EURASIP Journal on Advances in Signal Processing

RESEARCH Open Access

Distributed localization using Levenberg-Marquardt algorithm

Shervin Parvini Ahmadi[1]*, Anders Hansson[1] and Sina Khoshfetrat Pakazad[2]

*Correspondence:
shervin.parvini.ahmadi@liu.se
[1]Department of Electrical Engineering, Linköping University, Linköping, Sweden
Full list of author information is available at the end of the article

Abstract
In this paper, we propose a distributed algorithm for sensor network localization based on a maximum likelihood formulation. It relies on the Levenberg-Marquardt algorithm where the computations are distributed among different computational agents using message passing, or equivalently dynamic programming. The resulting algorithm provides a good localization accuracy, and it converges to the same solution as its centralized counterpart. Moreover, it requires fewer iterations and communications between computational agents as compared to first-order methods. The performance of the algorithm is demonstrated with extensive simulations in Julia in which it is shown that our method outperforms distributed methods that are based on approximate maximum likelihood formulations.

Keywords: Distributed localization, Maximum likelihood estimation, Message passing, Dynamic programming, Levenberg-Marquardt, Nonlinear least-squares

1 Introduction

The problem we investigate in this paper is that of determining the locations of all sensors in a network, given noisy distance measurements between some sensors. It is usually assumed that the positions of some of the sensors are known, which are referred to as anchors. This is called a localization problem. Specifically, we present a distributed algorithm that solves the maximum likelihood estimation problem for localization when the measurements are corrupted with Gaussian noise. Our algorithm is based on applying the Levenberg-Marquardt algorithm to the resulting nonlinear least-squares problem. It requires a good initialization, and we initialize it with an approximate estimate obtained from the algorithm proposed in [1], which is based on a convex relaxation of our nonlinear least-squares problem formulation.

Over the years there has been a considerable interest in wireless sensor network localization using inter-sensor distance or range measurements [2–4]. Wireless sensor networks are small and inexpensive devices with low energy consumption and computing resources. Each sensor node comprises sensing, processing, transmission, and power units, some with mobilizers [5, 6]. The applications are many, e.g., natural disaster relief, patient tracking, military targets, automated warehouses, weather monitoring, smart space, monitoring environmental information, detecting the source of pollutants, and mobile peer-to-peer computing to mention a few, as well as underwater applications [7]. The information collected through a sensor network can be used more effectively if it is know where it is coming from and where it needs to be sent. Therefore, it is often very useful to know the positions of the sensor nodes in a network. The use of global positioning system is a very expensive solution for this [8]. Instead, techniques to estimate node positions are needed that rely just on the measurements of distances and angles between neighboring nodes [2, 9]. Deployment of a sensor network for these applications can be done in a random fashion, e.g., dropped from an airplane in a disaster management application, or manually, e.g., fire alarm sensors in a facility or sensors planted underground for precision agriculture [5].

Localization in this setting is mostly based on optimizing some cost function which is dependent on the model uncertainties. The most widely used technique is based on maximizing a likelihood function, know as maximum likelihood estimation, which in general is equivalent to a non-convex optimization problem of high dimensionality [10, 11]. Both centralized and distributed algorithms have been used to solve the problem [12]. The centralized algorithms require that each sensor/agent sends its information to a central unit where an estimate of the sensors position can be computed using for example second-order optimization methods. Then the results are sent back to the agents.

The disadvantage of these algorithms is that the processing in the central unit can be computationally heavy, specially when the number of sensors are large. Distributed algorithms overcome this obstacle. These algorithms enable us to solve the problem through collaboration and communication between several computational agents, which could correspond to the sensors, without the need for a centralized computational unit. The disadvantage of these algorithms is that they might not result in as accurate estimates of the positions as for centralized algorithms. Moreover, they might require excessive communication between the senors, specially for large network sizes. The algorithm we propose in this paper is somewhere between a centralized and distributed algorithms in the sense that, instead of having one central unit as in a centralized algorithms, the sensors are grouped together in a structured way, where one computational agent is assigned for each group. The sensors then send their measurements to their groups computational agent and those agents in turn carry out the computations by communicating with one another. As a result, neither the computations in the proposed algorithm are as heavy as in centralized algorithms, nor the communication burden is as intensive as in distributed algorithms in which all the adjacent sensors communicate together in order to find a solution to the localization problem.

There have been various techniques developed to distribute the computations. We will now survey different distributed methods for localization. First, we discuss approaches which are based on the original non-convex maximum likelihood formulation. They all solve the maximum likelihood problem exactly. The authors in [13], propose a distributed multidimensional scaling algorithm which minimizes multiple local nonlinear least-squares problems. Each local problem is solved using quadratic majorizing functions. In [14], the authors present two distributed optimization approaches, namely a distributed gradient method with Barzilai-Borwein step sizes, and a distributed Gauss-Newton approach. In [15], a decentralized algorithm is devised based on the incremental subgradient method. In [11], the authors propose a distributed alternating direction method of multipliers approach. To this end, an equivalent equality constrained problem of the original nonlinear least-square problem is considered by introducing duplicate variables in the optimization problem which allows for a distributed solution. In [16], the authors reformulate the problem to obtain a gradient Lipschitz cost which in turn enables them to propose a distributed algorithm based on a majorization-minimization approach. The main shortcoming of the surveyed approaches is that they take many iterations to converge, and hence are slow, since many communications are required to reach a solution. The reason for this is either because they are based on first-order optimization methods, or as is the case for the Gauss-Newton method, a consensus algorithm is used in order to compute the search direction. Also it is difficult to effectively initialize the algorithms, since the likelihood function might have several local maxima. The latter problem can be overcome by using some approximate algorithm for localization which is easy to initialize. Then the solution from this approximate method is used to initialize the non-convex optimization problem solver. Good approximate problems can be obtained from convex relaxations of the maximum likelihood problem.

We will now continue the survey with methods based on convex relaxations of the maximum likelihood formulation. These are not only used for initialization of non-convex formulations, but are also of interest per se assuming that the approximation provides a good enough approximate localization. A good survey of semi-definite programming relaxation methods is given in [4]. The authors in [3], and [12], use the relaxation in [4] to devise distributed algorithms based on the alternating direction method of multipliers and second-order cone programming approaches, respectively. A nice property of these algorithms is that they have convergence guarantees. This, however, comes at the cost of solving a semi-definite programming at every iteration of the algorithm which imposes a substantial computational burden. In [17], a distributed algorithm in which only linear system of equations have to be solved at each iteration is proposed. They distribute the computations using message-passing over a tree. Another way to decrease the computational cost at each iteration is to consider a disk relaxation of the localization problem instead of an semi-definite programming relaxation. Based on this idea, the authors in [1] and [18], devise distributed algorithms for solving the resulting problem which rely on projection based methods and Nestrov's optimal gradient method, respectively. In [19], the authors propose a hybrid approach based on the disk relaxation in [1] and a semi-definite programming relaxation, by fusing range and angular measurements. It should be stressed that the solutions from relaxation based methods do not provide global optima. The quality of the solutions are highly dependent on how tight the relaxation is.

As mentioned above solutions from relaxed formulations may be used to initialize the non-relaxed formulations. In [2], a semi-definite programming relaxation of the problem, combined with a regularization term is used to initialize a gradient-descent method for solving the exact maximum likelihood problem. In [20], the authors propose a hybrid solution to the localization problem. To this end, they apply a distributed alternating direction method of multipliers approach in two stages. In the first stage, they use the disk relaxation formulation of the localization problem as in [1], and then there is a smooth transition to the second stage where they use the original non-convex formulation as in [11].

Although the algorithm in [20], has faster convergence rate than what is presented in [1], the number of communications per sensor is not significantly lower, and in addition to that there is an extra communication overhead because of the existence of several duplicate variables which needs to be passed among the sensors. Notice that the number of duplicate variables in each sensor is proportional to the number of sensors that a senor can communicate with, which causes a considerable amount of computations and communications, specially if the network size is large. Also note that for the algorithms in both [1], and in [20], the computations are distributed in such a way that each sensor has to carry out its own computations and exchange messages with adjacent sensors. The authors in [20] argue that this way of distributing the computations might require an excessive communication burden, specially for large network sizes. Because of this, they discuss the possibility of devising a regional reinterpretation of their algorithm, where the sensors are partitioned in regions and there is one computational agent per region which is responsible for carrying out the computations and exchanging messages with the adjacent computational agents. We will see later that in our proposed algorithm, we also distribute the computations in such a way that not every sensor has to be involved in the computations.

In this paper, we propose a distributed algorithm based on the Levenberg-Marquardt method [21], with a localization accuracy which is better than the algorithm in [1], but with much fewer communications per sensor/agent. The accuracy is better since we solve the maximum likelihood problem and not only an approximation of it. This will show that the claim in [19], that the algorithm in [1] has equal localization accuracy compared to the one in [20], is debatable. We use an approximate estimate obtained from the relaxation based algorithm presented in [1], as the initial starting point for our algorithm. We will see that since the number of communications between agents in our algorithm is far less than the algorithm presented in [1], our algorithm can be utilized on top of the algorithm in [1], in order to improve the estimate in terms of accuracy with much less iterations than what are used in [1], and achieving better accuracy. Note that both algorithms in [1] and [20] which are based on Nesterov's gradient and alternating direction method of multipliers approaches, respectively, are first-order methods whereas the Levenberg–Marquardt algorithm is a pseudo-second-order method as it uses approximate Hessian information. It is known that in general second-order methods require fewer iterations in order to converge than first-order methods. The reason is that second-order methods use both gradient and curvature information of the objective function, whereas first-order methods rely solely on the gradient of the objective function. As a result the number of communications between agents in the distributed Levenberg-Marquardt algorithm is expected to be lower than for the algorithms in [1] and [20].

1.1 Contributions

We propose a distributed algorithm for localization that solves the localization problem to highest accuracy using few communication and computations.

