is a large, non-convex, constrained optimization problem. Large networks contain many thousands of sensors, whose locations should be determined accurately and quickly.

In this work, we also consider a special case of the SNL problem, when sensors are moving. This moving sensors scenario occurs in many applications e.g., wildlife tracking (animal networks), vehicle tracking etc. In animal networks, sensors are installed on animals to help them track. Although any algorithm that can be used for stationary SNL problem can also be used for moving case, one can do better if it takes into account the movement of the sensors, and is able to exploit the information available along the movement trail. In this work, we present an algorithm that is especially suitable for moving sensors. Although focus is on moving sensors, there is no reason why one cannot apply the proposed algorithm for stationary case, however in our experiments, we found this algorithm to be more effective for the moving case.

Our work is inspired by Agarwal et al. [8]. In their work, they describe a distance embedding algorithm where all pairwise distances are available and there are no anchors. In this work, we modify this algorithm to exploit the available anchors to position the remaining sensors more accurately, and to handle the sparse and noisy distance measurements. We experiment with the moving scenario for a variety of networks, and show that

the proposed algorithm outperforms the existing state-of-art SNL algorithms across almost all experiments. Our algorithm comes with many attractive properties:

1) It is scalable. It is the first algorithm test on more than 100,000 sensors with *noisy* distance measurements. In particular we localize 120,000 sensors in less than 20 minutes.
2) It is decentralized. Each step can be run on each sensor with only neighborhood information.
3) It is guaranteed to converge. The cost function monotonically decrease after the initialization.
4) It can refine other algorithms' outputs. The algorithm can take advantage of a *warm start*, (e.g. the output of another algorithm) and will quickly converge to a local optimum.
5) It is effective for real-time tracking. It takes only a fraction of second for a tracking problem with 1000 sensors; See results in Section IV-E.
6) It is general. It preserves all generalities of [8], including for various spaces and cost functions.

II. Related Work

In the last one decade, there has been a considerable amount of work in stationary SNL, however we are not aware of any algorithm that is especially designed for moving sensors. In this section we discuss some of the recent work for SNL problem, especially for the *large scale noisy* problems. For more detailed discussion on the related work, we refer readers to [9], [7]

The literature on solving SNL problems can fit in two broad categories: ones that performs convex relaxation on the cost function (see equation (1) below) using semi-definite programming (SDP); and ones that directly minimizes the non-convex cost function using multi-dimensional scaling (MDS)-style algorithms.

In the SDP-type algorithms, one of the most notable work is of Biswas Ye [1]. They propose a semi-definite relaxation of the SNL problem, now called Biswas-Ye relaxation. Although effective, this relaxation can only be used for small problems with exact distance measurements. Authors later extended this work for larger scale [10], [11] and noisy problems [2]. This was further extended by Carter et al. [3] to solve the even larger problems by breaking large problems into a series of small sub-problems, each solved using an SDP.

To handle even larger problem using SDP relaxation, Wang et. al. [12] proposed further relaxation of SNL problem, where unlike the Biswas-Ye relaxation, one does not require the entire distance matrix to be positive semi-definite. Instead, the problem is broken into smaller subproblems, and the semideifinite constraint is enforced only on the subproblems. They proposed two such relaxations edge-based and node-based; the edge-based (ESDP) appears more successful. Although considerably faster than the original Biswas-Ye relaxation, it still takes significant time compared to the other more recent approaches. For $n = 4,000$ sensors (and with noise $\sigma = 0.01$, neighborhood radius $R = 0.035$ in a unit cube) it took more than 10 minutes which is slower than the some of the recent approaches [5], [6] but better than the SOCP approach of [4] and Carter et. al. [3]. A noise-aware version of ESDP called ρ-ESDP was proposed by Pong and Tseng (2009) [5] in which they use the Log-barrier Penalty Coordinate Gradient Descent (LPCGD) method to solve the SDP relaxation problem. In our experiments, we have found this to be the best baseline among all competing algorithms both in terms of speed and accuracy. Since the SNL problem typically has sparse input, a sparse version of the Biswas-Ye relaxation is proposed by [6] called SFSDP. This allows one to handle the sparse noisy SNL problem efficiently. In their work Kim et al. could handle the problem up to 18,000 sensors and 2000 anchors in less than 10 minutes. Some of the more recent work in SNL world is done by Krislock [7] where smaller SDPs are solved to embed the sensors in a successive manner. This algorithms can handle enormous numbers of sensors (experiments were performed on sensor sets of size up to 100,000), and is remarkably fast and accurate for the exact distance measurements, but when there is a noise in the measurement, algorithms performs very poorly, both in terms of time and accuracy (see results in Section IV-C).

