Distributed Localization Using Noisy
Distance and Angle Information[*]

Amitabh Basu
Stony Brook University
Stony Brook, NY 11794-4400

Jie Gao
Stony Brook University
Stony Brook, NY 11794-4400

Joseph S. B. Mitchell
Stony Brook University
Stony Brook, NY 11794-3600

Girishkumar Sabhnani
Stony Brook University
Stony Brook, NY 11794-4400

ABSTRACT

Localization is an important and extensively studied problem in ad-hoc wireless sensor networks. Given the connectivity graph of the sensor nodes, along with additional local information (e.g. distances, angles, orientations etc.), the goal is to reconstruct the global geometry of the network. In this paper, we study the problem of localization with noisy distance and angle information. With no noise at all, the localization problem with both angle (with orientation) and distance information is trivial. However, in the presence of even a small amount of noise, we prove that the localization problem is **NP**-hard. Localization with accurate distance information and *relative angle* information is also hard. These hardness results motivate our study of approximation schemes. We relax the non-convex constraints to approximating convex constraints and propose linear programs (LP) for two formulations of the resulting localization problem, which we call the weak deployment and strong deployment problems. These two formulations give upper and lower bounds on the location uncertainty respectively: No sensor is located outside its weak deployment region, and each sensor can be anywhere in its strong deployment region without violating the approximate distance and angle constraints. Though LP-based algorithms are usually solved by centralized methods, we propose distributed, iterative methods, which are provably convergent to the centralized algorithms solutions. We give simulation results for the distributed algorithms, evaluating the convergence rate, dependence on measurement noises, and robustness to link dynamics.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*graphs and networks*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Design, Theory

Keywords

Sensor networks, Localization, Linear Programming

1. INTRODUCTION

In recent years wireless sensor networks have revealed great potential to provide economical and practical solutions for long-term extensive data gathering and monitoring. One of the fundamental calibration procedures for sensor networks is localization, i.e., determining either absolute or relative locations of the sensor nodes. Node locations are important features for the integrity of the sensor readings. They are also helpful for many network operations such as clustering, topology control and geographical routing. Sensor node locations can be found by extra hardware support, such as GPS (global positioning system) receivers [11]; however, due to the high cost of such hardware, it is desirable to derive sensor locations from purely local measurements.

There has been extensive work on localization algorithms [22, 23, 15, 16, 17, 21, 24, 25, 2, 13, 8, 7]. The majority of them use distance estimations to derive relative node locations or absolute locations when anchor nodes are available. Localization without anchor nodes, however, is a much harder problem, both in theory and in practice. Given the lengths of the edges of a unit disk graph, finding an embedding in the plane so that non-neighboring nodes are separated by distances of at least 1 is known to be **NP**-complete [3, 12, 1]. One of the major problems in using only distance information is resolving the flipping ambiguity. In the example of Figure 1, using only edge lengths we can fix the shapes of triangles $\triangle abc$ and $\triangle bcd$, but we cannot determine the "flip" of triangle $\triangle bcd$ relative to triangle $\triangle abc$. When the network is large, this flipping ambiguity issue can be so severe that many optimization-based ap-

proaches easily get stuck at local minima corresponding to configurations that are far from the ground truth. To avoid this problem, Moore *et al.* [13] identified robust quadrilaterals that have a unique shape and combined them to derive the global geometry. However, in order to collect enough quadrilaterals to cover the majority of the nodes, simulations on uniformly distributed sensor nodes show that it is usually required to have a dense sensor network, with average degree 10 or more [13]. In a sparse network, this quadrilateral-based approach can be augmented with input from mobile robots, which helps to create enough distance measurements [19].
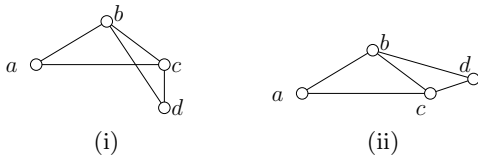

Figure 1: A connectivity graph with two distinct embeddings having the same set of edge lengths.

Another direction to solve the ambiguity of embedding with distances and incorrect flipping problem is to use angle information. Indeed, if we measure not only the (accurate) distance between two neighboring nodes but also the (accurate) angle between two incoming links with clockwise or counter-clockwise orientation, then localization is trivial – each node can find the locations of all of its neighbors in its local coordinate system, and then "gluing" these local coordinate systems together yields the global geometry. Angles between two incoming links can be measured by using multiple ultrasound receivers (e.g., the Angle of Arrival (AOA) technique [20]), laser transmitters and receivers (as in the SmartDust system), or directional antennas. In practice, the distance and angle measurements inevitably have noise.

In this paper we study the problem of localizing sensor nodes with noisy distance and angle measurements. Given a pair (i, j) of neighboring sensors, we define the notion of a *region of uncertainty* for a node pair. This region represents the noise in the distance and angle measurements in the sense that, if we fix the location of one node of the pair, the other node is somewhere within this region. See Figure 2 for an example. The localization problem in this setup is
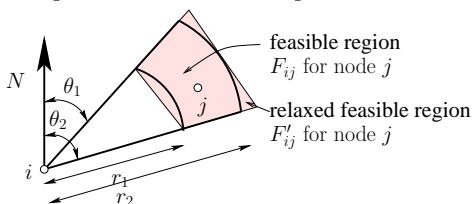

Figure 2: Region of uncertainty for node j in the local coordinate space of node i. The relaxed feasible region is shown shaded.

to find locations for the sensor nodes such that for each neighboring pair (i, j), the nodes fall in the feasible regions of each other. Some of the nodes may serve as *anchor nodes*, whose exact locations are known. If there are no anchor nodes, we arbitrarily fix one sensor node as the origin and one of its adjacent edges as the horizontal axis, in order to factor out global rotation and translation.

We first address the complexity of the localization problem. When we have accurate distance and angle measurements, each feasible region shrinks to a single point, and the localization problem is trivial (see above). On the other hand, with unbounded noise in distance and angle measurements, the embedding problem is **NP**-complete (with the unit disk graph assumption), as shown in [3, 12]. It remains **NP**-complete if we have exact angle information but unbounded noise in distance [4]. We are interested in the spectrum of possibilities in noise models to find out at what point does the embedding problem become hard. Surprisingly, even with an arbitrarily small noise in distance and angle measurements, the problem is hard, as we show using a reduction from 3SAT. Furthermore, with accurate distance measurements and accurate *relative* angle measurements, in which we know the absolute value of the angle between two incoming links but we do not know the orientation (the sign of the angle), the localization problem is also hard. This suggests that under certain reasonable noise models, the localization problem is very hard in theory unless both angle and distance information is known exactly.

The hardness of the localization problem mainly comes from the fact that the feasible region associated with a node pair is not convex. Thus, one can relax the feasible region to a convex enclosing shape, e.g., a minimum enclosing trapezoid [5]; see Figure 2. The localization problem under these relaxed constraints is polynomially solvable using linear programming, since all of the constraints can be represented by linear inequalities. Specifically, for n nodes with m edges, we have $2n$ variables for the coordinates of all nodes and $8m + 2$ linear constraints. With the relaxed constraints, we define two problems, the *weak deployment* and the *strong deployment* problem, which focus on different optimization criteria. In the weak deployment problem, we are to determine, for each node, its set of all possible placements (in the global coordinate system), assuming that all other nodes are placed in any feasible location (satisfying all noise constraints of all pairs of neighboring nodes). This *weak deployment region* represents an upper bound on the positional uncertainty of the node; no node can be located outside its weak deployment region. In the strong deployment problem, we are to determine, for each node, its *strong deployment region*, which has the property that for *any* choice of node locations, one within each of the strong deployment regions, the measurement constraints are satisfied for all pairs of neighboring nodes. The strong deployment regions represent a lower bound on the uncertainty inherent in the system – there is no way to tell where exactly each node is within its strong deployment region, based solely on the available constraints. The two problems are correlated: The weak deployment solution is non-empty if and only if the strong deployment solution is non-empty.

The use of linear programming to solve localization problems represents a centralized solution; distributed algorithms are more suitable for sensor network applications. Fortunately, our deployment problems have special structure, with all constraints being specified between neighboring pairs of nodes. This enables us to propose distributed algorithms that run in iterations, but provably converge to the same solution as the global LP. For the weak deployment problem, each node keeps its current feasible deployment region and updates it as it receives new information from its neighbors. At each iteration, a node sends its updated feasible region

to all of its one-hop neighbors. Each node then solves a local problem with the constraints from its neighbors and the new constraints enforced by the feasible deployment regions on itself and its neighbors. This local algorithm will output new feasible regions for each node. Iteratively, the feasible region of each node shrinks until the system converges. We prove that this iterative algorithm, which is implemented in terms of simple Minkowski sums, converges to the same solution as the global LP. The distributed iterative algorithm applies to the strong deployment problem as well, by reducing it to solving a weak deployment problem.

