SPASELOC: AN ADAPTIVE SUBPROBLEM ALGORITHM FOR SCALABLE WIRELESS SENSOR NETWORK LOCALIZATION

MICHAEL W. CARTER[*], HOLLY H. JIN[†], MICHAEL A. SAUNDERS[‡], AND YINYU YE[§]

Abstract. An adaptive rule-based algorithm, SpaseLoc, is described to solve localization problems for ad hoc wireless sensor networks. A large problem is solved as a sequence of very small subproblems, each of which is solved by semidefinite programming relaxation of a geometric optimization model. The subproblems are generated according to a set of sensor/anchor selection rules and a priority list. Computational results compared with existing approaches show that the SpaseLoc algorithm scales well and provides excellent positioning accuracy.

Key words. sensor localization, semidefinite programming, large-scale optimization

AMS subject classifications. 49M37, 65K05, 90C30

1. Introduction.

Ad hoc wireless sensor networks may contain hundreds or even tens of thousands of inexpensive devices (sensors) that can communicate with their neighbors within a limited radio range. By relaying information to each other, they can transmit signals to a command post anywhere within the network. They have many practical uses in areas such as military applications [14], environment or industrial control and monitoring [6, 8], wildlife monitoring [23], and security monitoring [14]. For example, Southern California Edison's Nuclear Generating Station in San Onofre, California has deployed wireless mesh networked sensors from Dust Networks to obtain real-time trend data [8]. These data are used to predict which motors are about to fail, so they could be preemptively rebuilt or replaced during scheduled maintenance periods. The use of a wireless sensor network saves the station money and avoids potential machine shutdown. Implementation of a sensor localization algorithm would provide a service that eliminates the need to record every sensor's location and its associated ID number in the network.

Wireless sensor networks are potentially important enablers for many other advanced applications. A huge variety of applications lie ahead. By 2008, there could be 100 million wireless sensors in use, up from about 200,000 in 2005, according to the market-research company Harbor Research. The worldwide market for wireless sensors, it says, will grow from $100 million in 2005 to more than $1 billion by 2009 [17]. This is motivating great effort in academia and industry to explore effective ways to build sensor networks with feature-rich services [11].

One of the important inputs these services build upon is the exact locations of all sensors in the network. The need for *sensor localization* arises because accurate positions are known for only some of the sensors (which are called *anchors*). If the networks are to achieve their purpose, the positions of the remaining sensors must be determined. One approach to localizing these sensors with unknown positions is to use known anchor locations and distance measurements that neighboring sensors and

[*]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada M5S 3G8 (carter@mie.utoronto.ca).

[†]Department of Management Science and Engineering, Stanford University, Stanford, CA 94305-4026 (hollyjin@stanford.edu), and Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada M5S 3G8. Partially supported by Robert Bosch Corporation.

[‡]Department of Management Science and Engineering, Stanford University, Stanford, CA 94305-4026 (saunders@stanford.edu).

[§]Department of Management Science and Engineering, Stanford University, Stanford, CA 94305-4026 (yinyu-ye@stanford.edu). Submitted to SIOPT, Dec 27, 2004. Revised November 18, 2005.

$$\underbrace{1 \quad \le \quad i \quad < \quad j \quad \le \quad s}_{s \text{ sensors}} \Big| \underbrace{s{+}1 \quad \le \quad k \quad \le \quad n}_{m \text{ anchors}}$$

FIG. 1.1. *Indexing of sensors and anchors.*

anchors obtain among themselves. The mathematical problem is to *estimate* sensor positions using a sparse data matrix of noisy distance measurements. This leads to a large, non-convex, constrained optimization problem. Large networks may contain many thousands of sensors, whose locations should be determined accurately and quickly.

1.1. Problem definition.

Sensor localization in ad hoc wireless sensor network aims to find the locations of all sensors in the network, given pair-wise distance measurements among some of the sensors, and known locations of some of the sensors. The sensors with known locations are called *anchors*. From now on, *sensor* generally means *unpositioned sensor*, excluding anchors. A *node* is any sensor or anchor in the network.

We use a constrained optimization approach to estimate the sensors' locations. The following input, output, and objectives are considered.

Input

Total points: n, the total number of nodes in the network.

Unknown points: s sensors, whose locations $x_i \in \mathcal{R}^2$, $i = 1, \ldots, s$ are to be determined. (We assume the points are on a plane here, but the approach is extended to three dimensions in Jin's thesis [13].)

Known points: m anchors, whose locations $a_k \in \mathcal{R}^2$, $k = s+1, \ldots, n$ are known. (Note that we put anchors at the end of the total points' list without loss of generality, and that $n = s + m$. Index k is specific for indexing anchors. Refer to Figure 1.1 for nodes indexing.)

Known distance measurements: The readings of certain ranging devices for estimating the distance between two points. $\widehat{d}_{ij}$ is the distance measurement between two sensors x_i and x_j $(i < j \le s)$, and $\widehat{d}_{ik}$ is the distance measurement between some sensor x_i and anchor a_k $(i \le s < k)$. The distance measurements are constant data and generally have errors.

Output

Locations: Estimated locations x_i for s sensors.

Objectives

Accuracy: Minimal errors in the estimated sensor positions.

Speed: Fast enough for real-time applications (e.g., networks with moving sensors).

Scalability: Suitable for large-scale deployment (with tens of thousands of nodes).

1.2. Notation.

The Euclidean distance between two vectors v and w is defined to be $\|v - w\|$, where $\| \cdot \|$ always means the 2-norm. Nodes are said to be *connected* if the associated measurements $\widehat{d}_{ij}$ or $\widehat{d}_{ik}$ exist. The remaining elements of $\widehat{d}$ are zero. If a measurement does exist between node i and j but it is zero (i and j are at the same spot), we do not set $\widehat{d}_{ij}$ to zero: we set it to machine precision ϵ instead to distinguish from the case of $\widehat{d}_{ij} = 0$ when two nodes' distance is beyond the sensor device's measuring range.

1.3. Related research work. Sensor localization in ad hoc wireless network has been a booming research area recently. Hightower and Boriello [11] give an extensive review of the area and available methods. There are many ways to solve the localization problem [5, 7, 9, 12, 16, 18, 19, 20, 21], with two main ones based on triangulation and optimization.

Triangulation methods estimate node positions based on distance measurements between neighboring nodes, and some algorithms use iterative steps to localize all sensors.

Early work using optimization techniques is reported by Doherty et al. [7]. Ideally the Euclidean distance between neighboring nodes should be fitted in some near-equality sense to the distance measurements:

$$\|x_i - x_j\| \approx \widehat{d}_{ij} \quad \text{and} \quad \|x_i - a_k\| \approx \widehat{d}_{ik}. \tag{1.1}$$

Doherty et al. formulate a convex optimization model by treating the constraints as $\|x_i - x_j\| \le \widehat{d}_{ij}$ and $\|x_i - a_k\| \le \widehat{d}_{ik}$, and by including certain other convex constraints. This formulation takes advantage of available optimization algorithms, including those for convex optimization. However, the method needs sufficient anchors to be positioned on the boundary of the localization area for it to work effectively.

Biswas and Ye [2] work with the near-equality constraints (1.1), and most importantly they introduced a semidefinite programming (SDP) relaxation method in order to retain the benefits of convex optimization. They report that their method yields more accuracy under all conditions than the approach in [7].

The SDP relaxation approach can solve small problems effectively. The paper reports a few seconds of laptop execution time for a 50-node localization problem. However, the number of constraints in the SDP model is $O(n^2)$, where n is the number of nodes in the network. Even a few hundred-node problem leads to excessive memory and computation time by available SDP solvers such as DSDP (Benson, Ye, and Zhang [1]) and SeDuMi (Sturm [22]). These solvers are effective for SDP problems with dimension and number of constraints up to a few thousand.

Tseng [24] has presented a second-order cone programming (SOCP) relaxation model that permits solution for problem sizes up to a few thousand using available SOCP solvers. However, the additional relaxation of the original model usually generates larger error rates, and the run-times are high. The author reports CPU times of 330 seconds for 1000 nodes and 3 hours for 2000 nodes using SeDuMi 1.05 [22] and MATLAB 6.1 on a Linux PC.

Biswas and Ye [3] propose a decomposition scheme to overcome the scalability issue with SDP solvers. The anchors in the network are first partitioned into many clusters according to their physical positions, and sensors are assigned to these clusters if they have a direct connection to one of the anchors. Each cluster formulates a subproblem, and the subproblems are solved *independently* on each cluster using the SDP relaxation of [2]. The paper reports results for randomly generated sensor networks of 4000 sensors partitioned into 100 clusters strictly according to their geographic locations. Sensors with distance connections to more than one cluster are included in multiple clusters. The final estimation of their locations is determined by the cluster that gives the least estimated errors. An execution time of about 4 minutes on a 1.2GHz Pentium laptop is reported for this sized problem. The time could be reduced by using multiple CPUs.

Although Biswas and Ye [3] make large-scale sensor network localization possible by decomposing the large-scale problem geographically, there are shortfalls in this approach that may prevent its large-scale deployment. First of all, for any real deployment of a wireless sensor network, localization algorithms should run in real-time, where minutes could be too long and multiple CPUs could be too expensive. Secondly, since the partition is strictly based on geographic locations, sensors near the border lines of a cluster may not be positioned as accurately as they would be using other approaches. This is due to the fact that each cluster may include only partial connection information for the bordering sensors if the bordering sensors have connections with multiple clusters. Furthermore, the SDP relaxation approach that their decomposition method is based on provides poor accuracy on certain topologies with low anchor density or small radio range for even medium-size networks (refer to section 4.2).

1.4. SpaseLoc. A basic tool that we have developed during this research is a rule-based iterative algorithm named SpaseLoc (sub-problem algorithm for sensor localization). It is effective for networks involving tens of thousands of sensors and beyond.

To solve a large localization problem (defined as the *full_problem*), SpaseLoc proceeds iteratively by estimating only a portion of the total sensors' locations at each iteration. Some anchors and sensors are chosen according to a set of rules. They form a sensor localization *subproblem* that can be treated similarly to the basic SDP formulation of Biswas and Ye [2]. The solution from the subproblem is fed back to the *full_problem* and the algorithm iterates again until all sensors are localized.

Computational results show that SpaseLoc can solve small or large problems with excellent accuracy and scalability. It is capable of localizing 4000 nodes with great accuracy in under 20 seconds, and 10000 nodes in about a minute on a 2.4GHz laptop.

2. The subproblem SDP model.

This section reviews the quadratic programming formulation of the sensor localization problem and the SDP relaxation model of Biswas and Ye [2] that the SpaseLoc subproblem is based on. Error analysis is also reviewed here as reference for later sections.

2.1. Euclidean distance model. Consider a network of sensors and anchors labeled as in Figure 1.1. For any point in the network, there could be three types of distance measurements. Since we generally do not need the distance information between two anchor points, we exclude this type of measurement from now on.

The other types of distance measurements are the two we need for the localization model. First is the distance measurement between two sensors (i and j) with unknown positions; second is the distance measurement between a sensor (i) and an anchor (k) with known position. Corresponding to these two types of distances, we define sets N_1, $\overline{N}_1$, N_2 and $\overline{N}_2$ as follows:

- N_1 includes pairwise sensors (i,j) if $i<j$ and there exists a distance measurement $\widehat{d}_{ij}$:
$$N_1 = \{(i,j) \text{ with known } \widehat{d}_{ij} \text{ and } i<j\}.$$
- $\overline{N}_1$ includes pairwise sensors (i,j) with unknown measurement $\widehat{d}_{ij}$ and $i<j$:
$$\overline{N}_1 = \{(i,j) \text{ with unknown } \widehat{d}_{ij} \text{ and } i<j\}.$$
- N_2 includes pairs of sensor i and anchor k if there exists a measurement $\widehat{d}_{ik}$:
$$N_2 = \{(i,k) \text{ with known } \widehat{d}_{ik}\}.$$
- $\overline{N}_2$ includes pairs of sensor i and anchor k with unknown measurement $\widehat{d}_{ik}$:
$$\overline{N}_2 = \{(i,k) \text{ with unknown } \widehat{d}_{ik}\}.$$

The full set of nodes and pair-wise distance measurements form a graph $G = \{V, E\}$, where $V = \{1, 2, \dots, s, s+1, \dots, n\}$ and $E = N_1 \cup N_2$.