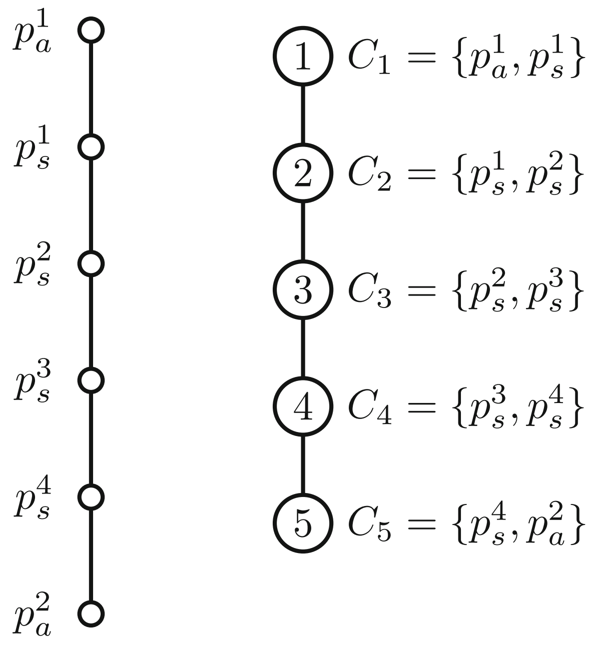
1.2 Example

In order to introduce the notation and to exemplify what results will be derived, a simple one-dimensional example will be considered. Relevant applications of this are e.g. a metro line in-between two stations with anchors at the stations or a mine tunnel with anchors at the intersection of tunnels. The anchors are positioned at p_a^1 and p_a^2 and the position of the other sensors $p_s^j, j = 1, 2, 3, 4$, are such that $p_a^1 \leq p_s^1 \leq \cdots \leq p_s^4 \leq p_a^2$. Moreover, we assume that each sensor can measure the distance of the adjacent sensors. We depict this in what is known as the inter-sensor measurement graph shown to the left in Fig. 1. The nodes represent the sensors and anchors, and there is and edge between two nodes if they can measure the distance to one another. Assume that we are given measurements R_{ij} between sensors i and j and measurements Y_{ij} between sensors i and anchors j with Gaussian measurement errors with zero mean and unit variance. Then, the maximum-likelihood problem of estimating the positions of the sensors is equivalent to

$$\begin{aligned}\underset{P}{\text{minimize}} \quad & \left(p_s^1 - p_a^1 - \mathcal{Y}_{11}\right)^2 + \left(p_s^2 - p_s^1 - \mathcal{R}_{12}\right)^2 \\ & + \left(p_s^3 - p_s^2 - \mathcal{R}_{23}\right)^2 + \left(p_s^4 - p_s^3 - \mathcal{R}_{34}\right)^2 \\ & + \left(p_a^2 - p_s^4 - \mathcal{Y}_{42}\right)^2\end{aligned}$$

where $P = [p_s^1 \ldots p_s^4]$, which is a linear least-squares problem.

The maximal subgraphs of the inter-sensor measurement graph in Fig. 1, which are complete, i.e. contains an edge from every node to every other node, are called cliques and given by $C_1 = \{p_a^1, p_s^1\}$, $C_2 = \{p_s^1, p_s^2\}$, $C_3 = \{p_s^2, p_s^3\}$, $C_4 = \{p_s^3, p_s^4\}$ and $C_5 = \{p_s^4, p_a^2\}$. They can be arranged in a tree as is seen to the right in Fig. 1. This tree is called a clique tree. It is not unique, but it can always be arranged in such a way that any element in the intersection of two cliques will also be elements of cliques on the path between the two cliques. This is called the clique intersection property. It is not possible in general for any inter-sensor measurement graph to derive a clique tree. For this to be possible, the graph has to be what is called chordal. We will discuss this in more detail later. However, for our example, the graph is chordal, i.e. any cycle of length four or more in the graph has a chord. It is now possible to solve the least-squares problem over this clique tree by using each of the cliques as computational agents. This is done by associating terms of the


p_a^1
p_s^1
p_s^2
p_s^3
p_s^4
p_a^2
1 $C_1 = \{p_a^1, p_s^1\}$
2 $C_2 = \{p_s^1, p_s^2\}$
3 $C_3 = \{p_s^2, p_s^3\}$
4 $C_4 = \{p_s^3, p_s^4\}$
5 $C_5 = \{p_s^4, p_a^2\}$
Fig. 1 Inter-sensor measurement graph to the left and a corresponding clique tree to the right

objective function with different cliques. A valid assignment is that the variables of the terms that are assigned to a clique should belong to the clique. Therefore, we assign

$$f_1(p_s^1) = \left(p_s^1 - p_a^1 - \mathcal{Y}_{11}\right)^2$$

to C_1,

$$f_2\left(p_s^1, p_s^2\right) = \left(p_s^2 - p_s^1 - \mathcal{R}_{12}\right)^2$$

to C_2,

$$f_3\left(p_s^2, p_s^3\right) = \left(p_s^3 - p_s^2 - \mathcal{R}_{32}\right)^2$$

to C_3,

$$f_4\left(p_s^3, p_s^4\right) = \left(p_s^4 - p_s^3 - \mathcal{R}_{34}\right)^2$$

to C_4, and

$$f_5\left(p_s^4\right) = \left(p_s^4 - p_a^2 - \mathcal{Y}_{23}\right)^2$$

to C_5. Hence, the least-square problem is equivalent to

$$\underset{P}{\text{minimize}} \quad f_1\left(p_s^1\right) + f_2\left(p_s^1, p_s^2\right) + f_3\left(p_s^2, p_s^3\right) + f_4\left(p_s^3, p_s^4\right) + f_5\left(p_s^4\right)$$

We then start with the leaf clique C_5 and its corresponding function $f_5\left(p_s^4\right)$ and minimize it with respect to the variables that are not shared with its parent. There is no such variable and hence the minimization is not carried out. We then let $m_{54}\left(p_s^4\right) = f_5\left(p_s^4\right)$, which is called a message function or value function. This is added to the objective function corresponding to the parent of C_5, i.e. to $f_4\left(p_s^3, p_s^4\right)$. Notice that any quadratic function can be represented with a matrix and a vector, and hence this is the only information that has to be passed to the parent. We then again minimize the resulting function with respect to the variables that are not shared with its parent, i.e. p_s^4. Since the problem is convex and quadratic, this is equivalent of solving a linear equation, and after back substitution of the solution, the objective function value will be a quadratic function of p_s^3, which we denote by $m_{43}\left(p_s^3\right)$. We then add the message function to the objective function corresponding to the parent of C_4 i.e. to $f_3\left(p_s^2, p_s^3\right)$ and repeat the procedure until we reach the root clique. For the root clique C_1, we now can optimize $f_1\left(p_s^1\right) + m_{21}\left(p_s^1\right)$ with respect to the remaining variable p_s^1, where $m_{21}(p_s^1)$ is a message from the child clique. By parsing this solution down the clique tree the remaining optimal variables can be computed assuming that the parametric solutions have been stored in the nodes of the clique tree.

The fact that the problem is convex and quadratic, makes it easy to compute the messages. In general, this is not the case, but we will use this procedure not for the optimization problem itself but for computing the search directions in a non-linear least-square method, in particular the Levenberg-Marquardt method [21]. These equations are linear equations and correspond to a quadratic approximation of the problem at the current iterate of the Levenberg-Marquardt method. All other computations in the Levenberg-Marquardt method also distribute over the clique tree. We see that what we are doing in this example is nothing but serial dynamic programming. In general, the clique tree will not be a chain, and then we will carry out dynamic programming or message passing over a tree, see [22] for details. The clique tree is not unique. For the example we can just as well make C_5 the root and C_1 the leaf. Moreover, we can take

C_3 as root and get two branches with C_1 and C_5 as leafs. This will facilitate parallel computations.

1.3 Outline

In Section 2, we review the maximum likelihood formulation of the localization problem. In Section 3, we discuss how to find the clique tree and how to assign subproblems, in order to distribute the computations for a general optimization method. In Section 4, we review the Levenberg-Marquardt algorithm for solving non-linear least-square problems. In Section 5, we discuss how to distribute the computations in the Levenberg-Marquardt algorithm using the clique tree. Numerical experiments are presented in Section 6 ,and we conclude the paper in Section 7.

1.4 Notations and definitions

We denote by $\mathbb{R}$, the set of real scalars and by $\mathbb{R}^{n\times m}$, the set of real $n \times m$ matrices. The transpose of a matrix A is denoted by A^T. We denote the set of positive integers $\{1, 2, \ldots, p\}$, with $\mathbb{N}_p$. With x_i, we denote the ith componenet of the vector x. For a square matrix X, we denote with $\text{diag}(X)$, a vector with its elements given by the diagonal elements of X.

A graph is denoted by $G(V, \mathcal{E})$, where $V = \{1, \ldots, n\}$ is its set of vertices or nodes and $\mathcal{E} \subseteq V \times V$denotes its set of edges. Vertices $i, j \in V$ are adjacent if $(i, j) \in \mathcal{E}$, and we denote the set of adjacent vertices of i by $Ne(i) = \left\{j \in V | (i, j) \in \mathcal{E}\right\}$. A graph is said to be complete if all its vertices are adjacent. An induced graph by $V' \subseteq V$ on $Q(V, \mathcal{E})$, is a graph $Q_I(V', \mathcal{E}')$, where $\mathcal{E}' = \mathcal{E} \cap V' \times V'$. A clique C_i of $Q(V, \mathcal{E})$ is a maximal subset of V that induces a complete subgraph on Q, i.e., no clique is properly contained in another clique [23]. Assume that all cycles of length at least four of $Q(V, \mathcal{E})$ have a chord, where a chord is an edge between two non-consecutive vertices in a cycle. This graph is then called chordal [24, Ch. 4]. It is possible to make graphs chordal by adding edges to the graph. The resulting graph is then referred to as a chordal embedding. Let $C_Q = \left\{C_1, \ldots, C_q\right\}$ denote the set of its cliques, where q is the number of cliques of the graph. Then there exists a tree defined on C_Q such that for every $C_i, C_j \in C_Q$ where $i \neq j$, $C_i \cap C_j$ is contained in all the cliques in the path connecting the two cliques in the tree. This property is called the clique intersection property [23]. Trees with this property are referred to as clique trees.