In the second category (MDS-style), the work that is most related to us is by Costa et al. [13]. They present a distributed algorithm dwMDS which uses iterative refinement, and finds a minimum of a global cost function using a majorization algorithm. Although similar to our algorithm, its biggest limitation is that it does not make smart use of available anchors, and thus is not suitable when there are lots of anchors and distance measurements are noisy. Another related work in the MDS-style approaches is done by Moore et al. [14] which is based on trilateralization. This requires each node to have a degree or 10 or more between any pair of neighbors, or more if the distance measurements are noisy.

III. Our Approach

Problem 1 (Sensor Network Localization (SNL))**.** *Consider a graph $G = (V, E, D)$ with vertex set of size $|V| = n$, where D_{ij} is the estimated distance for each edge $(i, j) \in E$, and where there are m anchor vertices $(v_{n-m+1}, \ldots, v_n)$ with known and fixed locations. The*

Algorithm 1 PLACEMENT$(D, X_0, \epsilon_x, \epsilon_d, \sigma, N_{out}, N_{in})$

$X \leftarrow$ AddAnchors$(D, X_0, \epsilon_x, \epsilon_d, \sigma, N_{out}, N_{in})$
for $t = 1$ **to** N_{out} **do**
  for $i = 1$ **to** $|X|$ **do**
    $U = \{x_j : (i, j) \in E\}$
    $S = \{D_{ij} : x_j \in U\}$
    $x_i \leftarrow$ place-center$(x_i, U, S, N_{in}, \epsilon_x)$
Return X

Algorithm 2 place-center$(x_i, U, S, N_{in}, \epsilon_x)$

for $t = 1$ **to** N_{in} **do**
  $x_{old} \leftarrow x_i$
  for $j = 1$ **to** $|U|$ **do**
    $x'_j \leftarrow$ move $x_j \in U$ towards x_i by S_j
  $x_i \leftarrow$ Mean$(x'_1 \dots x'_{|U|})$
  if $\|x_i - x_{old}\|_2 < \epsilon_x$ **then**
    Return x_i
Return x_i

Algorithm 3 AddAnchors$(D, X_0, \epsilon_x, \epsilon_d, \sigma, N_{out}, N_{in})$

$X \leftarrow X_0$
isAncr $:= |X_0| \times 1$ vector, TRUE if x_i is anchor.
for $t = 1$ **to** N_{in} **do**
  for $i = 1$ **to** n **do**
    if isAncr$[i] =$ FALSE **then**
      $U = \{x_j \mid (i, j) \in E, \text{isAncr}[j] = \text{TRUE}\}$
      if $|U| \geq d + 2$ **then**
        $S = \{D_{ij} : x_j \in U\}$
        $c = 0$
        while isAncr$[i] \neq$ TRUE & $c < 30$ **do**
          $c = c + 1$
          $x_i \leftarrow$ place-center$(x_i, U, S, t, \epsilon_x)$
          if $C_i(x_i, S, U) < \max(\sigma, \epsilon_d)$ **then**
            isAncr$[i] \leftarrow true$
          else
            Reinitialize x_i
Return X

SNL problem seeks a mapping $\mu : V \to \mathbb{R}^d$ to minimize

$$\sum_{(i,j)\in E} \Big(\|\mu(v_i) - \mu(v_j)\|_2 - D_{ij}\Big)^2 \qquad (1)$$

over all choices of μ that restricts anchor vertices to their known locations $(x_{n-m+1}, \dots, x_n)$. For notation, we label $X = \{x_i = \mu(v_i)\}$ for all vertices.

We solve SNL problem *directly* without converting it into a related problem and without relaxing any constraints. We propose an algorithm inspired by [8], and call it PLAce CEnter with Missing ENTries (PLACEMENT). Let X be $d \times n$ matrix such that the last m columns of this matrix are anchors. Let X_0 be another $d \times n$ matrix that will be used to initialize X. Let ϵ_x and ϵ_d be terminating parameters of the algorithm, and N_{in}, N_{out} be the number of iterations. Let

$$C_i(x, U, S) = \sum_{(i,j)\in E} \Big(\|x - \mu(v_j)\|_2 - D_{ij}\Big)^2$$

be the embedding cost of one point v_i at location x where $U \subset V$ is the set of neighboring of x and $S \subset D$ are the corresponding distances of x to its neighbors U.