Another advantage of our distributed localization algorithm is its robustness to network dynamics. Sensor networks have high link dynamics, due to channel fading, node mobility, power conservation and communication interference. For the case of directional antennas, the antenna rotates between different sectors, potentially making previously available links unavailable. Our iterative algorithm only requires a mild condition on network connectivity. As long as the network is jointly connected, i.e., the union of the communication networks at different time steps is connected, the algorithm will converge to a reasonably good solution. The robustness to link variation is confirmed by simulation results. We note that this robustness is mainly due to the iterative nature of the solution. We also evaluate the convergence rate through simulation. With reasonable measurement noises, the algorithm converges rather fast.

2. PROBLEM SETUP

We have a collection of n sensor nodes and a connectivity graph $G = (V, E)$, where $|V| = n, |E| = m$. We assume that each node has a compass; thus, each knows the universal north and the local coordinate systems for all nodes are aligned. For each pair of sensor nodes i, j, we can measure both the distance between them and the angle of edge ij with the universal north. These measurements are noisy, so we only have lower and upper bounds (r_1 and r_2) on the distance, and upper and lower bounds (θ_1 and θ_2) on the angle. This uncertainty in the measurement is captured by the *region of uncertainty*, F_{ij}, which is a frustum in which j can lie with respect to i's local coordinate system; see Figure 2. We note that for each pair of neighboring nodes i, j, there are two regions of uncertainty – one, F_{ij}, is the region of uncertainty for node j with respect to i, and one, F_{ji}, is the region of uncertainty for node i with respect to j.

We define the localization problem with noise in the following way. Given, for each node, regions of uncertainty for its neighbors in its *local coordinate system*, find *possible deployment regions* for each node in a global coordinate system. Note that this problem is more general than the usual definition of the localization problem, which attempts to compute the "best" exact coordinates for the nodes. Our goal is to obtain *regions* for possible deployment that are consistent with the measurement data. Specifically we define deployment regions in two different ways that focus on different optimization criteria.

Weak deployment problem: Find the (set-theoretic) *maximal* deployment regions W_i such that, for any point $p \in W_i$, there exists a feasible placement of all other nodes $j \neq i$ that is consistent with placing node i at p. We factor out global translation by fixing one node (without loss of generality, node 1) at the origin.

Strong deployment problem: Find deployment regions S_i, of a specified shape, such that all constraints implied by the regions of uncertainty for neighboring pairs are satisfied if we locate node i at *any point* within the deployment region S_i, for each $i = 1, 2, \ldots, n$. The objective function is to maximize (maximally scale) the size of the smallest deployment region.

The motivation behind the above definitions is in obtaining *upper* and *lower* bounds on the uncertainty of node locations. Weak deployment regions W_i capture all possible feasible solutions to the given input; i.e., no placement outside these regions can possibly be the actual coordinates of the nodes, thus providing an upper bound on the solution space. Analogously, strong deployment regions S_i give a means of quantifying the "best we can do" with the inherent error in the input: Any combination of points within these regions simultaneously satisfies all constraints; based on the measurement data, one cannot tell where the real node location is within the strong deployment region. So the strong deployment provides a lower bound on the uncertainty in the system. The weak and strong deployment problems are correlated. The strong deployment region for each node is always a subset of the weak deployment region, $W_i \subseteq S_i$. Further, the weak deployment region is non-empty if and only if the strong deployment region is non-empty.

In section 5, we show that it is **NP-Complete** to solve the problem with nonconvex frustums F_{ij} as defined above; thus, we approximate these regions with convex polygons F'_{ij}, as in Figure 2. By the relaxation we can formulate the problem by linear programs as shown in section 3. We denote by $\mathcal{P}$ the original problem and $\mathcal{P}'$ the relaxed problem. Since $F_{ij} \subseteq F'_{ij}$ for all pairs i, j, the weak deployment region W'_i for $\mathcal{P}'$ is always a superset of W_i for $\mathcal{P}$. A feasible solution for $\mathcal{P}$ is always feasible for $\mathcal{P}'$, but not vice versa.

Relaxing non-convex constraints to convex constraints was previously utilized by Bădoiu et al. [5]. In their paper, they do not consider finding *regions* for deployment; they instead mention that any feasible solution to the LP is a valid assignment of node coordinates. They also adapt the method to minimize the angle and distance errors and different metrics, e.g., ℓ_1 and the ℓ_∞ norms. On the other hand, computing feasible regions by collecting constraints on the possible position of a sensor node has been considered in both single-hop and multi-hop scenarios [10, 9, 18]. These methods use either only connectivity information [18] or (absolute or relative) ranging information [10, 9].

3. LP FORMULATIONS FOR WEAK AND STRONG DEPLOYMENT

In this section we focus on the problem $\mathcal{P}'$, by relaxing the regions of uncertainty to convex regions.

3.1 Weak Deployment

We first present the linear program for the weak deployment. The decision variables in the LP are the $2n$ coordinates of the n nodes (x_i, y_i). The set of inequalities given by the sides of the convex region of uncertainty, F'_{ij}, define the constraints of the LP. Let the set of lines bounding F'_{ij} be denoted by L_{ij} and any element of this set be denoted by $\ell^k_{ij}(x, y)$, $k \in \{1, \ldots, |L_{ij}|\}$. Now we can constrain node j to be within F'_{ij} by putting the constraint $\ell^k_{ij}(x_j, y_j) \leq 0$ or $\ell^k_{ij}(x_j, y_j) \geq 0$, depending on which side of ℓ^k_{ij} is feasible.

Two additional constraints, $x_1 = 0$ and $y_1 = 0$, "pin-down" node 1 to be at the origin.

Each feasible solution of the localization problem is an assignment of locations to all the nodes and is a point in $\mathbb{R}^{2n}$. The linear constraints specify a feasible solution subspace F', which is a convex polytope in $\mathbb{R}^{2n}$. The projection of the feasible region F' of the LP onto the (x_i, y_i) space (the coordinate space of node i), denoted by W_i', is the solution to the weak deployment problem for $\mathcal{P}'$. Since it is a projection into two dimensions of a convex polytope, the weak deployment region W_i' for node i is a 2D convex polygon. In general, the complexity of the feasible polytope $F' \subseteq \mathbb{R}^{2n}$ of an LP can be exponential in n in the worst case. Our goal is to compute only the projections W_i', for each i, of the polytope F'. One approach is to compute, using an LP solver, a low-complexity convex polygon to approximate each projection W_i'. For example, by the appropriate choice of objective function in the LP over region F', we can compute the extreme points of W_i' along specified set of directions; the corresponding intersection of halfplanes gives an outer approximation to W_i'. The more choices of directions (and calls to the LP optimizer), the better the approximation to the weak deployment region W_i'. For example, we can find the bounding box for each of the deployment regions by solving 4 LPs over F' with 4 different objective functions, namely, **max** x_i, **min** x_i, **max** y_i and **min** y_i. In fact, if the regions of uncertainty F_{ij}' are given as axis-aligned rectangles, instead of trapezoids, then this approach gives the exact weak deployment regions. This approach the approach we used in simulations. More generally, in section 4.1 below, we show (Lemma 2) that the exact weak deployment regions are obtained by optimizing over F' in each of the directions determined by the set of all outward normals to the regions of uncertainty F_{ij}'. This implies that we can solve the weak deployment problem exactly (in polynomial time) by making a number of calls to an LP, with the number being linear in the total number of sides of the convex polygonal regions F_{ij}'.

3.2 Strong Deployment

We now show how to obtain an LP formulation for solving the strong deployment problem. Let the deployment regions of two neighboring nodes i, j be denoted by S_i', S_j'. By the convexity of the problem $\mathcal{P}'$, any set of strong deployment regions is a set of convex polygons. The first challenge we face for the strong deployment problem is that for *any* $p \in S_i'$ and *any* $q \in S_j'$, the constraints must be satisfied. This leads to an infinite number of constraints for all pairs of such points. To formulate a finite constraint set for the LP, we can use the following lemma, whose proof is omitted due to space constraints.