2 Maximum likelihood localization

The localization problem that we consider in this paper can be formulated as a network of n_s sensors with unknown positions $p_s^i \in \mathbf{R}^d$, $i \in \mathbb{N}_{n_s}$, and n_a anchors with known positions $p_a^i \in \mathbf{R}^d$, $i \in \{n_s + 1, \ldots, n_s + n_a\}$, where $d \in \{1, 2, 3\}$ is the dimension of the localization problem. The goal is to find the position of the sensors. We assume that the sensors are capable of performing computations and that they also can measure their distance to some of the adjacent sensors and/or anchors. However, later we will see it is enough to assume that some of the sensors are capable of performing computations for the proposed distributed algorithm. Let us define the set of neighbors of each sensor i, $Ne_r(i) \subseteq \mathbb{N}_{n_s}$, as the set of sensors to which this sensor has an available range measurement. In a similar fashion let us denote the set of anchors to which sensor i can measure its distance to by

$Ne_a(i) \subseteq \{n_s + 1, \ldots, n_s + n_a\}$. Define the inter-sensor and anchor range measurements for each sensor, $i \in \mathbb{N}_{n_s}$,

$$\begin{aligned} \mathcal{R}_{ij} &= \mathcal{D}_{ij}\left(p_s^i, p_s^j\right) + E_{ij}, \quad j \in Ne_r(i), \\ \mathcal{Y}_{ij} &= \mathcal{Z}_{ij}\left(p_s^i, p_a^j\right) + V_{ij}, \quad j \in Ne_a(i) \end{aligned} \tag{1}$$

respectively, where $\mathcal{D}_{ij} = ||p_s^i - p_s^j||_2$ and $\mathcal{Z}_{ij} = ||p_s^i - p_a^j||_2$ are the noise-free sensor distance and the noise-free anchor-sensor distance, respectively. The quantities $E_{ij} \sim \mathcal{N}(0, \sigma)$ and $V_{ij} \sim \mathcal{N}(0, \sigma)$ are the measurement noises. It is assumed that the inter-sensor and the anchor-sensor measurement noises are independent. With these definitions, the maximum likelihood problem for localization can be written as

$$\underset{P}{\text{minimize}} \quad \frac{1}{2}\left\{\sum_{i=1}^{n_s}\Big(\sum_{j\in Ne_r(i)} \Omega_{ij}^2\left(p_s^i, p_s^j\right) + \sum_{j\in Ne_a(i)} \Omega_{ij}^2\left(p_s^i\right)\Big)\right\} \tag{2}$$

where $P = \left(p_s^1, \ldots, p_s^{n_s}\right) \in \mathbb{R}^{dn_s}$, and where

$$\begin{aligned} \Omega_{ij}\left(p_s^i, p_s^j\right) &= \mathcal{D}_{ij}\left(p_s^i, p_s^j\right) - \mathcal{R}_{ij}, \quad j \in Ne_r(i) \\ \Omega_{ij}\left(p_s^i\right) &= \mathcal{Z}_{ij}\left(p_s^i, p_a^j\right) - \mathcal{Y}_{ij}, \quad j \in Ne_a(i) \end{aligned}$$

for $i \in \mathbb{N}_{n_s}$. The problem is a nonlinear least-square problem and hence is non-convex. It is in general NP hard [25], and although the problem is guaranteed to have a global minimum [1], it is difficult find it [3]. The goal, therefore, is to find good local minimum for the problem.

There are also work reported in which only sensor to anchor measurements are considered and not inter-sensor ones, see, e.g. [26–28], in which a range-free based convex method, a convex relaxation based method using range measurements and a sensor selection based method using range and angle measurements, are proposed, respectively.

3 Clique tree and assignment strategy

In order to solve the problem in (2) in a distributed way, similar to the approach for the one-dimensional example in Section 1, we base our computations on a clique tree which will be used as the computational graph.

Let us assume that if sensor/anchor i can measure its distance to sensor/anchor j, so can j measure its distance to i. This then allows us to describe the range measurement available using an undirected graph $G(V, \mathcal{E})$ with vertex set $V = \{1, \ldots, n_s + n_a\}$ and edge set $\mathcal{E} \subset V \times V$. An edge $(i,j) \in \mathcal{E}$, if and only if there is a range measurement between i and j. We assume that the graph is connected. Consider the network with 4 sensors and 4 anchors in Fig. 2. The sparsity graph for this network is shown in the top graph of Fig. 3. The graph is not chordal and therefore as mentioned in Section 1, we first find a chordal embedding of the graph before we are able to obtain a corresponding clique tree. One possibility is to add an edge between nodes 2 and 3 to obtain a chordal graph. A corresponding clique tree is shown in the bottom graph of Fig. 3. For more complicated networks, one may use general purpose algorithms to generate the chordal embedding and subsequently the clique tree. Although the problem of finding

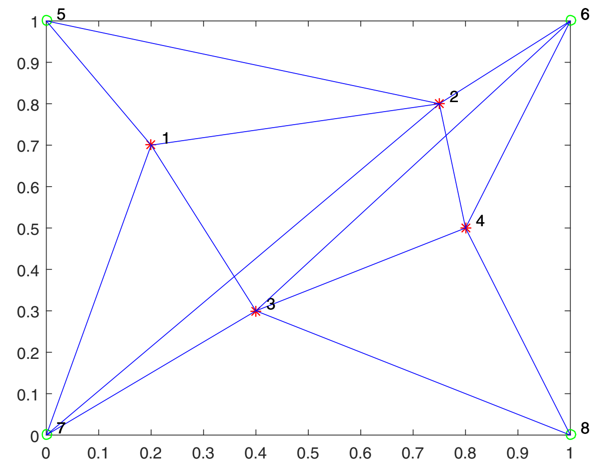


Fig. 2 A sensor network with 4 sensors (red crosses) and 4 anchors (green circles). An edge between two nodes, implies existence of a range measurement between two nodes

a chordal embedding of a graph by adding a minimal number of edges is NP-hard, sub-optimal methods can be used [29]. One such method is given in [30]. A MATLAB code for chordal embedding and the corresponding clique tree which is based on the approach proposed in [30], is provided in [31]. Once the clique tree is found, we choose one of the cliques as the root of the tree. Once the root of the tree is specified, the terms of the problem in (2) are assigned to the cliques. We use the assignment strategy given in Algorithm 1. The purpose of the algorithm is to have a balanced distribution of the terms in the objective function. The resulting assignment relies on the ordering of the cliques. Consequently, different ordering of the cliques may result in different assignments of terms in the objective function.

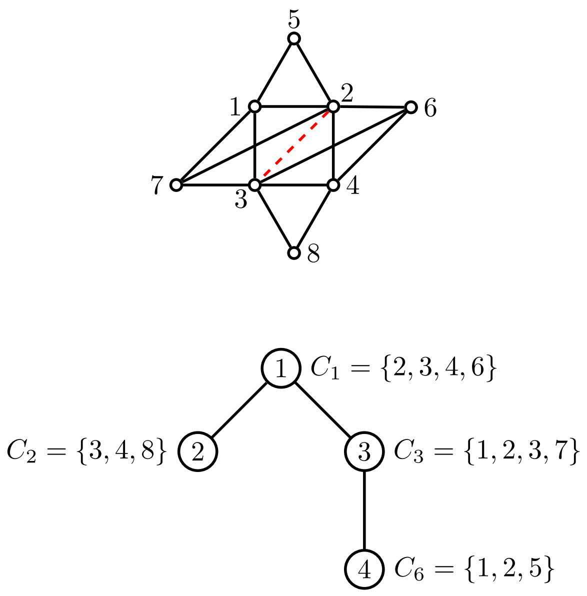


Fig. 3 Sparsity graph (on the top) and Clique tree (on the bottom) for the network in 2

Algorithm 1 A simple assignment strategy

```
1: Given the graph $G(V, \mathcal{E})$, the cliques $\{C_1, \ldots, C_q\}$ and $\Omega = \{\Omega_{ij} \mid (i,j) \in \mathcal{E}_r\}$ for the
   problem in (2)
2: while $\Omega$ is not empty do
3:   for $k = 1, \ldots, q$ do
4:     for $(i,j) \in C_k$ do
5:       if $\Omega_{ij}$ is not assigned then
6:         Assign it to agent $k$
7:         $\Omega = \Omega \backslash \Omega ij$
8:         break
9:       end if
10:     end for
11:   end for
12: end while
```

The clique tree can be constructed in a distributed way. We start with the anchors and they start by communicating with sensors that they can communicate with. This will give us the initial knowledge of the sparsity graph. Then the senors that the anchors can communicate with can do the same and tell the anchors about what neighbors they have found, and then it goes on like that. This will give us the sparsity graph, and we can centrally at the anchors compute the clique tree and distribute out that information. It is clear that if we lose a measurement, assuming that does not make the sparsity graph disconnected, then we can still have the same clique tree, by just making an embedding of the missing measurement. However, if we get an extra measurement, then if we want to include it we have to recompute the clique tree. However, if the new sensor is only providing measurements to sensors in the same clique, it can form a new clique with the sensors it can communicate with, and the clique tree can be easily augmented with this clique. If all the sensors of a clique fail, then the sensor network and the corresponding sparsity graph will be disconnected, see e.g. Theorem 3.7. in [32], which violates our assumptions.

Remark 1 *Note the each clique is a grouping of sensors. Adding artificial edges between sensors in order obtain a chordal embedding is a way to virtually group them. In other words, the added edges corresponds to saying that terms in the objective function are function of variables which they are actually not. Later when we assign different terms of the objective function to the clique tree those added edges are of no relevance. The only purpose of adding edges was to be able to obtain a clique tree.*

Remark 2 *It should be noted that for the distributed Levenberg–Marquardt algorithm which is discussed later, what clique is chosen as root does not affect the number of communications required for converging to a solution. However, it affects how computations can be carried out in parallel, see [33].*

4 Levenberg–Marquardt algorithm

We will now discuss the Levenberg-Marquardt algorithm and how it can be used to solve the maximum likelihood formulation in (2). Consider the nonlinear least-square problem

$$\underset{x}{\text{minimize}} \quad F(x) \tag{3}$$

where $F(x) = \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2$ and $f_i : \mathbb{R}^n \to \mathbb{R}$. The problem in (2) can be written as in (3), where each and every Ω_{ij} in (2) corresponds to a f_i in (3) and $x = P$. We assume that the terms f_is are differentiable. Necessary condition for a local minimum x is that

$$\nabla F(x) = \sum_{i=1}^{m} f_i(x).\nabla f_i(x) = J(x)^T f(x) = 0$$

where $J(x)_{m\times n}$ is the Jacobian of $f(x) = (f_1(x), \dots, f_m(x))$. One of the well-known and efficient methods to solve this problem is the Levenberg-Marquardt algorithm which is a variation of the Gauss-Newton algorithm, [21, Ch. 10]. The method has been very successful in practice [34, Ch. 10], with a convergence rate which is better than linear and sometimes it can even be quadratic.

