Localizability of Wireless Sensor Networks: Beyond Wheel Extension

Buddhadeb Sau[1] and Krishnendu Mukhopadhyaya[2]

[1] Department of Mathematics, Jadavpur University, Kolkata, India, bsau@math.jdvu.ac.in
[2] ACM Unit, Indian Statistical Institute, Kolkata, India, krishnendu@isical.ac.in

Abstract. A network is called *localizable* if the positions of all the nodes of the network can be computed uniquely. If a network is localizable and embedded in plane with *generic configuration*, the positions of the nodes may be computed uniquely in finite time. Therefore, identifying localizable networks is an important function. If the complete information about the network is available at a single place, localizability can be tested in polynomial time. In a distributed environment, networks with *trilateration orderings* (popular in real applications) and *wheel extensions* (a specific class of localizable networks) embedded in plane can be identified by existing techniques. We propose a distributed technique which efficiently identifies a larger class of localizable networks. This class covers both trilateration and wheel extensions. In reality, exact distance is almost impossible or costly. The proposed algorithm based only on connectivity information. It requires no distance information.

Key words: Wireless sensor networks, graph rigidity, localization, localizable networks, distributed localizability testing.

1 Introduction

A *sensor* is a small sized and low powered electronic device with limited computational and communicating capability. A *sensor network* is a network containing some ten to millions of sensors. Wireless sensor networks (WSNs) have wide-ranging applications in problems such as traffic control, habitat monitoring, battlefield surveillance (e.g., intruder detection or giving assistance to mobile soldiers etc.), fire detection for monitoring forest-fires, disaster management, to alert the appearance of phytoplankton under the sea, etc. WSNs can help in gathering information from regions, where human access is difficult. To react to an event detected by a sensor, the knowledge about the position of the sensor is necessary. Sensor deployment may be random. For example, they may be dropped from an air vehicle with no pre-defined infrastructure. In such cases, the positions of the sensors are completely unknown to start with. The problem of finding the positions of nodes in a network is known as *network localization*.

A wireless ad-hoc network embedded in $\mathbb{R}^m$ (m-dimensional Euclidean space) may be represented by a *distance graph* $G = (V, E, d)$ whose vertices represent computing devices. A pair of nodes, $\{u, v\} \in E$ if and only if the Euclidean distance between u and v ($d(u,v) = |u - v|$) is known. Determining the coordinates of vertices in an embedding of G in $\mathbb{R}^m$ may be considered as graph realization problem [1,2,3,4]. A *realization* of a distance graph $G = (V, E, d)$ in $\mathbb{R}^m$ is an injective mapping $p : V \to \mathbb{R}^m$ such that $|p(u) - p(v)| = d(u,v), \forall \{u, v\} \in E$ (i.e., one-to-one assignment of coordinates $(x_1, \ldots, x_m) \in \mathbb{R}^m$ to every vertex in V so that the $d(u,v)$ represents the distance between u and v. The pair (G, p) is called a *framework* of G in $\mathbb{R}^m$. Two frameworks (G, p) and (G, q) are *congruent* if $|p(u) - p(v)| = |q(u) - q(v)|$, $\forall u, v \in V$ (i.e., preserving the distances between all pairs of vertices). A framework (G, p) is *rigid*, if it has no smooth deformation [5] preserving the edge lengths. The distance graph G is

generically globally rigid [5], if all realizations with *generic configuration*s (set of points with coordinates not being *algebraically dependent*) are congruent. A set $A = \{\alpha_1, \ldots, \alpha_m\}$ of real numbers is algebraically dependent if there is a non-zero polynomial h with integer coefficients such that $h(\alpha_1, \ldots, \alpha_m) = 0$. The graph G is termed as *globally rigid*, if all realizations of G are congruent. In this work, we consider realizations of a distance graph only in plane with generic configurations. Here onwards, all the discussions and results are concerned only in $\mathbb{R}^2$.

In real applications, positions of the nodes of a network may either be 1) estimated [6,7,8,9,10] within some tolerable error level or 2) uniquely realized. If all realizations of a network are congruent, the network is called *localizable*. A distance graph is localizable if and only if it is globally rigid. Starting from three *anchor*s (nodes with known unique position), all the nodes in a globally rigid graph can be uniquely realized. If the nodes cannot be localized using the given information, in several applications location estimation may serve well. Localization of a network is NP-hard [11] even when it is localizable [12]. Jackson and Jordán [12] proved that a graph is globally rigid, if and only if it is 3-*connected* and *redundantly rigid*. A graph is 3-connected, if at least three vertices must be removed to make the graph disconnected. A graph is redundantly rigid, if it remains rigid after removing any edge. Localizability of a graph can be answered in polynomial time [12] by testing the 3-connectivity and redundant rigidity when complete network wide information is available in a single machine. To gather complete network information in a single machine is infeasible or very costly. On the contrary, finding methods to recognize globally rigid graphs in distributed environments based on local information is still a challenging problem [13].

The rest of the paper is organized as follows. Section 2 describes motivation and contribution of this work. Section 3 introduces some classes of localizable graphs which include trilateration graph and wheel extension as their special cases. Section 4 defines the problem. It also describes a mapping of the problem into triangle bar recognition. Section 5 describes the proposed distributed algorithm of localizability testing. Section 6 proves correctness and performance analysis of the algorithm. Finally, we conclude in Section 7.

2   Background and our contribution

The most commonly used technique for localization is *trilateration* [14,15]. It efficiently localizes a *trilateration graph* starting from three anchors. A trilateration graph is a graph with a *trilateration ordering*, $\pi = (u_1, u_2, \ldots, u_n)$ where u_1, u_2, u_3 form a K_3 and every u_i ($i > 3$) is adjacent to at least three nodes before u_i in π. However, not all localizable networks admit trilateration ordering. Fig. 1 (a) and 1 (b) respectively show a *wheel graph* and a *wheel extension*
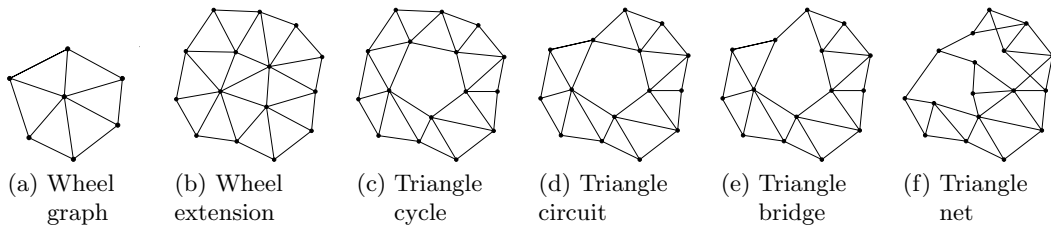

| (a) Wheel graph | (b) Wheel extension | (c) Triangle cycle | (d) Triangle circuit | (e) Triangle bridge | (f) Triangle net |

Fig. 1. Examples of some localizable graphs having different properties

graph which have no trilateration ordering. A wheel W_n with n vertices is a graph consisting of a cycle with $n-1$ nodes and a vertex which is adjacent to all vertices on the cycle. A *wheel extension* is a graph having an ordering $\pi = (u_1, u_2, \ldots, u_n)$ of nodes where u_1, u_2, u_3 form a K_3 and each u_i, $i > 3$, lies in a wheel subgraph containing at least three nodes before u_i in π. A wheel extension is generically globally rigid and its localizability can be identified efficiently and distributedly [13]. However, there are many more localizable graphs which do not have

wheel extensions. For example, Fig. 1 (c), 1 (d), 1 (e) and 1 (f) are examples of graphs which are generically globally rigid, but do not have wheel extensions.

The main contributions of this paper are as follows. It introduces some elementary class of localizable graphs *triangle cycle, triangle circuit, triangle bridge, triangle notch and triangle net*. Using these elementary classes of graphs, we build up a new family of generically globally rigid graphs called *triangle bar*. Trilateration graphs and wheel extensions are special cases of triangle bars. We propose an efficient distributed algorithm that recognizes triangle bars starting from a K_3 based only on connectivity information. It requires no distance information. In real applications, exact node distance is impossible or costly. However, several localizable graphs still fall outside the class triangle bar. To the best of our knowledge, distributedly recognizing an arbitrary localizable network still is an open problem.