Introduce α_{ij} to be the difference between the measured squared distance $(\widehat{d}_{ij})^2$ and the squared Euclidean distance $\|x_i - x_j\|^2$ from sensor i to sensor j. Also, let α_{ik} be the difference between the measured squared distance $(\widehat{d}_{ik})^2$ and the squared Euclidean distance $\|x_i - a_k\|^2$ from sensor i to anchor k. Intuitively, we seek a solution for which the magnitude of these differences is small.

Lower bounds r_{ij} or r_{ik} are imposed if $(i,j) \in \overline{N}_1$ or if $(i,k) \in \overline{N}_2$. Typically each r_{ij} or r_{ik} value is the radio range (also known as *radius*) within which the associated sensors can detect each other.

Biswas and Ye [2] formulate the sensor localization problem as minimizing the ℓ_1 norm of the squared-distance errors α_{ij} and α_{ik} subject to mixed equality and inequality constraints:

$$
\begin{array}{llcll}
\underset{x_i, x_j, \alpha_{ij}, \alpha_{ik}}{\text{minimize}} & \displaystyle\sum_{(i,j)\in N_1} |\alpha_{ij}| \; + \sum_{(i,k)\in N_2} |\alpha_{ik}| & & & \\
\text{subject to} & \|x_i - x_j\|^2 - \alpha_{ij} & = & (\widehat{d}_{ij})^2, & \forall\,(i,j) \in N_1, \\
& \|x_i - a_k\|^2 - \alpha_{ik} & = & (\widehat{d}_{ik})^2, & \forall\,(i,k) \in N_2, \\
& \|x_i - x_j\|^2 & \geq & r_{ij}^2, & \forall\,(i,j) \in \overline{N}_1, \\
& \|x_i - a_k\|^2 & \geq & r_{ik}^2, & \forall\,(i,k) \in \overline{N}_2, \\
& x_i,\ x_j \in \mathcal{R}^2, & & \alpha_{ij},\ \alpha_{ik} \in \mathcal{R}, & \\
& i,j = 1, \dots, s, & & k = s+1, \dots, n. &
\end{array}
\tag{2.1}
$$

The above model is a non-convex constrained optimization problem. As yet there is no effective solution method. In the following subsections, we review Biswas and Ye's [2] relaxation method for solving this problem approximately.

2.2. The Euclidean distance model in matrix form. The distance model (2.1) is reformulated into (2.2) (refer to Biswas and Ye [2]) by introducing matrix variables as follows:

$$
\begin{array}{lrcll}
\text{minimize} & \displaystyle\sum_{(i,j)\in N_1} (\alpha_{ij}^+ + \alpha_{ij}^-) \; + \sum_{(i,k)\in N_2} (\alpha_{ik}^+ + \alpha_{ik}^-) & & & \\
\text{subject to} & e_{ij}^T\, Y\, e_{ij} - \alpha_{ij}^+ + \alpha_{ij}^- & = & (\widehat{d}_{ij})^2, & \forall\,(i,j) \in N_1, \\
& \begin{pmatrix} e_i \\ -a_k \end{pmatrix}^T \begin{pmatrix} Y & X^T \\ X & I \end{pmatrix} \begin{pmatrix} e_i \\ -a_k \end{pmatrix} - \alpha_{ik}^+ + \alpha_{ik}^- & = & (\widehat{d}_{ik})^2, & \forall\,(i,k) \in N_2, \\
& e_{ij}^T\, Y\, e_{ij} & \geq & r_{ij}^2, & \forall\,(i,j) \in \overline{N}_1, \\
& \begin{pmatrix} e_i \\ -a_k \end{pmatrix}^T \begin{pmatrix} Y & X^T \\ X & I \end{pmatrix} \begin{pmatrix} e_i \\ -a_k \end{pmatrix} & \geq & r_{ik}^2, & \forall\,(i,k) \in \overline{N}_2, \\
& Y & = & X^T X, & \\
& \alpha_{ij}^+,\ \alpha_{ij}^-,\ \alpha_{ik}^+,\ \alpha_{ik}^- & \geq & 0, & \\
& i,j = 1, \dots, s, & & k = s+1, \dots, n, &
\end{array}
\tag{2.2}
$$

where

- $X = (x_1\ x_2\ \dots\ x_s)$ is a $2 \times s$ matrix to be determined;
- e_{ij} is a zero column vector except for 1 in position i and -1 in position j, so that

$$\|x_i - x_j\|^2 \;=\; e_{ij}^T\ X^T X\ e_{ij};$$

- e_i is a zero column vector except for 1 in position i, so that

$$\|x_i - a_k\|^2 \;=\; \begin{pmatrix} e_i \\ -a_k \end{pmatrix}^T \begin{pmatrix} X & I \end{pmatrix}^T \begin{pmatrix} X & I \end{pmatrix} \begin{pmatrix} e_i \\ -a_k \end{pmatrix};$$

- Y is defined to be $X^T X$;
- The substitutions $\alpha_{ij} = \alpha_{ij}^+ - \alpha_{ij}^-$ and $\alpha_{ik} = \alpha_{ik}^+ - \alpha_{ik}^-$ are made to deal with $|\alpha_{ij}|$ and $|\alpha_{ik}|$ in the normal way.

2.3. The SDP relaxation model. The approach of Biswas and Ye [2] is to relax the constraint $Y = X^T X$ to be $Y \succeq X^T X$, for which an equivalent matrix inequality is (Boyd et al. [4])

$$Z_I \equiv \begin{pmatrix} Y & X^T \\ X & I \end{pmatrix} \succeq 0. \tag{2.3}$$

With the definitions

$$A_I \;=\; \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \qquad b_I \;=\; \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix},$$

where $\mathbf{0}$ in A_I is a zero column vector of dimension s, problem (2.2) is relaxed to a linear SDP:

$$\begin{array}{lrcll}
\text{minimize} & \multicolumn{4}{l}{\displaystyle\sum_{(i,j)\in N_1} (\alpha_{ij}^+ + \alpha_{ij}^-) \;+ \sum_{(i,k)\in N_2} (\alpha_{ik}^+ + \alpha_{ik}^-)} \\
\text{subject to} & \operatorname{diag}(A_I^T\ Z\ A_I) & = & b_I, & \\
& \begin{pmatrix} e_{ij} \\ \mathbf{0} \end{pmatrix}^T Z \begin{pmatrix} e_{ij} \\ \mathbf{0} \end{pmatrix} - \alpha_{ij}^+ + \alpha_{ij}^- & = & (\widehat{d}_{ij})^2 & \forall\ (i,j) \in N_1, \\
& \begin{pmatrix} e_i \\ -a_k \end{pmatrix}^T Z \begin{pmatrix} e_i \\ -a_k \end{pmatrix} - \alpha_{ik}^+ + \alpha_{ik}^- & = & (\widehat{d}_{ik})^2 & \forall\ (i,k) \in N_2, \\
& \begin{pmatrix} e_{ij} \\ \mathbf{0} \end{pmatrix}^T Z \begin{pmatrix} e_{ij} \\ \mathbf{0} \end{pmatrix} & \ge & r_{ij}^2 & \forall\ (i,j) \in \overline{N}_1, \\
& \begin{pmatrix} e_i \\ -a_k \end{pmatrix}^T Z \begin{pmatrix} e_i \\ -a_k \end{pmatrix} & \ge & r_{ik}^2 & \forall\ (i,k) \in \overline{N}_2, \\
& Z \succeq 0, \qquad \alpha_{ij}^+,\ \alpha_{ij}^-,\ \alpha_{ik}^+,\ \alpha_{ik}^- & \ge & 0, & \\
& i,j = 1,\dots,s, & & k = s+1,\dots,n, &
\end{array} \tag{2.4}$$

where the constraint $\mathrm{diag}(A_I^T Z A_I) = b_I$ ensures that the matrix variable Z's lower right corner is a 2-dimensional identity matrix I, so that Z takes the form of Z_I in (2.3).

Initially, Biswas and Ye [3, 2] omit the $\geq$ inequalities involving r_{ij} and r_{ik}, and solve the resulting problem to obtain an initial solution Z_1. (The inequality constraints increase the problem size dramatically, and Z_1 is likely to satisfy most of them.) They then adopt an "iterative active-constraint generation technique" in which inequalities violated by Z_k are added to the problem and the resulting SDP is solved to give Z_{k+1} ($k = 1, 2, \ldots$). The process usually terminates before all constraints are included. Further study of this approach is presented in section 4.1.

2.4. SDP model analysis. Let $\bar{Z} = \begin{pmatrix} \bar{Y} & \bar{X}^T \\ \bar{X} & I \end{pmatrix}$ be a feasible solution of the relaxed SDP (2.4). Biswas and Ye [2] give conditions under which $\bar{X}$ and $\bar{Y}$ solve problem (2.2) exactly, when exact distance measurements are assumed:

- $\bar{Z}$ is the unique optimal solution of (2.4), including all inequality constraints.
- In (2.4), there are $2n + n(n+1)/2$ exact pair-wise distance measurements.

These conditions ensure that $\bar{Y} = \bar{X}^T\bar{X}$. In practice, distance measurements have noise and we only know that the SDP solution satisfies $\bar{Y} - \bar{X}^T\bar{X} \succeq 0$. This inequality can be used for error analysis of the position estimation provided by the relaxation. For example, $\mathrm{trace}(\bar{Y} - \bar{X}^T\bar{X}) = \sum \tau_i$, where

$$\tau_i \equiv \bar{Y}_{ii} - \|\bar{x}_i\|^2 \geq 0, \tag{2.5}$$

is a measure of deviation of the SDP solution from the desired constraint $Y = X^T X$ (ignoring off-diagonal elements). The individual trace τ_i can be used to evaluate the position estimation $\bar{x}_i$ for sensor i. In particular, we interpret a smaller τ_i to mean higher accuracy in the estimated position x_i. Further explanation is given in [2].

3. SpaseLoc: A scalable localization algorithm.

When the number of nodes in (2.4) is large, applying a general SDP solver such as DSDP5.0 [1] or SeDuMi [22] would not scale well. In this section, we present a sequential subproblem approach named SpaseLoc to solve the full localization problem iteratively.

3.1. Adaptive subproblem approach. We call the overall sensor localization problem including all sensors and anchors the *full_problem.* At each iteration, SpaseLoc selects from the *full_problem* a subset of the unpositioned sensors and a subset of the anchors to form a localization *subproblem.* We call the selected sensors in the subproblem *subsensors*, and the selected anchors in the subproblem *subanchors*, These subsensors and subanchors, together with their known distance measurements and known anchors' locations, form a sub SDP relaxation model to be solved using the same formulation as in (2.4).

In our adaptive approach, the subanchors and subsensors for each subproblem are chosen dynamically according to rule sets. (Rather than using predefined data, every new iteration's subproblem generation is based on the previous iteration's results.) The resulting SDP subproblems are of varying but limited size. Currently they are solved by Benson, Ye, and Zhang's SDP solver DSDP5.0 [1].

SpaseLoc is a *greedy algorithm* in the sense that each subproblem determines the final estimate of the associated sensor positions.