Now let us linearize $F(x)$ in the neighborhood of x as

$$F(x + \Delta x) \simeq F_{lin}(\Delta x) = F(x) + f(x)^T J(x)\Delta x + \frac{1}{2}\Delta x^T J(x)^T J(x)\Delta x \tag{4}$$

Applying the Gauss-Newton algorithm solves the problem in (3) in an iterative fashion by minimizing (4) at each iterate, i.e.

$$\underset{\Delta x}{\text{minimize}} \quad \frac{1}{2}\Delta x^T J(x)^T J(x)\Delta x + f(x)^T J(x)\Delta x \tag{5}$$

A drawback, however, with this approach is that a nearly rank-deficient $J(x)$ may lead to ill-conditioning. One can circumvent this by using the Levenberg-Marquardt algorithm where we solve the problem in (3) in an iterative fashion by minimizing a damped version of (4) at each iterate, i.e.

$$\underset{\Delta x}{\text{minimize}} \quad \frac{1}{2}\Delta x^T (J(x)^T J(x) + \mu I)\Delta x + f(x)^T J(x)\Delta x \tag{6}$$

where μ is a damping parameter and the current iterate of x is updated by adding Δx to it. The size of the damping parameter μ determines the behavior of the algorithm, meaning that for large values of μ, the algorithm behaves like the steepest descent method which is suitable when the current iterate is far from the solution, whereas for small values of μ the algorithm behaves like the Gauss-Newton method which is suitable when the current iterate is close to the solution. In addition, it should be pointed out that in the final iterations of the algorithm if the value $F(x)$ in (3) is very small, then the algorithm behaves like the Newton method. The reason follows from the fact that if f_is are close to zero, then $J^T(x)J(x)$ is a good approximation of the Hessian $\nabla^2 F(x)$ since

$$\nabla^2 F(x) = J(x)^T J(x) + \sum_{i=1}^{m} f_i(x) f_i''(x) \approx J(x)^T J(x)$$

As a result the obtained direction is a Newton direction.

The strategy for updating μ is controlled by a parameter called $\mathcal{Q}$ defined as

$$\mathcal{Q} = \frac{F(x) - F(x + \Delta x)}{F_{lin}(0) - F_{lin}(\Delta x)} = \frac{F(x) - F(x + \Delta x)}{\frac{1}{2}\Delta x^T(\mu\Delta x - J(x)^T f(x))} \tag{7}$$

where Δx is the solution to the above optimization problem. For details, see [35]. It should be noted that this strategy is inherited from the well-known trust-region method and also note that the Levenberg-Marquardt algorithm is sometimes viewed as a trust-region method. See [21, Ch. 10], for details. A suitable choice for the initial value of μ is

$$\mu_0 = \tau \times ||\text{diag}(J(x_0)^T J(x_0))||_\infty$$

where the value of τ, as suggested in [35], depends on how good the approximation x_0 is compared to the local minimizer x^*. If it is known to be a good approximation, then a small value can be chosen, e.g. 10^{-6}, otherwise one can choose 10^{-3} or even a larger value.

5 Distributed computations

We will now discuss how the different computations in the Levenberg-Marquardt algorithm can be distributed over the clique tree. Since each f_i corresponds to a Ω_{ij}, we have also assigned the f_is to different cliques of the clique tree. Let ϕ_k be the set of i for which f_i have been assigned to clique C_k. Define $\bar{F}_k\left(x_{C_k}\right) = \frac{1}{2}\sum_{i\in\phi_k}\left(f_i(x)\right)^2$, where x_{C_k} is the sub-vector of x that contains those components of x that $f_i(x)$ for $i \in \phi_k$ depend on. We will make this dependence somewhat more clear by defining the vector valued functions $\bar{f}^k\left(x_{C_k}\right) = \left(\bar{f}_i^k\left(x_{C_k}\right)\right)_{i\in\phi_k}$ for $k \in \mathbb{N}_q$, where $\bar{f}_i^k : \mathbf{R}^{|C_k|} \to \mathbf{R}$ are defined such that $\bar{f}_i^k\left(x_{C_k}\right) = f_i(x)$ for all $x \in \mathbf{R}^n$, where $i \in \phi_k$, $k \in \mathbb{N}_q$. Then we have $\bar{F}_k\left(x_{C_k}\right) = \frac{1}{2}\sum_{i\in\phi_k}\left(\bar{f}_i^k\left(x_{C_k}\right)\right)^2$. Morover,

$$F(x) = \sum_{k=1}^{q} \bar{F}_k\left(x_{C_k}\right) \tag{8}$$

We see that we have obtained a sum of nonlinear least squares objective functions that are coupled through common components of x. Now let $J_k\left(x_{C_k}\right)$ be the Jacobains of $\bar{f}^k$, and let us define the matrices E_k as the zero-one matrices that are such that $E_k x = x_{C_k}$, for all $x \in \mathbf{R}^n$, $k \in \mathbb{N}_q$. It then follows that

$$\begin{aligned}\nabla F(x) &= \sum_{k=1}^{q} E_k^T J_k\left(x_{C_k}\right)^T \bar{f}^k\left(x_{C_k}\right)\\ J(x)^T J(x) &= \sum_{k=1}^{q} E_k^T J_k\left(x_{C_k}\right)^T J_k\left(x_{C_k}\right) E_k\end{aligned} \tag{9}$$

We see how the matrixes E_k distribute the gradients and the approximate Hessians of the individual functions for the different cliques over the gradient vector and the approximate Hessian matrix of the overall problem.

We will now discuss how the above structure can be used to solve the problem in (6) in a distributed way using the clique tree. The only remaining challenge is how to distribute μI over the cliques. To this end, we introduce modifications of E_k that we call $\bar{E}_k$. They are obtained by identifying the rows which E_k have in common with $E_{\text{par}(k)}$, where par(k) is the parent of the kth clique in the clique tree. Then $\bar{E}_k$ is defined to be equal to E_k, except for these rows, which are set equal to zero. Let us also define $\Delta x_{C_k} = E_k \Delta x$. It is then straightforward to conclude that the problem in (6) can be written as

$$\underset{\Delta x}{\text{minimize}} \quad \sum_{k=1}^{q} \frac{1}{2}\Delta x_{C_k}^T H_k \Delta x_{C_k} + r_k^T \Delta x_{C_k} \tag{10}$$

where

$$H_k = E_k^T J_k \left(x_{C_k}\right)^T J_k \left(x_{C_k}\right) E_k + \mu \bar{E}_k \bar{E}_k^T$$
$$r_k = J_k \left(x_{C_k}\right)^T \bar{f}^k \left(x_{C_k}\right)$$

Notice that this optimization problem has the same sparsity graph and corresponding clique tree as the original problem in (2). Therefore, Δx can be obtained using message passing over the clique tree. See [22] for details regarding message passing.

A distributed version of the Levenberg-Marquardt algorithm is presented in Algorithm 2, in which it is straightforward to show that $||\nabla F(x)||$ and $\mathcal{Q}$ in (7) can be calculated distributedly as

$$||\nabla F(x)|| = \sqrt{\sum_{k=1}^{q} ||r_k||^2}$$
$$\mathcal{Q} = \frac{\sum_{k=1}^{q} \bar{F}_k \left(x_{C_k}\right) - \bar{F}_k \left(x_{C_k} + \Delta x_{C_k}\right)}{\frac{1}{2}\sum_{k=1}^{q} \mu \Delta x^T \bar{E}_k^T \bar{E}_k \Delta x - \Delta x^T E_k^T r_k} \tag{11}$$

Here, $\bar{E}_k \Delta x$ contains a subset of the components in Δx_{C_k}, which is only available in clique C_k and not its parent clique $C_{\mathrm{par}(k)}$.

Remark 3 *It should be pointed out that the proposed algorithm requires the objective function to be differentiable, which is not the case for the problem in (2) because of the non differentiability of* $\mathcal{D}_{ij}\left(p_s^i, p_s^j\right)$ *and* $\mathcal{Z}_{ij}\left(p_s^i, p_a^j\right)$ *at* $p_s^i = p_s^j$ *and* $p_s^i = p_a^j$. *Nevertheless, we can still use the algorithm by imposing an extra condition in Line 6 of Algorithm 2 such that the* $x_{C_{q_i}}$ *update is acceptable only if the next iterate satisfies* $p_s^i \neq p_s^j$ *and* $p_s^i \neq p_a^j$ *for all* $i \neq j$. *The authors in [36], discuss why Quasi-Newton methods are practical and efficient for optimizing non-smooth functions.*

Remark 4 *It is worth mentioning that for Gauss-Newton method in [14], which is a special case of the Levenbergh-Marquardt algorithm when* $\mu = 0$, *the search directions are not computed as efficiently as with the message passing approach used here.*

Remark 5 *Concerning the computational complexity of the distributed method and its centralized counterpart, we first realize that the centralized counterpart of the algorithm is exactly the same as Algorithm 2, expect that the search direction in Line 3 and* $\mathcal{Q}$ *in Line 4 are computed using Problem 6 and Eq. 7, respectively, and the variable update in Line 6 is done in a centralized manner. Given that at each iteration of both methods, the resulting search directions (Problem 6 for centralized and Problem 10 for distributed method) and the* $\mathcal{Q}$ *values (Eq. 7 for centralized and Eq. 11 for distributed method) are identical, the distributed method will converge to the same solution as its centralized counterpart. Hence in order to compare the complexity and the computational cost the methods, it is enough to compare them for one iteration of the algorithm. Next we compare the computational complexity of Line 3 and Line 4 in Algorithm 2 for both methods. Line 3, i.e. computation of search directions, which is the major computational burden for both methods, is carried out by solving the linear system of equations* $(J(x)^T J(x) + \mu I)\Delta x = -f(x)^T J(x)$. *In the centralized method, this is typically done by factorizing* $J(x)^T J(x) + \mu I$, *which is the dominant computation, followed by back/forward substitutions. Common factorizations for this purpose are* LDL^T, *LU, and QR factorizations, which lead to a computational complexity of*

Algorithm 2 Distributed Levenberg–Marquardt algorithm using the clique tree

```
1: Given the clique tree, $k = 0$, $\nu = 2$, $x = x_0$, $\epsilon$ and $\mu = \mu_0$
2: while $||\nabla F(x)|| > \epsilon$ do
3:     Solve (10) using message passing over the clique tree
4:     Calculate $\mathcal{Q}$ using (11)
5:     if $\mathcal{Q} > 0$ then
6:         $x_{C_{q_i}} = x_{C_{q_i}} + \Delta x_{C_{q_i}}$ for all $q_i = 1, \dots, q$
7:         $\mu = \mu \times \max\{\frac{1}{3}, 1 - (2\mathcal{Q} - 1)^3\}$
8:         $\nu = 2$
9:     else
10:        $\mu = \mu \times \nu$
11:        $\nu = 2 \times \nu$
12:    end if
13:    $k = k + 1$
14: end while
```

at most $\mathcal{O}(n^3)$, where $\mathcal{O}(\cdot)$ is the so-called Big-O notation, see [37] for details. In the distributed method, however, we solve the linear system of equations using message passing over a clique tree. The message passing scheme can be viewed as a multi-frontal LDL^T factorization technique [22], leading to a computational complexity of at most $\mathcal{O}(n^3)$. To be specific, conducting an upward pass from the leave of the clique tree to the root at each iteration, is equivalent to block-diagonalizing the matrix $J(x)^T J(x) + \mu I$, with the number of blocks being equal to the number of cliques in the clique tree. In addition, conducting a downward pass from the root of the leaves, can be viewed as the back substitution part when solving the linear system of equations. The computational complexity of the downward pass is negligible compared to the upward pass. Finally, Line 4 in Algorithm 2, i.e. the computation of $\mathcal{Q}$, has computational complexity of $\mathcal{O}(mn)$ for both the centralized and distributed methods, which is negligible compared to the cost associated with the factorization. To conclude, the proposed distributed method and its centralized counterpart have similar computational complexity of $\mathcal{O}(n^3)$.