3   Rigidity and localizability of triangle bar

Unique realizability is closely related to graph rigidity [1,2]. The realizability testing of a distance graph $G = (V, E, d)$ is NP-hard [11]. We expect data are consistent to have a realization, if the distance information is collected from an actual deployment of devices. A realization of G may be visualized as a frame constructed by a finite set of hinged rods. The junctions and free ends are considered as vertices of the realization and rods as the edges. With perturbation on the frame, we may have a different realization preserving the edge distances. The realizations obtained by flipping, rotating or shifting the whole structure are congruent. By *flip*, *rotation* or *shift* on a realization, we mean a part of the realization is flipped, rotated or shifted. If two globally rigid graphs in $\mathbb{R}^2$ share exactly one vertex in common, one of them may be rotated around the common vertex keeping the other fixed. Such a vertex is called a *joint*. If two globally rigid graphs in $\mathbb{R}^2$ share exactly two vertices, rotation about these vertices is no longer possible, but one of the graphs may be flipped, about the line joining the two common vertices, keeping the other fixed. This pair of vertices is called a *flip*.

Lemma 1 ([16]). *If two globally rigid subgraphs, B_1 and B_2, of a graph embedded in plane share at least three non-collinear vertices, then $B_1 \cup B_2$ is globally rigid.*

In this section, we formally introduce some elementary classes of localizable graphs: *triangle cycle, triangle circuit, triangle bridge, triangle notch, triangle net*. Using these elementary classes, a larger class of localizable graphs *triangle bar* is formally defined.

3.1   Triangle cycle, triangle circuit and triangle bridge

Let $\mathcal{T} = (T_1, T_2, \ldots, T_m)$ be a sequence of distinct triangles such that for every i, $2 \le i \le m-1$, T_i shares two distinct edges with T_{i-1} and T_{i+1}. Such a sequence $\mathcal{T}$ of triangles is called a *triangle stream* (Fig. 2 (a)). $G(\mathcal{T})$ is the graph constructed by taking the union of the T_is in
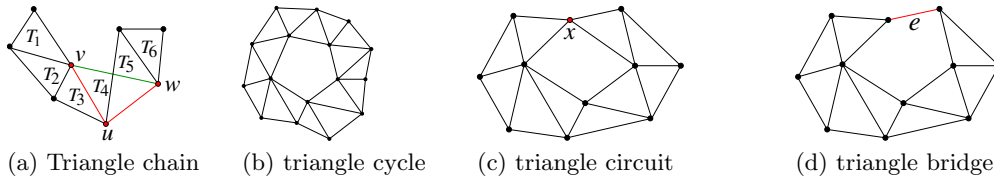

(a) Triangle chain   (b) triangle cycle   (c) triangle circuit   (d) triangle bridge

Fig. 2. Examples of triangle chain, triangle cycle, triangle circuit and triangle bridge

$\mathcal{T}$. A node u of a triangle T_i is termed a *pendant* of T_i, if the edge opposite to u in T_i is shared by another triangle in $\mathcal{T}$. This shared edge is called an *inner side* of T_i. Each triangle T_i has at least one edge which is not shared by another triangle in $\mathcal{T}$. Such a non-shared edge is

called an *outer side* of T_i. In Fig. 2 (a), $T_4 = \{u, v, w\}$ has two pendants v and w. It has two inner sides uw and uv and one outer side vw. If each of T_1 and T_m has unique and distinct pendants, then $G(\mathcal{T})$ is termed a *triangle chain*. Fig. 2 (a) shows an example of triangle chain. By construction, a triangle chain involves only flips; hence rigid. If T_1 and T_m share a common edge other than those shared with T_2 and T_{m-1}, then the union $G(\mathcal{T})$ is called a *triangle cycle*. In a triangle cycle, each triangle has exactly two inner and one outer sides. Fig. 2 (b) shows an example of a triangle cycle. Every wheel graph is a triangle cycle.

If $G(\mathcal{T})$ is not a triangle cycle and T_1 and T_m have a unique pendant in common, then $G(\mathcal{T})$ is called a *triangle circuit* (Fig. 2 (c)). The common pendant is called a *circuit knot*. x is the circuit knot of the triangle circuit. Let $\mathcal{T} = (T_1, T_2, \ldots, T_m)$ be a triangle stream corresponding to a triangle chain. T_1 and T_m have unique and distinct pendants. We connect these pendants by an edge e. $G(\mathcal{T}) \cup \{e\}$ is called a *triangle bridge* (Fig. 2 (d)). The edge e is called the *bridging edge*. The *length of a triangle stream* $\mathcal{T}$ is the number of triangles in it and is denoted by $l(\mathcal{T})$.

Lemma 2. *1) Every triangle cycle has a spanning wheel or triangle circuit (a wheel or triangle circuit which is a spanning subgraph of the triangle cycle). 2) Every triangle circuit has a spanning triangle bridge (a triangle bridge which is a spanning subgraph of the triangle circuit).*

Proof. See Appendix A.1 for the first part. For the second part, see Appendix A.2. □

We have seen that a rigid realization in $\mathbb{R}^2$ may have *flip ambiguity*, i.e., it may yield another configuration by applying flip operation only. In $\mathbb{R}^2$, if a rigid realization admits no flip ambiguity, then it is globally rigid. Using this, we shall prove the generically global rigidity as follows.

Lemma 3. *Triangle cycle, circuit and bridge are generically globally rigid.*

Proof. A triangle cycle has a spanning wheel or triangle circuit (Lemma 2). A wheel graph is generically globally rigid. A triangle circuit always has a spanning triangle bridge (Lemma 2). If we can prove that a triangle bridge is generically globally rigid, the result will follow.

Let $G(\mathcal{T})$ be a triangle bridge with the triangle stream $\mathcal{T} = (T_1, T_2, \ldots, T_n)$ and the bridging edge e. Consider a generic configuration of $G(\mathcal{T})$ (Fig. 3). Note that $G(\mathcal{T})$ contains $n + 2$ nodes. $G(\mathcal{T}) - e$ is a *spanning triangle chain* (a triangle chain which is a spanning
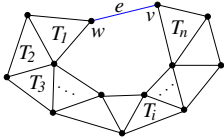

Fig. 3. A generic configuration of a triangle bridge $G(\mathcal{T})$

subgraph of the triangle bridge) of $G(\mathcal{T})$. Since $G(\mathcal{T}) - e$ can only have flips, i.e., no smooth deformation. Therefore, $G(\mathcal{T})$ also admits no smooth deformation.

Consider a different realization of $G(\mathcal{T})$ obtained from the current realization through a sequence of flip operations. Without loss of generality, we assume that T_1 remains fixed with positions (x_0, y_0), (x_1, y_1) and (x_2, y_2) for w and two other nodes in T_1 respectively. If T_2 is involved in a flip, then the only possibility is the flip that is taken with respect to the inner edge with T_1. Only one point of T_2 changes its position. Let (x_3, y_3) and (x_3', y_3') be the positions of this point in the original and the modified configuration respectively. From elementary coordinate geometry, x_3' and y_3' can be expressed in the form of $\frac{\phi_3}{\psi_3}$ and $\frac{\xi_3}{\eta_3}$ where ϕ_3, ψ_3, ξ_3 and η_3 are non-zero polynomials of x_1, y_1, x_2, y_2, x_3 and y_3 with integer coefficients such that $\psi_3 \neq 0$ and $\eta_3 \neq 0$. Once, the positions of T_1 and T_2 in the second configuration are computed (fixed) then only one point of T_3 may need to be computed. This node again

may be involved in a flip with respect to the inner edge of T_2. If (x_4, y_4) and (x'_4, y'_4) are the positions of this point in the original and modified configurations respectively, x'_4 and y'_4 can be expressed in the form of $\frac{\phi'_4}{\psi'_4}$ and $\frac{\xi'_4}{\eta'_4}$ where ϕ'_4, ψ'_4, ξ'_4 and η'_4 are non-zero polynomials of x_2, y_2, x'_3, y'_3, x_4 and y_4 with integer coefficients such that $\psi'_4 \neq 0$ and $\eta'_4 \neq 0$. In this expression, if we substitute x'_3 and y'_3 by expressions involving x_1, y_1, x_2, y_2, x_3 and y_3 (obtained from the previous equations), x'_4 and y'_4 can be expressed in the form of $\frac{\phi_4}{\psi_4}$ and $\frac{\xi_4}{\eta_4}$ where ϕ_4, ψ_4, ξ_4 and η_4 are non-zero polynomials of x_1, y_1, x_2, y_2, x_3, y_3, x_4 and y_4 with integer coefficients such that $\psi_4 \neq 0$ and $\eta_4 \neq 0$. Proceeding in this way, finally (x'_{n+1}, y'_{n+1}), the position of v, can be expressed in the form of $\frac{\phi}{\psi}$ and $\frac{\xi}{\eta}$ where ϕ, ψ, ξ and η are non-zero polynomials of x_is and y_is, $1 \leq i \leq n+1$, the coordinates of the nodes in the first configuration, with integer coefficients such that $\psi \neq 0$ and $\eta \neq 0$.