LEMMA 1. *Assume that every pair of corners c_i, c_j from S_i', S_j' satisfies the uncertainty constraints; i.e., c_j lies in F_{ij}' if the origin is placed at c_i and c_i lies in F_{ji}' if the origin is placed at c_j. Then, any pair of points from S_i' and S_j' satisfies the uncertainty constraints for nodes i and j.*

The above lemma shows that if we satisfy the constraints for every pair of corners of the deployment regions, then the regions are valid under the strong deployment condition. Hence, the strong deployment condition can be enforced by simply enforcing them on the finite set of corners.

Another subtle issue for the strong deployment problem is that we can trade off part of the deployment region of one node to obtain a larger deployment region for another node. Thus, our formulation of the strong deployment problem is subject to an optimization criterion. In this paper we impose one objective to make specific our choice of strong deployment regions: we restrict the shape of each strong deployment region to be a convex polygon of some fixed shape (e.g., an axis-aligned square) and maximize the minimum-size (scale) strong deployment region. In other words, we treat the regions S_i' for all i equally (all the regions S_i' have the same shape) and then maximize the minimum region size so that all of the constraints are satisfied.

If we enforce the regions S_i' to be rectangles (similar formulation can be given for any other convex shape), the decision variables of the LP are the center of the rectangle (x_i, y_i), and the height (h_i) and width (w_i). The four corners can now be obtained in terms of these parameters and the constraint equations can be obtained for every pair of corners just as in the weak deployment LP. Again, one of the centers must be "pinned down" (say, $x_1 = 0, y_1 = 0$) to eliminate global translation. Additionally, we have constraints of the type: $h_i \geq r$, $w_i \geq r$ for all i. The objective function of our LP is **max** r, where r is the scale factor for the sizes of the boxes. Thus, the strong deployment problem is solvable in polynomial time, using linear programming methods.

4. DISTRIBUTED ALGORITHMS FOR WEAK AND STRONG DEPLOYMENT

The LP formulations outlined above for weak and strong deployment are centralized algorithms. The LP formulation, however, has special structure that can be exploited in devising distributed algorithms that are more appropriate to sensor network applications: Each constraint only involves 4 of the variables in the LP, namely, the x- and y-coordinates of two neighboring nodes. Based on this structure of the constraint matrix, we devise distributed methods for the weak and strong deployment problems.

4.1 A Distributed Method for Weak Deployment

In our distributed method, a node uses only the local constraints to refine its *region of deployment*. Only the constraints of the centralized LP that involve this particular node *and* one of its neighbors are considered. We also include the constraints induced by the current regions of deployment of a node and its neighbors. Since the region of deployment is represented by a convex polygon, these constraints are linear. A localized halfplane intersection problem is formed at each node and solved iteratively. The deployment regions of all the nodes are refined by solving these local constraint problems, and this updated information about the network is used in the next iteration for the construction of the local deployment regions.

4.1.1 The Algorithm

1. Fix one of the nodes, without loss of generality node 1, at the origin in order to eliminate global translation. For each node $i \neq 1$, initialize the deployment regions R_i to be the whole plane $\mathbb{R}^2$. The deployment region for node 1 is initialized to (and remains throughout the algorithm) a single point – the origin.

2. For any node j, let $C_b(j)$ be the linear constraints of the sides of its *current deployment region* R_j on x_j, y_j. For each node i, we formulate a local linear linear constraint problem. Denote the set of neighbors of i by $N(i)$. Let $C_g(i)$ be all the linear constraints that only involve node i and one of its neighbors. Create a new set of constraints $U(i) = C_b(i) \cup \bigcup_{j \in N(i)} C_b(j)$. The set of local linear constraints, denoted by $L(i)$, is then $U(i) \cup C_g(i)$. At node i, we find the projection of the intersection of the constraints $L(i)$ onto the $x_i\text{-}y_i$ plane. This is the updated deployment region for node i. Do this local computation for all nodes.

3. If none of the deployment regions were updated in the previous step, then stop; otherwise, go to step (2).

4.1.2 Implementation in Terms of Minkowski Sums

The distributed method for weak deployment can be conveniently interpreted in terms of Minkowski sums of convex polygons. Recall that each node i has a current deployment region R_i, which is a convex polygon. Given a polygon P with a fixed reference point $\mathbf{p}$ inside it, we denote by $P(\mathbf{v})$ the translated polygon by a vector $\mathbf{v} \in \mathbb{R}^2$, i.e., $\mathbf{p}$ is translated to $\mathbf{p} + \mathbf{v}$. Now we can define the feasible region for node j in the reference frame of node i as a translated polygon $F'_{ij}(\mathbf{v}_{ij})$, as shown by the shaded region in Figure 3 (i). If node i's location is fixed, then node j must stay inside
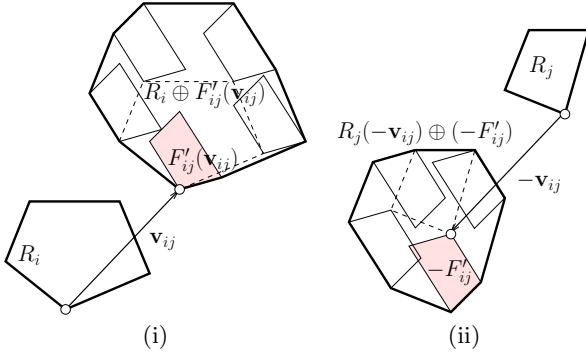

Figure 3: (i) The shaded region is the feasible region of node j for a particular location of node i. Node j stays inside the Minkowski sum of two polygons R_i and $F'_{ij}(\mathbf{v}_{ij})$. (ii) The constraint for edge ij also restricts that only points in $R_j(-\mathbf{v}_{ij}) \oplus (-F'_{ij})$ are possible deployment locations for node i.

the feasible region $F'_{ij}(\mathbf{v}_{ij})$. Now node i stays inside the region R_i. Thus node j stays in the feasible region $F'_{ij}(\mathbf{v}_{ij})$ for every possible position of node i in R_i, as shown in Figure 3 (i). Formally, for two convex polygons P and Q, the Minkowski sum is defined as

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\},$$

where $p + q$ is the sum of the vectors p, q. Then node j stays in the Minkowski sum of two polygons $R_i \oplus F'_{ij}(\mathbf{v}_{ij})$, as shown in Figure 3 (i). This puts a new constraint on the location of node j. Namely, the new deployment region should be $R_j \bigcap \left(R_i \oplus F'_{ij}(\mathbf{v}_{ij})\right)$.

Furthermore, it is not just the deployment region for node j that may shrink. The constraint may also shrink the deployment region R_i. Namely, if a part of the polygon

$R_i \oplus F'_{ij}(\mathbf{v}_{ij})$ is not feasible for node j, this indicates that part of R_i is not feasible. Intuitively, for a valid deployment point in R_j, there should be a point p in the deployment region R_i such that node j stays inside the feasible region F'_{ij} with node i at position p. This implies that only the points inside the polygon $R_j(-\mathbf{v}_{ij}) \oplus (-F'_{ij})$ are valid deployment points, as shown in Figure 3 (ii). Thus, the deployment region for node i shrinks to $R_i \bigcap \left(R_j(-\mathbf{v}_{ij}) \oplus (-F'_{ij})\right)$.

The Minkowski sum can be implemented efficiently. If P and Q are convex polygons with m and n vertices, both operations result in convex polygons with complexity $O(m + n)$ vertices and can be implemented in time $O(m + n)$ [6].

The distributed weak deployment method can be implemented using this Minkowski sum interpretation. At each iteration, a node i first obtains the current deployment regions of all of its neighbors $N(i)$. For each neighbor j, both the constraint $F'_{ij}(\mathbf{v}_{ij})$ and $F'_{ji}(-\mathbf{v}_{ij})$ will shrink the deployment region R_i. The algorithm stops when the deployment regions do not change. We summarize the algorithm in the pseudo-code below. Each node has a flag *not_done(i)* that indicates whether it needs to perform an iteration. Essentially, if a node i shrinks its deployment region, it informs its neighbors by setting their flags to 1.