6 Results and discussion

In this section we compare performance of the proposed distributed Levenberg-Marquardt algorithm, referred as LV algorithm, which is implemented in Julia [38], with two algorithms. The first algorithm is a convex relaxation based distributed algorithm presented in [1]. We refer to it as Disk algorithm since the approach is based on what is known as the disk relaxation approach. The second algorithm presented in [16], is a distributed algorithm which directly optimizes the non-convex maximum likelihood problem. We refer to it as StableML algorithm. We do not conduct a comparison with other algorithms, since a thorough comparison with Disk has been conducted in [1], which illustrated the superiority of that algorithm to high performance algorithms in [18] and [3] both in accuracy and number of communications among agents. Furthermore, in [3], the authors show the superiority of their algorithm to the one proposed in [12].

6.1 Simulation data

We conduct experiments for networks of sensors with connected inter-sensor measurement graphs in three simulation setups. In all setups we consider several sensors which are randomly distributed, and 9 anchors which are uniformly distributed in a two-dimensional area. We generate the noisy range measurements as

$$\mathcal{R}_{ij} = | \parallel \left(p_s^*\right)^i - \left(p_s^*\right)^j \parallel + E_{ij}|, \quad j \in Ne_r(i)$$
$$\mathcal{Y}_{ij} = | \parallel \left(p_s^*\right)^i - \left(p_a^*\right)^j \parallel + V_{ij}|, \quad j \in Ne_a(i)$$

where $\left(p_s^*\right)^i$ is the true location of the ith sensor, $E_{ij} \sim \mathcal{N}(0,\sigma)$ and $V_{ij} \sim \mathcal{N}(0,\sigma)$. We assume that all noises are Gaussian and mutually independent. We consider 10, 30 and 50 fixed sensors for the first, second and third setup, respectively which are distributed in a 1×1 area. We consider four different measurement noise standard deviations (σ), 0.01, 0.05, 0.1 and 0.3, and for each setup, 25 realizations of each noise level that are generated across Monte Carlo runs. We assume there exists a measurement between two sensors or between a sensor and an anchor if the distance between them is less than the communication range which is chosen to be between 0.3 and 0.4 depending on the number of sensors in the network, to ensure that the generated graph is loosely connected. For instance for the first network with 10 sensors, we choose a large communication range, e.g. close to 0.4. By doing so, the average number of edges connected to each sensor turned out to be 7.40, 8.63 and 11.92 for the first, second and third network, respectively. As an example we depict the resulting sensor network for the third setup with 50 sensors and the corresponding clique tree in Figs. 4 and 5, respectively. As can be seen, the inter-sensor measurement graph is connected.

6.2 Performance assessment

Before evaluating the performance of the two aforementioned algorithms, we want to stress the importance of the number of measurements for the quality of estimates. This, in our sensor network application, means that the more measurements are available between sensor i and sensor and/or anchor j, the better estimate will be achieved. Notice that in

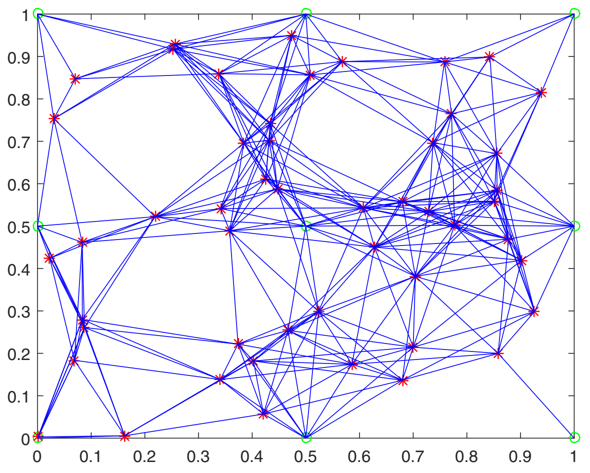


Fig. 4 The sensor network with 50 sensors and 9 anchors, considered for our experiment. The sensor nodes are marked with red crosses and the anchors are marked with green circles. An edge between two nodes, implies existence of a range measurement between two nodes

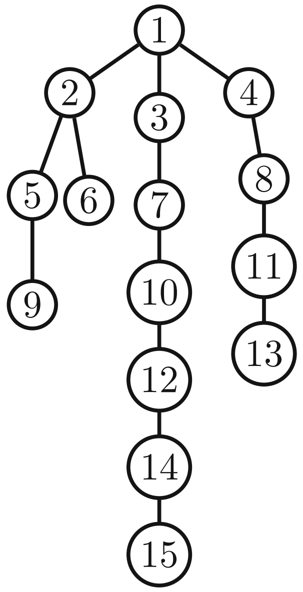


Fig. 5 Clique tree for the network in Fig. 4

the maximum likelihood formulation (2), we have assumed that there is a single measurement between sensor i and sensor and/or anchor j, in particular $\mathcal{R}_{ij}$ and/or $\mathcal{Y}_{ij}$. Let us now assume that there are N measurements available between sensor i and sensor and/or anchor j, in particular $\mathcal{R}_{ij}^k$ and/or $\mathcal{Y}_{ij}^k$ for $k \in \mathbb{N}_N$, where the superscript k denotes the index of the measurement. With these definitions, the maximum likelihood problem becomes

$$\underset{P}{\text{minimize}} \quad \sum_{k=1}^{N}\left\{\sum_{i=1}^{n_s}\Big(\sum_{j\in Ne_r(i)} \left(\mathcal{D}_{ij}\left(p_s^i, p_s^j\right) - \mathcal{R}_{ij}^k\right)^2 + \sum_{j\in Ne_a(i)} \left(\mathcal{Z}_{ij}\left(p_s^i, p_a^j\right) - \mathcal{Y}_{ij}^k\right)^2\Big)\right\} \tag{12}$$

Compared to the problem in (2), although they have the same clique tree, this problem requires N times more computations in order to calculate the gradient of the objective function. It can however be shown that solving the problem in (12) is equivalent to solving (2), by replacing the single measurement with the average of measurements over k, i.e. $\bar{\mathcal{R}}_{ij} = \sum_{k=1}^{N} \mathcal{R}_{ij}^k$ and $\bar{\mathcal{Y}}_{ij} = \sum_{k=1}^{N} \mathcal{Y}_{ij}^k$, instead of $\mathcal{R}_{ij}$ and $\mathcal{Y}_{ij}$, respectively. This follows from the fact that $\bar{\mathcal{R}}_{ij}$ and $\bar{\mathcal{Y}}_{ij}$ are sufficient statistics for estimation of x_s^i which follows from the Neyman-Fisher factorization theorem. For details see [39, Ch. 5].

In our simulations, we choose N to be 1, 10, and 100. In the experiments, we run our proposed algorithm based on the formulation in (2) using the average of measurements, i.e. $\bar{\mathcal{R}}_{ij}^k = \sum_{k=1}^{N} \mathcal{R}_{ij}^k$ and $\bar{\mathcal{Y}}_{ij}^k = \sum_{k=1}^{N} \mathcal{Y}_{ij}^k$. We refer to the obtained estimate as LV estimate. The Disk and StableML algorithms also run for single measurement, and therefore we use the average of the measurements for these algorithms as well. We refer to these estimates as Disk and StableML estimates, respectively.

The Disk and StableML algorithms are terminated if the norm of the gradient of their cost functions are below 10^{-6}. This threshold was chosen based on the experience of the authors in [1] and [16], so as to guarantee that Disk and StableML generate accurate enough solutions. For the proposed distributed Levenberg–Marquardt algorithm, we choose $\tau = 10^{-6}$ and $\epsilon = 10^{-6}$. It should be noted that the LV and StableML algorithms are sensitive to the initial starting point, and therefore they should be initialized not very far from optimum to ensure convergence to a good local minimum. Here we use an approximate solution obtained from the Disk algorithm as the initial starting point for both LV and StableML algorithms. To this end, we terminate the Disk algorithm if the norm of the gradient of its cost function is below 10^{-1}. It should be noted that in average the number of iterations for convergence to this low accuracy is $2-3\%$ of the iterations needed for terminating the Disk algorithm when requiring the norm of the gradient to be below 10^{-6}. There might also be other cheap ways of initializing the algorithm, but we have not investigated that in this paper. Moreover, the chordal embedding and the corresponding clique tree for all networks are generated using the MATLAB functions in the toolbox [31]. The generated clique trees for the networks with 10, 30, and 50 sensors, have 8, 12 and 15 cliques, respectively.

Let $P_s = [p_s^i]_{i=1}^{n_s}$, be the vector obtained by stacking the estimated sensor ith position. Also let $P_s^* = \left[(p_s^*)^i\right]_{i=1}^{n_s}$, be the vector obtained by stacking the true position of ith sensor. We will compare the performance of two different estimates using the root mean squared error (RMSE) per sensor defined as

$$\mathrm{RMSE} = \sqrt{\frac{1}{Q \times n_s} \sum_{q=1}^{Q} \| P_s^* - P_s(q) \|^2} \tag{13}$$

where n_s is the number of sensors and Q is the number of problem instances. The argument q refers to the qth experiment. Figures 6, 7 and 8 illustrate the RMSE results for different noise levels for the networks with 10, 30 and 50 sensors, respectively. Notice that the plots for the LV (red) and StableML (green) estimates are similar. It can be seen from the figures that the LV and StableML estimates perform equally well and in general they outperform the Disk estimate for low noise levels and as the number of measurements N increases, they perform even considerably better than the Disk estimate. Note that as we will see later, the good performance of the StableML estimate comes at the price that it requires far more communications for convergence compared to the LV estimate. Note also that, although we have not included the figures for the objective function values for the nonlinear LS problem, for all the simulations it is the case that the LV and StableML estimates have ended up in lower objective function values in (2) than the Disk estimate.