Since v and w are adjacent, $d(w, v)$ (the Euclidean distance between w and v) remains preserved in both configurations. In terms of the coordinates,
$$(x'_{n+1} - x_0)^2 + (y'_{n+1} - y_0)^2 = (x_{n+1} - x_0)^2 + (y_{n+1} - y_0)^2,$$
$$\eta^2(\phi - x_0\psi)^2 + \psi^2(\xi - y_0\eta)^2 = \eta^2\psi^2(x_{n+1} - x_0)^2 + \eta^2\psi^2(y_{n+1} - y_0)^2.$$
So the coordinates in the original configuration are algebraically dependent. It contradicts that the configuration is generic. Therefore, no flip is possible. $\qquad\square$

3.2 Triangle notch and triangle net

Consider a sequence $\mathcal{T} = (T_1, T_2, \ldots, T_m)$ of triangles. Suppose, for $i = 2, 3, \cdots, m$, each T_i shares exactly one edge with exactly one T_j, $1 \leq j < i$. The node opposite to this sharing edge is called a *pendant* of T_i in $\mathcal{T}$. Fig. 4 shows an example of such a sequence and x is a pendant of T_2. T_1 has no pendant. For $2 \leq i \leq m$, each T_i has exactly one pendant in $\mathcal{T}$. The graph
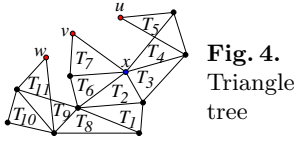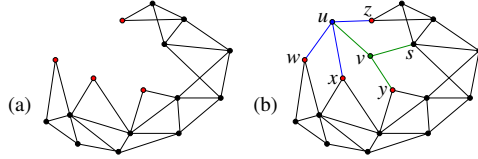

Fig. 4. Triangle tree

(a)

(b)

Fig. 5. (a) Triangle tree (b) u and v are Extended nodes

$G(\mathcal{T})$ corresponding to such a sequence $\mathcal{T}$, is called a *triangle tree*. Fig. 4 is an example of a triangle tree with 11 triangles. $G(\mathcal{T})$ contains no triangle cycle. Otherwise, there always exists a T_j which shares two edges with some triangles before T_j in $\mathcal{T}$. If a triangle T_i shares no edge with T_j, $j > i$, is called a *leaf triangle*. A leaf triangle shares exactly one edge with other triangles in $\mathcal{T}$. It has a unique pendant, called a *leaf knot*. T_5, T_7 and T_{11} are leaf triangles and u, v and w are leaf knots. By construction, any realization of a triangle tree is rigid.

Definition 1. *Let $G(\mathcal{T})$ be a triangle tree. A node v, outside $G(\mathcal{T})$, is called an extended node of $G(\mathcal{T})$, if v is adjacent to at least three nodes, each being i) a pendant in $G(\mathcal{T})$; or ii) an extended node of $G(\mathcal{T})$. Each of the edges which connect the extended node to a pendant or an extended knot of $G(\mathcal{T})$ is called an extending edge.*

Fig. 5 (a) is a triangle tree, say $G(\mathcal{T})$. Fig. 5 (b) consists of a replica of the graph in Fig. 5 (a) and some more nodes and edges. Fig. 5 (a) does not contain u of Fig. 5 (b). u is adjacent to three pendants w, x and z. So u is an extended node of $G(\mathcal{T})$. The edges uw, ux and uz are the extending edges of u. Similarly, v is adjacent to an extended node u and two pendants s and y. So v is also an extended node of $G(\mathcal{T})$; where vu, vs and vy are the extending edges.

Definition 2. *A graph G is called a triangle notch, if it can be generated from a triangle tree $G'(\mathcal{T})$, where G' is proper subgraph of G, by adding only one extended node v where all the leaf knots of $G'(\mathcal{T})$ are adjacent to v. The extended node v is called the apex of G.*

Fig. 6 (b) shows an example of a triangle notch with the apex v. The triangle tree from which it is generated is separately shown in Fig. 6 (a).
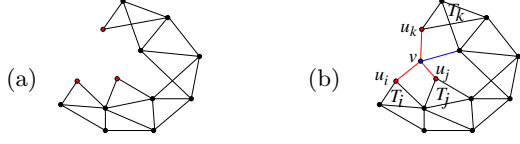

Fig. 6. (a) Triangle tree $G(\mathcal{T})$ (b) Triangle notch with apex v

Lemma 4. *A triangle notch is generically globally rigid.*

Proof. See Appendix A.3. □

Lemma 5. *Let G be a graph obtained from a triangle tree $G'(\mathcal{T})$ by adding extended nodes, where G' is a proper subgraph of G. Any extended node along with all pendants and extended nodes adjacent to it lie in a generically globally rigid subgraph.*

Proof. If the extended node v is adjacent to only pendants of $G'(\mathcal{T})$, then these pendants are leaf knots of some triangle tree $G''(\mathcal{T}')$ where the triangles of $\mathcal{T}'$ are all taken from $\mathcal{T}$. $G''(\mathcal{T}') \cup \{v\}$ forms a triangle notch. Fig. 7 (a) shows an example of such a case. By Lemma 4,
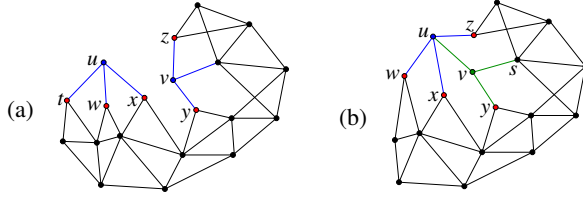

Fig. 7. u and v are extended nodes where (a) u, v are adjacent to pendants only, (b) u, v are adjacent to both pendant and extended nodes

$G''(\mathcal{T}') \cup \{v\}$ is generically globally rigid. Now consider the case when v is adjacent to at least one extended node. Let u be an extended node which is adjacent to v (Fig. 7 (b)). Since u is also an extended node of $G'(\mathcal{T})$, we assume that u lies in a generically globally rigid subgraph G_1 of G and is generated from a triangle tree $G''(\mathcal{T}')$ by adding extended nodes (including u), where $\mathcal{T}'$ contains triangle only from $\mathcal{T}$. Consider a generic configuration $\mathcal{P}$ of G_1. If G_1 admits any flip operation in $\mathcal{P}$ to yield a different configuration $\mathcal{P}'$, then proceeding in a manner similar to that in the proof of Lemma 4, we can show that at least three nodes (pendants or extended nodes adjacent to v) are algebraically dependent. □

Definition 3. *A graph G is called a triangle net, if it may be generated from a triangle tree $G'(\mathcal{T})$ by adding one or more extended nodes and satisfying the following conditions:*

1. *G contains no triangle cycle, triangle circuit or triangle bridge; and*
2. *there exists an extended node u such that every leaf knot of $G'(\mathcal{T})$ is connected to u by a path (extending path) containing only extending edges.*

The last extended node added to generate the triangle net is called an apex of the triangle net.
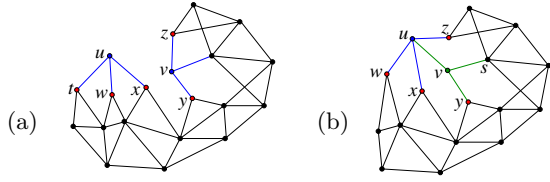
Fig. 8. (a) Not a triangle net
(b) A triangle net

In Fig. 8 (a), u and v are two extended nodes. The leaf knots t, w and x are connected to u by extending paths. Other leaf knots y and z are connected to v by extending paths. No extending path exists between the y and u, and x and v. So the graph shown in Fig. 8 (a) is not a triangle net. Fig. 8 (b) contains two extended nodes u and v. All the leaf knots w, x, y and z are connected to u by extending paths. Thus Fig. 8 (b) is an example of a triangle net. The graph shown in Fig. 8 (b), is generated from a triangle tree by adding extended nodes u and then v. So v is an apex of G. Triangle notch is a special case of triangle net.