Initialize: $R_i = \mathbb{R}^2$ and not_done(i) = 1
while (not_done(i)) do
    $N(i)$ = set of neighbors of node i
    Collect deployment regions R_j, for $j \in N(i)$
    $M = \bigcap_{j \in N(i)} R_j \oplus F'_{ji}(\mathbf{v}_{ji})$
    $N = \bigcap_{j \in N(i)} R_j(-\mathbf{v}_{ij}) \oplus (-F'_{ij})$
    $R'_i = R_i \cap M \cap N$
    if ($R_i \neq R'_i$) {$R_i = R'_i$, $\forall j \in N(i)$: not_done(j) = 1}
    else not_done(i)= 0

From the Minkowski interpretation of the construction of regions R_i, and the fact that the Minkowski sum of two convex polygons is bounded by translates of edges from the two polygons, we immediately get the following lemma, which justifies the claim made in section 3 about the global LP:

LEMMA 2. *The edges bounding the regions R_i at each iteration of the distributed weak deployment algorithm, and therefore also the edges bounding the weak deployment regions R^*_i, have orientations among those of the set of all edges bounding the input constraint regions F'_{ij}.*

4.1.3 Convergence

We now show that the distributed weak deployment method converges in a finite number of iterations to the same solution given by the global (centralized) LP-based solution; the proof is based on the three lemmas that follow.

THEOREM 3. *The distributed weak deployment method converges to the solution of the weak deployment problem.*

LEMMA 4. *At iteration k, the deployment region $R_i^{(k)}$ is a superset of R^*_i, the deployment region of the global LP.*

PROOF. We prove this claim inductively on the number of iterations. Denote by M the global LP algorithm. Suppose in the global LP algorithm, the weak deployment region of node i is R^*_i. R^*_i is the projection of the feasible region of M in $\mathbb{R}^{2n}$ on the x_i, y_i plane. At the initial step, each node i has a deployment region $\mathbb{R}^2$, which obviously include R^*_i.

Suppose at the k-th iteration the current deployment region $R_i^{(k)}$ is a superset of R_i^*, for every node i. Now we claim that the deployment region in the next iteration, $R_i^{(k+1)}$, is also a superset of R_i^*.

Consider the local constraint region at node i, denoted by $M_i^{(k)}$, the constraints are $L(i) = U^{(k)}(i) \bigcup C_g(i)$, where $C_g(i)$ is the subset of the constraints in the global LP that involves node i and its neighbors, $U^{(k)}(i)$ is the constraints imposed by the current deployment regions $R_i^{(k)}$ and $R_j^{(k)}$, for $j \in N(i)$. Now we define a global LP with the extra constraints $U^{(k)}(i)$ besides those in M, denoted by M'. Since $R_i^{(k)}$ is a superset of R_i^* for every node i, the constraints $U^{(k)}(i)$ are redundant constraints in the problem M'. Thus we can safely throw away the constraints $U^{(k)}(i)$ with the feasible region unchanged. This says, the feasible region of M' in $\mathbb{R}^{2n}$ is the same as that of the original LP M. Now notice that we can throw away the constraints that does not involve node i. This will only enlarge the feasible region. Now since we don't have any constraints involving nodes other than i or $N(i)$. We can throw away those variables with the projection for node i unchanged. This is exactly the local constraint formulation $M_i^{(k)}$. The weak deployment region for problem $M_i^{(k)}$ is a superset of R_i^*. This finishes the induction. $\square$

LEMMA 5. *When the algorithm stops, the weak deployment region is the same as R_i^* for every node i.*

PROOF. The proof is by contradiction. Let R_i denote the deployment region produced by the local constraint problem for node i. Suppose there is a node i with R_i a strict superset of R_i^*, and let $p \in R_i \setminus R_i^*$. This implies that if node i is at p, then there is no feasible assignment of locations to all nodes. Now we prove that the iterative algorithm can continue and improve (reduce) the deployment region R_i of node i, contradicting the fact that it stopped.

We first define by $\hat{F}'_{ij}(p)$ the mutually feasible region of node j if node i is at p – this is the collection of points q that are inside the region of uncertainty of p with the condition that p is also inside the region of uncertainty of q. We note that $\hat{F}'_{ij}(p)$ is a convex polygon whose shape only depends on F'_{ij} and F'_{ji}. Now consider a self-disjoint path $\mathcal{P}$ in the graph G from node i to some node ℓ and propagate information along $\mathcal{P}$. We can define the feasible region of node ℓ, denoted by $F_\ell(\mathcal{P}, p)$, as the collection of positions of node ℓ enforced by the constraints of edges of $\mathcal{P}$ and the fact that node i is at location p. Essentially $F_\ell(\mathcal{P}, p) = \oplus_{jk \in \mathcal{P}} \hat{F}'_{jk}$ in the reference space with p at the origin. It can be observed that $F_\ell(\mathcal{P}, p)$ is a convex polygon whose shape only depends on the polygons F'_{jk}, F'_{kj}, for all of the edges jk on the path $\mathcal{P}$.

We now propagate the constraints starting from point p and compute the feasible regions of a node ℓ along each path from node i. For each node ℓ, we take the intersection, denoted $H_\ell(p)$, of all of these feasible regions, each corresponding to a different path from node i. Suppose $H_\ell(p) \neq \emptyset$ for each node ℓ, and that the origin o is not in $H_1(p)$ for the node 1; then, we can find a feasible location assignment for all of the nodes when node i is at p. This shows that p lies in the set R_i^*, which contradicts our choice of p.

Thus, it must be the case either that (1) for some node ℓ, $H_\ell(p) = \emptyset$, or that (2) $H_1(p)$ does not include the origin, o.

We first show that (1) cannot happen. Indeed, the shape of the regions $F_\ell(\mathcal{P}, p)$ does not depend on the position of node i. Now we choose p^* as a feasible location for node i, $p^* \in R_i^*$. Correspondingly there is a feasible position for node ℓ, say at q^*. Then $F_\ell(\mathcal{P}, p)$ and $F_\ell(\mathcal{P}, p^*)$ have the same shape. Thus $H_\ell(p)$ and $H_\ell(p^*)$ have the same shape. However, $H_\ell(p^*)$ is not empty – it includes the feasible position q^* for node ℓ. This completes the argument for case (1).

Now consider case (2): $H_1(p)$ does not include the origin o. Then there is a path from node i to node k, say $\mathcal{P}$, such that $F_1(\mathcal{P}, p)$ does not include the origin. Define $F_1^{-1}(\mathcal{P}, o)$ to be the collection of positions for node i such that $F_1(\mathcal{P}, p)$ includes the origin. Point p is not included in $F_1^{-1}(\mathcal{P}, o)$. By the definition of mutual feasibility, $F_1^{-1}(\mathcal{P}, o) = F_i(\mathcal{P}^{-1}, o)$, where $\mathcal{P}^{-1}$ is the inverse path of $\mathcal{P}$. This means that if we fix node 1 at the origin, then p cannot be a feasible position for node i by the constraints along the path $\mathcal{P}^{-1}$. Indeed, we can apply the local constraint region algorithm starting from 1 along the path $\mathcal{P}^{-1}$. At some nodes on $\mathcal{P}^{-1}$ we can shrink the feasible regions – at least we can shrink the current deployment region R_i, since p should definitely be excluded. This shows that the algorithm could not have stopped (converged) with the region R_i. $\square$

LEMMA 6. *The distributed weak deployment algorithm terminates after a finite number of steps.*

PROOF. We first make the following claim. For a node j, recall that $H_j(1)$ is the intersection of the feasible regions computed along each simple path from the origin o (node 1) to j. Now we claim that the region $H_j(1)$ is the weak deployment region R_j^* for node j. If otherwise, suppose that there is a point $p \in H_j(1)$ that is not in R_j^*. Then by the same argument as in Lemma 5, it must be the case that $H_1(p)$ does not include the origin; thus, $H_j(1)$ does not contain p, which is a contradiction.

Now for each node i, we keep a list $\mathcal{L}_i = \{\mathcal{P}\}$ of simple paths that start from the origin and end at node i. Only the paths in $\mathcal{L}_i$ contribute to the current feasible region of node i. Initially the lists $\mathcal{L}_i$, for all i, are initialized to be $\{i\}$. After an iteration that updates the feasible region of node j by using the feasible region of node i, the list of paths for node j will include new paths $\mathcal{P}_j$, for all $\mathcal{P} \in \mathcal{L}_i$.

Now we claim that the current feasible region of node j, $R_j = \cap_{\mathcal{P} \in \mathcal{L}_j} F_j(\mathcal{P}, o)$. This is proved inductively. The claim is true for the base case. For a neighbor j of the origin o, the feasible region will shrink from $\mathbb{R}^2$ to $\hat{F}'_{1j}(o)$. The list $\mathcal{L}_j$ will contain a new path $1j$. Now for an iteration from node i to j, we will intersect R_j with $R_i \oplus \hat{F}'_{ij}$. By the induction hypothesis, $R_i \oplus \hat{F}'_{ij} = \left(\cap_{\mathcal{P} \in \mathcal{L}_i} F_i(\mathcal{P}, o) \right) \oplus \hat{F}'_{ij} = \cap_{\mathcal{P} \in \mathcal{L}_i} \left(F_i(\mathcal{P}, o) \oplus \hat{F}'_{ij} \right)$. Recall that $F_i(\mathcal{P}, o) = \oplus_{\ell k \in \mathcal{P}} \hat{F}'_{\ell k}$. Thus, $R_j = \cap_{\mathcal{P} \in \mathcal{L}_j} F_j(\mathcal{P}, o)$, proving the claim.