Let us also evaluate the bias-variance trade-off for different estimates for the lowest noise level ($\sigma = 0.01$) and the highest noise level ($\sigma = 0.3$). Now, recall the relation between mean square error (MSE), variance and bias of the estimate

$$\underbrace{\frac{1}{Q \times n_s} \sum_{q=1}^{Q} \| P_s^* - P_s(q) \|^2}_{\mathrm{MSE}} = \underbrace{\frac{1}{n_s} \| P_s^{\mathrm{mean}} - P_s^* \|^2}_{\mathrm{Bias}^2} + \underbrace{\frac{1}{Q \times n_s} \sum_{q=1}^{Q} \| P_s(p) - P_s^{\mathrm{mean}} \|^2}_{\mathrm{Variance}} \tag{14}$$

where $P_s^{\mathrm{mean}} = \frac{1}{Q} \sum_{q=1}^{Q} P_s(q)$. These values for the cases with $\sigma = 0.01$, $\sigma = 0.3$, and $N = 100$ are illustrated in Figs. 9, 10, and 11, for the networks with 10, 30 and 50 sensors, respectively. The observations from the results are the followings. In general the LV and StableML estimates have similar biases, variances and MSEs, although in some cases the

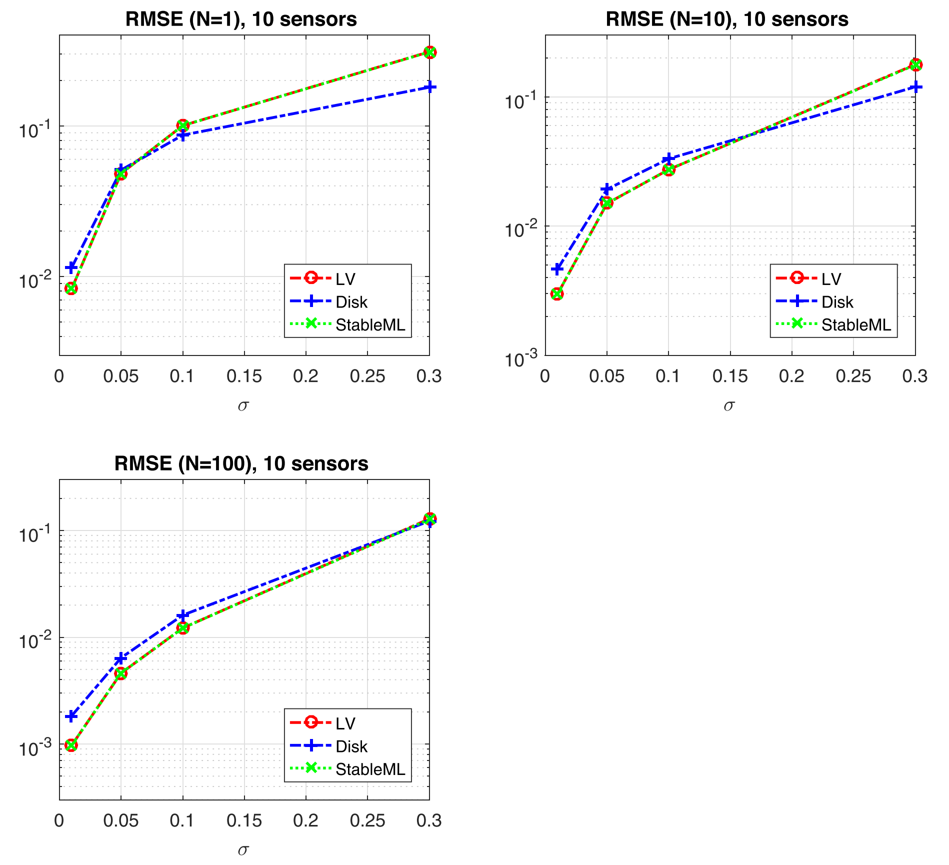


Fig. 6 The RMSE per sensor results for different noise levels, namely 0.01, 0.05, 0.1, and 0.3, for the network with 10 sensors. The top left, top right and bottom left plots correspond to $N = 1$, $N = 10$, and $N = 100$, respectively

LV estimate has slightly lower bias and larger variance compared to the StableML estimate. For $\sigma = 0.01$, LV and StableML estimates have lower biases and MSEs compared to the Disk estimate. Irrespective of the value of σ, LV and StableML estimates have the smaller biases in all cases. Whenever Disk estimate beats LV and StableML estimates in terms of MSE, it has larger bias and smaller variance. We can therefore draw the conclusion that LV and StableML estimates are the method with smallest bias. The only way to beat LV and StableML estimates in terms of MSE is to have a larger bias and smaller variance. This is what is expected, since LV and StableML estimates are maximum likelihood estimates.

In order to compare the number communications between the algorithms, first we have to note that the way the computations are distributed in Disk and StableML algorithms are similar. However, they are different than the way the computations are distributed in the LV algorithm. In the distributed LV algorithm, it is enough to assume that one sensor per clique is capable of carrying out the computations. Hence, we have as many computational agents as the number of cliques. This is related to how we distribute our computations which is based on the clique tree. In the distributed Disk and StableML algorithms, however, it is assumed that each sensor is capable of carrying out computations, and therefore those algorithms have as many computational agents as the number of sensors. It should be noted that, we can use even fewer computational agents than the

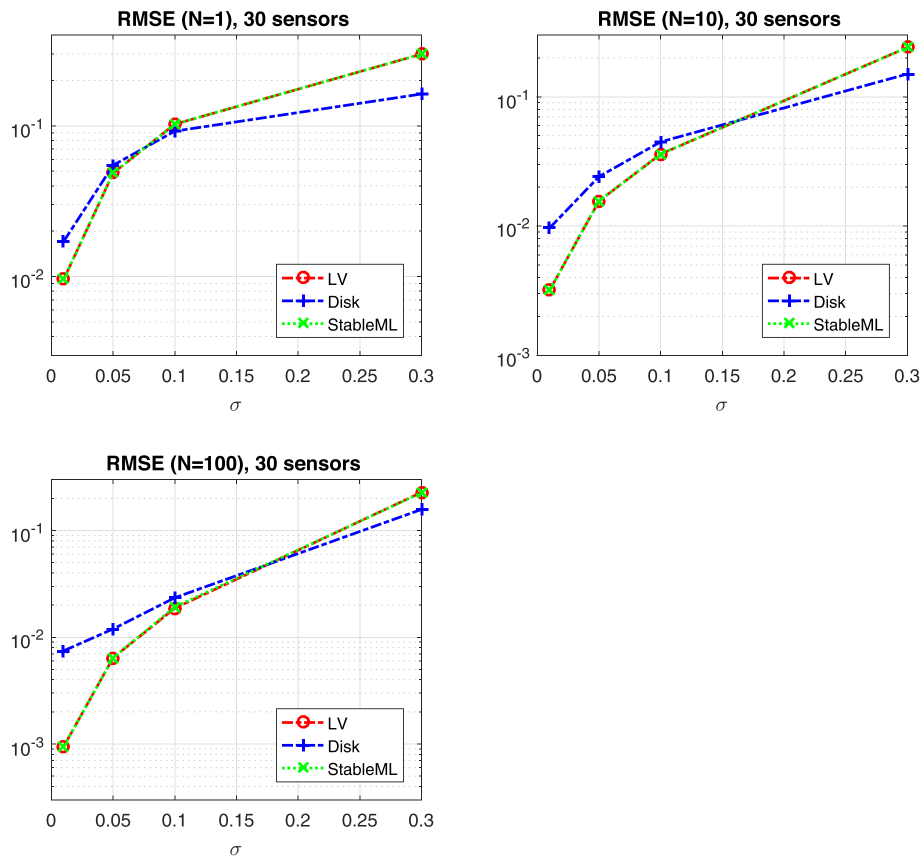


Fig. 7 The RMSE per sensor results for different noise levels, namely 0.01, 0.05, 0.1, and 0.3, for the network with 30 sensors. The top left, top right and bottom left plots correspond to $N = 1$, $N = 10$, and $N = 100$, respectively

number of cliques in the distributed LV algorithm. This is possible because we can always merge neighboring cliques in the tree. Hence, in our approach we have an upper limit on the number of computational agents that we may use, but no lower limit.

The advantage of having one computational agent per clique is that the number of communications in general is lower compared to the case where we have one computational agent per sensor. This is a nice property especially when the communication is costly. The computations, however, in the former case is heavier than the latter case. Nevertheless, the effort spent for the computations is considerably less than the effort spent for sending and receiving messages between agents which include complex operations such as coding, decoding, synchronization, etc. [20]. See [40], for an estimation of energy consumption between two sensors in wireless communication. Notice also that one important factor which affects the number of communications between agents is the number of iterations needed to get a certain accuracy. As discussed in Section 1, the former case which is based on a pseudo-second-order method, requires fewer iterations, and so fewer communications compared to the latter case which is based on a first-order method. The former case also requires inter-clique communications when the sensor readings are sent to the sensor which is responsible for carrying out the computations. Notice however that as discussed before, not all but just the average of the sensor readings needs to be sent as the average

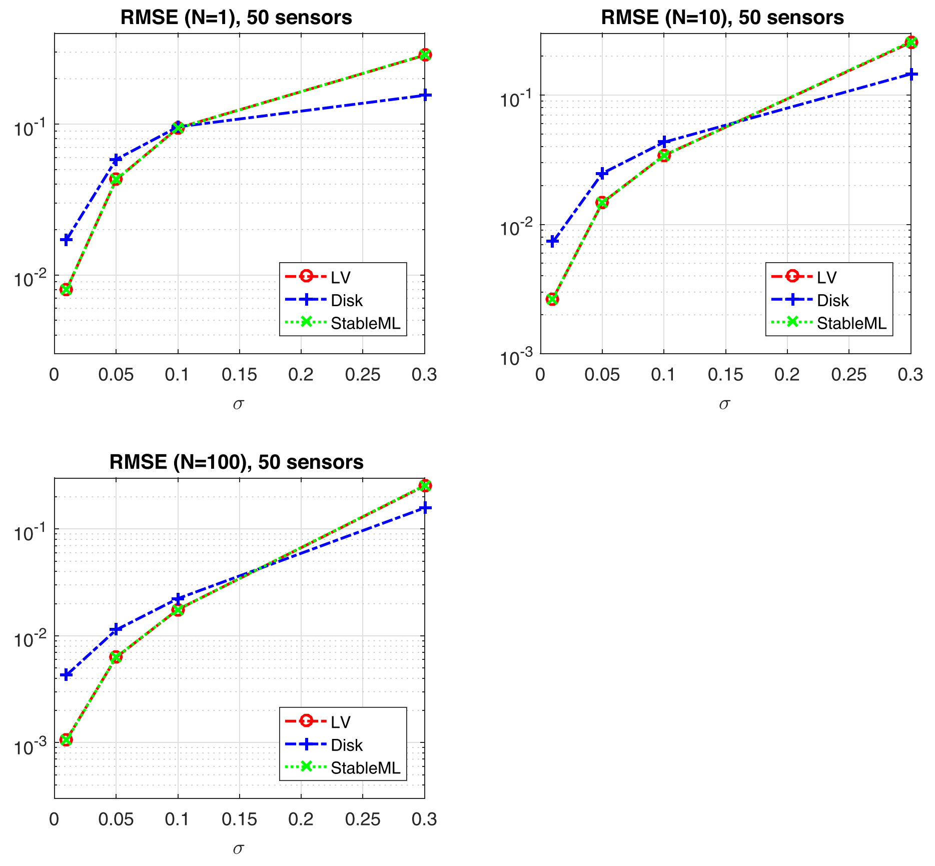


Fig. 8 The RMSE per sensor results for different noise levels, namely 0.01, 0.05, 0.1 and 0.3, for the network with 50 sensors. The top left, top right and bottom left plots correspond to $N = 1$, $N = 10$ and $N = 100$, respectively