Lemma 6. *A triangle net is generically globally rigid.*
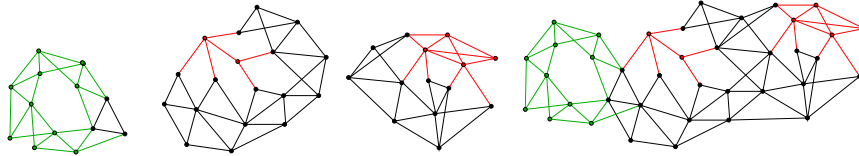
Proof. See Appendix A.4. □

3.3 Triangle bar

A graph G is called a *triangle bar*, if it satisfies one of the followings:

1. G can be obtained from a triangle cycle, triangle circuit, triangle bridge or triangle net by adding zero or more edges, but no extra node;
2. $G = B_i \cup B_j$ where B_i and B_j are triangle bars which share at least three nodes; or
3. $G = B_i \cup \{v\}$ where B_i is a triangle bar and v is a node not in B_i, and adjacent to at least three nodes of B_i.

Note that triangle cycle, triangle circuit, triangle bridge and triangle net are also triangle bars. These triangle bars will be referred as *elementary bars*.

Fig. 9 shows some examples of triangle bars. The first figure is a triangle cycle. Next two are triangle nets. The fourth figure shows an example of a triangle bar which is obtained by

Fig. 9.
Examples
of triangle bar

stitching the first three elementary bars through common triangles.

Theorem 1. *Triangle bar is generically globally rigid.*

Proof. From Lemma 3 and 6, elementary bars are generically globally rigid. Suppose two triangle bars B_i and B_j share three nodes. Since all the nodes are in generic position, these three nodes are non-collinear. Using Lemma 1, $B_i \cup B_j$ is generically globally rigid.

Let a triangle bar B be obtained from another triangle bar B' by adding a node v which is adjacent to three nodes in B'. In a generic realization of B', any node placed with three given distances from known positions has a unique location. So B is generically globally rigid. □

Theorem 2. *Trilateration graph and wheel extension are triangle bars.*

Proof. See Appendix 2. □

4 Problem statement

A triangle bar is a class of graphs which includes trilateration and wheel extension graphs as special cases. Starting from a triangle of three reference nodes, we find a maximal triangle bar. Let $G(\mathcal{T})$ be a triangle tree where $\mathcal{T} = (T_1, T_2, \ldots, T_n)$. Three nodes in T_1 are chosen as the reference nodes. This triangle is called *seed triangle*. Our goal is to identify a maximal triangle bar containing T_1 and then mark the nodes in this triangle bar as localizable.

Problem 1. Consider a distance graph $G = (V, E, d)$, generically embedded in plane with a seed triangle T_1. Find a maximal triangle bar containing T_1 in a distributed environment and mark the nodes of the triangle bar as localizable.

We solve the problem involving only connectivity information. No distance information is used. Here onwards, we ignore the distance function d and consider the graph $G = (V, E)$. The stated problem is solved by exploiting flips of triangles in G. In order to solve the problem, we introduce the notion of *flip-triangle graph* of G.

4.1 Flip-triangle graph

Given a graph $G = (V, E)$, we construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{t_1, t_2, \ldots, t_N\}$ where t_i represents a triangle T_i in G and $\{t_i, t_j\} \in \mathcal{E}$ if and only if T_i and T_j share an edge in G. The graph $\mathcal{G}$ is termed as flip-triangle graph of G, in short $FTG(G)$. If no ambiguity occurs, we use t_i to denote a vertex of $FTG(G)$ and T_i to refer the corresponding triangle in G. A maximal tree in $FTG(G)$ is called a *flip-triangle tree* (FTT). A connected $FTG(G)$ has unique FTT. Let $\mathcal{T} = (T_1, T_2, \ldots, T_m)$ be a sequence of triangles in G and $\tau = (t_1, t_2, \ldots, t_m)$ be the corresponding sequence of nodes in $\mathcal{G}$. If no ambiguity occurs, $\mathcal{T}$ also means the subgraph obtained from the union of T_is in $\mathcal{T}$. Similarly, τ means corresponding subgraph in $\mathcal{G}$. We describe some properties which are useful for developing the proposed algorithm.

Proposition 1. $\mathcal{T}$ *is a triangle cycle of length n in G if and only if τ is an n-cycle in $\mathcal{G}$.*

Proof. Let $\mathcal{T}$ be a triangle cycle in G. By construction, pairs of nodes t_i and t_{i+1} for $1 \leq i \leq n-1$, and t_m and t_1 are adjacent in $\mathcal{G}$. For some i, j ($|i - j| > 1$) (except t_1 and t_m), if t_i and t_j are adjacent in $\mathcal{G}$ then T_i and T_j share an edge. This contradicts that $\mathcal{T}$ is a triangle cycle in G. Therefore, τ is an n-cycle in $\mathcal{G}$.

Conversely, let τ be an n-cycle in $\mathcal{G}$. For some i, j ($|i - j| > 1$) (except T_1 and T_m), if T_i and T_j share an edge in G then t_i and t_j are adjacent in $\mathcal{G}$. It contradicts that τ is an n-cycle in $\mathcal{G}$. Therefore, $\mathcal{T}$ is a triangle cycle in G of length n. $\square$

Proposition 2. $\mathcal{T}$ *is a triangle tree in G if and only if τ is a tree in $\mathcal{G}$.*

Proof. Let $\mathcal{T}$ be a triangle tree in G. By construction, τ is connected subgraph in $\mathcal{G}$. In view of Proposition 1, τ contains a cycle in $\mathcal{G}$ if and only if $\mathcal{T}$ contains a triangle tree in G. Hence the result follows. $\square$

Proposition 3. $\mathcal{T}$ *is a maximal triangle tree in G if and only if τ is an FTT in $FTG(G)$.*

Proof. Let $\mathcal{T}$ be a maximal triangle tree in G. τ is a tree in $\mathcal{G}$ (Proposition 2). If τ is not an FTT in $FTG(G)$, there exists a tree τ' containing τ as a proper subtree in $\mathcal{G}$. The triangle tree $\mathcal{T}'$ corresponding to τ' also contains $\mathcal{T}$ as a proper subgraph in G. This is contradicts that $\mathcal{T}$ is a maximal triangle tree in G.