This claim shows that the feasible region of a node j will not shrink unless the list $\mathcal{L}_j$ is increased. Combined with the first claim, it shows that the algorithm converges after a finite number of steps – essentially when the list for each node i has already contained all possible simple paths from the origin to node i. Since there is a finite number of simple paths from the origin to each node i in the network. The list $\mathcal{L}_i$ cannot grow indefinitely. Thus, the algorithm stops after a finite number of iterations. $\square$

4.1.4 Finding a Single Feasible Solution

We note that the goal for the weak deployment problem is to provide an upper bound on the solution space. If we want to obtain a feasible location assignment, we can start

with the weak deployment regions and produce one feasible solution. Essentially, we iterate through the nodes in some arbitrary order. At the beginning of the ith iteration, all the nodes from 1 to $i-1$ have been assigned locations and the rest of the nodes have feasible regions that are compatible with the assignment for node $1, \ldots, i-1$. Now we fix node i at a point in its current feasible region and run the weak deployment algorithm with the partial assignment. The deployment regions for the rest of the nodes will shrink. At the end of the algorithm, we have a feasible assignment to all the nodes.

4.2 A Distributed Method for Strong Deployment

We now give an algorithm for strong deployment using distributed methods. The strong deployment problem can be reduced to the weak deployment problem, by Lemma 1. Recall that we specify the shape of the strong deployment region (e.g, a rectangle with fixed aspect ratio) and search for the maximum size regions for all the nodes. We describe the algorithm by using a rectangular deployment region. The algorithm for deployment regions of other shapes is similar. We first assume that we have guessed the maximum scale of the strong deployment region. Now we show how to find all the strong deployment regions in a distributed manner. For the strong deployment region S_i, we fix one point, called the center, as a reference point for the position of S_i. In the reference frame with the center at the origin, the corners of the rectangle are represented by vectors $\mathbf{c_k}$, $k = 1, 2, 3, 4$. We look for the placement of the centers of the S_i's such that the constraints are satisfied for every choice of location assignment in S_i.

Consider two nodes i an j. For each placement of the center for S_i, we consider possible placements of the center for S_j. By Lemma 1, node j must stay completely inside the region $R = \bigcap_k F'_{ij}(\mathbf{v_{ij}} + \mathbf{c_k})$, which is the intersection of the feasible regions for each corner of S_j, as shown by the shaded region in Figure 4. This defines a region H_{ij} for the center of j's deployment region to be such that the deployment region S_j lies completely inside R (see Figure 4). It is easy to compute H_{ij} in linear time. We consider H_{ij} to be the "feasibility region" for node j's center with respect to node i's center. Using these new feasibility regions for the centers of the neighboring node regions, we can now use our distributed weak deployment algorithms to find regions for the center of the strong deployment regions. Using ideas from the previous subsection, we obtain a feasible solution from these regions, and we have a solution to the strong deployment problem.

Note that we had assumed the knowledge of the maximum size of the S_i's. In order to determine the size, we can use repeated doubling and binary search, running the method $O(\log r + B)$ times, where r is the actual value of the objective function and B is the number of bits used for representing r.

5. HARDNESS OF LOCALIZATION

In this section we study the computational complexity of localization by noisy measurements. Given a graph $G = (V, E)$ together with noisy measurement of edge lengths $\mathcal{R}$ and angles Θ (with respect to a universal north), the localization problem asks whether one can find an assignment $\mathcal{E}$ of locations to all the vertices that satisfy the constraints.
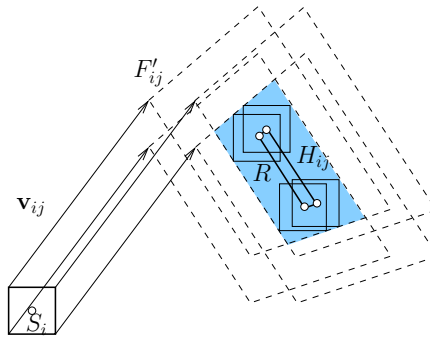

Figure 4: The feasibility region H_{ij} for center of S_j with respect to placement of S_i.

In particular, suppose $\mathcal{E}(u)$ is the embedded point in $\mathbb{R}^2$ for node u. For each edge uv, we have

$$r_1(u, v) \leq r(\mathcal{E}(u), \mathcal{E}(v)) \leq r_2(u, v),$$

$$\theta_1(u, v) \leq \theta(\mathcal{E}(u), \mathcal{E}(v)) \leq \theta_2(u, v),$$

where $r(\mathcal{E}(u), \mathcal{E}(v))$ and $\theta(\mathcal{E}(u), \mathcal{E}(v))$ are the length and angle of edge uv, $r_1(u, v), r_2(u, v), \theta_1(u, v), \theta_2(u, v)$ are the lower and upper bound on the edge length and angle of uv respectively, as shown in Figure 2.

We first note that this formulation includes a number of interesting subcases whose computational complexity is known. One subcase is unit disk graph embedding, where neighboring nodes are embedded to have distance no more than 1 and non-neighboring nodes are of at least distance 1 apart. It is shown by Breu and Kirkpatrick [3] that unit disk graph embedding with only the connectivity information, i.e., with infinite noise in the angle measurement and only a upper (lower) distance bound on the neighboring (non-neighboring) pairs, is **NP**-complete. In fact, a relaxed version, quasi-unit disk graph embedding is still hard, where neighboring nodes are within distance 1 and non-neighboring nodes are of at least distance $\sqrt{3/2}$ away [12]. Unit disk graph embedding remains **NP**-hard even if we have exact angle information [4].

However, with exact angle and distance information, i.e., $r_1 = r_2$, $\theta_1 = \theta_2$, the localization problem is trivially in **P**. We wish to examine this spectrum and find out at what point does the localization problem become hard. We first show that even with ε noise in distance measurement and δ noise in angle measurement, $\varepsilon > 0, \delta > 0$ arbitrarily small, the problem immediately becomes hard. Further, the embedding problem remains hard if we have accurate distance information but relative angle information. In other words, we know the magnitude of the angle measurements but do not know the sign, thus allowing flips. Therefore, unless **P** = **NP**, essentially only the case with exact information about both angle and distance is polynomially tractable. We summarize the hardness results in Figure 5.

5.1 Hardness of Embedding with Noisy Distance and Angle Measurements

Our reduction is from 3SAT. A 3SAT problem consists of a set of Boolean variables and clauses. Each clause has at most three literals, either negated variables or unnegated. A clause is satisfied if one of the literal has value 1. The 3SAT problem asks for an assignment to the variables such that all

Input	Hardness	ref.
UDG graph only	NP-hard	[3, 12]
UDG graph with $O(1)$-hop distances	NP-hard	[1]
UDG graph with $O(1)$-hop angles	NP-hard	[4]
Noisy $O(1)$-hop distances and angles	NP-hard	this paper
Accurate distances and relative angles	NP-hard	this paper
$O(1)$-hop angles & distances	in P	[4]
$\Omega(n^2)$ pairs distances	in P	[2, 25]
all pairs angles	in P	[4]

Figure 5: A summary of the hardness results.

the clauses are satisfied. We use the special version of the $3SAT$, in which each variable occurs in at most 3 clauses. This version is known to be **NP**-Hard too. An instance of 3SAT can be formulated as an rectilinear plane graph G with the clauses and variables as vertices and an edge between a variable node and a clause node if that variable appears in the clause. This graph can be obtained in polynomial time. See Figure 6 for an example. We now show that for each 3SAT instance we can construct a planar graph such that the planar graph has a valid embedding under noisy angle and distance constraints if and only if we can solve the 3SAT problem.
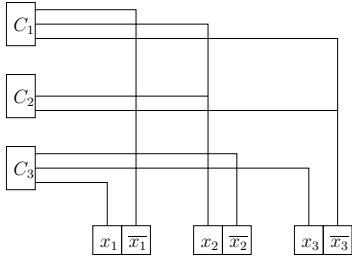

Figure 6: The graph G_C of a 3SAT instance $(\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$.