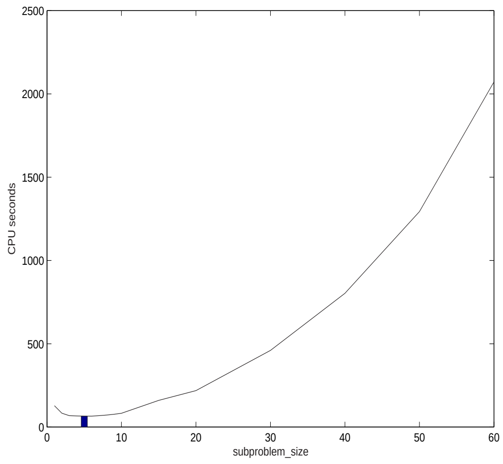

FIG. 3.1. *SpaseLoc execution time as a function of subproblem_size: total nodes = 10000, anchors = 100, radius = 0.02068.*

3.2. The SpaseLoc algorithm. The main steps of SpaseLoc are listed below, followed by explanations of the steps and definitions of new terms used therein.

A0 Set *subproblem_size*.
A1 Subproblem creation: Select subsensors and subanchors to be included in the subproblem.
A2 Formulate SDP relaxation model (2.4) based on the chosen subsensors and subanchors, together with the known distances among them and the subanchors' known positions.
A3 Call SDP solver to obtain optimal solution for the subsensors' positions.
A4 Classify positioned subsensors according to their τ_i value.
A5 If all sensors in the network become positioned or are determined to be outliers, go to step A6. Otherwise, return to step A1 for the next iteration.
A6 Output all sensor locations and report outliers if any. Stop.

In step A0, *subproblem_size* specifies a limit on the number of unpositioned sensors to be included in each subproblem. It can range from 1 to an upper limit value that is potentially solvable by the SDP solver. In our experiments, the upper limit is 150. The most effective *subproblem_size* seems to change with the *full_problem* size, the model parameters such as *radius*, and the SDP solver used. We perform an approximate linesearch to find *subproblem_size* that corresponds to the minimum time, since empirically the total execution time with all other parameters fixed is essentially a convex function of *subproblem_size*.

For example, when *full_problem* size is 10000 with 100 anchors, *radius* 0.02068, and no noise, *subproblem_size* 5 seems to give the best execution time with the DSDP5.0 solver (refer to Figure 3.1). The search time for *subproblem_size* is not included as part of the SpaseLoc execution time.

Step A1 involves choosing a subset of unpositioned sensors (no more than *subproblem_size*) and an associated subset of nodes with known positions. The latter can include a subset of the original anchors and/or a subset of sensors already positioned by a previous subproblem (we define them as *acting anchors*). The rules for choosing subsensors and subanchors in this iteration are discussed in sections 3.4–3.5.

In step A4, the error in sensor i's positioning is estimated by its individual trace τ_i as discussed in section 2.4. Subsensors whose τ_i value is within a given tolerance τ are labeled as positioned and treated as acting anchors for the next iteration, whereas subsensors whose positioning error is higher than the tolerance are also labeled as positioned but are not used as acting anchors in later iterations. These new acting anchors are labeled with different acting levels as explained in section 3.4. The value of τ has an impact on the localization accuracy. Bigger values allow more positioned sensors to be acting anchors, but with possibly greater transitive errors. Smaller values may increase the estimation accuracy for some of the sensors, but could lead to more outliers. A rule of thumb is to use a small τ for networks with high anchor density to achieve potentially more accuracy, and a bigger τ for networks with low anchor density to avoid potential outliers.

In step A5, an unpositioned sensor is called an *outlier* when it does not have any distance information for the algorithm to decide its location. If a sensor has no connection to any anchors, it is classified as an outlier. In addition, if a connected cluster of sensors has no connection to any anchors, then all sensors in the cluster will be outliers.

The next sections explain the subproblem creation procedure used by step A1 above. Section 3.3 lists steps S1–S9 of the creation procedure itself. Section 3.4 presents rules RS1–RS4 for subsensor selection in step S5. Section 3.5 presents rules RA1–RA3 for subanchor selection in step S8. Section 3.6 illustrates the method for independent subanchor selection used in rules RA2–RA3. Sections 3.7–3.8 discuss the routines used in step S7 to localize sensors that have less than 3 connected anchors.

3.3. Subproblem creation procedure. As explained, *subproblem_size* is a predetermined parameter that represents the maximum number of unpositioned sensors that can be selected as subsensors in a subproblem. When there are more than *subproblem_size* unpositioned sensors, we have a choice to make among them.

The subproblem creation procedure makes sure that the choice of subsensors is based first on the number of connected anchors they have, and second on the type of connected anchors such as original anchors and different levels of acting anchors as defined by a priority list (section 3.4), and the choice of subanchors is based on a set of rules (section 3.5). The main steps are listed below, followed by explanations of the steps and definitions of new terms used.

S1 Specify *MaxAnchorReq*.

S2 Initialize *AnchorReq* = *MaxAnchorReq*.

S3 Loop through unpositioned sensors, finding all that are connected to at least *AnchorReq* anchors. If *AnchorReq* ≥ 3, determine if there are 3 *independent* subanchors; if not, go to next sensor.[1] Enter each found sensor into a *candidate subsensor list*, and enter its connected anchors into a corresponding *candidate subanchor list.* Each sensor in the candidate subsensor list has its own candidate subanchor list (so there are as many candidate subanchor lists as the number of sensors in the candidate subsensor list). Let *sub_s_candidate* be the length of the candidate subsensor list.

S4 If *sub_s_candidate* = *subproblem_size*, the candidate subsensor list becomes the chosen subsensors list. Go to step S8.

S5 If *sub_s_candidate* > *subproblem_size*, the choice of subsensors is further based on subsensor selection rules RS1–RS4 described in section 3.4. After ex-

[1]See section 3.6 for dependency definition and independent anchor selection.

actly *subproblem_size* subsensors are selected from the candidate list according these rules, go to step S8.

S6 If *sub_s_candidate* $<$ *subproblem_size* and the candidate list is not null, go to step S8.

S7 Now *sub_s_candidate* $= 0$. Reduce *AnchorReq* by 1.
If *AnchorReq* ≥ 3, go to step S3 for another round of subproblem creation.
If *AnchorReq* $= 2$, apply the procedure in section 3.7 then go to step S3.
If *AnchorReq* $= 1$, apply the procedure in section 3.8 then go to step S3.
Otherwise, *AnchorReq* $= 0$ and *sub_s_candidate* $= 0$ indicates that there are still unpositioned sensors left that are not connected to any positioned nodes. We classify them as outliers and exit this procedure to continue at step A6 of section 3.2.

S8 Now that we have a subsensor list and the candidate subanchor lists, choose subanchors using selection rules RA1–RA3 presented in section 3.5.

S9 The subsensors and subanchors are selected and the subproblem creation routine finishes here. Continue at step A2 in section 3.2.

In step S1, *MaxAnchorReq* determines the initial (maximum) value of *AnchorReq*. It is useful for scalability when connectivity is dense. A smaller *MaxAnchorReq* would generally cause fewer subanchors to be included in the subproblem, thus reducing the number of distance constraints in each SDP subproblem and hence reducing execution time for each iteration. For instance, under ideal conditions (where there is no noise), even if a sensor has 10 distance measurements to 10 anchors, we don't need to include all 10 anchors because we can use 3 to localize that sensor accurately.

In the presence of noise, a bigger *MaxAnchorReq* should reduce the average estimation error. For example, if there is a large distance measurement error from one particular anchor, since *MaxAnchorReq* anchors are all taken into consideration for deciding the sensor's actual position, the large error would be averaged out. Another consideration for setting *MaxAnchorReq* is the trade-off between estimation accuracy and execution speed. If we are in a static environment and would like to have sensor positioning as accurate as possible under noise conditions, we might choose a large *MaxAnchorReq*. However, in a real-time environment involving moving sensors, where speed might take priority, we would consider a smaller *MaxAnchorReq*.

In step S2, *AnchorReq* is a dynamic parameter that may decrease in later steps.

In step S6, the subproblem will contain less than *subproblem_size* subsensors, and this is perfectly acceptable. The alternative is to reduce *AnchorReq* by 1 and find more subsensor candidates that have fewer distance connections. However, this approach might reduce the accuracy of the algorithm, because we do want to localize the subsensors as accurately as possible as the iteration progresses, and the newly localized subsensors could be further used as acting anchors for the next iteration.

In step S7, *AnchorReq* is iteratively reduced by 1 from *MaxAnchorReq* to 0 eventually. This approach allows sensors with at least *AnchorReq* connections to anchors to be positioned before sensors with fewer connections to anchors. As we know, under no-noise conditions, a sensor's position can be uniquely determined by at least 3 independent distance measurements to 3 anchors. If a sensor has only 2 distance measurements to 2 anchors, there are two possible locations; and if there is only 1 distance measurement to an anchor, the sensor can be anywhere on a circle. In this situation, we use heuristic subroutines described in sections 3.7–3.8 to include the sensor's anchors' connected neighboring nodes in the subproblem in order to improve the estimation accuracy.

TABLE 3.1
An example: priority list when MaxAnchorReq=3.

Priority value	Level 1 anchor	Level 2 anchor	Level 3–7 anchor	Level 8–10 anchor	Resulting level
1	≥ 3	any	any	any	2
2	$= 2$	*total* ≥ 1		any	3
3	$= 1$	*total* ≥ 2		any	4
4	0	≥ 3	any	any	5
5	0	$= 2$	*total* ≥ 1	any	6
6	0	≤ 1	*total* ≥ 2 if level 2 anchor = 1, else *total* ≥ 3	any	7
7	*total* ≥ 3, at least one of the 3 anchors is acting level 8 or 9, or 10				8
8	*total* $= 2$				9
9	*total* $= 1$				10

3.4. Subsensor selection priority list. In step S5, when the number of sensors in the candidate subsensor list is bigger than *subproblem_size*, the choice of subsensors is further based on the types of anchors each sensor is connected to.

First, we introduce the concept of *sensor priority*. We assign a priority to each sensor in the candidate subsensor list. A sensor with a smaller priority value is selected to be localized before one with a bigger priority value. A sensor's priority is based on the types of anchors the sensor is directly connected to. Next, in order to define different types of anchors, we introduce the concept of *anchor acting levels*. All anchors including acting anchors are assigned certain acting levels. Original anchors are always set to acting level 1. Every acting anchor is set to an acting level after it has been localized as a sensor. The acting level depends on the priority of the sensor that becomes this acting anchor. Essentially, acting anchors are set with acting levels depending on the levels of the anchors that localized them.

The priority rules for selecting subsensors from a candidate subsensor list are as follows:

RS1 When *AnchorReq* ≥ 3 and a sensor has at least 3 connected anchors that are independent, the sensor's priority depends on the lowest acting level among all the connected anchors and the number of anchors in this level. The lower the acting level and the larger the number of anchors in that level, the higher the sensor's priority.

RS2 If the sensor has 3 connected anchors that are dependent, it is ranked with the same priority as when the sensor is connected to only 2 anchors.

RS3 Sensors with 2 anchor connections are ranked with equal priority, independent of the acting levels of the 2 connected anchors. (This can be easily expanded to be more granular according to the connected anchors' acting levels.) Sensors in this category are assigned lower priority than any sensors that have at least 3 independent anchor connections.

RS4 Sensors with 1 anchor connection are ranked with equal priority, independent of the acting level of the connected anchor. (Again, this can be more granular according to the connected anchor's acting level.) Sensors in this category are assigned lower priority than any sensors that have at least 2 anchor connections.

Table 3.1 illustrates the priority list for an example where *MaxAnchorReq* $= 3$ and the sensor's priority is determined by the number of its connected anchors that have the lowest acting level among all the sensor's connected anchors. We can certainly add more granularity by further classifying the sensor's second and third connected anchors' acting levels. Although more categorizations of the priorities should increase localization accuracy under most noise conditions, more computational effort is required to handle more levels of priorities. In Table 3.1, we assume we will generate only 9 levels of priorities.