Conversely, let τ is an FTT. If $\mathcal{T}$ is not maximal triangle tree in G, there is a $\mathcal{T}'$ containing $\mathcal{T}$ as a proper subgraph. $\mathcal{T}'$ corresponds a tree τ' in $\mathcal{G}$ (Proposition 2) while τ' contains τ as a proper subtree. This is a contradiction. Hence the result follows. $\square$

4.2 Solution plan

Consider a graph $G = (V, E)$. A triangle bar may be identified in G by three rules as in its definition. First, we find elementary bars in G. If possible, then we stitch them via three common nodes to form a larger triangle bar; or extend a triangle bar $\mathcal{B}$ successively by adding a new node which is adjacent to at least three nodes of $\mathcal{B}$. After computing the $FTG(G)$, the stated problem is solved in a distributed set up as follows:

1. We identify all the components of $\mathcal{G}$. For each component $\mathcal{G}'$ in $\mathcal{G}$, we compute a corresponding spanning tree $FTT(\mathcal{G}')$ which is a maximal subtree in $\mathcal{G}$.
2. Finding triangle cycles is equivalent to finding the cycles in $\mathcal{G} = FTG(G)$ (Proposition 1). We identify a *set of base cycles* (a minimal set of cycles such that any cycle of the graph may be obtained by union of some base cycles and deleting some parts).
3. A triangle chain is also a triangle tree. The generator chains of triangle circuits and bridges and generator trees of triangle nets are uniquely identified by subtrees in $FTG(G)$ (Proposition 2). We identify other elementary bars in G from the FTTs by suitable extensions.
4. Finally, we stitch or extend these elementary bars to form a maximal triangle bar in G containing T_1; then we mark the nodes in this triangle bar as localizable.

5 Localizability testing

This section describes a distributed technique to find the maximal triangle bar with a seed triangle T_1 in three phases. This triangle bar is reported as the localizable subgraph of G.

5.1 Representation of graph and flip-triangle graph

Each node contains data structures suitable for describing and storing necessary information for the execution of the algorithm. We assume that each node contains a unique number as its identification (called *node-id*) and a list (`nbrs`) of node-ids of its neighbours. The node contains no edge distance information. To represent a triangle in computer, we define a data structure, with type name **Trngl**, containing: 1) node-ids of the nodes of the triangle; and 2) a list of adjacent nodes in $FTG(G)$ (i.e., triangles sharing its edge in G). In a distributed environment, the node with minimum node-id among three nodes of a triangle is designated as the *leader*. The leader contains all the information of the triangle and processes them. Each node v additionally contains a list (`trngls`) of all the triangles containing v as the leader.

5.2 Communication protocols for G and $FTG(G)$

A communication between two adjacent nodes in $FTG(G)$ (i.e., two triangles sharing an edge in G) means communication between their leaders which may involve at most 2-hop communication in G. Intermediate communications via other nodes uses standard communication tools for G (i.e., communication within G). By a communication between a node $s_i \in G$ and a node $t_j \in FTG(G)$ (i.e., triangle T_j), we mean the communication between s_i and the leader of T_j. The nodes of G and $FTG(G)$ use different types of signals to indicate the types of the contents of the messages. We list these signals as follows:

Signal types	Significance of the symbol
visit	On arrival of this signal a node of $FTG(G)$ wake up and starts processing.
visitNode	On arrival of this signal a node of G wake up and starts processing.
child	A node in G sends a `child` signal to its parent to register itself as a child in parent.
cycle	A node in G or $FTG(G)$ sends a `cycle` signal on identification of an elementary bar.

5.3 Phase I: Computing the *FTG*

Phase-I of the algorithm sets up the basic structure of the $FTG(G)$ using the procedures RECVNBRLIST( ) and RECVTRIANGLE( ) described with pseudo-codes as follows.

1: **procedure** RECVNBRLIST( )   /* s_i = current node */
2:     **Wake** on arrival of neighbours (nbrs$_j$) form s_j
3:     **for** (each common s_k in nbrs$_j$ and nbrs$_i$) **do**   /* A triangle $T(\triangle s_i s_j s_k)$ is identified. */
4:         **if** (s_i is the leader of T) **then**   /* T corresponds new node in $FTG(G)$ */
5:             **for** (each $T' \in s_i$.trngls sharing an edge with T) **do**   /* Found a new edge TT' */
6:                 **Push** T' into T.trngls and T into T'.trngls as a neighbour of each other in $\mathcal{G}$.
                    /* Both of these push operations occur in s_i, since s_i is the leader T and T'. */
7:             **end for**
8:             **Store** the new triangle T into s_i.trngls in its leader.
9:             **Send** the triangle T to s_j and s_k to find and set edges with other triangles.
10:         **end if**
11:     **end for**
12: **end procedure**

1: **procedure** RECVTRIANGLE( )   /* s_i = current node */
2:     **Wake** on arrival of a triangle $T(\triangle s_j s_k s_l)$ such that $j < k < l$
3:     **for** (each $T' \in s_i$.trngls sharing an edge with T) **do**   /* Found a new edge TT' for $\mathcal{G}$ */
4:         **Push** T into T'.trngls as a neighbour of T' in $FTG(G)$.   /* s_i is the leader of T' */
5:     **end for**
6:     **if** ($i \in \{j, k, l\}$, say $i = l$)   /* Note that beyond 1-hop, $i \notin \{j, k, l\}$ */
7:         **Send** the triangle T to all neighbours of the current node except s_j and s_k
8: **end procedure**

Proposition 4. *1. All the processes in Phase I are synchronized.*
2. The algorithm guarantees the progress and finite termination of Phase I.
3. Number of communications from each node is thrice the number of neighbours.
4. Phase I of the algorithm computes the $FTG(G)$.

Proof. On arrival of a triangle message T, RECVTRIANGLE( ) wakes up. If the node is the leader of T, it stores the triangle information. It also checks, if T shares an edge with the existing triangles in the list trngls. Note that each triangle in the list trngls share an edge with T. We call these triangles as *neighbour triangles* of T.

   Synchonization: Each node starts by sending its neighbour list (nbrs) to every neighbour. When a node s_i receives the neighbour list from a node s_j, it identifies all triangles which contain s_i and s_j as two nodes. Each of two processes in Phase I is atomic. Any order of insertions of triangles into the neighbour triangle list will finally give the same result.

   Progress and finite termination: Every node executes RECVNBRLIST( ) exactly one for a neighbour list each neighbour. Each **For** loop runs over neighbour lists which are finite in size. Since the processes are atomic and loops runs on finite neighbour list, progress is guaranteed. In RECVTRIANGLE( ), **If** block conditions are false beyond 1-hop from s_i and stops resending T. Assumed channels are reliable, every message reaches its destination in finite time. We use Lamport's logical clock. In each node, the value of logical clock does not exceed thrice the number of neighbours; one neighbour list and two triangle messages from each neighbour. It also follows the number of communications from a node.

   Computation of $FTG(G)$: Each node receives a neighbour list of a neighbour exactly once. Consider an arbitrary triangle $T(\triangle s_i s_j s_k)$ in G (Fig. 10) while $i < j < k$. Let T is received by s_i, the leader node of T. RECVNBRLIST( ) inserts T into s_i.trngls. No other node incorporates

T, though s_j and s_k also identify T. It may be adjacent to the edges in $FTG(G)$ due to sharing its edges with other triangles. The inner **for** loop in RECVNBRLIST( ) sets up the edges which
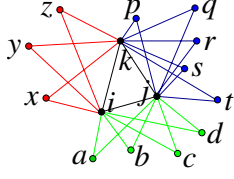

Fig. 10. Examples showing all possible triangle communications

are obtained while T shares an edge with the triangles whose leader is s_i. s_i also sends a triangle message T to s_j and s_k to find and set other edges with T in $FTG(G)$. T shares no edge in G beyond 1-hop. The edges in $FTG(G)$ between T and other triangles with leaders other than s_i are set by RECVTRIANGLE( ). When s_j (or s_k) receives T, it checks whether T shares any triangle in its list **trngls** sends to sends T to its neighbours other than s_i and s_j. Thus T reaches all the nodes which contains a triangle that may share an edge with T.   □

5.4   Phase II: Finding *FTT*s and elementary bars

For finding elementary bars, we may assume that G is connected; otherwise, we proceed with the component containing the seed triangle T_1. Phase-II identifies the FTT of $\mathcal{G}$ containing t_1(i.e., T_1 in G); then elementary bars. This process is triggered by t_1 in $\mathcal{G}$ sending a **visit** signal to its adjacent nodes in $\mathcal{G}$.

Representation of *FTT*s of *G* and elementary bars: For this purpose, each node in $\mathcal{G}$ (a triangle in G) contains additional five fields:

1. **status** (assumes **0** or **visited**, initially **0**) is used to indicate the status regarding the processing of the node. After the required processing of a node, **status** is set to **visited**.
2. **elementLst** is a list of elementary bars in which this particular node (triangle) is a constituent part; initially it is empty.
3. **parent** contains the immediate ancestor of the triangle in a FTT of G. The seed triangle has no parent; and it is treated as root node.
4. **children** is a list containing direct descendants in $\mathcal{G}$. These are used to set the trees in $\mathcal{G}$.
5. **hearLst** holds the list of received triangles with some extra information that help in identifying elementary bars.