We have the following building blocks for construction. Given any $\varepsilon, \delta > 0$, for each edge the (absolute) error on the distance measurement and angle measurement (with respect to the universal north) are ε and δ respectively. In other words, for an edge $e = uv$ the measurement length is ℓ_e and the angle with respect to the universal north direction is θ_e. The upper and lower bound on the distance between the embedded positions $\mathcal{E}(u), \mathcal{E}(v)$ are $\ell_e + \varepsilon/2$ and $\ell_e - \varepsilon/2$, respectively. Similarly the angle of $\mathcal{E}(u)\mathcal{E}(v)$ is bounded by $\theta_e + \delta/2$ and $\theta_e - \delta/2$.
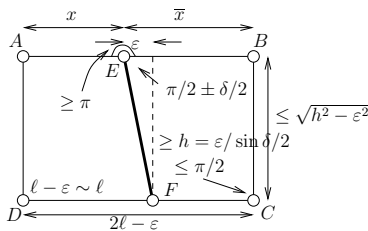

Figure 8: Variable.

- **Truth-Setters:** These are the truth-setting components for the variables. See Figure 8. Basically we have a fixed-length rectangle $ABCD$ with a tilted edge EF inside, whose length is longer than the vertical side

length of the bounding box. There are only 2 possible embeddings for this component – one with the center segment sloping right and one sloping left. One in which it slopes to the left is setting the variable to 1. The right-sloping configuration corresponds to setting this variable to 0. For the positive literal, the left half of the top edge is used; the negative literal corresponds to the right half. A literal has value 1 (or 0) if its corresponding length is $\ell - \varepsilon$ (or ℓ).

This block is realized by a graph with appropriate constraints on the edges. The four inner angles of the polygon $AEBCFD$ at vertices A, B, C, D are in between $\pi/2 - \delta$ and $\pi/2$. The angles $\angle AEB$ and $\angle CFD$ (measured counter-clockwise) are upper bounded by π. Since the sum of all the inner angles of polygon $AEBCFD$ is 4π, both AEB and DFC are flat and $ABCD$ is a rectangle. The lengths of AD and BC are between $\sqrt{h^2 - \varepsilon^2} - \varepsilon$ and $\sqrt{h^2 - \varepsilon^2}$. The length of the tilted edge EF is between h and $h + \varepsilon$, where $h = \varepsilon/\sin \delta/2$. The angle of EF with the vertical line is at most $\delta/2$. The four horizontal edges AE, EB, DF, FC have lengths between $\ell - \varepsilon$ and ℓ. One can verify that indeed there are only two ways to embed this graph so that all of the constraints are satisfied. These two embeddings correspond to the variable being set to 0 and 1, respectively.

- **Propagators and Cross-overs:** These are for propagating the truth value along the edges of G to the clauses. They attach themselves to the truth-setting components and the clauses. Cross-overs are for the crossing edges in G. See Figure 7 (i) and Figure 7 (ii). To realize them, we restrict the angles between all adjacent edges to be upper bounded by $\pi/2$. For a valid embedding in the plane, all of them must be $\pi/2$. Thus all the edges are axis-aligned.

- **Bends:** These are for the corners in the edges in G. See Figure 7 (iii). The diagonals ensure that the same truth value gets propagated around the bend.

- **Clause:** We need to ensure that the three variables are not simultaneously 0, i.e. their lengths in the truth-setting components are not simultaneously ℓ. See Figure 7 (iv). We first set the inner angles of the polygon in Figure 7 (iv) accordingly such that the shape is a rectangle. Moreover, the height of the rectangle is upper bounded by $3\ell - \varepsilon$. Thus, one of the input literals must take a value 1. This is exactly the 3SAT satisfiability condition.

Now with the basic building blocks we can establish the reduction from 3SAT to our localization problem. For each 3SAT instance, we can find a localization problem such that the localization problem has a feasible solution if and only if the 3SAT instance is satisfiable. This shows that the localization problem with angle/distance measurements, even with small noise, is still NP-hard.

5.2 Hardness of Embedding with Accurate Distance and Relative Angle Measurements

Here we consider the problem of embedding with exact (or noisy) angle and distance information, but only relative angle information. In other words, we know the magnitude
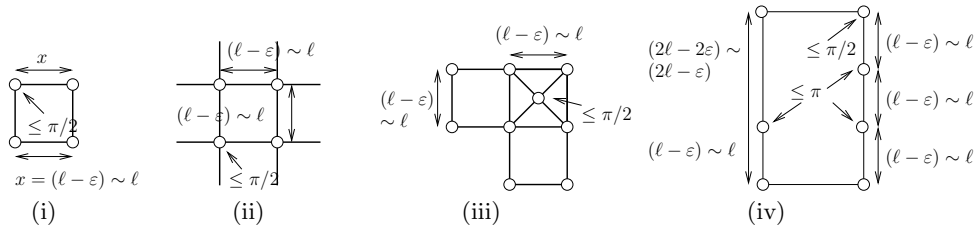
Figure 7: (i) Propagator; (ii) Cross-over; (iii) Corner; (iv) Clause.

of the angle measurements but do not know the sign, thus allowing flips; e.g., node i knows that node j is 30 degrees from north, but does not know if it is 30 degrees clockwise or counter-clockwise of north. We prove that this version of the embedding problem is also hard, under the unit disk graph assumption. We again reduce from $3SAT$ and use a similar construction as in [3]. The corresponding building blocks for the construction are illustrated in the figures below.
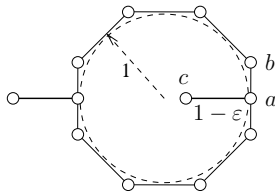
Figure 9: Basic octagon.

The construction exploits the fact that, because of the UDG assumption, two non-neighboring nodes cannot be too close to each other in the final embedding. As before, consider the graph G described earlier that represents the logical formula. The truth setting is reflected now as an orientation of the edges (as opposed to the width in the earlier construction). We use a basic octagon and a chain of such octagons to represent each edge. Notice that we only have relative angle information; the small "twig" attached at the boundary of the octagon can be either completely inside or outside. Note that due to the UDG assumption, all of the little twigs in the octagons have to be oriented in the same direction, since two twigs cannot co-exist in an octagon; see Figure 9 and Figure 11. The basic building blocks are:

- **Truth-Setters** Refer to Figure 10. There are only two possible embeddings, if the UDG constraint is to be respected. The arms emerging from the left correspond to the positive literal and the arms on the right side correspond to the negative literal. If the twigs in the arms are pointing out of the gadget, then that literal is set to 0.

- **Propagators and Corners** The basic chain of octagons is used. See Figure 11.

- **Clauses** The basic property of the clause gadget is that the three incoming arms cannot all possibly be oriented inwards (corresponding to a truth value of 0). See Figure 11 (iii) and (iv).

- **Cross-Overs** When two edges in the 3SAT graph cross, the orientations of the edges are propagated.

We again argue that the 3SAT problem is satisfiable if and only if the localization problem with accurate distances and relative angle information has a feasible solution.
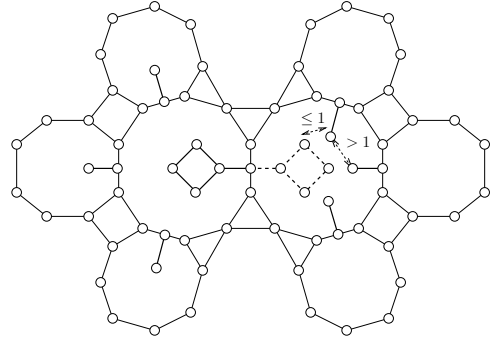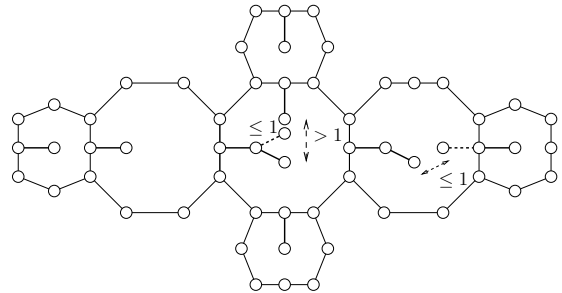

Figure 10: Variable.

Figure 12: Cross-Over

6. SIMULATION

We implemented our distributed weak deployment algorithm to evaluate its performance, especially the convergence rates, dependence on average degree, network size, anchor density, etc. Moreover, we test the method's robustness under link variations. We focused on the weak deployment problem, implemented with Minkowski sums, since the algorithm for the strong deployment problem uses the same mechanism but simply a different notion of feasible regions.