Each item in the table represents the number of anchors with different acting levels that is needed at each priority. The last column represents the resulting acting anchors' acting levels for subsequent iterations. For example, if a sensor has at least three independent connections to anchors, and if 2 of the anchors are original anchors (acting level 1) and at least 1 of the connected anchors is at any acting level from 2 to 7, this sensor belongs to priority 2 as listed in row 2 of the table. Also, when this sensor is positioned, it becomes acting anchor level 3. The sensors that connect to two anchors belong to the second last priority (8 in the table), and sensors that connect to only one anchor belong to the last priority (9 in this case). In addition, if a sensor connects to at least 3 independent anchors, among which at least one anchor belongs to level 8, 9, or 10, this sensor will be classified as the third last priority as listed in row 7.

3.5. Subanchors selection. In step S8, for each unpositioned subsensor, only *AnchorReq* of the connected anchors are allowed to be included in the subproblem. We use the following rules to select subanchors from a candidate subanchor list that contains more than *AnchorReq* anchors.

RA1 Original anchors are selected first, followed by acting anchors with lower acting level.
RA2 The subanchors chosen should be linearly independent.
RA3 Among independent anchors in the candidate subanchor list, we use distance scale-factors to encourage selection of the closest subanchors.

Rules RA2 and RA3 are implemented as in section 3.6. Rule RA3 is based on the assumption that under noise conditions, we trust the shorter distance measurements more than the longer ones. This is specially true for ranging devices based on RF (radio frequency) strength.

For certain applications, it may be beneficial to choose *MaxAnchorReq* large in order to increase the localization accuracy, though it could impact the algorithm speed.

3.6. Independent subanchors selection. Suppose sensor i is connected to K $(K > 3)$ anchors at locations a_{ik} with corresponding distance measurements $\widehat{d}_{ik}$ $(k = 1, \dots, K)$. Define the matrices

$$A = \begin{pmatrix} 1 & 1 & \dots & 1 \\ -a_{i1} & -a_{i2} & \dots & -a_{iK} \end{pmatrix}, \quad D_1 = \mathrm{diag}(1/\sqrt{1 + \|a_{ik}\|^2}), \quad D_2 = \mathrm{diag}(1/\widehat{d}_{ik}).$$

We select an independent subset by a QR factorization with column interchanges [10]: $B = AD_1D_2$, $BP = QR$, where Q is orthogonal, R is upper-trapezoidal, and P is a permutation chosen to maximize the next diagonal of R at each stage of the factorization. (D_1 normalizes the columns of A, and D_2 biases them in favor of anchors that are closer to sensor i.) If the 3rd diagonal of R is larger than a predefined threshold (10^{-4} is used in our simulation), the first 3 columns of AP are regarded as independent, and the associated anchors are chosen. Otherwise, all subsets of 3 among the K anchors are regarded as dependent. (In MATLAB, R and P are obtained by a command of the form `[Q,R,P] = qr(B)`.)

3.7. Geometric subroutine (two connected anchors). This section illustrates the heuristic techniques used in step S7 of section 3.3 to localize sensors connected to only two anchors.

When a sensor's connected anchors are also connected to other anchors, this subroutine may improve the accuracy of the sensor's positioning, as illustrated by an example in Figure 3.2.

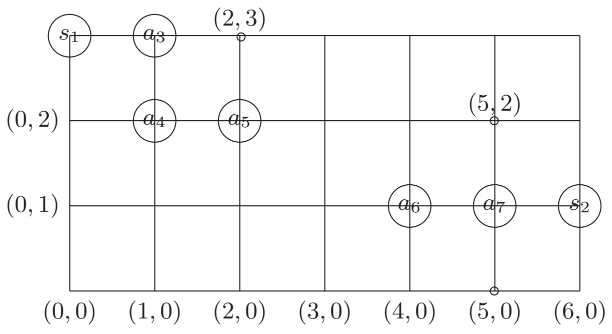

FIG. 3.2. *Sensors with connections to at most two anchors.*

In this example, assume s_1 and s_2 are sensors with unknown locations, and $a_3(1,3)$, $a_4(1,2)$, $a_5(2,2)$, $a_6(4,1)$, $a_7(5,1)$ are anchors with known positions in brackets. Assume that the sensors' radio range is $\sqrt{2}$, and we are also given two distance measurements $\widehat{d}_{13} = 1$ and $\widehat{d}_{14} = \sqrt{2}$ for sensor s_1 and one measurement $\widehat{d}_{27} = 1$ for sensor s_2.

Given two distances $\widehat{d}_{13}$ and $\widehat{d}_{14}$ to two anchors $a_3(1,3)$ and $a_4(1,2)$, we know that s_1 should be either at $(0,3)$ or $(2,3)$. If we only use s_1, $a_3(1,3)$, $a_4(1,2)$ in an SDP subproblem, SDP relaxation will give a solution near the middle of the two possible points, which would be very close to point $(1,3)$. If there is any anchor (a_5) that is near s_1's connected anchors (a_3 and a_4) with any of the two possible sensor' points within their radio range (point $(2,3)$ is within a_5's range), that point $(2,3)$ must not be the real location of s_1, or else s_1 would be connected to this anchor (a_5) as well. Thus we can infer that s_1 must be at the other point $(0,3)$.

Inspired by the above observation, when a sensor has at most 2 connected anchors, we include these anchors' connected anchors in the subproblem (we call them the connected anchors' neighboring anchors) together with the sensor and its directly connected anchors. By including the neighboring anchors, we might hope that the inequality constraints in the SDP relaxation model (2.4) would push the estimation towards the right point. However, because of the relaxation, enforcing inequalities in (2.4) is not equivalent to enforcing them in the distance model (2.2). The added inequality constraints only push the original solution near $(1,3)$ a tiny bit towards s_1's real location $(0,3)$, and the solution essentially stays at around $(1,3)$.

Given the ineffectiveness of the SDP relaxation approach under this condition, we propose instead a geometric approach as illustrated in Figure 3.3. Assume $s_1(x_x, x_y)$ has measurements $\widehat{d}_{12}$ to anchor $a_2(a_{2x}, a_{2y})$ and $\widehat{d}_{13}$ to anchor $a_3(a_{3x}, a_{3y})$. We also assume $\widehat{d}_{12} \le \widehat{d}_{13}$ (we can always swap the two indexes otherwise). Let a_l ($l = 4, \dots, k$) be a_2 and/or a_3's neighboring anchors with radio range r_{1l} ($l = 4, \dots, k$), and let d_{23} be the known (exact) Euclidean distance between a_2 and a_3.

- If two circles centered at a_2 and a_3 with radii $\widehat{d}_{12}$ and $\widehat{d}_{13}$ intersect each other ($\widehat{d}_{12} + \widehat{d}_{13} \ge d_{23}$ and $\widehat{d}_{13} - \widehat{d}_{12} \le d_{23}$) as in Figure 3.3(a):

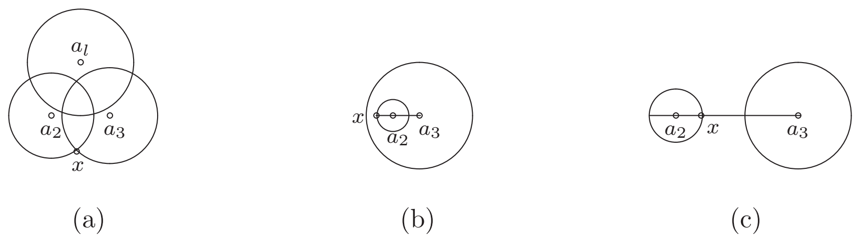

FIG. 3.3. *(a) Sensor with two anchors' circles intersecting. (b) Sensor with two anchors, a_2's circle in a_3's. (c) Sensor with two anchors' circles disjoint.*

- Two possible locations of s_1 are given by solutions x^* and x^{**} of the equations

$$\|x - a_2\|^2 = \widehat{d}_{12}^2 , \qquad \|x - a_3\|^2 = \widehat{d}_{13}^2 .$$

- Sensor s_1's position is selected from x^* and x^{**}, whichever is further away from any neighboring anchor. Thus, for $l = 4$ to k,

```
     if $\|x^* - a_l\|^2 < r_{1l}^2$,   then $x = x^{**}$ and stop
else if $\|x^{**} - a_l\|^2 < r_{1l}^2$,   then $x = x^*$ and stop.
```

Otherwise, $x = (x^* + x^{**})/2$ and stop.

• Under noise conditions, the a_2 circle may be inside the a_3 circle ($\widehat{d}_{12} + \widehat{d}_{13} \geq d_{23}$ and $\widehat{d}_{13} - \widehat{d}_{12} > d_{23}$) as in Figure 3.3(b).

- The solutions x^* and x^{**} of the following equations give two possible points for s_1 on the a_2 circle:

$$\begin{aligned}(x_x - a_{2x})^2 + (x_y - a_{2y})^2 &= \widehat{d}_{12}^2,\\ (a_{2x} - a_{3x})(x_y - a_{2y}) &= (a_{2y} - a_{3y})(x_x - a_{2x}),\end{aligned}$$

where x is on the line through a_2 and a_3 represented by the second equation.

- If $\|x^* - a_3\| < \|x^{**} - a_3\|$, then $x = x^{**}$; otherwise $x = x^*$. This guarantees that the point further from a_3 is chosen. Note that we base the sensor's estimation on the closest anchor (a_2 here since $\widehat{d}_{13} \geq \widehat{d}_{12}$), assuming that a shorter measurement is generally more accurate than longer ones, given similar anchor properties.

The same approach applies when the a_3 circle is inside the a_2 circle ($\widehat{d}_{12} - \widehat{d}_{13} > d_{23}$).

• Under noise conditions, the a_2 and a_3 circles may again have no intersection ($\widehat{d}_{12} + \widehat{d}_{13} < d_{23}$) as in Figure 3.3(c).

- The solutions x^* and x^{**} of the following equations give two possible points for s_1 on the circle for the anchor with smaller radius. Let's assume $\widehat{d}_{12} \leq \widehat{d}_{13}$:

$$\begin{aligned}(x_x - a_{2x})^2 + (x_y - a_{2y})^2 &= \widehat{d}_{12}^2,\\ (a_{2x} - a_{3x})(x_y - a_{2y}) &= (a_{2y} - a_{3y})(x_x - a_{2x}),\end{aligned}$$

where x is on the line through a_2 and a_3 represented by the second equation.

- If $\|x^* - a_3\| > \|x^{**} - a_3\|$, then $x = x^{**}$; otherwise $x = x^*$. This guarantees that the point closer to a_3 (in between a_2 and a_3) is chosen.

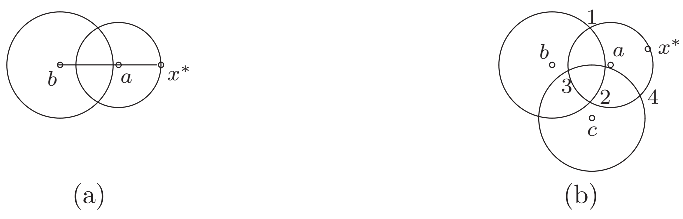

FIG. 3.4. *(a) Sensor with one anchor connection a and one neighboring anchor b. (b) Sensor with one anchor connection a and two neighboring anchors b, c.*

3.8. Geometric subroutine (one connected anchor). Similar inefficiency occurs in the SDP solution when a sensor connects to only *one* anchor. The SDP solver under this condition gives a solution for the sensor to be in the same location as the sensor's connected anchor. In reality, the sensor could be anywhere on the circle. The SDP gives an average point, at the center of the circle, and that is where the connected anchor is. Even if the anchor's neighboring anchor is included in the SDP subproblem, the inequality constraints are not active most of the time because the SDP solution may not provide optimal solutions all the time.

