Chapter 4
Range-Based Network localization

4.1 Computation Organization

This section defines the taxonomy of range-based approaches based on their computational organization. Centralized algorithms are designed to run on a central machine with plenty of resources. Network nodes collect physical measurements and deliver back to a base station for analysis. Centralized algorithms resolve the computational limitations of individual nodes. This benefit, however, comes from accepting the communication cost of transmitting data back to the base station. Unfortunately, communication generally consumes more energy than computation in most hardware platforms.

In contrast, distributed algorithms are designed to run in-network, using massive parallelism and internode communication to compensate for the lack of centralized computing power, while at the same time reducing the expensive node-to-sink communication. Often distributed algorithms use a subset of measurement data to locate nodes one by one, yielding an approximation of a corresponding centralized algorithm where all the data are considered and used to compute the positions of all nodes simultaneously. There are two important categories of distributed approaches. The first group, beacon-based distributed algorithms, typically starts a localization process with beacons and the nodes in vicinity of beacons. In general, nodes measure distances to a few beacons and then determine their locations. In some algorithms, the newly localized nodes become beacons to help locating other nodes in the following process. In such an iterative fashion, location information diffuses from beacons to the border of a network, which can be viewed as a top-down manner. The second group of approaches performs in a bottom-up manner, in which localization is originated in a local group of nodes in relative coordinates. After gradually merging such local maps, it finally achieves entire network localization in global coordinates.

Y. Liu and Z. Yang, *Location, Localization, and Localizability: Location-awareness Technology for Wireless Networks*, DOI 10.1007/978-1-4419-7371-9_4,

4.2 Centralized Localization Approaches

4.2.1 Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) [55] is originally developed for mathematical psychology. The intuition behind MDS is simple. Suppose there are n points, suspended in a volume. We do not know the positions of the points, but we know the distance between each pair of points. MDS is an $O(n^3)$ algorithm that uses the law of cosines and linear algebra to reconstruct the relative positions of the points based on the pairwise distances. The algorithm has three stages:

1. Generate an $n \times n$ matrix M, whose (i, j) entry contains the estimated distance between nodes i and j. (Simply run Floyd's all-pairs shortest-path algorithm.)
2. Apply classical metric MDS on M to determine a map that gives the locations of all nodes in relative coordinates.
3. Transform the solution into global coordinates based on a number of anchor nodes.

The goal of metric MDS is to find a configuration of points in a multidimensional space such that the interpoint distances are related to the provided proximities by some transformation (e.g., a linear transformation). The computation of metric MDS is as follows.

Let p_{ij} refer to the proximity measure between objects i and j. The Euclidean distance between two points $X_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ and $X_j = (x_{j1}, x_{j2}, \ldots, x_{jm})$ in an m-dimensional space is given by

$$d_{ij} = \sqrt{\sum_{k=1}^{m} (x_{ik} - x_{jk})^2}.$$

When a geometrical model (the coordinates of points) fits the proximity data M perfectly, the corresponding Euclidean distances are related to the proximities by a transformation $d_{ij} = f(p_{ij})$. In classical metric MDS, a linear transformation model is assumed, i.e., $d_{ij} = a + bp_{ij}$.

The distances D are determined so that they are as close to the proximities P as possible, under a least-squares metric. In this case, define $I(P) = D + E$, where $I(P)$ is a linear transformation of the proximities and E is a matrix of errors (residuals). Since D is a function of the coordinates X, the goal of classical metric MDS is to calculate the X such that the sum of squares of E is minimized, subject to suitable normalization of X. In classical metric MDS, P is shifted to the center and coordinates X can be computed from the double centered P through singular value decomposition (SVD). For an $n \times n$ P matrix for n points and m dimensions of each point, we have

$$-\frac{1}{2}\left(p_{ij}^2-\frac{1}{n}\sum_{j=1}^{n}p_{ij}^2-\frac{1}{n}\sum_{i=1}^{n}p_{ij}^2+\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}p_{ij}^2\right)=\sum_{k=1}^{m}x_{ik}x_{jk}.$$

The double-centered matrix on the left-hand side (call it B) is symmetric and positive semidefinite. Performing singular value decomposition on B gives $B = VAV$. The coordinate matrix becomes $X = VA^{1/2}$.

Retaining the first r largest eigenvalues and eigenvectors ($r < m$) leads to a solution for lower dimensions. This implies that the summation over k in the above equation runs from 1 to r instead of m. It is the best low-rank approximation in the least-squares sense. For example, for a 2D network, we take the first two largest eigenvalues and eigenvectors to construct the best 2D approximation; while the first three largest ones for 3D case.

MDS performs well on RSS data, getting performance on the order of half the radio range when the neighborhood size n_{local} is higher than 12 [56]. The main problem with MDS, however, is its poor asymptotic performance, which is $O(n^3)$ on account of stages 1 and 2.