The main algorithm is presented in Algorithm 1. This algorithm consists of two stages. In the first stage, it uses the existing anchors to find more anchors using Algorithm 3. When it can no longer add anchors, it invokes the second stage to embed the remaining points and to refine their location. The significant difference between this algorithm and that of Agarwal et al. is the AddAnchors step. For sparse networks, the algorithm proposed in Agarwal et.al is very susceptible to local minima. This problem of local minima is alleviated using the AddAnchors subroutine and a good initialization. This AddAnchors subroutine is also critical to give a faster embedding when the number of anchors is large. Consider a case when 90% of the total sensors are anchors, in such a case, algorithm would most likely not enter the second step as all remaining sensors would be found in the AddAnchors step. The part after AddAnchors is same as in Agarwal et al., and we refer readers to the original paper [8] for more details.

We now describe the AddAnchors algorithm (Algorithm 3). It first considers points connected to at least $d + 2$ anchors. Since we know the location of the anchors, localizing a point with respect to anchors is a relatively easy problem, and only minimizing the distance with respect to the anchors should lead to a better solution. All such points that are connected to at least $d+2$ anchors are localized using place-center by only considering the distances from anchors; and when these points localized up to a user-defined tolerance $(\max(\sigma, \varepsilon_d))$, they are added to the set of the anchors. The process is repeated for N_{in} iterations.

A. MDS is Gradient Descent

We can show that the algorithm presented in [8] is exactly a gradient descent algorithm with learning rate $\lambda = 1/2n$. The cost function is for a point x_i is

$$C(x_i) = \sum_j \Big(\|x_i - x_j\| - D_{i,j}\Big)^2$$

Taking the gradient with respect to x_i

$$\frac{\partial C(x_i)}{\partial x_i} = \sum_j 2\Big(\|x_i - x_j\| - D_{i,j}\Big)\frac{\partial \|x_i - x_j\|}{\partial x_i}$$
$$= \sum_j 2\Big(1 - \frac{D_{i,j}}{\|x_i - x_j\|}\Big)(x_i - x_j).$$

The new location of x_i with learning rate $\lambda = \frac{1}{2n}$ is

$$x_i^* = x_i - \lambda \frac{\partial C(x_i)}{\partial x_i}$$
$$= x_i - \frac{1}{2n}\sum_j 2\Big(1 - \frac{D_{i,j}}{\|x_i - x_j\|}\Big)(x_i - x_j)$$
$$= \frac{1}{n}\sum_j \Big(x_j + (x_i - x_j)\frac{D_{i,j}}{\|x_i - x_j\|}\Big) = \frac{1}{n}\sum_j \hat{x}_j.$$

These $\hat{x}_j$ are interpolated points between x_j and x_i at the fractional distance $q = D_{i,j}/\|x_i - x_j\|$. Although this algorithm is simply the gradient descent with learning rate $1/2n$, notably *it always monotonically reduces the cost*, which is not true for general gradient descent algorithms.

IV. Experiments

We experiment with different sensor networks, and show the behavior of our algorithm PLACEMENT compared to the other algorithms. We only directly compare against two algorithms Krislock (2010) [7] and Pong and Tseng (2009) [5], since they have been shown to dominate other algorithms (e.g. [2], [6], [12]), and other popular algorithms (e.g. [15]) cannot handle anchor points.

- **Pong and Tseng (2009)** [5] - This is the best algorithm for dealing with noisy measurements.
- **Krislock (2010)** [7] - This algorithm is the most scalable, but does not handle noise well.

For all of the baselines (including cited dominated ones (e.g. [2], [6], [12], [15]) for which we do not report numbers), we asked authors to provide their code or downloaded publicly available code. We implemented our algorithm in C. All the runtimes reported in experiments are wall clock times. For baselines, all the parameters were taken to be the default values or whatever suggested by authors. For our algorithm, we chose the following values of the parameters. $\epsilon_x =$ 1e-7, $\epsilon_d =$ 1e-5; $N_{out} = 20, N_{in} = 10$ for $\sigma = 0$, otherwise $N_{out} = 40, N_{in} = 5$.

A. Data Generation

Following the previous work [12], [7], [5], [6], artificial data is generated by first generating n points uniformly at random in $[0, 1]^2$ box, and then selecting m points uniformly at random as anchors. Call these set of points as X_0. Here 0 denotes the time. To model moving points, we have them move in a random walk described by a Gaussian distribution on each step. Specifically, each step we generate the set X_t from X_{t-1} by $X_t = X_{t-1} + \eta\mathcal{N}(0, 1))$, where η is the perturbation factor and $\mathcal{N}(0, 1)$ is the Gaussian random variable with 0 mean and unit variance. Only sensors move, anchors locations remain fixed. For each set of points except X_0 which is used for initialization; edges are generated based on the radio-distance R. Two nodes are connected if their distance is less than radio-distance

$$D'_{ij} = \begin{cases} \|x_i - x_j\|_2, (i,j) \in E & \text{if } \|x_i - x_j\|_2 \le R \\ \text{unspecified} & \text{otherwise.} \end{cases}$$

Then we add Gaussian noise to each specified distance. For all $(i, j) \in E$ we set $D_{ij} = D'_{i,j}(1 + \sigma\mathcal{N}(0, 1))$, where σ is the noise factor. In the moving case, this σ-error is added to the measurement after each time step.