We propose a heuristic for estimating a sensor's location with only one connecting anchor. The idea is to use one neighboring anchor's radio range information to eliminate the portion of the circle that the sensor would not be on, and then calculate the middle of the other portion of the circle to be the sensor's position. For the example in Figure 3.2, because we know the distance between s_2 and a_7 is 1, we know that s_2 could be anywhere on the circle surrounding a_7 with a radius of 1. Knowing a_7's neighboring anchor node a_6 is not connected to s_2, we know that s_2 would not be in the area surrounding a_6 with a radius of $\sqrt{2}$. Thus, s_2 could be anywhere around the half circle including points $(5, 2)$, $(6, 1)$, $(5, 0)$. The heuristic chooses the middle point between the two circles' intersection points $(5, 2)$ and $(5, 0)$, which happens to be $(6, 1)$ in this example. The heuristic gives better accuracy for the sensor's location than the SDP solution under most conditions. The procedure follows:

- Assume s has one distance measurement $\widehat{d}$ to anchor a, and b is the closest connected neighboring anchor to a with radio range r (refer to Figure 3.4(a)). We assume $a = (a_x, a_y)$, $b = (b_x, b_y)$, $x = (x_x, x_y)$.
- The solutions x^* and x^{**} of the following equations give two possible points s on the circle:

$$\begin{aligned} (x_x - a_x)^2 + (x_y - a_y)^2 &= \widehat{d}^2, \\ (a_x - b_x)(x_y - a_y) &= (a_y - b_y)(x_x - a_x), \end{aligned}$$

  where x is on the line through a and b represented by the second equation.
- If $\|x^* - b\| < r$, then $x = x^{**}$; otherwise $x = x^*$. This guarantees that the point further from b is chosen.

The above heuristic provides a simple way of estimating a sensor's location when the sensor connects to only one anchor. A more complicated approach can be adopted when the connected anchor has more than one neighboring anchor, which can increase the accuracy of the sensor's location. We call it an arc elimination heuristic. The idea is to loop through each of the neighboring anchors and find the portion of the circle that the sensor won't be on, and eliminate that arc as a possible location of the sensor. Eventually, when one or more plausible arcs remain, we choose the middle of the largest arc to be the sensor's location. For example, assume we add one more neighboring anchor c to sensor s's anchor a from the previous example in Figure 3.4(a). The new scenario is shown in Figure 3.4(b). First, we find the intersections (points 1 and 2) of two circles: one at a with radius $\widehat{d}$, the other at b with radius r. We know that the 1–2 portion of the arc closer to point b won't be the location of s. Second, we find the intersections (points 3 and 4) of two circles: one at a with radius $\widehat{d}$, the other at c with radius r. We know that the arc 3–4 closer to point c won't be the location of s. Thus we deduce that s must be somewhere on the arc 1–4 further away from b or c. The estimation of s is given in the middle of the arc 1–4. As we see, this method should provide more accuracy than the one-neighboring-anchor approach.

4. Computational results. This section explains the simulation method and the setup for experimenting with the SpaseLoc algorithm, then presents results for various parameter settings.

For the simulation, a total number of nodes n (including s sensors and m anchors) is specified in the range 50 to 10000. The positions of these nodes are assigned with a uniform random distribution on a square region of size $r \times r$ where $r = 1$, or put on the grid points of a regular topology such as a square or an equilateral triangle on the same region. $\mathit{MaxAnchorReq} = 3$ is used in the simulation. The m anchors are randomly chosen from the given n nodes. We assume all sensors have the same radio range (*radius*) for any given test case. Various radio ranges were tested in the simulation.

Euclidean distances $d_{ij} = \|x_i - x_j\|$ are calculated among all sensor pairs $(i,\ j)$ for $i < j$. We then use $\widehat{d}_{ij}$ to simulate measured distances, where $\widehat{d}_{ij}$ is d_{ij} times a random error simulated by *noise_factor* $\in [0, 1]$. For a given *radius* $\subseteq [0, 1]$ it is defined as follows:

- If $d_{ij} \le \mathit{radius}$, then $\widehat{d}_{ij} = d_{ij}(1 + \mathtt{rn} * \mathit{noise\_factor})$, where $\mathtt{rn}$ is normally distributed with mean zero and variance one. (Any numbers generated outside $(-1, 1)$ are regenerated.)
  In practical networks, depending on the technologies that are being used to obtain the distance measurements, there may be many factors that contribute to the noise level. For example, one way to obtain the distance measurement is to use the received radio signal strength between two sensors. The signal strength could be affected by media or obstacles in between the two sensors. In this study, *noise_factor* is a normally distributed random variable with mean zero and variance one. This model could be replaced by any other noise model in practice.
- If $d_{ij} > \mathit{radius}$, the bound $r_{ij} = 1.001 * \mathit{radius}$ is used in the SDP model.

In the simulation, we define the average estimation error to be $\frac{1}{s}\sum_{i=1}^{s} \|\bar{x}_i - x_i\|$, where $\bar{x}_i$ is from the SDP solution and x_i is the ith node's true position. In a practical setting, we wouldn't know the node's true location x_i. Instead, we would use the node's trace τ_i (2.5) to gauge the estimation error.

To convey the distribution of estimation errors and trace, we also give the 95% quartile.

Factors such as noise level, radio range, and anchor densities can directly impact localization accuracy. The sensors' estimated positions are derived directly from the given distance measurements. If the noise level in these measurements is high, the estimation accuracy cannot be high. We also need sufficiently large radio range to achieve accurate positioning, because too small a range could cause many sensors to be unreachable. Finally, more anchors in the network should help with the estimation accuracy because there are more reference points.

In the following subsections, we present simulation results (most results averaged over 10 runs) to show the accuracy and scalability of the SpaseLoc algorithm. We observe the impact of various radio ranges, anchor densities, and noise levels on the accuracy and performance of the algorithm. Computations were performed on a laptop computer with 2.4GHz CPU and 1GB system memory, using MATLAB 6.5 [15] for SpaseLoc and a Mex interface to DSDP5.0 (Benson, Ye, and Zhang [1]) for the SDP solutions.

4.1. Effect of inequality constraints in SDP relaxation model. As we discussed in section 3.7, because of the $Y \succeq X^T X$ constraint relaxation, enforcing the r_{ij}^2 and r_{ik}^2 inequality constraints in (2.4) is not equivalent to enforcing them in the distance model (2.2). In order to observe the effectiveness of including these inequality constraints, we conduct simulations with the following three strategies, according to the number of times we check for violated inequality constraints and then include them to obtain new solution.

- I2 corresponds to solving the SDP problem with all equalities (and no inequalities), and then adding violated inequality constraints and re-solving one or more times until all inequalities are satisfied. The final solution is an optimal solution to problem (2.4).
- I1 corresponds to solving the SDP problem with all equalities and then adding all violated inequality constraints at most once.
- I0 corresponds to solving the SDP problem with equality constraints only. (No inequality constraints are ever added.) The final solution is optimal for problem (2.4) without the inequality constraints involving r_{ij}^2 and r_{ik}^2.

Our experimental results show that the added inequality constraints do not always provide better positioning accuracy, but greatly increase the execution time. In this section, we illustrate the inequality constraints' impact through two simulation examples: one with no noise but low connectivity; the other with full connectivity but with noise.

In our first example, we run simulation results on a network of 100 randomly uniform-distributed sensors with radius 0.2275 and 10 randomly selected anchors. One of the sensors happens to be connected to only two other nodes. The sensors are localized with the full SDP and with SpaseLoc, using each of the I2, I1, I0 strategies in turn. And for SpaseLoc, we also examine each case with or without our geometric routines. The results are shown in Figure 4.1 and Table 4.1.

Figure 4.1 shows there is a sensor (s) that is connected with only 2 anchors. For full SDP shown in (a), no violated inequalities are ever found, so full SDP with I2, I1, or I0 has only one SDP call and always generates the same results. For SpaseLoc in (b) with I0 and no geometric routine, SDP is called 46 times (with no subsequent check for violated constraints). It produces the same estimation accuracy as the full SDP approach but with much improved performance. In (c), SpaseLoc with I2 or I1 produces the same results, which means violated inequalities are found only once. Comparing (b) and (c), we see that including violated inequalities does improve the estimation accuracy. Best of all in (d), SpaseLoc with I0 and our geometric routines localizes all sensors with virtually no error.

Table 4.1 shows that adding violated inequalities increases execution time slightly for SpaseLoc.

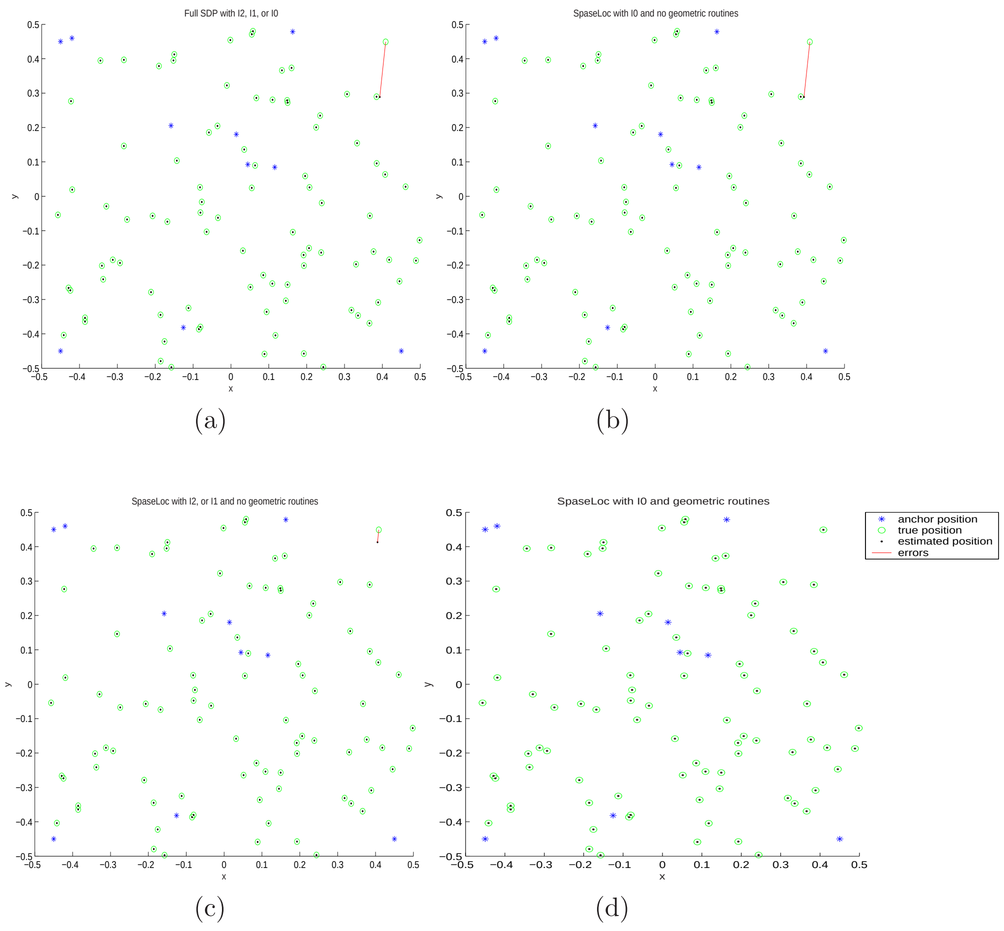

FIG. 4.1. *Inequality impact on accuracy: 100 nodes, 10 anchors, no noise, radius 0.2275.*

TABLE 4.1
Inequality impact on accuracy and speed: 100 nodes, 10 anchors, no noise, radius 0.2275.

Methods	Error	95% Error	Time	SDP's
Full SDP with I2, or I1 or I0	1.80e-3	1.74e-10	11.63	1
SpaseLoc with I0 and no geometric routines	1.80e-3	1.09e-08	0.40	46
SpaseLoc with I2 or I1 and no geometric routines	4.01e-4	1.09e-08	0.44	47
SpaseLoc with I0 and geometric routines	1.28e-7	8.78e-09	0.41	45

In our second example, in order to observe the effectiveness of the inequality constraints under noise conditions, we run simulations for a network of 100 nodes whose true locations are at the vertices of an equilateral triangle grid. 10 anchors are positioned along the middle grid-points of each row, and the radius is 0.25. A *noise_factor* of 0.1 is applied to the distance measurements. The sensors are localized with either full SDP or SpaseLoc using I2, I1, I0 in turn without geometric routines. (Although we do not activate the geometric routines in this experiment, they are not a factor here because the localization error is not caused by low connectivity but by the noisy measurements.) The results are shown in Figure 4.2 and Table 4.2. Figure 4.2 (b) and (c) correspond to strategy I2 for full SDP and SpaseLoc.