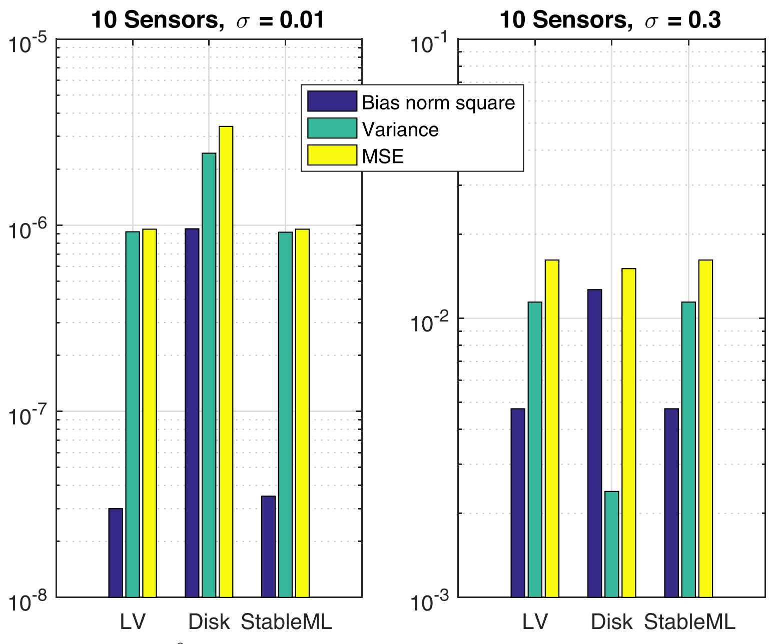


Fig. 9 MSE, Variance and Bias^2 values in (14) for $\sigma = 0.01$ (left figure) and $\sigma = 0.3$ (right figure), for the network with 10 sensors and $N = 100$

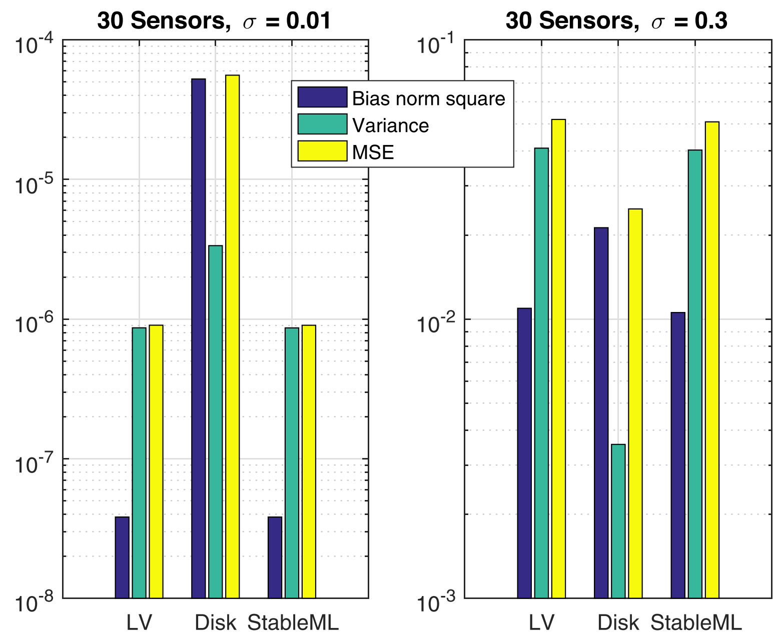


Fig. 10 MSE, Variance and Bias^2 values in (14) for $\sigma = 0.01$ (left figure) and $\sigma = 0.3$ (right figure), for the network with 30 sensors and $N = 100$

is sufficient statistics to the maximum likelihood estimate. The disadvantage of the former case is that if a computational agent fails or if the communication with the neighbor agents is lost, the clique tree should be computed from scratch, whereas in the latter case the failed agent can be dismissed and the algorithm will continue working.

Remark 6 *It should be pointed out that although for the distributed LV algorithm it is enough to have one sensor per clique which is capable of carrying out the computations, there are two advantages of having more than one sensor per clique for this purpose. One is that we can distribute the energy consumption of the sensors by letting them take turns for carrying out the computations. The second advantage is that in case a sensor which is responsible for the computations fails, there is a backup sensor ready to take over. How to trigger this can be done in the following way. If the parent/child clique does not get a message from its child/parent clique for a pre-specifed period of time, it implies that a failure has happened and another sensor is requested to take over the computations.*

In general we have two types communication in the distributed LV algorithm. The first type relates to the fact that within each clique, each sensor needs to send the information regarding the distance to its adjacent sensors to the sensor which is responsible for carrying out the computations. The second type of communication is about exchanging messages between the sensors which are responsible for carrying out the computations. Theses messages can be expressed with matrices which are symmetric, and vectors. In particular, we need three upwards and downwards pass through the clique tree at each iteration. To be more specific, we require one upwards and one downwards pass through the clique tree in order to calculate the search direction in (10) (Line 3 in Algorithm 2) and one upwards pass through the clique tree in order to calculate different terms of

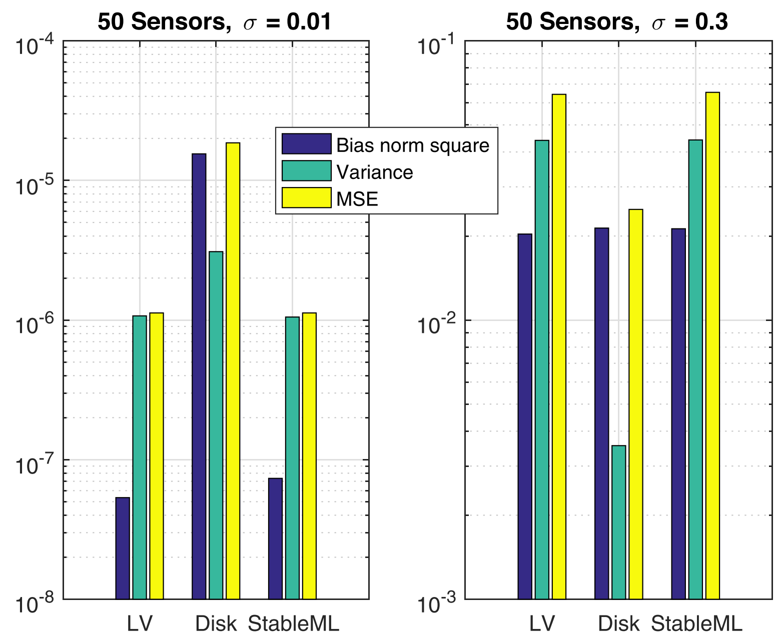


Fig. 11 MSE, Variance and Bias^2 values in (14) for $\sigma = 0.01$ (left figure) and $\sigma = 0.3$ (right figure), for the network with 50 sensors and $N = 100$

$\nabla F(x)$ (Line 2 in Algorithm 2) and $\mathcal{Q}$ (Line 4 in Algorithm 2) using (11). For the considered networks with 10, 30 and 50 sensors, for the first upwards pass, the communicated messages are a symmetric matrix and a vector with the average sizes of $(7 \times 7, 7 \times 1)$, $(18 \times 18, 18 \times 1)$ and $(43 \times 43, 43 \times 1)$, respectively. Also, the maximum sizes of the matrix and the vector are $(10 \times 10, 10 \times 1)$, $(26 \times 26, 26 \times 1)$ and $(72 \times 72, 72 \times 1)$, respectively. For the downwards pass, the communicated messages are a vector with the same size as the vector communicated in the first upwards pass. Finally, for the last upwards pass, the communicated messages are three scalars which can be combined in a (3×1) vector. For the distributed disk relaxation method, however, according to the Algorithm 1 in [1], at each iteration every sensor or agent needs to communicate a (2×1) vector which is called ω_i with its adjacent agents or sensors. It is obvious that for the largest network the amount of data to be sent in the distributed LV algorithm is roughly 400–500 more than for the Disk algorithm. However, as discussed before, what is most costly in many applications is the number of times contact is established and not how much information is sent. We will now compare the total number of communications for different algorithms. The total number of communications required for each algorithm to converge to a solution are depicted in Figs. 12 and 13 for all networks with $\delta = 0.05$ and $\delta = 0.1$, respectively. Both figures correspond to the case with $N = 100$. It is seen that the LV algorithm requires roughly two orders of magnitude fewer communications for computing the solution compared to the Disk algorithm and the StableML algorithm requires even more communications compared to the Disk algorithm. The shaded areas depict the maximum and minimum values out of 25 problem instances.

It is worth pointing out that for high noise levels, in order to get a better performance in terms of RMSE, one can proceeds as follows. Given N measurements between sensor i and sensor and/or anchor j, we can run the LV algorithm for each of the measurements

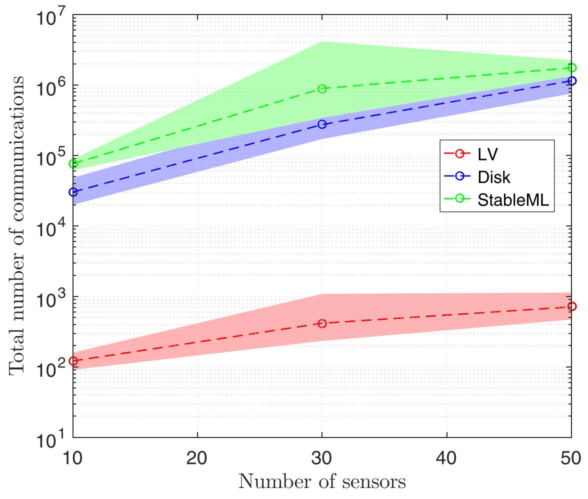


Fig. 12 Total number of communications required for each algorithm to converge to a solution for $\delta = 0.05$ and $N = 100$

using the formulation in (2), which will result in N estimates. We can then compute the final estimate as the average of the estimates. We can also do the same procedure using the Disk algorithm. By doing so, although for high noise levels the results will again be improved, compared to the case where we run the algorithm once using the average of the measurements, the estimate obtained from using the LV algorithm outperforms the estimate obtained form the Disk algorithm in terms of RMSE. We verified this in the simulations, however, for the sake of brevity we do not present the results in the paper. Notice that the number of communications for this approach is much more than the case where we run the algorithm once by using the average of the measurements, since we have to run the algorithm N times.