B. Evaluation

We evaluate the algorithms based on their ability to position sensors at their actual locations, (not based on the cost function). We measure two quantities, RMSE (Root Mean Square Error) and MaxErr (Maximum Error) defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n-m}\sum_{i=1}^{n-m}\|x_i - x_i^{true}\|^2}$$
$$\text{MaxErr} = \max_i \|x_i - x_i^{true}\|$$

We perform two types of experiments. First, we consider only one time step, where data is initialized at X_0, and after one step, we solve the SNL on X_1 (each sensor moves with $\eta\mathcal{N}(0, 1)$). Second, we consider the problem over multiple time steps, e.g. at $t = 1, 2, \ldots, 20$.

C. Results

In reporting results, there are five free parameters to generate a problem i.e., n, m, R, σ and η. We vary all five parameters, and for each set of parameters we generate six random problems, and present the mean. All of the above parameters control different properties of the network. n and m and R control the sparseness/density of the network. A higher value of R means more edges and hence a dense graph. σ controls the amount of noise in the distances while η controls the movement of sensors (only speed). All times are reported in seconds unless otherwise noted.

First set of results is reported in Table I. In this set of results, we fix the network structure i.e., value of m and R, but changes the amount of noise σ and the

Table I
RELATIVE COMPARISON OF ALL THREE ALGORITHMS FOR DIFFERENT VALUES OF n, σ AND η

n	m	R	σ	η	**PLACEMENT**			**Pong and Tseng (2009)**			**Krislock (2010)**		
					RMSE	MAXERR	Time	RMSE	MAXERR	Time	RMSE	MAXERR	Time
1000	20	0.06	0	0.01	2.29e-3	1.80e-2	0.54	7.67e-2	3.62e-1	12.75	8.44e-14	67e-13	0.87
1000	20	0.06	0	0.1	2.75e-2	1.39e-1	1.40	8.25e-2	3.73e-1	12.14	2.52e-13	4.20e-12	0.83
1000	20	0.06	0	0.2	4.10e-2	2.07e-1	1.49	7.18e-2	3.67e-1	10.44	1.66e-13	3.23e-12	0.87
1000	20	0.06	0.1	0.01	5.65e-3	3.60e-2	0.73	7.27e-2	3.49e-1	5.62	5.54e+1	1.08e+3	0.92
1000	20	0.06	0.1	0.1	3.09e-2	1.36e-1	1.93	8.63e-2	4.13e-1	6.07	7.95e+2	2.0e+4	1.10
1000	20	0.06	0.1	0.2	3.82e-2	1.59e-1	3.47	7.24e-2	3.42e-1	5.30	7.64e+3	9.5e+4	0.84
1000	20	0.06	0.3	0.01	1.85e-2	9.35e-2	1.80	7.86e-2	3.24e-1	4.40	7.29e+1	1.72e+3	1.44
1000	20	0.06	0.3	0.1	3.38e-2	1.39e-1	1.11	8.56e-2	3.35e-1	4.49	1.16e+2	1.66e+3	1.78
1000	20	0.06	0.3	0.2	4.58e-2	1.92e-1	2.54	7.20e-2	3.17e-1	4.24	9.05	1.18e+2	1.52
4000	20	0.035	0	0.01	1.90e-3	3.10e-2	2.31	5.5e-2	3.05e-2	133.09	1.98e-13	1.92e-12	3.27
4000	20	0.035	0	0.1	1.77e-2	9.63e-2	8.36	5.94e-2	3.17e-2	138.59	1.45e-13	9.7e-13	3.26
4000	20	0.035	0	0.2	2.96e-2	1.48e-1	8.19	7.51e-1	3.73e-1	153.85	1.47e-13	9.91e-13	3.16
4000	20	0.035	0.1	0.01	2.69e-3	2.24e-2	1.78	7.11e-2	3.68e-1	96.96	2.64e+1	8.54e+2	8.84
4000	20	0.035	0.1	0.1	1.75e-2	8.99e-2	15.89	5.9e-2	3.18e-2	85.31	5.88e+3	6.57e+4	6.50
4000	20	0.035	0.1	0.2	2.44e-2	1.24e-1	21.11	7.26e-2	3.45e-1	88.08	1.76e+1	3.12e+2	10.70
4000	20	0.035	0.3	0.01	5.50e-3	3.07e-2	0.82	7.23e-2	3.45e-1	60.62	2.90e+1	4.37e+2	10.79
4000	20	0.035	0.3	0.1	1.69e-2	1.05e-2	12.51	7.64e-2	3.46e-2	62.90	1.02e+2	2.25e+3	19.14
4000	20	0.035	0.3	0.2	2.62e-2	1.36e-1	21.12	9.28e-2	3.96e-1	64.24	1.28e+2	2.72e+3	33.18