Besides the computation cost, the classical metric MDS has the other two main drawbacks. First, the computation is inherently centralized, which constrains the scalability of MDS. Second, for irregularly shaped networks, the shortest path distance between two nodes does not correspond well to their Euclidean distance. Consequently, the distance estimation error will introduce huge errors in the localization result. To address the problem, researchers propose a distributed MDS-based algorithm, called MDS-MAP(P) [57]. The main idea of MDS-MAP (P) is to build a local map at each node of the immediate vicinity and then merge these maps together to form a global map. Specifically, MDS-MAP(P) includes the following five steps:

1. Set the range for local maps, R_{lm}. For each node, neighbors within R_{lm} hops are involved in building its local map. The value of R_{lm} affects the amount of computation, as well as the quality. Generally, setting $R_{lm} = 2$ can obtain satisfactory results. The overall complexity of computing each local map is $O(k^3)$, where k is the average number of neighbors. Thus the complexity of computing n local maps is $O(k^3 n)$, where n is the number of nodes.
2. Compute local maps for individual nodes. For each node, do the following:
   (a) Compute shortest paths between all pairs of nodes in its local mapping range R_{lm}. The shortest path distances are used to construct the distance matrix for MDS.
   (b) Apply MDS to the distance matrix and retain the first two (or three) largest eigenvalues and eigenvectors to construct a 2D (or 3D) local map.
   (c) Refine the local map. Using the node coordinates in the MDS solution as the initial point, least-squares minimization is performed to make the distances between nearby nodes match the measured ones.
3. *Merge local maps*. Local maps can be merged sequentially or in parallel. First, randomly pick a node and make its local map the core map. Then, grow the core

map by merging maps of neighboring nodes to the core map. Each time a neighbor's map with the maximal number of common nodes with the core map is selected. Eventually the core map covers the whole network. If the merges are chosen carefully, the complexity of this step is $O(k^3 n)$, where k is the average number of neighbors and n is the number of nodes.

4. *Refine the global map (optional)*. Using the node coordinates in the global map as the initial solution, least-squares minimization is applied to make the distances between neighboring nodes match the measured ones. This step leads to $O(n^3)$ cost.
5. Given sufficient anchor nodes (three or more for 2D networks, four or more for 3D networks), transform the global map to an absolute map based on the absolute positions of anchors. For r anchors, the complexity of this step is $O(r^3 + n)$.

To summarize, MDS-MAP(P) computes small relative maps using local information, instead of a global map using pairwise distances between any two nodes. Comparing with the centralized version, MDS-MAP(P) reduces the computational complexity and can handle the nonconvexity of network deployment, thus the localization accuracy is improved.

4.2.2 Semidefinite Programming (SDP)

Semidefinite programming (SDP) is pioneered by Doherty [39]. In this algorithm, geometric constraints between nodes are represented as linear matrix inequalities (LMIs). Once all the constraints in the network are expressed in this form, the LMIs can be combined to form a single semidefinite program, which is solved to produce a bounding region for each node. The advantage of SDP is its elegance on concise problem formulation, clear model representation, and elegant mathematic solution.

The mathematical expression of SDP is as follows. Suppose there are m anchors $a_k \in R^2$, $k = 1, \ldots, m$, and n unknown nodes $x_j \in R^2$, $j = 1, \ldots, n$. For a pair of two points in N_e, we have a Euclidean distance measure d_{kj} between a_k and x_j or d_{ij} between x_i and x_j; and for a pair of two points in N_l, we have a distance lower bound $\underline{r}_{kj}$ between a_k and x_j or $\underline{r}_{ij}$ between x_i and x_j; and for a pair of two points in N_u, we have a distance upper bound $\bar{r}_{kj}$ between a_k and x_j or $\bar{r}_{ij}$ between x_i and x_j. Then, the localization problem is to find x_js such that

$$\begin{aligned}
&\| x_i - x_j \|^2 = d_{ij}^2, \| a_k - x_j \|^2 = d_{kj}^2, \forall (i,j), (k,j) \in N_e \\
&\| x_i - x_j \|^2 \geq \underline{r}_{ij}^2, \| a_k - x_j \|^2 \geq \underline{r}_{kj}^2, \forall (i,j), (k,j) \in N_l \\
&\| x_i - x_j \|^2 \leq \bar{r}_{ij}^2, \| a_k - x_j \|^2 \leq \bar{r}_{kj}^2, \forall (i,j), (k,j) \in N_u
\end{aligned}$$

Since these measures and bounds are typically noisy, the coordinates x_js are chosen to minimize the sum of errors:

$$\begin{aligned}\min & \sum_{i,j\in N_e, i<j} \left| \| x_i - x_j \|^2 - d_{ij}^2 \right| \\ & + \sum_{k,j\in N_e} \left| \| a_k - x_j \|^2 - d_{kj}^2 \right| \\ & + \sum_{i,j\in N_l, i<j} (\| x_i - x_j \|^2 - \underline{r}_{ij}^2)_- \\ & + \sum_{k,j\in N_l} (\| a_k - x_j \|^2 - \underline{r}_{kj}^2)_- \\ & + \sum_{i,j\in N_u, i<j} (\| x_i - x_j \|^2 - \bar{r}_{ij}^2)_+ \\ & + \sum_{k,j\in N_u} (\| a_k - x_j \|^2 - \bar{r}_{kj}^2)_+,\end{aligned}$$

where $(u)_-$ and $(u)_+$ are defined as $(u)_- = \max\{0,-u\}$ and $(u)_+ = \max\{0, u\}$, respectively.