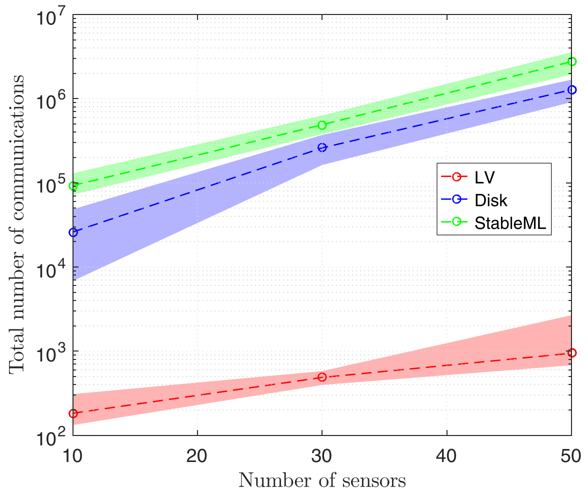


Fig. 13 Total number of communications required for each algorithm to converge to a solution for $\delta = 0.1$ and $N = 100$

7 Conclusion

In this paper we proposed a distributed algorithm for maximum likelihood estimation for the localization problem which relies on the Levenberg-Marquardt algorithm and message passing over a tree. We discussed how the tree can be generated in a distributed way and also we discussed how one should proceed if a measurement between sensors is lost, or if a new sensor is added to the network. The resulting algorithm requires much fewer iterations than first-order distributed methods in order to converge to an accurate solution, which in turn leads to fewer number of communications among computational agents. The algorithm also outperforms other distributed algorithms in terms of accuracy if estimates of small bias with limited number of communications are important. Only by a larger bias and smaller variance can other methods beat our method in terms of RMSE.

The size of messages communicated between cliques depend on the number of sensors and anchors two adjacent cliques share. In terms of parallel computations, the more branches we have in the clique tree, the faster the computations can be carried out. Therefore, we intend to investigate different approaches for finding clique trees in future research. In addition, different ways of initializing the algorithm in a cheap way will be investigated.

Abbreviations
LV: Levenberg-Marquardt; RMSE: Root mean squared error; MSE: Mean squared error

Acknowledgements
Not applicable

Authors' contributions
SKP initiated the research on using message passing for solving localization problems. AH came up with the idea of extending the message passing idea to the exact non-linear least squares formulation. Then SPA showed how this could be done for the Levenberg-Marquardt algorithm. He also performed the simulations and conducted the performance analysis, under supervision of AH. SPA wrote the majority of the manuscript. AH wrote the example in the Introduction, proofread the manuscript several times and provided feedback. SKP also proofread the manuscript. All authors read and approved the final manuscript.

Funding
This research has been supported by Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, which is gratefully acknowledged. Open access funding provided by Linköping University.

Availability of data and materials
The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Electrical Engineering, Linköping University, Linköping, Sweden. [2]C3.ai, Redwood City, CA, USA.

Received: 24 November 2020 Accepted: 7 July 2021
Published online: 23 August 2021

References
1. C. Soares, J. Xavier, J. Gomes, Simple and fast convex relaxation method for cooperative localization in sensor networks using range measurements. IEEE Trans. Signal Process. 63(17), 4532–4543 (2015)

2. P. Biswas, T.-C. Liang, K.-C. Toh, Y. Ye, T.-C. Wang, Semidefinite programming approaches for sensor network localization with noisy distance measurements. IEEE Trans. Autom. Sci. Eng. 3(4), 360–371 (2006)
3. A. Simonetto, G. Leus, Distributed maximum likelihood sensor network localization. IEEE Trans. Signal Process. 62(6), 1424–1437 (2014)
4. Z. Wang, S. Zheng, Y. Ye, S. Boyd, Further relaxations of the semidefinite programming approach to sensor network localization. SIAM J. Optim. 19(2), 655–673 (2008)
5. J. N. Al-Karaki, A. E. Kamal, Routing techniques in wireless sensor networks: a survey. IEEE Wirel. Commun. 11(6), 6–28 (2004)
6. C. Wang, Q. Yin, H. Chen, Robust chinese remainder theorem ranging method based on dual-frequency measurements. IEEE Trans. Veh. Technol. 60(8), 4094–4099 (2011)
7. B. Liu, H. Chen, Z. Zhong, H. V. Poor, Asymmetrical round trip based synchronization-free localization in large-scale underwater sensor networks. IEEE Trans. Wirel. Commun. 9(11), 3532–3542 (2010)
8. H. Chen, F. Gao, M. Martins, P. Huang, J. Liang, Accurate and efficient node localization for mobile sensor networks. Mob. Netw. Appl. 18(1), 141–147 (2013)
9. W. Zhang, Q. Yin, H. Chen, F. Gao, N. Ansari, Distributed angle estimation for localization in wireless sensor networks. IEEE Trans. Wirel. Commun. 12(2), 527–537 (2012)
10. G. Han, J. Jiang, C. Zhang, T. Q. Duong, M. Guizani, G. K. Karagiannidis, A survey on mobile anchor node assisted localization in wireless sensor networks. IEEE Commun. Surv. Tutor. 18(3), 2220–2243 (2016)
11. T. Erseghe, A distributed and maximum-likelihood sensor network localization algorithm based upon a nonconvex problem formulation. IEEE Trans. Signal Inf. Process. Netw. 1(4), 247–258 (2015)
12. Q. Shi, C. He, H. Chen, L. Jiang, Distributed wireless sensor network localization via sequential greedy optimization algorithm. IEEE Trans. Signal Process. 58(6), 3328–3340 (2010)
13. J. A. Costa, N. Patwari, A. O. Hero III, Distributed weighted-multidimensional scaling for node localization in sensor networks. ACM Trans. Sens. Netw. (TOSN). 2(1), 39–64 (2006)
14. G. C. Calafiore, L. Carlone, M. Wei, in *49th IEEE Conference on Decision and Control (CDC)*, Distributed optimization techniques for range localization in networked systems (IEEE, New York, 2010), pp. 2221–2226
15. M. G. Rabbat, R. D. Nowak, in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Decentralized source localization and tracking [wireless sensor networks], vol. 3 (IEEE, New York, 2004), p. 921
16. C. Soares, J. Xavier, J. Gomes, in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Distributed, simple and stable network localization (IEEE, New York, 2014), pp. 764–768
17. S. Khoshfetrat Pakazad, E. Özkan, C. Fritsche, A. Hansson, F. Gustafsson, Distributed localization of tree-structured scattered sensor networks. arXiv preprint arXiv:1607.04798, 14 pages (2016)
18. M. R. Gholami, L. Tetruashvili, E. G. Ström, Y. Censor, Cooperative wireless sensor network positioning via implicit convex feasibility. IEEE Trans. Signal Process. 61(23), 5830–5840 (2013)
19. B. Q. Ferreira, J. Gomes, C. Soares, J. P. Costeira, FLORIS and CLORIS: Hybrid source and network localization based on ranges and video. Signal Process. 153, 355–367 (2018)
20. N. Piovesan, T. Erseghe, Cooperative localization in WSNs: A hybrid convex/nonconvex solution. IEEE Trans. Signal Inf. Process. Netw. 4(1), 162–172 (2016)
21. J. Nocedal, S. Wright, *Numerical Optimization*. (Springer, USA, 2006)
22. S. Khoshfetrat Pakazad, A. Hansson, M. S. Andersen, I. Nielsen, Distributed primal–dual interior-point methods for solving tree-structured coupled convex problems using message-passing. Optim. Methods Softw. 32(3), 401–435 (2017)
23. J. R. Blair, B. Peyton, in *Graph Theory and Sparse Matrix Computation*, An introduction to chordal graphs and clique trees (Springer, USA, 1993), pp. 1–29
24. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. (Elsevier, The Netherlands, 2004)
25. J. J. Moré, Z. Wu, Global continuation for distance geometry problems. SIAM J. Optim. 7(3), 814–836 (1997)
26. H. Chen, Q. Shi, R. Tan, H. V. Poor, K. Sezaki, Mobile element assisted cooperative localization for wireless sensor networks with obstacles. IEEE Trans. Wirel. Commun. 9(3), 956–963 (2010)
27. G. Wang, H. Chen, Y. Li, N. Ansari, NLOS error mitigation for TOA-based localization via convex relaxation. IEEE Trans. Wirel. Commun. 13(8), 4119–4131 (2014)
28. Z. Dai, G. Wang, H. Chen, Sensor selection for TDOA-based source localization using angle and range information. IEEE Trans. Aerosp. Electron. Syst. 57(4), 2597–2604 (2021)
29. X. Xie, Z. Geng, Q. Zhao, Decomposition of structural learning about directed acyclic graphs. Artif. Intell. 170(4-5), 422–439 (2006)
30. F. V. Jensen, F. Jensen, in *Uncertainty Proceedings 1994*, Optimal junction trees (Elsevier, Morgan Kaufmann, 1994), pp. 360–366
31. X. Xie, A Recursive method to learn Bayesian network. MathWorks (2020). https://se.mathworks.com/matlabcentral/fileexchange/20678-a-recursive-method-to-learn-bayesian-networkl
32. L. Vandenberghe, M. S. Andersen, Chordal graphs and semidefinite optimization. Found. Trends Optim. 1(4), 241–433 (2015)
33. S. P. Ahmadi, A. Hansson, in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, Parallel exploitation for tree-structured coupled quadratic programming in julia (IEEE, New York, 2018), pp. 597–602
34. J. E. Dennis Jr, R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, vol. 16. (Siam, USA, 1996)
35. K. Madsen, H. B. Nielsen, O. Tingleff, *Methods for non-linear least squares problems*, 2nd ed., (2004)
36. A. S. Lewis, M. L. Overton, Nonsmooth optimization via quasi-Newton methods. Math. Program. 141(1-2), 135–163 (2013)
37. G. H. Golub, C. F. Van Loan, *Matrix Computations*, vol. 3. (JHU press, Baltimore, 2013)
38. J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A fresh approach to numerical computing. SIAM Review. 59(1), 65–98 (2017)
39. S. M. Kay, *Fundamentals of Statistical Signal Processing*. (Prentice Hall PTR, USA, 1993)

40. H. Chen, B. Liu, P. Huang, J. Liang, Y. Gu, Mobility-assisted node localization based on TOA measurements without time synchronization in wireless sensor networks. Mob. Netw. Appl. 17(1), 90–99 (2012)

Publisher's Note