Table II
RELATIVE COMPARISON WHEN THE NETWORK STRUCTURE IS CHANGED, FOR DIFFERENT m AND R

n	m	R	σ	η	**PLACEMENT**			**Pong and Tseng (2009)**		
					RMSE	MAXERR	Time	RMSE	MAXERR	Time
1000	5	0.02	0.1	0.1	1.18e-1	3.51e-1	0.03	1.12e-1	3.39e-1	0.50
1000	5	0.06	0.1	0.1	2.77e-2	1.55e-1	1.55	1.96e-1	5.45e-1	17.69
1000	5	0.15	0.1	0.1	6.04e-3	1.55-2	18.59	1.26e-1	5.86e-1	9.43
1000	20	0.02	0.1	0.1	1.17e-1	3.56e-1	0.03	1.11e-1	3.67e-1	0.64
1000	20	0.06	0.1	0.1	3.39e-2	1.36e-1	2.50	8.20e-2	3.49e-1	5.79
1000	20	0.15	0.1	0.1	1.74e-2	1.32e-1	12.74	5.60e-1	3.00e-1	1.93
1000	50	0.02	0.1	0.1	1.15e-2	3.47e-1	0.04	1.09e-2	3.00e-1	1.93
1000	50	0.06	0.1	0.1	2.94e-2	1.44e-1	3.03	3.26e-1	2.39e-1	1.9
1000	50	0.15	0.1	0.1	3.48e-3	1.18e-2	7.80	1.49e-2	1.18e-1	1.21
1000	200	0.02	0.1	0.1	1.09e-1	3.45e-1	0.05	1.03e-1	3.45e-1	0.35
1000	200	0.06	0.1	0.1	1.45e-2	9.24e-2	1.32	1.43e-2	1.26e-1	0.56
1000	200	0.15	0.1	0.1	2.91e-3	1.03e-2	0.52	2.89e-3	1.00e-2	1.05
4000	5	0.02	0.1	0.1	4.28e-2	2.74e-1	3.38	2.61e-1	6.23e-1	286.11
4000	5	0.1	0.1	0.1	4.94e-4	1.31e-2	64.55	1.23e-1	5.03e-1	267.35
4000	20	0.02	0.1	0.1	4.19e-2	2.85e-1	3.85	1.12e-1	3.52e-1	78.08
4000	20	0.1	0.1	0.1	7.20e-2	2.45e-1	175.18	2.36e-2	2.64e-1	48.99
4000	50	0.02	0.1	0.1	4.09e-2	2.58e-1	4.26	8.60e-2	3.39e-1	52.04
4000	50	0.1	0.1	0.1	7.59e-3	1.04e-1	100.69	1.51e-3	5.55e-3	16.11
4000	200	0.02	0.1	0.1	3.70e-2	2.89e-1	4.03	4.12e-2	2.89e-1	12.27
4000	200	0.1	0.1	0.1	1.66e-3	7.41e-3	12.43	1.41e-3	5.98e-3	9.69

amount of perturbation η. We repeat these experiments for different values of $n = 1000$ and 4000. From this table, first observation we make is that Krislock (2010) performs poorly when there is noise in the data. This algorithm is extremely good for non-noisy data, and can give almost perfect position for sensors, however if one only cares for the near-perfect position (of the order of $1e^{-2}$), PLACEMENT is nearly as fast. But, since Krislock (2010) performs poorly for the noisy data (large σ), Pong and Tseng (2009) will be our main baseline for the remaining experiments.

On comparing Pong and Tseng (2009) with PLACEMENT, we observe that PLACEMENT outperforms Pong and Tseng (2009) in all cases both in terms speed and accuracy. In some cases (e.g., non-noisy cases) the difference in time is significant. Note that time taken by Pong and Tseng (2009) for $n = 4000$ even with $\eta = 0.2$ perturbation is 153 seconds while PLACEMENT can find a *better* embedding in only 8 seconds. This time difference between Pong and Tseng (2009) and PLACEMENT is consistent for other cases as well. We emphasize here that both algorithms PLACEMENT and Pong and Tseng (2009) require an initial estimation of the locations but Pong and Tseng (2009) is not able to exploit the good estimate of the initial locations as much as PLACEMENT. Notice that when there is small perturbation $\eta = 0.01$, PLACEMENT is significantly faster than Pong and Tseng (2009)—for $n = 4000$, $\sigma = 0$, $\eta = 0.01$ Pong and Tseng (2009) takes 133 seconds while PLACEMENT only takes 2.31 seconds, while providing the better embedding both in terms of RMSE and MAXERR.