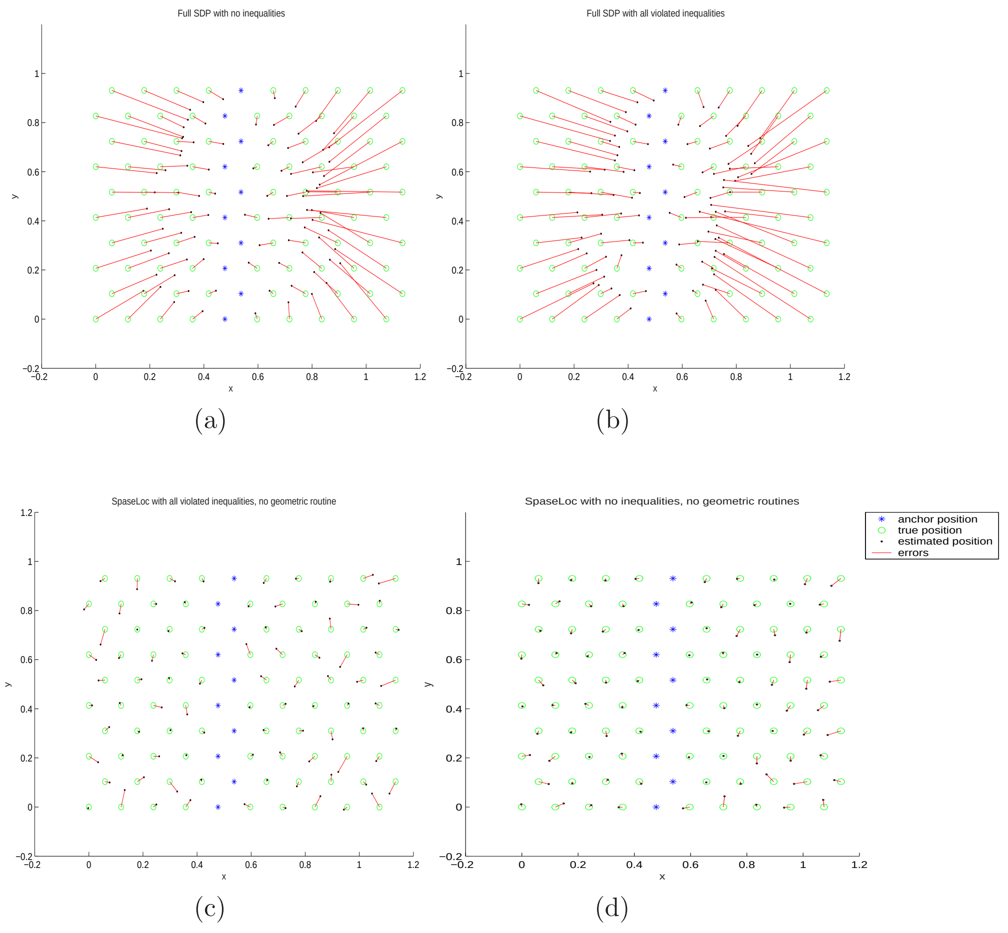

FIG. 4.2. *Inequality impact on accuracy: 100 nodes, 10 anchors, noise_factor 0.1, radius 0.25.*

TABLE 4.2
Inequality impact on accuracy and speed: 100 nodes, 10 anchors, noise_factor 0.1, radius 0.25.

Methods	Error	Time	SDP's
Full SDP with I2	0.1403	134.50	4
Full SDP with I1	0.1292	34.20	2
Full SDP with I0	0.1268	13.87	1
SpaseLoc with I2	0.0154	0.71	48
SpaseLoc with I1	0.0154	0.65	46
SpaseLoc with I0	0.0165	0.42	30

As we can see, adding violated inequalities for full SDP not only increases the execution times dramatically, but also increases the localization error. For SpaseLoc, adding violated inequalities improves the estimation accuracy slightly. Note that I2 had 2 more SDP calls but did not improve the accuracy over I1.

In summary, the first experiment shows that when the errors are caused by *low connectivity*, SpaseLoc with geometric routines and no inequality constraints (I0) outperforms SpaseLoc with inequalities (I1 or I2) and all of the full SDP options. Given this observation, from now on we only use SpaseLoc with geometric routines, which means the geometric routines are used instead of SDP to localize sensors connected to fewer than 3 anchors.

The second experiment indicates that under noise conditions, although adding violated inequalities does not seem to improve the estimation accuracy for full SDP, it does improve accuracy for SpaseLoc.

In the subsequent sections, we continue to examine the inequality constraints' effects on accuracy and speed.

4.2. Accuracy and speed comparison: Full SDP vs SpaseLoc. For very small networks, the full SDP solution is both accurate and efficient. (This is vital to the performance of SpaseLoc, as many small subproblems must be solved using SDP.) However, the performance of the pure SDP approach deteriorates rapidly with network size.

Table 4.3 shows the localization results using full SDP (a) and using SpaseLoc (b) for a range of examples with various numbers of nodes whose true locations in the network are at the vertices of an equilateral triangle grid. Anchors are positioned along the middle grids of each row. A *noise_factor* of 0.1 is applied to the distance measurements.

Let's first look at the impact of I2, I1, and I0 on estimation accuracy. As we can see from Table 4.3 (a), for full SDP, 5 errors with I2 are bigger than with I1, and 8 errors with I1 are bigger than with I2. Comparing I2 with I0, we see that for each strategy, 8 errors are bigger than the errors for the other strategy. It appears that full SDP with added inequalities does not improve the estimation accuracy in this simulation. For SpaseLoc, I2 and I1 generate almost equivalent estimation accuracy, I0 has 8 errors that are bigger than with I1, and at the same time, I1 has 7 errors that are bigger than with I0. It is therefore hard to judge the effectiveness of the added inequalities.

Now let's compare full SDP with SpaseLoc. Figures 4.3–4.4 plot results for full SDP with I0 and SpaseLoc with I0 for two of these examples: 9 and 49 nodes, including 3 and 7 anchors positioned at the grid-point in the middle of each row. As we can see from these two figures and Table 4.3, for localizing 4 and 9 nodes, full SDP and SpaseLoc show comparable performance. Beyond that size, the contrast grows rapidly. For localizing 49 nodes, SpaseLoc is more than 10 times faster than the full SDP method, with more than 4 times better accuracy. For 400 nodes, SpaseLoc is about 1000 times faster and 20 times more accurate with strategy I0, about 2000 times faster with I1, and 6000 times faster with I2, with similar accuracy improvements. Thus, the full SDP model becomes less effective as problem size increases. In fact, for problem sizes above 49 nodes, the average estimation error becomes so large that the computed solution is of little value.

It may seem non-intuitive that SpaseLoc's greedy approach could produce smaller errors than the full SDP method. However, all of the SDP problems and subproblems of the form (2.4) are *relaxations* of Euclidean models of the form (2.2). It shows experimentally that the relaxations are *tighter* in SpaseLoc's subproblems than in the single large SDP.

In the following sections, we run more simulations only with SpaseLoc.

Table 4.3
Accuracy and speed comparison between full SDP and SpaseLoc.

(a) Full SDP

Number	Radio	Error			Time (sec)			SDP calls		
of nodes	range	I2	I1	I0	I2	I1	I0	I2	I1	I0
4	2.24	0.0317	0.0317	0.0317	0.01	0.01	0.01	1	1	1
9	1.12	0.0800	0.0800	0.0704	0.05	0.07	0.02	2	2	1
16	0.75	0.0680	0.0703	0.0837	0.35	0.21	0.10	3	2	1
25	0.56	0.1170	0.1170	0.0938	1.26	0.80	0.37	3	2	1
36	0.45	0.0561	0.0618	0.0719	3.02	1.88	0.81	3	2	1
49	0.40	0.1190	0.1190	0.1190	5.42	5.33	2.10	2	2	1
64	0.40	0.0954	0.0919	0.1218	21.60	9.21	3.43	4	2	1
81	0.40	0.0885	0.0894	0.1380	59.05	19.66	7.26	5	2	1
100	0.25	0.1403	0.1292	0.1268	140.26	34.20	13.87	4	2	1
121	0.40	0.1091	0.1088	0.1157	182.74	81.62	23.24	3	2	1
144	0.21	0.1891	0.1899	0.1480	584.23	168.43	37.76	4	2	1
169	0.40	0.1217	0.1141	0.1283	692.12	278.72	71.87	4	2	1
196	0.18	0.1286	0.1275	0.1404	1081.35	461.97	151.52	4	2	1
225	0.40	0.1571	0.1589	0.1568	2408.67	752.75	232.31	5	2	1
256	0.15	0.1370	0.1375	0.1429	3260.33	1089.52	356.86	5	2	1
324	0.14	0.1685	0.1685	0.1685	2620.20	2659.18	962.66	2	2	1
361	0.13	0.1833	0.1842	0.1734	15281.26	5051.05	1391.04	4	2	1
400	0.12	0.1968	0.1970	0.1819	20321.60	5950.34	1662.22	4	2	1

(b) SpaseLoc

Number	Radio	Error			Time (sec)			SDP calls		
of nodes	range	I2	I1	I0	I2	I1	I0	I2	I1	I0
4	2.24	0.0317	0.0317	0.0317	0.01	0.01	0.01	1	1	1
9	1.12	0.0766	0.0766	0.0670	0.11	0.32	0.03	3	3	2
16	0.75	0.0599	0.0599	0.0612	0.07	0.07	0.04	7	7	4
25	0.56	0.0495	0.0495	0.0516	0.11	0.11	0.08	10	10	7
36	0.45	0.0250	0.0250	0.0262	0.22	0.21	0.12	17	17	10
49	0.40	0.0283	0.0283	0.0282	0.21	0.21	0.19	15	15	14
64	0.40	0.0193	0.0193	0.0202	0.41	0.38	0.25	28	27	19
81	0.40	0.0186	0.0187	0.0192	0.56	0.53	0.31	41	40	24
100	0.25	0.0154	0.0154	0.0165	0.71	0.65	0.42	48	46	30
121	0.40	0.0152	0.0152	0.0143	0.86	0.82	0.51	58	57	37
144	0.21	0.0133	0.0133	0.0123	0.99	0.93	0.61	65	63	44
169	0.40	0.0108	0.0108	0.0110	1.22	1.16	0.74	81	80	52
196	0.18	0.0113	0.0112	0.0100	1.30	1.24	0.86	85	83	61
225	0.40	0.0099	0.0099	0.0097	1.88	1.68	0.96	126	118	70
256	0.15	0.0075	0.0075	0.0077	2.01	1.79	1.10	136	126	80
324	0.14	0.0075	0.0075	0.0075	1.64	1.64	1.48	102	102	102
361	0.13	0.0073	0.0073	0.0069	2.20	2.16	1.65	139	139	114
400	0.12	0.0070	0.0070	0.0064	3.31	2.98	1.78	217	204	127

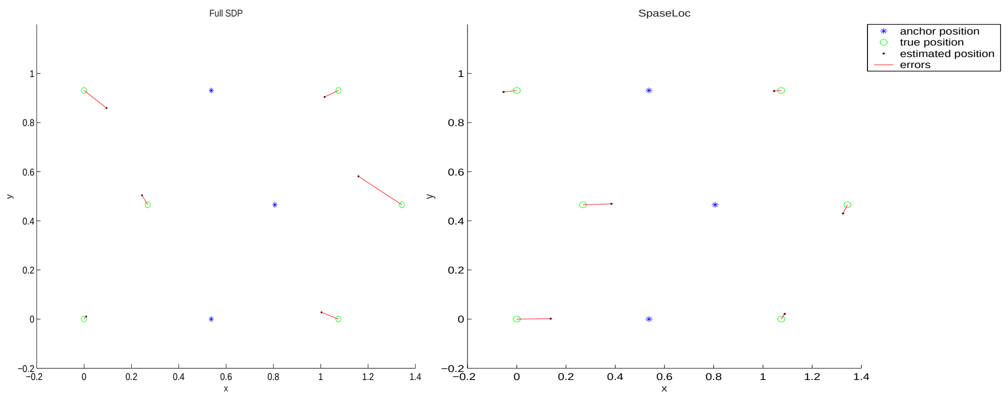

FIG. 4.3. *9 nodes on equilateral-triangle grids, 3 anchors, 0.1 noise, radius 1.12.*