Each node in G also contains similar five fields to store the information regarding pendants, extended knots and elementary bars.

Finding the *FTT* and base cycles in $\mathcal{G}'$: The FTT of $\mathcal{G}$ containing T_1 are identified by distributed BFS on $\mathcal{G}$. If we take any FTT and add to it a new edge from $\mathcal{E}$ a set of base cycles of $\mathcal{G}$ may be obtained. The details of these steps are described in procedure VISITTRIANGLE( ).

   On arrival of **visit** signal into $t_i \in \mathcal{V}$, VISITTRIANGLE( ) in leader node of T_i sets its status to **visited**. t_i.**parent** is assigned the value t_j. This helps in backtracking in the tree in $FTG(G)$. After visiting t_i, the process sends **visit** signal to all neighbours in $FTG(G)$. At the same time, t_i sends a **visitNode** signal with T_i to the pendant s_k of T_i with respect to T_j in G for finding elementary bars other than triangle cycles. It sends back a **child** signal to the sender t_j to inform itself as a child. The **child** signal is handled (handler is not described separately) by inserting its sender into the list **children**. If t_i is already visited by some other

```
1: procedure VISITTRIANGLE( )          /* t_i=
   current node of G' */
2:    Wake on a visit signal from t_j ∈ V
3:    if (status ≠ visited) then
4:        Set parent ← t_j, status ← visited
5:        Send a visit signal to all adjacent
              nodes in G' other than t_j
6:        Send a child signal to t_j to indicate
              that s_i is a child of s_j
7:        Send a visitNode signal to s_k ∈
              T_i − T_j with T_i qualified as
              triangle
8:    else    /* a base cycle is identified by t_i t_j
      */
9:        Push T_i T_j into elementLst to indi-
              cate that t_i lies in the cycle
              T_i T_j
10:       Send a cycle signal to t_j and
              parent with T_i T_j as elemen-
              tary bar-id
11:   end if
12: end procedure
```

```
1: procedure ELEMENTARYBARS(x_i)    /* x_i is
   the current (t_i ∈ V or s_i ∈ V) */
2:    Wake on arrival of a cycle signal with
              elementary bar identity as bar-id
3:    if (bar-id is qualified as delete) then
4:        Delete bar-id from elementLst
5:    else if (bar-id ∉ elementLst) then
6:        Push bar-id into elementLst
7:        Send a cycle signal to parent with
      bar-id
8:    else /* Elementary bars in G' are traced
      */
9:        Send a cycle signal with bar-id
              qualified as delete to its
              parent
10:   end if
11: end procedure
```

```
1: procedure VISITNODE( )    /* s_i=current node
   in G */
2:    Wake on a visitNode signal with X_j (= T_j
      or s_j)
3:    Push X_j into hearLst
4:    if (status ≠ visited) then
5:        Set parent ← X_j,  status ← visited
6:        if (X_j = T_j)              /* s_i ∈ T_j, let
      T_j = △s_i s_k s_m */
7:            Send a visitNode signal with s_i to
                  all s_m ∈ nbrs − {s_k, s_l}
8:        Send a child signal to parent to inform
              that s_i is a child
9:    else if (X_j = T_j) then    /* if parent = T_k
      */
10:       Send a cycle signal to t_j and t_k with
              T_j s_i T_k as elementary bar-id
11:   else if (X_j = s_j) then
12:       if (parent = T_k) then    /* a bridge or
      net */
13:           Send a cycle signal to s_j and
                  t_k with s_j s_i T_k as elementary
                  bar-id
14:       else if (parent = s_k) then
15:           if (mark = extended) then
16:               Send
      a visitNode signal with
      s_i to all s_m ∈ nbrs − {s_k}
17:               Send a cycle signal to s_j and s_k
                      with s_j s_i s_k as elementary
                      bar-id
18:           else if (size(hearLst) = 3) then
19:               Set mark ← extended
20:               for all (s_j ∈ hearLst) do
21:                   Send a cycle signal to s_k
                          and s_j with s_j s_i s_k as
                          bar-id
22:               end for
23:           end if
24:       end if
25:   end if
26: end procedure
```

node t_k, a base cycle in $FTG(G)$ is identified with $t_i t_j$ and sends a **cycle** signal with $t_i t_j$ to t_j and t_k. A cycle in $FTG(G)$ corresponds a triangle cycle in G (Proposition 1). A triangle cycle G, corresponding to a base cycle in $FTG(G)$, contains at least one new triangle which is not a part of any other triangle cycle in G. The outcome of the procedure may be summarized below in an proposition.

Proposition 5. VISITTRIANGLE( ) *identifies the FTT and triangle cycles of G containing T_1.*

Finding triangle circuits and triangle bridges in $\mathcal{G}'$: Triangle cycles are identified by handling **cycle** signals. In view of the Proposition 3, a FTT generate maximal triangle tree in G. These maximal trees provide maximal triangle bars including appropriate edges and extended knots. The steps are described in the procedure VISITNODE( ).

On arrival of a **visitNode** signal with X_j (either a triangle T_j or a node s_j), a node $s_i \in V$ wakes up and VISITNODE( ) stores X_js into a list `hearLst`. If s_i is being visited for the first time as a node in G, s_i set its `status` as **visited** and `parent` to X_j for backtracking. If $X_j = T_j$ and s_i is a pendant of T_j, s_i sends a **visitNode** signal with s_i to all neighbours other those in T_j. Otherwise, if s_i 1) receives a triangle T_j and its parent is a triangle T_k, a triangle circuit is identified; 2) receives a node s_j and its parent is a triangle T_k (either a pendant or extended knot), a triangle bridge or net is identified; and 3) if three **visitNode** signals, it identifies itself as an extended knot and as well as a triangle net. If s_i marks himself as an extended knot it sends **visitNode** signal with s_i to all neighbours except its parent for identifying other extended knots and nets. Note that **visitNode** signal with triangle is sent only to a pendant from the process VISITTRIANGLE( ). The final outcome of this process is described below.

Proposition 6. *All triangle circuits and triangle bridges in $\mathcal{G}'$ are identified by* VISITNODE*( ).*

Identifying triangle nets in $\mathcal{G}'$: On arrival of a **cycle** signal into x_i (either a node in $FTG(G)$ or pendant or extended knots in G), ELEMENTARYBARS( ) inserts elementary bar-id into the `elementLst` and in turn sends the same **cycle** signal to its parent until it finds a node in $FTG(G)$ containing same elementary bar-id in respective `elementLst`. If a matching bar-id is found, it sends **cycle** signal with this bar-id qualified as **delete**. If it finds bar-id qualified as **delete** and bar-id is deleted from `elementLst` and sends **delete** signal same bar-id qualified as **delete** to its parent until the root node t_1 in $FTG(G)$ is reached.

5.5 Phase III: Identifying the maximal triangle bar containing T_1

VISITNODE( ) and ELEMENTARYBARS( ) provide a maximal triangle bar for the FTT. Thus, we have obtained a set $\mathcal{M}$ of maximal triangle bars for the FTT containing T_1. Maximal triangle bars for a FTT do not share any triangle in this FTT. From $\mathcal{M}$, two maximal triangle bars M_1 and M_2 which have three nodes in common are replaced by $M_1 \cup M_2$. Repeat these until no replacement. This task may be achieved by sending a special signal from all nodes of M_1 (assuming that M_1 contains T_1) to their neighbours. Another M_i sends this special signal if it hears this signal from three nodes. Finally, a maximal triangle bar in G will be identified.