6.1 Experimental Setup

We tested the algorithm on the topological level; we do not explicitly consider issues on MAC layer, such as medium contention, packet loss and corruption, in evaluating the algorithm's convergence behavior. Due to the iterative nature of the algorithm, it can be easily adapted under an imperfect environment, as will be shown in section 6.4. The input was generated as follows.

- *Connectivity Graph:* We placed n nodes uniformly randomly in a squared region with size $[0, 400] \times [0, 400]$. Then we connected nodes within a chosen distance d of each other (simulating the communication range of sensor nodes). For our simulation, d is chosen (depending on n) such that the graph is connected.
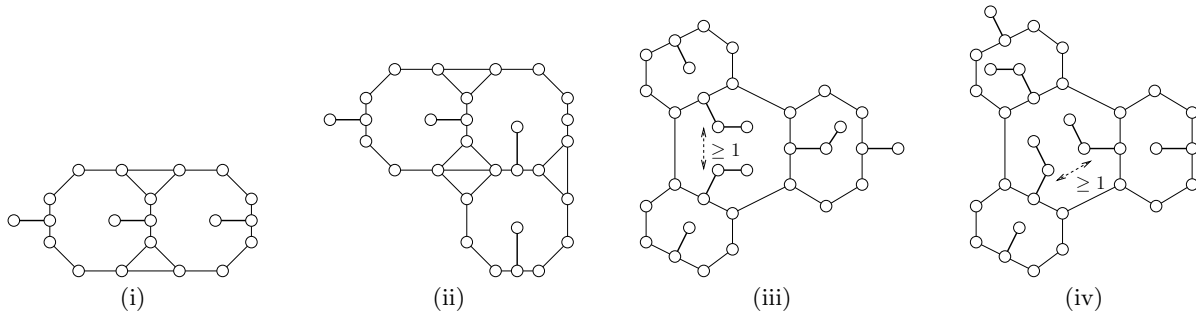
Figure 11: (i) Chain; (ii) Corner; (iii) and (iv) Two embedding of a clause gadget.

- *Noise Generation:* We currently use only axis-aligned rectangles as our feasibility regions. This is only for the purpose of simulation; as we described earlier, the algorithm works for arbitrary convex regions. These axis-aligned noise rectangles are generated by the following scheme. We first fix the distance and angle error at ε and δ respectively. Then we use the bounding box of the noise frustum shown in Figure 2. the dependence of the algorithm's output on ε and δ are studied in our experiments.

6.2 Convergence Rates

Many factors can potentially affect the convergence rate. We investigated the effect of these input parameters on the convergence rate, measured by the number of iterations it takes for the algorithm to converge. This evaluates roughly the number of message exchanges per node.

1. *Number of nodes* We keep the average degree the same and increases the network size. The number of iterations grows with a slow linear dependence on the number of nodes of the input graph. Refer to Figure 13 (a).

2. *Average degree of the nodes* Increasing the average degree implies increasing the number of constraints on a node, and, not surprisingly, increasing the rate of convergence of the regions; see Figure 13 (b).

3. *Number of fixed nodes* In the original definition of the weak deployment problem, one of the nodes is fixed to rule out global translation. However, in general, we may have the information about the location of more than one node. These can be incorporated in our algorithm and as expected, the number of iterations decreases as the number of anchor nodes increases. See Figure 13 (c). Of course if the fixed nodes are in a close neighborhood of each other, there would not be significant change in the number of iterations. This is the cause of the plateau regions in the graph.

6.3 Size of Deployment Regions

We measure the size of the (weak) deployment regions by the area of the rectangle. We use two statistics for the area of the regions obtained: *maximum* and *average*. We again vary the different parameters and plot the effect on size. To get an intuitive idea of the results, we include a snapshot of our simulation in Figure 14.

1. *Average degree of nodes* As seen in Figure 13 (d), the *maximum* and the *average* area of deployment regions both steadily decrease as the average degree increases.

This is not surprising, since an increase in average degree implies an increase in the number of edges and, thus, the number of constraints. Hence, it is natural for the regions to shrink in size.

2. *Number of fixed nodes* As before, we fix the coordinates of more than one node. As expected, the regions shrink in size as the number of fixed nodes increases; see Figure 13 (e). As in the case of convergence rates, there are plateau regions when the fixed nodes are in a small vicinity of each other.

6.4 Robustness to Link Failures

One interesting thing to study is how the convergence rate is affected when links are unstable, that is they go down at some instant and come back up again after some time. Our results confirm that if the joint union of all the intermediate graphs is connected, we get very close to the solution without any link failures. Our experiments were run as follows. We assign a probability $p > 0$ to each individual link and we start with all of the links active. At each iteration, a link toggles between active and inactive with probability p (that is they become inactive if they were on, and turned back on if they are off). We study the number of iterations and the size statistics of the regions for different probability p. See Figure 13 (f) and Figure 13 (g). There is *practically no variation* in the size statistics, i.e. *maximum* and *average* area, showing that our method is highly robust even for a high percentage of link failures. The number of iterations jumps up immediately when link dynamics are introduced; but then decreases rapidly until it becomes almost stable with slight fluctuations for $p > 0.20$. It may appear odd that the convergence rate when p is small is even slower. The reason is that, when the graph is disconnected at some instant, with a small p it takes longer to reconnect the graph.

6.5 Comparisons to SDP Methods

We compared our methods to the localization method for noisy data proposed by Biswas and Ye [2]. The algorithm in [2] uses semi-definite programming (SDP) for localization with noisy distance measurements between the nodes. The non-convex constraints are relaxed to yield an SDP formulation. In both SDP and our methods, upper and lower bounds are provided for the distance of a neighboring sensor. These bounds are generated using a multiplicative Gaussian distribution. The synthesized range is $d(1 \pm |N(0,1) * f|)$, where d is the actual distance and $f = 0.2$. The SDP algorithm however, does not use any angle information. By a comparison with SDP we show that even a rough angle
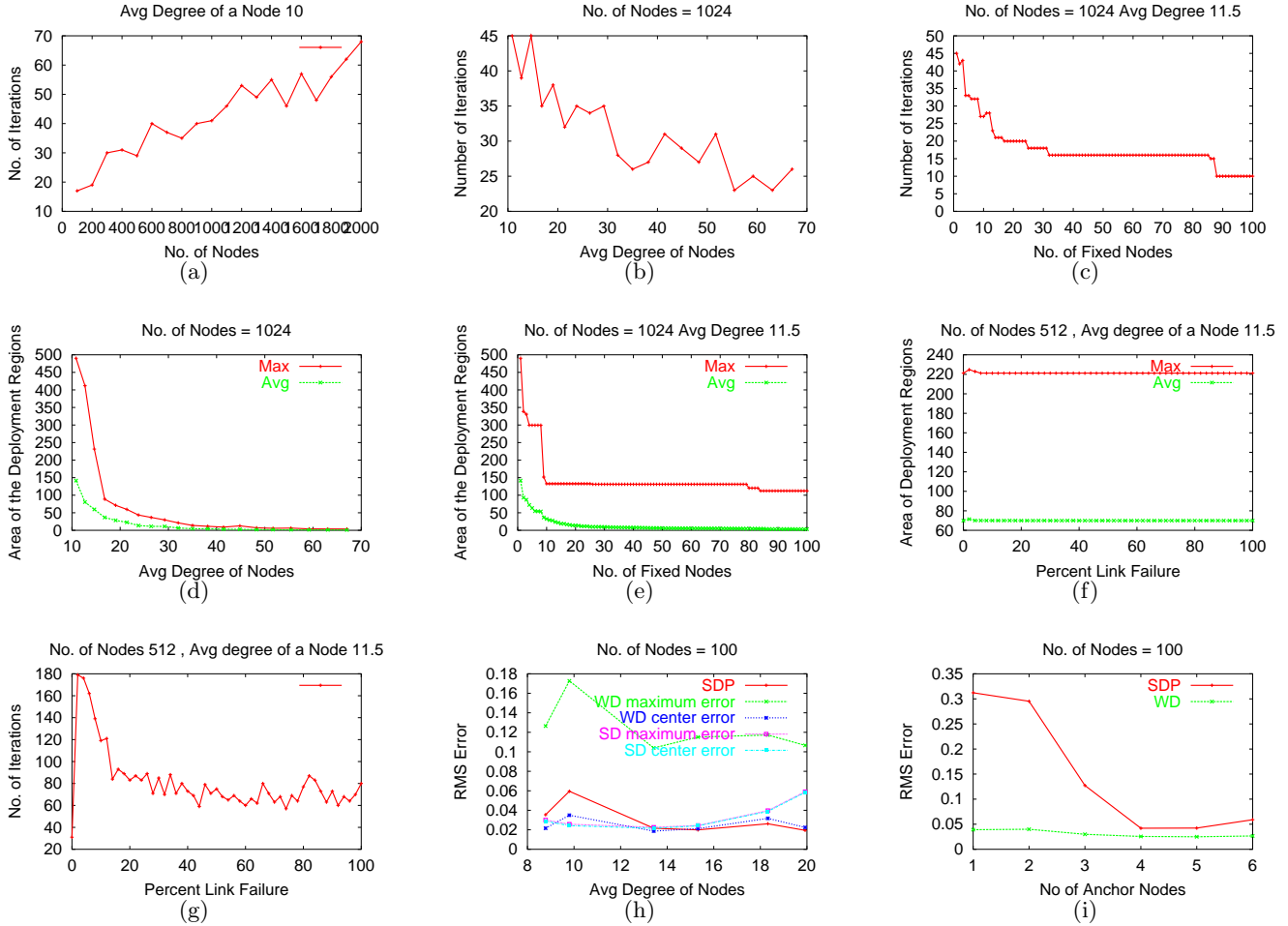
Figure 13: (a) Number of iterations vs. number of nodes; **(b)** Number of iterations v.s. average degree of nodes; **(c)** Number of iterations vs. number of fixed nodes; **(d)** Area statistics vs. average degree of nodes; **(e)** Area statistics vs. number of fixed nodes; **(f)** Area statistics vs. percentage of link failure; **(g)** Number of iterations vs. percentage of link failure; **(h)** and **(i)** Comparison of SDP with our distributed algorithms.