In our second set of results, we study the behavior of different algorithms as the network properties are varied, specifically the number of anchors m and the radio distance R. We fix the noise level σ at 0.1 and perturbation η at 0.1, so as not to bias any algorithm. Note from the previous table that our algorithm will perform very well for small perturbation $\eta = 0.01$ and for small noise, hence we do not keep σ and η to be too small to favor PLACEMENT. The results of this set of experiments are reported in Table II. Due to the poor performance of Krislock (2010) on noisy data, we exclude it from the table. From this set of experiments, we observe that PLACEMENT has a better performance than Pong and Tseng (2009) when the network is sparse, i.e., when R and m both are small. As we increase R, the performance of PLACEMENT still remains better as long as m is small but when we increase both, Pong and Tseng (2009) starts to catch up. This is mainly because of the relatively poor performance of Pong and Tseng (2009) for small m. Note that all the experiments reported in [5] considered m at 10% of n. Here we keep m to be much smaller, we believe this to be a more practical scenario. When both m and R are large (e.g. $m = 200$, $R = 0.15$), there is not much difference in the performance of Pong and Tseng (2009) and PLACEMENT because, for large m, subroutine AddAnchors finds more anchors making the problem simpler, and thus PLACEMENT runs faster.

Table IV
RESULTS ON LARGE SCALE NOISY DATA ($m = 20$).

n	R	σ	η	PLACEMENT		
				RMSE	MAXERR	Time
20000	0.018	0.02	0.01	1.5e-3	2.1e-2	14
20000	0.018	0.02	0.1	1.1e-2	6.9e-2	79
20000	0.018	0.1	0.01	1.8e-3	2.4e-2	13
20000	0.018	0.1	0.1	1.1e-2	6.4e-2	87
40000	0.015	0.02	0.01	1.2e-3	1.7e-2	46
40000	0.015	0.02	0.1	8.6e-3	5.6e-2	316
40000	0.015	0.1	0.01	1.3e-3	1.8e-3	50
40000	0.015	0.1	0.1	8.4e-3	5.1e-2	305
80000	0.010	0.02	0.01	1.4e-3	1.6e-2	123
80000	0.010	0.02	0.1	8.2e-3	4.7e-2	699
80000	0.010	0.1	0.01	1.4e-3	1.6e-2	125
80000	0.010	0.1	0.1	8.3e-3	4.4e-2	729
120000	0.008	0.02	0.01	1.4e-3	1.5e-2	197
120000	0.008	0.02	0.1	8.0e-3	4.8e-2	1043
120000	0.008	0.1	0.01	1.4e-3	1.3e-3	202
120000	0.008	0.1	0.1	8.0e-3	4.3e-2	1085

In certain instances, it is not clear which algorithm performs best, mainly because our algorithm is an iterative algorithm, and one can increase/decrease the time by running it for more/less iterations. For example, $n = 1000, m = 50, R = 0.15$ or $n = 1000, m = 5, R = 0.15$, although PLACEMENT takes more time, but it also gives better accuracy. One can reduce the time by compromising on the accuracy.

D. Large Scale Experiments

We also perform experiments on much larger problems in the noisy setting. To the best of our knowledge, there has not been any work on this type of problem at this scale for noisy data. So for the large scale, there is no baseline algorithm. But still to see the behavior of both baselines for noisy sparse problem, we run them for $n = 10,000$ and results are reported in Table III. From this table, it is clear that PLACEMENT not only runs significantly faster than both Pong and Tseng (2009) and Krislock (2010) but also gives better

Table III
RELATIVE PERFORMANCE ON LARGE SCALE NOISY DATA. "-" INDICATES THAT JOB DID NOT COMPLETE WITHIN 24 HOURS.

n	m	R	σ	η	**PLACEMENT**			**Pong and Tseng (2009)**			**Krislock (2010)**		
					RMSE	MAXERR	Time	RMSE	MAXERR	Time	RMSE	MAXERR	Time
10000	20	0.02	0.02	0.01	2.6e-3	2.9e-2	5	6.0e-2	2.6e-1	598	1.1e+2	1.9e+3	39
10000	20	0.02	0.02	0.1	1.5e-2	8.1e-2	31	6.7e-2	3.0e-1	582	2.5e+3	1.8e+5	33
10000	20	0.02	0.1	0.01	2.9e-3	3.0e-2	5	7.1e-2	3.0e-1	592	3.6e+2	2.2e+4	126
10000	20	0.02	0.1	0.1	1.5e-2	8.4e-2	28	6.43e-2	2.67e-1	858	-	-	-

accuracy for both RMSE and MaxErr. Results for even larger problems, $n = 20,000$ to $n = 120,000$ are presented in Table IV. We can handle the sensors up to $n = 120,000$ in less than 20 minutes, with a reasonable accuracy.