Proposition 7. *The maximal triangle bar of G containing T_1 is identified in polynomial time with $O(|E|)$ one-hop communications over the network.*

Mark localizable nodes: At the end of Phase-II when t_1.`elementLst` is empty, t_1 set its `status` as **localizable**. The triangle T_1 (i.e., t_1) triggers the Phase-III by sending the `elementLst` to all its adjacent nodes (`children`) in $\mathcal{G}$. On arrival of an elementary bar list into x_i ($t_i \in FTG(G)$ or $s_i \in G$), MARKLOCALIZABLE( ) starts execution. If the current node, x_i, lies in an elementary

---

1: **procedure** MARKLOCALIZABLE(x_i)   /* x_i is the current ($t_i \in \mathcal{V}$ or $s_i \in V$) */
2:     **Wake** on arrival of an elementary bar list, say $barLst$
3:     **if** ($barLst \cap$ `elementLst` $\neq \emptyset$ and `status` $\neq$ **localizable**) **then**
4:         Set `status` $\leftarrow$ **localizable**
5:         **Send** a message with `elementLst` to all nodes in x_i.`children`
6:     **end if**
7: **end procedure**

---

bar in the received list and is not marked as localizable, the process marks x_i as localizable and sends x_i.`elementLst` to all nodes in x_i.`children`.

Theorem 3. *If an elementary bar contains T_1, then all the nodes (in G) of these bars are marked as localizable through* MARKLOCALIZABLE*( ) (identifying the bar through T_1) in the complete run of the algorithm.*

Proof. The result follows from the statements in Proposition 1, 2, 5 6. □

6 Performance analysis of the algorithm

Synchronization, Correctness and Progress: Phase I computes the $FTG(G)$. Its synchronization, progress, finite termination and correctness of are proved in Proposition 4. Phase II uses BFS tool to find the trees in a connected $FTG(G)$. This ensures that the FTTs are found correctly (Proposition 3). Theorems 1 and 3 establish the correctness of Phase II and III of the algorithm.

Each of the procedures in the algorithm may contain loop which run over a list either `nbrs` or `trngls`. The sizes of these lists do not exceed the number of neighbours of a node. The loops executes without waiting for any signal. Therefore, the finite termination of each procedure in any individual node is guaranteed. Since the transmission medium is reliable, every message sometime reaches the destination. Thus, executions in the whole system terminate in finite time.

Time complexity: We have used Lamport's logical clock. The maximum value of this logical clock in any node does not exceed thrice the number of its neighbours (Proposition 4). Phase I communicates no message beyond 2-hop. The running time complexity of RECVNBRLIST( ) in each node is $O(n)$ in the worst case. Therefore, the worst case time complexity of Phase I is $O(n^2)$ in total. Time complexity for communication in Phase II and III is guided by the BFS of $FTG(G)$ in distributed way. A visit signal from t_1 will reach any other t_i in $FTG(G)$ along its shortest path in between them. In worst case, it may be equal $|V|$. This dominates time required to find extended knots. Thus, the worst case time complexity is equal to the number of nodes in G, i.e., $O(n)$. Thus the overall time complexity of the execution in the whole system is $O(n^2)$.

Energy complexity: Since message communication dominates the leading consumer of energy, we only count the communications for energy analysis. Every node sends the neighbour list `nbrs` only once. It counts n (number of nodes in the network) transmissions. A node executes RECVNBRLIST( ) once for each of its neighbours. RECVNBRLIST( ) sends a triangle message for a newly obtained triangle whose leader is a different. The total number of these triangle messages is $|E|$ maximum. RECVNBRLIST( ) also sends a **flip** message for a newly obtained flip with a triangle whose leader is a different node. The total number of such **flip** messages does not exceed $|E|$. Thus the number message transmissions in Phase I is $O(|E|)$. It is easy to see that, in Phase II and Phase III, each node communicates with its neighbours constant number of times. Hence, the total energy dissipation is $O(|E|)$ in worst.

7 Conclusion

In this paper, we consider the problem of localizability of nodes as well as networks. We do not compute the positions of nodes. So, exact distances are not necessary and error in distance measurements does not affect the localizability testing. We propose an efficient distributed technique to solve this problem for a specific class of networks, triangle bar. The proposed technique is better than both trilateration and wheel extension techniques. We also illustrate

some network scenarios which are wrongly reported as not localizable by wheel extension technique, but the proposed algorithm recognizes them as localizable in a distributed environment. The proposed algorithm runs with $O(|V|^3)$ time complexity and energy complexity of $O(|E|)$ in the worst case.

In centralized environment localizability testing can be carried out in polynomial time. Though the proposed algorithm recognizes a class of localizable networks distributedly, several localizable networks remains unrecognized by this technique. For localizability testing, we only consider a maximal triangle bar of G which includes the anchor triangle T_1 in $G = (V, E, d)$. Our future plan is to extend localizability testing considering other triangle bars in $FTG(G)$.

References

1. Laman, G.: On graphs and rigidity of plane skeletal structures. Journal of Engineering Mathematics **4**(4) (December 1970)
2. Hendrickson, B.: Conditions for unique graph realizations. SIAM J. Comput **21** (1992) 65–84
3. Čapkun, S., Hamdi, M., Hubaux, J.P.: Gps-free positioning in mobile ad hoc networks. Cluster Computing **5** (2002) 157–167
4. Aspnes, J., Eren, T., Goldenberg, D., Morse, A., Whiteley, W., Yang, Y., Anderson, B., Belhumeur, P.: A theory of network localization. Mobile Computing, IEEE Transactions on **5**(12) (December 2006) 1663–1678
5. Connelly, R.: Generic global rigidity. Discrete and Computational Geometry **33**(4) (2005) 549–563
6. Moore, D., Leonard, J., Rus, D., Teller, S.: Robust distributed network localization with noisy range measurements. In: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, New York, NY, USA, ACM (2004) 50–61
7. Biswas, P., Toh, K.C., Ye, Y.: A distributed sdp approach for large-scale noisy anchor-free graph realization with applications to molecular conformation. SIAM J. Sci. Comput. **30**(3) (2008) 1251–1277
8. Cakiroglu, A., Erten, C.: Fully decentralized and collaborative multilateration primitives for uniquely localizing wsns. EURASIP Journal on Wireless Communications and Networking **2010, Article ID 605658, 7 pages** (2010)
9. Kwon, O.H., Song, H.J., Park, S.: Anchor-free localization though flip error resistant map stitching in wireless sensor network. IEEE Transactions on Parallel and Distributed Systems **99**(PrePrints) (2010)
10. Zhu, Z., So, A.C., Ye, Y.: Universal rigidity: Towards accurate and efcient localization of wireless networks. In: IEEE INFOCOM 2010. (2010)
11. Saxe, J.: Embeddability of weighted graphs in k-space is strongly np-hard. In: Proc. 17th. Allerton Conference in Communications, Control and Computing. (1979) 480489
12. Jackson, B., Jordán, T.: Connected rigidity matroids and unique realizations of graphs. Journal of Combinatorial Theory Series B **94**(1) (2005) 1–29
13. Yang, Z., Liu, Y., Li, X.: Beyond trilateration: on the localizability of wireless ad hoc networks. IEEE/ACM Trans. Netw. **18**(6) (December 2010) 1806–1814
14. Niculescu, D., Nath, B.: Dv based positioning in ad hoc networks. Journal Telecommunication Systems **22** (2003) 267–280
15. Savvides, A., Han, C., Strivastava, M.: Dynamic fine-grained localization in ad-hoc networks of sensors. In: Proc. of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001), Rome, Italy, ACM (July 2001) 166–179
16. Sau, B., Mukhopadhyaya, K.: Length-based anchor-free localization in a fully covered sensor network. In: Proc. of the First international conference on COMmunication Systems And NETworks. COMSNETS'09, Piscataway, NJ, USA, IEEE Press (2009) 137–146

A   Appendix

A.1   Proof of the first part of Lemma 2

Proof. A wheel graph W_n with n nodes is a triangle cycle with $n-1$ triangles. A triangle cycle with three or four triangles is a wheel W_4 or a wheel W_5 (Fig. 11) respectively. These wheels show the existence of spanning wheels for some triangle cycles. Consider a sufficiently large triangle cycle $G(\mathcal{T})$ with the triangle stream $\mathcal{T} = (T_1, T_2, \ldots, T_n)$ which has no spanning wheel. If we consider three consecutive triangles in $\mathcal{T}$, then we have at least one node with degree at least four. Consider such a node v with $deg(v) \geq 4$. If $deg(v) = 4$, then the edges incident on v lie in three consecutive triangles T_i, T_{i+1}, T_{i+2} (Fig. 12 (a)). Deleting the outer side e of
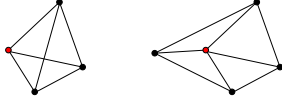