range of $\pi/4$ can help a lot in improving the localization results.

We compare the localization error, in particular, the root-mean-square error in both methods. Since our algorithms compute *regions* instead of actual points, we use either the center of the deployment region or the furthest point from the actual location as an estimate of the sensor location. The second metric gives an upper bound on localization error. We compare the SDP result with our weak and strong deployment regions under both error metrics; see Figure 13 (h). As expected, the farthest points in the *weak deployment* regions are the most erroneous. The SDP method performs worse than our algorithm for low average degree and gets successively better when the average degree increases. However, our algorithm remains competitive, and the quality of our results is not dependent on the density of the graph.

We also note that the method proposed in [2] produces results whose quality depends significantly on the number of anchor nodes. We show that our algorithms perform considerably better, showing that having noisy angle information in addition to noisy distances alleviates the dependence on

anchors. See Figure 13 (i).

7. CONCLUSION AND FUTURE WORK

In this paper we initiated an algorithmic study of the sensor localization problem with noisy distance and angle information. Interesting open problems remain.

Since the localization problem is NP-hard, we use convex polygons to (outer) approximate the regions of uncertainty. Can we bound the approximation ratio caused by this relaxation? Little seems to be known about approximation algorithms for localization; we know only of Moscibroda *et al.* [14], which gives a polylogarithmic approximation ratio. Another natural question is to bound the convergence rate of our distributed weak deployment algorithm.
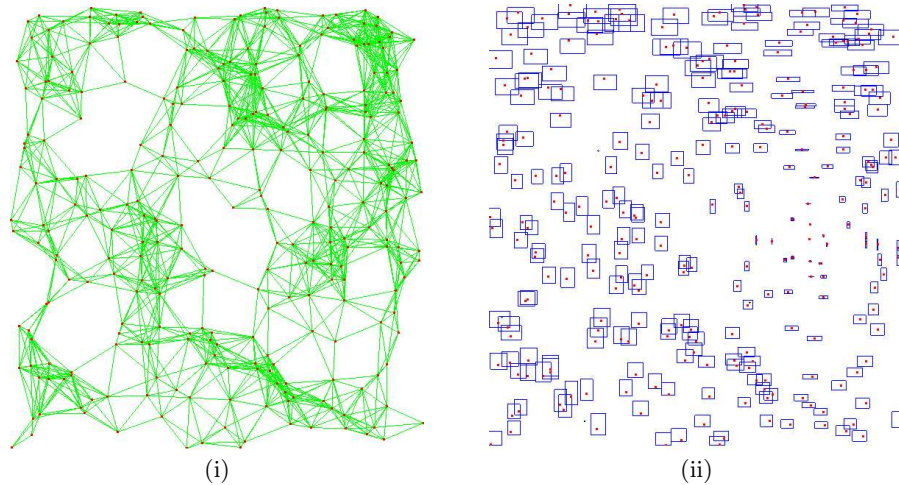
(i) (ii)

Figure 14: (i) The connectivity graph; (ii) Weak deployment regions. (280 nodes, average degree is 11.7)

LP-based methods of localization, which led to the development of our methods. J. Gao thanks A. Jiang for suggesting to study the hardness of localization with noisy measurements.

8. REFERENCES

[1] J. Aspnes, D. Goldenberg, and Y. R. Yang. On the computational complexity of sensor network localization. In *Proc. 1st Internat. Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 32–44, 2004.

[2] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proc. 3rd Internat. Symposium on Information Processing in Sensor Networks*, pages 46–54, 2004.

[3] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry. Theory and Applications*, 9(1-2):3–24, 1998.

[4] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *Proc. 6th ACM Internat. Symposium on Mobile Ad Hoc Networking and Computing*, pages 181–192, 2005.

[5] M. Bǎdoiu, E. D. Demaine, M. T. Hajiaghayi, and P. Indyk. Low-dimensional embedding with extra information. In *Proc. 20th Annual Symposium on Computational Geometry*, pages 320–329, 2004.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[7] L. Doherty, L. E. Ghaoui, and S. J. Pister. Convex position estimation in wireless sensor networks. In *Proc. IEEE INFOCOM*, vol 3, pages 1655–1663, 2001.

[8] C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. In *Proc. Internat. Symposium on Grpah Drawing*, pages 273–284, 2004.

[9] S. Guha, R. Murty, and E. G. Sirer. Sextant: a unified node and event localization framework using non-convex constraints. In *Proc. 6th ACM Internat. Symposium on Mobile Ad Hoc Networking and Computing*, pages 205–216, 2005.

[10] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proc. 9th Internat. Conference on Mobile Computing and Networking*, pages 81–95, 2003.

[11] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning Systems: Theory and Practice*. Springer, 5th ed., 2001.

[12] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit disk graph approximation. In *Proc. Joint Workshop on Foundations of Mobile Computing*, pages 17–23, 2004.

[13] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proc. 2nd Internat. Conference on Embedded Networked Sensor Systems*, pages 50–61, 2004.

[14] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proc. Joint Workshop on Foundations of Mobile Computing*, pages 8–16, 2004.

[15] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proc. IEEE GLOBECOM*, pages 2926–2931, 2001.

[16] D. Niculescu and B. Nath. *Ad hoc* positioning system (APS) using AOA. In *Proc. IEEE INFOCOM*, vol 22(1), pages 1734–1743, 2003.

[17] D. Niculescu and B. Nath. Error characteristics of ad hoc positioning systems (APS). In *Proc. 5th ACM Internat. Symposium on Mobile Ad Hoc Networking and Computing*, pages 20–30, 2004.

[18] R. O'Dell and R. Wattenhofer. Theoretical aspects of connectivity-based multi-hop positioning. In *Theoretical Computer Science*, 344: 47–68, 2005.

[19] N. B. Priyantha, H. Balakrishnan, E. D. Demaine, and S. Teller. Mobile-assisted localization in wireless sensor networks. In *Proc. INFOCOM*, 2005.

[20] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. 6th ACM Annual Internat. Conference on Mobile Computing and Networking*, pages 32–43, 2000.

[21] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proc. 9th ACM Internat. Conference on Mobile Computing and Networking*, pages 96–108, 2003.

[22] A. Savvides, C.-C. Han, and M. B. Strivastava. Dynamic fine-grained localization in *ad-hoc* networks of sensors. In *Proc. 7th ACM Internat. Conference on Mobile Computing and Networking*, pages 166–179, 2001.

[23] A. Savvides, H. Park, and M. Srivastava. The *n*-hop multilateration primitive for node localization problems. *ACM Mobile Networks and Applications*, 8(4):443–451, 2003.

[24] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. Localization from mere connectivity. In Proc. 4th ACM Internat. Symposium on Mobile Ad Hoc Networking and Computing, pages 201–212, 2003.

[25] A. M.-C. So and Y. Ye. Theory of semidefinite programming for sensor network localization. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 405–414, 2005.