E. Moving Problem

We now consider experiments on moving data over several (up to $t = 20$) time steps. Algorithms only take the *initial position* X_0 as input, and in the successive steps, the initial position is the position returned from the previous step (estimated location from $(t-1)$ step).

So far we have only considered the speed of the movement η not the pattern. When sensors are moving, there can be many moving patterns e.g., moving in a herd, moving in only one direction, disperse moving etc., and an algorithm's behavior might change according to the pattern. In our work, we experiment with following two moving patterns:

Moving-within-boundary: in this pattern, sensors move from their initial location by a small amount η, but at any given time all sensors remain inside a fixed boundary. This pattern would imitate if animals are left in a big fenced in ranch (note that animals are randomly distributed in the ranch), and then they start moving. They are not allowed to go out of the ranch.

Moving-without-boundary: this pattern is same as above except, now that there is no notion of boundary and animals are allowed to move freely. Since the size of the area sensors are located in is increasing with the time (and so are the distances), error would also increase (as seen in Figure 1) .

Results for the pattern moving-without-boundary is shown in Figure 1. For these experiments, we fix $n = 1000$ and $m = 5$, but change other parameters σ and η. Each subfigure in the figure is captioned with the average time (in seconds) taken in all 20 time steps, in the order of Pong and Tseng (2009) ($\eta = 0.01$), PLACEMENT ($\eta = 0.01$), Pong and Tseng (2009) ($\eta = 0.05$), PLACEMENT ($\eta = 0.05$). Each subfigure has four RMSE plots, two for $\eta = 0.01$ (red lines) and two for $\eta = 0.05$ (blue lines). From this figure, it is clear that PLACEMENT outperforms Pong and Tseng (2009) in terms of both speed and accuracy in all cases. This improvement is even more significant if one considers the time taken by both algorithms. As mentioned above, since we are dealing with the ground with no boundary, as we advance in time, inter-sensor distances increase and so the embedding error, but the relative improvement of PLACEMENT over Pong and Tseng (2009) remains consistent. Note that when $R = 0.06$, $\sigma = 0.01$, $\eta = 0.05$, $m = 5$ (Figure 1(a)), Pong and Tseng (2009) takes 19.1 seconds while PLACEMENT only takes 0.2 seconds. We remind readers that fast response time is important if one were to do a real-time tracking.

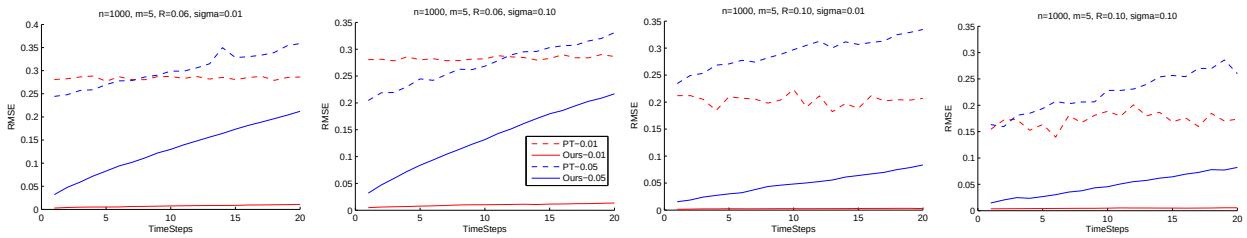
Results for the pattern moving-within-boundary is shown in Figure 2. In these cases as well, PLACEMENT consistently outperforms Pong and Tseng (2009) both in terms of speed and accuracy. Once again, compare the time for the case $m = 5$, $R = 0.06$, $\eta = 0.05$, $\sigma = 0.01$ (Figure 2(a)), Pong and Tseng (2009) takes 25.6 seconds while PLACEMENT only takes 0.9 seconds.

V. CONCLUSION AND DISCUSSION

We have presented an iterative scalable MDS-based algorithm for sensor network localization that can handle the noisy data more effectively than the previous scalable algorithms, and is guaranteed to reduce the cost function in every iteration. It can embed sensors up to $120,000$ in less than 20 minutes. Our algorithm requires and exploits an initial estimate of the sensor locations. This makes the algorithm especially effective for mobile sensor network location problems. Furthermore, our algorithm is naturally distributed, thus potentially avoiding large communication overhead associated with centralized algorithms.