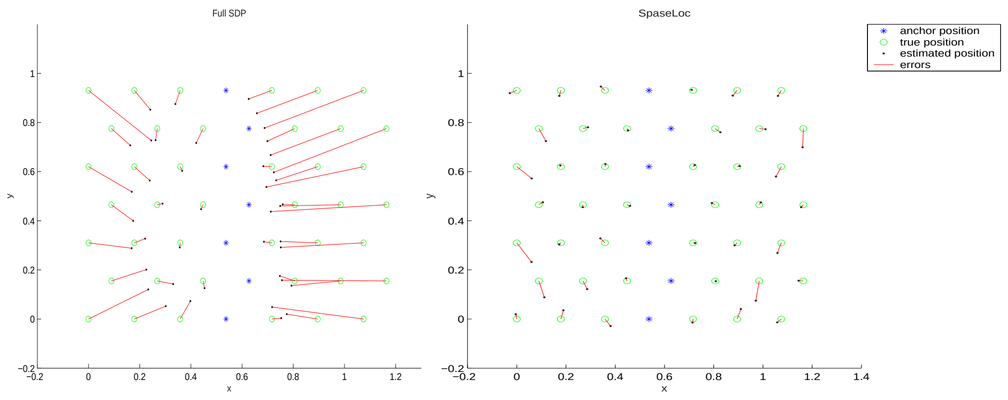

FIG. 4.4. *49 nodes on equilateral-triangle grids, 7 anchors, 0.1 noise, radius 0.40.*

4.3. Scalability. Table 4.4 shows simulation results for 49 to 10000 randomly uniform-distributed sensors being localized using SpaseLoc with strategies I2, I1, and I0. The node numbers 49, 100, 225, ... are squares k^2, and the *radius* is the minimum value that permits localization on a regular $k \times k$ grid. The number of anchors changes with the number of sensors in order to maintain the same anchor density. Noise is not included in this simulation.

We find that the three strategies I2, I1, I0 produce the same results. This is because the inaccuracy of the positioning estimation is purely caused by low connectivity, not by noisy distance measurements. Empirically we see that the program scales well: almost linearly in the number of nodes in the network. Indeed, the computational complexity of the SpaseLoc algorithm is of order n, the number of sensors in the network, even though the full SDP approach has much greater complexity, as we now show.

We know that in the full SDP model (2.4), the number of constraints is $O(n^2)$, and in each of iteration of its interior-point algorithm the SDP solver needs to solve a *sparse* linear system of equations whose dimension is the number of constraints. Figure 4.5 plots the CPU time for strategy I0 from Table 4.3 (a) as well as three curves of the form $time = a_p n^p$ for $p = 2, 3, 4$, where a_p is determined by a least-

Table 4.4
SpaseLoc scalability. Strategies I2, I1, and I0 generate same results.

Nodes	Anchors	*radius*	*sub_size*	Error	95% Error	Trace	95% Trace	Time	SDP's
49	7	0.3412	2	1.40e-08	9.86e-10	6.28e-09	4.10e-10	0.21	21
100	10	0.2275	2	1.28e-07	8.78e-09	2.68e-08	2.36e-09	0.45	45
225	15	0.1462	3	6.98e-07	7.74e-08	8.75e-08	1.19e-08	0.80	70
529	23	0.0931	3	2.87e-06	9.42e-08	2.68e-07	1.03e-08	1.99	169
1089	33	0.0620	3	2.50e-06	1.65e-07	1.27e-07	1.12e-08	4.12	350
2025	45	0.0451	3	2.70e-05	2.41e-07	2.28e-07	1.41e-08	8.55	658
3969	63	0.0330	3	6.32e-06	3.53e-07	1.30e-07	1.39e-08	19.51	1302
5041	71	0.0292	3	6.70e-06	5.06e-07	1.44e-07	1.80e-08	25.57	1656
6084	78	0.0266	3	6.95e-06	5.47e-07	1.50e-07	1.84e-08	33.38	2000
7056	84	0.0247	4	5.92e-06	6.43e-07	1.35e-07	1.90e-08	41.58	1743
8100	90	0.0230	5	3.92e-06	6.42e-07	8.23e-08	1.78e-08	50.34	1602
9025	95	0.0218	5	8.38e-06	6.74e-07	1.17e-07	1.74e-08	57.34	1788
10000	100	0.0207	5	4.70e-06	7.63e-07	1.12e-07	1.94e-08	65.08	1981

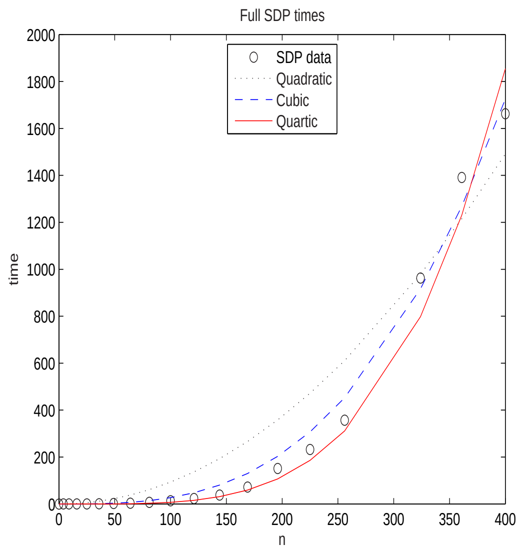

Fig. 4.5. *SDP computational complexity*

squares fit. It appears that the SDP complexity with strategy I0 lies somewhere between $O(n^3)$ and $O(n^4)$.

In SpaseLoc, we partition the full problem into p subproblems of size q or less, where $p \times q = n$. We generally set q to be much smaller than n, ranging from 2 to around 10 in most of our simulations. If τ represents the execution time taken by the full SDP method for a 10-node network, in the worst case the computation time for SpaseLoc is $\tau \times O(p)$. Thus, SpaseLoc is really linear in p in theory. Since we can assume q to be a parameter ranging from 2 to 10, with worst case 2, we know that $O(p) = O(n/q) \leq O(n/2) = O(n)$. Now we can see that SpaseLoc's computation time is $O(n)$.

In the remaining subsections we choose the middle network size from Table 4.4 (nodes = 3969) to observe the effect of varying radio range, noise factor, and number of anchors.

4.4. Radio range impact. With a fixed total number of randomly uniform-distributed nodes (3969, of which 63 are anchors), Table 4.5 shows the direct impact of radio range on accuracy and performance. (Noise is not included.) When *radius* is reduced to 0.02, the number of unreachable sensors (outliers) reaches 302, which is unacceptable. Clearly, the simulation could assist sensor network designers in selecting a radio range to achieve a desired estimation error and algorithm speed.

Strategies I2 and I1 produce the same results. I2, I1, and I0 generate the same results except for radius of 0.020, 0.028, and 0.030, when I2 and I1 have more SDP calls than I0. I2 and I1 produce slightly reduced average error for radius of 0.028 and 0.030 but the same 95% error, and obviously, I2 and I1 take more time than I0.

Table 4.5
Radio range impact: nodes = 3969, anchors = 63, no noise.

rad-	*sub_*	Out-	Error			95% Error			Time			SDP's		
ius	*size*	liers	I2	I1	I0	I2	I1	I0	I2	I1	I0	I2	I1	I0
0.020	28	302	5.26e-3	5.26e-3	5.26e-3	3.77e-3	3.77e-3	3.77e-3	8.40	7.72	7.30	390	390	388
0.022	13	68	2.93e-3	2.93e-3	2.93e-3	1.49e-3	1.49e-3	1.49e-3	10.92	10.48	9.79	539	539	539
0.024	9	17	1.38e-3	1.38e-3	1.38e-3	2.28e-4	2.28e-4	2.28e-4	12.94	13.02	11.95	684	684	684
0.026	6	7	4.36e-4	4.36e-4	4.36e-4	1.36e-6	1.36e-6	1.36e-6	14.90	14.77	13.49	783	783	783
0.028	3	2	1.20e-4	1.20e-4	1.21e-4	1.01e-6	1.01e-6	1.01e-6	16.31	16.36	14.79	1295	1295	1294
0.030	3	0	3.04e-5	3.04e-5	3.08e-5	6.69e-7	6.69e-7	6.69e-7	18.12	18.15	16.23	1303	1303	1302
0.032	3	0	1.06e-5	1.06e-5	1.06e-5	4.10e-7	4.10e-7	4.10e-7	18.22	18.22	18.22	1302	1302	1302
0.033	3	0	6.32e-6	6.32e-6	6.32e-6	3.53e-7	3.53e-7	3.53e-7	19.51	19.51	19.51	1302	1302	1302

4.5. Noise factor impact. With constant *radius* (0.033) and the same randomly distributed nodes (3969), Table 4.6 shows the impact of noise conditions on accuracy and performance. We see that more noise in the network has a direct impact on estimation accuracy. Simulations of this kind may help designers determine the measurement noise level that will give an acceptable estimation error.

We also see that strategies I2 and I1 (with added inequality constraints) provide consistent improvement for both average and 95% error, at the price of increased execution time.

Table 4.6
Noise factor impact: nodes = 3969, anchors = 63, radius = 0.033, subproblem_size = 3.

noise	Error			95% Error			Time			SDP's		
factor	I2	I1	I0	I2	I1	I0	I2	I1	I0	I2	I1	I0
0.1	2.57e-3	2.58e-3	3.18e-3	1.99e-3	2.00e-3	2.23e-3	31.93	30.45	22.00	1907	1860	1303
0.2	4.55e-3	4.60e-3	5.60e-3	3.76e-3	3.79e-3	4.40e-3	43.21	37.52	22.61	2520	2310	1303
0.3	6.49e-3	6.60e-3	8.04e-3	5.49e-3	5.58e-3	6.63e-3	52.96	42.43	23.86	2879	2471	1301
0.4	8.25e-3	8.45e-3	1.01e-2	7.06e-3	7.21e-3	8.62e-3	66.46	48.19	26.07	3075	2533	1302
0.5	1.00e-2	1.03e-2	1.23e-2	8.56e-3	8.78e-3	1.07e-2	86.74	56.13	29.45	3278	2571	1302

4.6. Number of anchors impact. With constant *radius* (0.033) and the same randomly distributed nodes (3969), Table 4.7 shows the impact of the number of anchors on accuracy and performance. (Noise is not included.) As we can see, when the radio range is sufficiently large, the number of anchors in the network has a very slight impact, improving the estimation accuracy in general, with no obvious impact on algorithm speed. This analysis is beneficial for designers to avoid the cost of deploying unnecessary anchors.

Strategies I2 and I1 produce identical results. I2, I1, and I0 achieve exactly the same 95% error, although when there are fewer than 30 anchors, the added inequality constraints improve the average error consistently.

Table 4.7
Number of anchors impact: nodes = 3969, radius = 0.033, no noise, subproblem_size = 3.