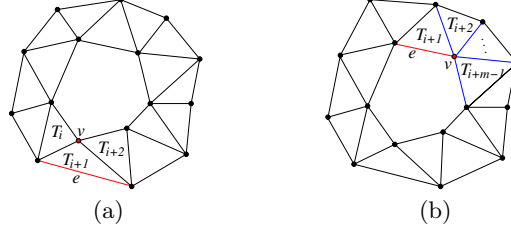**Fig. 11.** Triangle cycle with three and four triangles

Fig. 12. Triangle cycles without spanning wheel

T_{i+1} (i.e., $G(\mathcal{T}) - e$) gives a spanning triangle circuit of $G(\mathcal{T})$. Suppose, $deg(v) = m > 4$. The edges adjacent to v lie in $m - 1$ consecutive triangles T_{i+1}, T_{i+2}, …, T_{i+m-1} (Fig. 12 (b)). Deleting the outer side of T_{i+1} gives a spanning triangle circuit of $G(\mathcal{T})$.   □

A.2   Proof of the second part of Lemma 2

Proof. Let $G(\mathcal{T})$ be a triangle circuit with triangle stream $\mathcal{T} = (T_1, T_2, \ldots, T_n)$ and circuit knot v (Fig. 13). v is the only pendant for both T_1 and T_n. T_2 and T_3 share the edge f and w is the corresponding pendant in T_2. T_1 has two outer sides which are incident on v; one edge
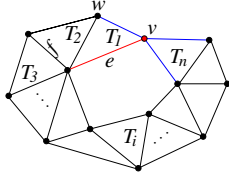

Fig. 13. Triangle circuit $\mathcal{T}$ gives a spanning triangle bridge $\mathcal{T} - e$

is incident on w and the other is incident on f. If e is the outer side of T_1 incident on f, then $G(\mathcal{T}) - e$ gives a spanning triangle bridge of $G(\mathcal{T})$.   □
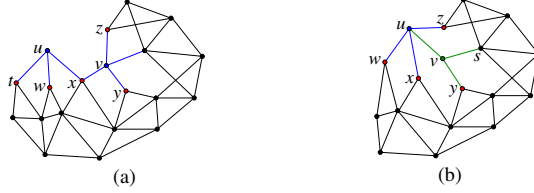
A.3   Proof of Lemma 4

Proof. Let G be a triangle notch generated from a triangle tree $G(\mathcal{T})$ with the apex v. G contains $n + 2$ nodes u_i, $i = 1, 2, \ldots, n + 2$. All the leaf knots of $G(\mathcal{T})$ are adjacent to v. Fig. 6 (b) shows an example of such a graph. The leaf knots u_i, u_j and u_k of $G(\mathcal{T})$ are adjacent to v. Consider a generic configuration $\mathcal{P}$ of G where v is realized as (x, y) and u_i as (x_i, y_i) for $i = 1, 2, \ldots, n + 2$. $G(\mathcal{T})$ can have only flips. If possible, let a flip operation on $\mathcal{P}$ generate a different configuration $\mathcal{P}'$ with coordinates (x', y') for v and (x_i', y_i') for u_i, $i = 1, 2, \ldots, n + 2$. Without loss of generality, we assume that the leaf triangle T_i, with a leaf knot u_i, remains

fixed in both the configurations $\mathcal{P}$ and $\mathcal{P}'$. Consider another leaf triangle T_j with leaf knot u_j. Let $\mathcal{T}_{ij}$ be the unique triangle stream from T_i to T_j in the graph $G(\mathcal{T})$. Proceeding in a manner similar to that in the proof of Lemma 3, each of x'_j and y'_j can be expressed in the form of $\frac{\phi}{\psi}$ where ϕ and ψ are two non-zero polynomials of the coordinates of the points in $\mathcal{T}_{ij} - \{u_i\}$ with integer coefficients such that $\psi \neq 0$. From elementary coordinate geometry in $\mathbb{R}^2$, x' and y' can also be expressed similarly in terms of the coordinates of the points in $\mathcal{T}_{ij}$. Similarly, from the triangle stream $\mathcal{T}_{jk}$, x'_k and y'_k have similar expressions involving the coordinates of the points in $\mathcal{T}_{jk} - \{u_i, v\}$. Since, the edge distance between u_k and v is given, then at least the coordinates of u_i, v and u_k are algebraically dependent. This contradicts the assumption that every three nodes in $\mathcal{P}$ are in general position. So the union of $\mathcal{T}_{ij}$, $\mathcal{T}_{jk}$ and v in $\mathcal{P}$ admits no flip; and the union is generically globally rigid. Since, the leaf triangles chosen are arbitrary and any triangle lies on at least one triangle stream between some pair of leaf triangles, the generic global rigidity of G follows. □

A.4 Proof of Lemma 6

Proof. Consider a triangle net G generated by a triangle tree $G(\mathcal{T})$ where $\mathcal{T} = (T_1, T_2, \ldots, T_n)$. If u_r is an apex of G, then the extended nodes have an ordering u_1, u_2, ..., u_r. Consider a generic configuration of G (e.g. Fig. 14). The extended node added to $G(\mathcal{T})$ is u_1 which is

Fig. 14. Triangle nets with extended nodes u and v where (a) u, v are adjacent to pendants only, (b) u, v are adjacent to both pendant and extended nodes

adjacent to only pendants. These pendants are leaf knots of a triangle tree, which is a subgraph of $G(\mathcal{T})$. This triangle tree is a triangle notch; hence it is generically globally rigid (Lemma 4).

Consider a leaf knot y (e.g., Fig. 14). Let P be an extending path which connects y to the apex u_r. In view of Lemma 5, considering the extended nodes along P and combining the corresponding generically globally rigid graphs, G is generically globally rigid. □

A.5 Proof of Theorem 2

Proof. Let G be a trilateration graph having a trilateration ordering $\pi = (u_1, u_2, \ldots, u_n)$ where u_1, u_2 and u_3 are in K_3. u_4 is adjacent to three nodes before u_4 in π. So u_1, u_2, u_3 and u_4 form a K_4 which is a triangle cycle and consequently a triangle bar. Suppose $\pi' = (u_1, u_2, \ldots, u_i)$, $4 \leq i < n$ forms a triangle bar $\mathcal{B}'$. The node u_{i+1} is adjacent to at least three nodes in $\mathcal{B}'$. Therefore, $\mathcal{B}' \cup \{v_{i+1}\}$ is a triangle bar. By mathematical induction, G is a triangle bar.

Consider a wheel extension graph G with a node ordering $\pi = (u_1, u_2, \ldots, u_n)$. u_1, u_2 and u_3 are in K_3 and u_i ($i \geq 4$) lies in a wheel containing at least three nodes in π before u_i. So, u_4 lies on a wheel, say W_1, which contains u_1, u_2 and u_3. If any, let u_j, $j > 4$, be the first node in π such that u_j does not lie on W_1. u_j lies on another wheel, say W_2, which shares at least three nodes with W_1. Therefore, $W_1 \cup W_2$ is generically globally rigid (by Lemma 1). Similarly, let u_k, if any, be the first node in π such that u_k does not lie on $W_1 \cup W_2$. Assume u_k lies on a wheel W_3 which shares three nodes with $W_1 \cup W_2$. By Lemma 1, $W_1 \cup W_2 \cup W_3$ is generically globally rigid. Proceeding in this way, we can obtain, $\mathcal{W} = (W_1, W_2, \ldots, W_m)$, a finite sequence of wheels such that each W_i shares at least three nodes on some W_js before W_i in $\mathcal{W}$ and $G = \bigcup_{i=1}^{m} W_i$, $m \geq 1$. Wheel graph is triangle cycle. Hence, G is a triangle bar. □