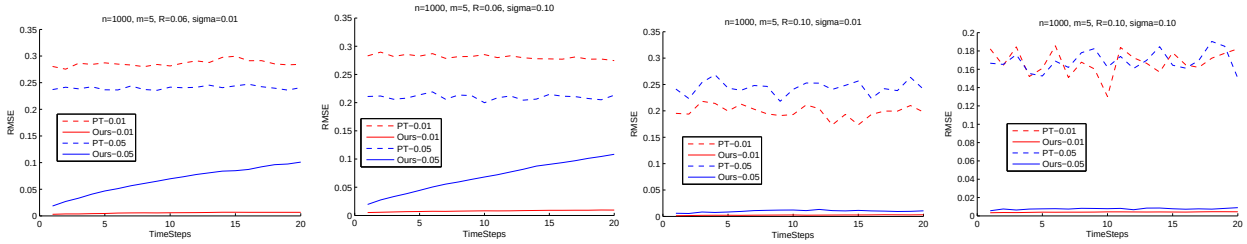
ACKNOWLEDGMENT

Suresh Venkatasubramanian is partially supported by NSF grants CCF-0953066 and CCF-1115677. Hal Daumé III is partially supported by NSF grants IIS-1117716 and IIS-1153487.



(a) $26.1, 0.3, 19.1, 0.2$ (b) $18.0, 0.2, 10.6, 0.2$ (c) $13.6, 0.7, 18.7, 2.7$ (d) $10.6, 0.4, 9.1, 2.9$

Figure 1. RMSE versus time steps for $m = 5$ for moving pattern moving-without-boundary



(a) $26.5, 0.4, 25.6, 0.9$ (b) $18.8, 0.2, 13.8, 0.7$ (c) $14.3, 0.7, 19.4, 5.8$ (d) $11.1, 0.4, 9.9, 6.0$

Figure 2. RMSE versus time steps for $m = 5$ for moving pattern moving-within-boundary

REFERENCES

[1] P. Biswas and Y. Ye, "Semidefinite programming for ad hoc wireless sensor network localization," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*. New York, NY, USA: ACM, 2004, pp. 46–54.

[2] P. Biswas, T.-C. Liang, K.-C. Toh, , Y. Ye, and T.-C. Wang, "Semidefinite programming approaches for sensor network localization with noisy distance measurements," *IEEE Transactions on Automation Science and Engineering*, vol. 3, pp. 360–371, 2006.

[3] M. W. Carter, H. H. Jin, M. A. Saunders, and Y. Ye, "Spaseloc: An adaptive subproblem algorithm for scalable wireless sensor network localization," *SIAM J. on Optimization*, vol. 17, no. 4, pp. 1102–1128, Dec. 2006.

[4] P. Tseng, "Second-order cone programming relaxation of sensor network localization," *SIAM J. on Optimization*, vol. 18, no. 1, pp. 156–185, 2007.

[5] T. Pong and P. Tseng, "Robust edge-based semidefinite programming relaxation of sensor network localization," U. of Washington, Tech. Rep., 2009.

[6] S. Kim, M. Kojima, and H. Waki, "Exploiting sparsity in sdp relaxation for sensor network localization," *SIAM J. on Optimization*, vol. 20, no. 1, pp. 192–215, Apr. 2009.

[7] N. Krislock, "Semidefinite facial reduction for low-rank euclidean distance matrix completion," Ph.D. dissertation, University of Waterloo, 2010.

[8] A. Agarwal, J. M. Phillips, and S. Venkatasubramanian, "Universal multi-dimensional scaling," in *KDD*, 2010.

[9] P. Biswas, "Semidefinite Programming Approaches to Distance Geometry Problems," Ph.D. dissertation, Department of Electrical Engineering, Stanford University., 2007.

[10] P. Biswas and Y. Ye, "A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization," *Multiscale optimization methods and applications*, pp. 69–84, 2006.

[11] P. Biswas, K.-C. Toh, and Y. Ye, "A distributed SDP approach for large-scale noisy anchor-free graph reailzation with applications to molecular conformation," *SIAM J. Sci. Comput.*, vol. 30, no. 3, pp. 1251–1277, 2008.

[12] Z. Wang, S. Zheng, Y. Ye, and S. Boyd, "Further relaxations of the semidefinite programming approach to sensor network localization," *SIAM J. Optim.*, pp. 655–673, 2008.

[13] J. Costa, N. Patwari, and A. Hero III, "Distributed weighted-multidimensional scaling for node localization in sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 1, pp. 39–64, 2006.

[14] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust distributed network localization with noisy range measurements," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 50–61.

[15] H. Fang and D. P. OLeary, "Euclidean distance matrix completion problems, year = 2010."