Anchors	Error			95% Error			Time			SDP's		
	I2	I1	I0	I2	I1	I0	I2	I1	I0	I2	I1	I0
5	1.97e-5	1.97e-5	2.09e-5	4.06e-7	4.06e-7	4.06e-7	22.01	22.21	19.05	1320	1320	1319
10	1.98e-5	1.98e-5	2.10e-5	4.10e-7	4.10e-7	4.10e-7	21.77	21.83	19.07	1319	1319	1318
20	2.00e-5	2.00e-5	2.14e-5	3.65e-7	3.65e-7	3.65e-7	22.12	21.80	19.11	1315	1315	1314
30	5.53e-6	5.53e-6	8.24e-6	3.99e-7	3.99e-7	4.01e-7	21.93	21.87	19.30	1314	1314	1313
40	4.15e-6	4.15e-6	4.15e-6	3.62e-7	3.62e-7	3.62e-7	21.97	21.92	19.29	1309	1309	1309
50	3.64e-6	3.64e-6	3.64e-6	3.73e-7	3.73e-7	3.73e-7	22.02	22.23	19.36	1306	1306	1306
60	6.23e-6	6.23e-6	6.23e-6	3.60e-7	3.60e-7	3.60e-7	22.11	22.13	19.49	1303	1303	1303

4.7. Number of anchors impact with noise. With the same *radius* (0.033) and the same randomly distributed nodes (3969), Table 4.8 shows the impact of the number of anchors on accuracy and performance when a *noise_factor* of 0.1 is included in the simulation. With this radio range (sufficiently large), more anchors give only slightly better estimation accuracy, with no obvious impact on algorithm speed in general. Compared with Table 4.7, the presence of noise does add execution time and cause more errors on average.

Regarding strategies I2, I1 and I0, the inequality constraints provide consistent improvement for both average and 95% error at the price of increased execution time. I2 requires more SDP calls than I1, but the improvement in estimation accuracy is very minimal.

Table 4.8
Number of anchors impact: nodes = 3969, radius = 0.033, noise_factor = 0.1, subprob_size = 3.

Anchors	Error			95% Error			Time			SDP's		
	I2	I1	I0	I2	I1	I0	I2	I1	I0	I2	I1	I0
5	2.71e-3	2.72e-3	3.31e-3	2.13e-3	2.14e-3	2.35e-3	32.61	30.80	21.93	1958	1920	1323
10	2.68e-3	2.68e-3	3.23e-3	2.06e-3	2.07e-3	2.28e-3	32.19	30.61	21.96	1916	1877	1321
20	2.66e-3	2.66e-3	3.27e-3	2.10e-3	2.10e-3	2.34e-3	32.09	30.42	21.93	1923	1881	1318
30	2.58e-3	2.59e-3	3.15e-3	2.01e-3	2.02e-3	2.22e-3	31.84	30.28	21.97	1885	1842	1313
40	2.61e-3	2.62e-3	3.29e-3	2.02e-3	2.03e-3	2.26e-3	32.41	30.51	22.27	1925	1862	1309
50	2.58e-3	2.60e-3	3.09e-3	2.03e-3	2.03e-3	2.21e-3	31.44	30.01	22.37	1856	1822	1307
60	2.56e-3	2.58e-3	3.18e-3	2.01e-3	2.02e-3	2.25e-3	32.04	30.42	21.90	1911	1870	1304

4.8. Anchors impact with noise and lower *radius*. With the same randomly distributed nodes and the same noise level but lower *radius* (0.026), Table 4.9 shows the impact of the number of anchors on accuracy and performance. Increased number of anchors results in slightly better estimation accuracy with no obvious impact on algorithm speed in general. In addition, decreased radio range reduces the execution time and causes more errors on average compared with Table 4.8. At the same time we start to see outlier sensors.

The same conclusion can be drawn for I2, I1, and I0 as in section 4.7.

Table 4.9
Number of anchors impact: nodes = 3969, radius = 0.026, noise_factor = 0.1, subprob_size = 5.

Anchors	Out-liers	Error			95% Error			Time			SDP's		
		I2	I1	I0	I2	I1	I0	I2	I1	I0	I2	I1	I0
5	3	3.32e-3	3.33e-3	3.96e-3	2.44e-3	2.45e-3	2.81e-3	24.43	23.10	15.28	1437	1399	920
10	2	3.19e-3	3.19e-3	3.77e-3	2.39e-3	2.39e-3	2.75e-3	24.66	23.28	15.24	1470	1434	929
20	9	3.17e-3	3.18e-3	3.81e-3	2.30e-3	2.31e-3	2.66e-3	24.59	23.11	15.29	1452	1408	930
30	4	3.13e-3	3.14e-3	3.65e-3	2.30e-3	2.30e-3	2.61e-3	24.49	23.15	15.33	1429	1401	931
40	2	3.09e-3	3.09e-3	3.65e-3	2.30e-3	2.30e-3	2.63e-3	24.17	22.85	15.22	1389	1355	901
50	2	3.15e-3	3.16e-3	3.59e-3	2.28e-3	2.29e-3	2.56e-3	24.05	22.70	15.16	1397	1360	904
60	7	3.01e-3	3.02e-3	3.51e-3	2.22e-3	2.23e-3	2.51e-3	24.71	23.30	15.33	1429	1385	911

5. Summary and extensions. We have shown that SpaseLoc achieves the aims of accuracy, speed, and scalability with a single processor on very large networks. It takes full advantage of the recent semidefinite programming (SDP) approach of Biswas and Ye [2]. The latter has computational complexity $O(n^p)$, where n is the network size and p is between 3 and 4, but we use it on multiple tiny subproblems to obtain an algorithm with essentially linear complexity. On a 2.4GHz laptop with 1GB memory, SpaseLoc maintains efficiency and provides accurate position estimation for networks with up to 10000 sensors.

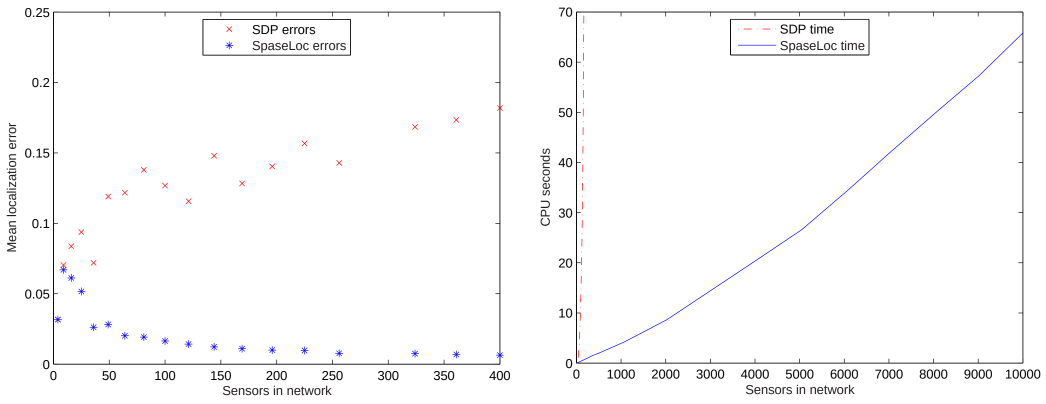

FIG. 5.1. *Accuracy and performance comparison.*

Figure 5.1 compares localization results for our SpaseLoc algorithm and the full SDP approach [2] for various sized networks. The left-hand figure shows a comparison in terms of estimation accuracy for localizing various sizes of networks when sensors are placed at the vertices of an equilateral triangle grid with 0.1 *noise_factor* added to distance measurements (refer to section 4.2). It shows clearly that SpaseLoc provides much improved positioning accuracy.

The right-hand graph summarizes results in terms of execution time on various network sizes. Data for the full SDP method is taken from Table 4.3, and data for SpaseLoc is taken from Table 4.4. The figure confirms near-linear complexity for SpaseLoc.

In Jin [13], SpaseLoc is used as a building block for more general localization algorithms. A dynamic version can estimate moving sensors' locations in real time, and a 3D version extends its utility further. For clustered and distributed environments, it is shown how to use SpaseLoc in parallel (on multiple large subproblems) to obtain essentially linear complexity on clustered networks of unlimited size. Finally, a preprocessor for SpaseLoc has been developed in [13] to localize sensors in anchorless networks.

Acknowledgements. We gratefully acknowledge valuable technical advice from Profs. Henry Wolkowicz, Scott Rogers, Daniel Frances, and two perceptive referees. Thanks also to Dr. Steve Benson for his expert advice on using the DSDP5.0 solver, and to Prof. Kenneth Holmström's for his help in fine-tuning some portions of the MATLAB implementation of SpaseLoc.

REFERENCES

[1] S. J. Benson, Y. Ye, and X. Zhang. DSDP website, http://www-unix.mcs.anl.gov/~benson/ or http://www.stanford.edu/~yyye/Col.html, 1998–2005.

[2] P. Biswas and Y. Ye. *Semidefinite programming for ad hoc wireless sensor network localization*, IPSN 2004, Berkeley, CA, April 26–27, 2004.

[3] P. Biswas and Y. Ye. *A distributed method for solving semidefinite programs arising from ad hoc wireless sensor network localization*, Working Paper, Depts of EE and MS&E, Stanford University, CA, October 30, 2003.

[4] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, SIAM, Philadelphia, 1994.

[5] N. Bulusu, J. Heidemann, and D. Estrin. *GPS-less low cost outdoor localization for very small devices*, Technical Report 00-729, Computer Science Department, University of Southern California, CA, April 2000.

[6] D. Culler and W. Hong. *Wireless sensor networks.* Comm. ACM, Vol. 47, No. 6, June 2004, pp. 30–33.

[7] L. Doherty, L. El Ghaoui, and K. Pister. *Convex position estimation in wireless sensor networks.* Proc. IEEE Infocom 2001, Anchorage, AK, April 2001, pp. 1655–1663.

[8] A. Dragoon. *Small wonders.* CIO Magazine, January 15, 2005.

[9] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. *An empirical study of epidemic algorithms in large-scale multihop wireless networks.* Report UCLA/CSD-TR-02-0013, Computer Science Department, UCLA, CA, 2002.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[11] J. Hightower and G. Boriello. *Location systems for ubiquitous computing*, IEEE Computer, 34(8) (2001), pp. 57–66.

[12] A. Howard, M. J. Mataric, and G. S. Sukhatme. *Relaxation on a mesh: a formalism for generalized localization*, in Proc. IEEE/RSJ International Conf. on Intelligent Robots and Systems (IROS01), 2001, pp. 1055–1060.

[13] H. H. Jin. *Scalable Sensor Localization Algorithms for Wireless Sensor Networks*, Ph.D. thesis, University of Toronto, Toronto, Canada, 2005. (Joint research conducted at Stanford University.)

[14] M. Lawlor. *Small systems, big business*, Signal Magazine, January 2005.

[15] Matlab 6.5, Release 13 with Service Pack 1. The MathWorks, Inc., Natick, MA, 2003.

[16] D. Niculescu and B. Nath. *Ad hoc positioning system*, IEEE GlobeCom, November 2001, pp. 2926–2931.

[17] A. Ricadela. *Sensors everywhere*, Information Week, January 24, 2005.

[18] C. Savarese, J. Rabaey, and K. Langendoen. *Robust positioning algorithm for distributed ad hoc wireless sensor networks*, in USENIX Technical Annual Conf., Monterey, CA, June 2002.

[19] A. Savvides, C.-C. Han, and M. B. Srivastava. *Dynamic fine-grained localization in ad hoc networks of sensors*, in ACM/IEEE International Conf. on Mobile Computing and Networking (MOBICON), July 2001, pp. 166–179.

[20] A. Savvides, H. Park, and M. B. Srivastava. *The bits and flops of the n-hop multilateration primitive for node localization problems*, in 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA, 2002, ACM Press, pp. 112–121.

[21] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. *Localization from mere connectivity*, MobiHoc 2003, Anapolis, MD, June 2003, ACM Press.

[22] J. F. Sturm. *Let SeDuMi seduce you*, http://fewcal.kub.nl/sturm/software/sedumi.html, October 2001.

[23] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. *Habitat monitoring with sensor networks.* Communications of the ACM, Vol. 47, No. 6, June 2004, pp. 34–44.

[24] P. Tseng. *SOCP relaxation for nonconvex optimization*, presented at ICCOPT 1, RPI, Troy, NY, August 2–5, 2004.