Let $X = [x_1\ x_2 \cdots x_n]$ be the $2 \times n$ matrix that needs to be determined. Then

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

$$\| a_i - x_j \|^2 = (a_i; e_j)^{\mathrm{T}} [I\ X]^{\mathrm{T}} [I\ X](a_i; e_j,)$$

where e_{ij} is a $n \times 1$ vector with 1 at the ith position, -1 at the jth position, and zeros elsewhere; and e_j is the vector of all zero except -1 at the jth position. Let $Y = X^{\mathrm{T}}X$. By introducing slack variables αs and βs, the softer error minimization problem can be rewritten as

$$\begin{aligned}\min \quad & \sum_{i,j\in N_e, i<j} (\alpha_{ij}^+ + \alpha_{ij}^-) + \sum_{k,j\in N_e} (\alpha_{kj}^+ + \alpha_{kj}^-) \\ & + \sum_{i,j\in N_l, i<j} \beta_{ij}^- + \sum_{k,j\in N_l} \beta_{kj}^- \\ & + \sum_{i,j\in N_u, i<j} \beta_{ij}^+ + \sum_{k,j\in N_u} \beta_{ij}^+ \\ \text{s.t.} \quad & e_{ij}^{\mathrm{T}} Y e_{ij} - d_{ij}^2 = \alpha_{ij}^+ - \alpha_{ij}^-, \forall i,j \in N_e, i < j, \\ & (a_k; e_j)^{\mathrm{T}} \begin{pmatrix} IX \\ X^T Y \end{pmatrix} (a_k; e_j) - d_{kj}^2 = \alpha_{kj}^+ - \alpha_{kj}^-, \forall k,j \in N_e, \\ & e_{ij}^{\mathrm{T}} Y e_{ij} - \underline{r}_{ij}^2 \geq -\beta_{ij}^-, \forall i,j \in N_l, i < j, \\ & (a_k; e_j)^{\mathrm{T}} \begin{pmatrix} IX \\ X^T Y \end{pmatrix} (a_k; e_j) - \underline{r}_{kj}^2 \geq -\beta_{kj}^-, \forall k,j \in N_l, \\ & e_{ij}^{\mathrm{T}} Y e_{ij} - \bar{r}_{ij}^2 \leq \beta_{ij}^+, \forall i,j \in N_u, i < j, \\ & (a_k; e_j)^{\mathrm{T}} \begin{pmatrix} IX \\ X^T Y \end{pmatrix} (a_k; e_j) - \bar{r}_{kj}^2 \leq \beta_{kj}^+, \forall k,j \in N_u, \\ & \alpha_{ij}^+, \alpha_{ij}^-, \alpha_{kj}^+, \alpha_{kj}^-, \beta_{ij}^-, \beta_{kj}^-, \beta_{ij}^+, \beta_{kj}^+ \geq 0, \\ & Y = X^{\mathrm{T}} X.\end{aligned}$$

Unfortunately, the above problem is not a convex optimization problem. Biswas et al. in IPSN'04 propose to convert this problem to a semidefinite program, by relaxing $Y = X^{\mathrm{T}}X$ to $Y \succeq X^{\mathrm{T}}X$. The expression $Y \succeq X^{\mathrm{T}}X$ is equivalent to

$$Z := \begin{pmatrix} I & X \\ X^{\mathrm{T}} & Y \end{pmatrix} \succeq 0.$$

Then, the problem can be written as a standard SDP problem:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i,j\in N_{\mathrm{e}},i<j} (\alpha_{ij}^{+} + \alpha_{ij}^{-}) + \sum_{k,j\in N_{\mathrm{e}}} (\alpha_{kj}^{+} + \alpha_{kj}^{-}) \\
& + \textstyle\sum_{i,j\in N_{\mathrm{l}},i<j} \beta_{ij}^{-} + \sum_{k,j\in N_{\mathrm{l}}} \beta_{kj}^{-} \\
& + \textstyle\sum_{i,j\in N_{\mathrm{u}},i<j} \beta_{ij}^{+} + \sum_{k,j\in N_{\mathrm{u}}} \beta_{ij}^{+} \\
\text{s.t.} \quad & (1;0;0)^{\mathrm{T}} Z (1;0;0) = 1 \\
& (0;1;0)^{\mathrm{T}} Z (0;1;0) = 1 \\
& (1;1;0)^{\mathrm{T}} Z (1;1;0) = 2 \\
& (0;e_{ij})^{\mathrm{T}} Z (0;e_{ij}) - \alpha_{ij}^{+} + \alpha_{ij}^{-} = d_{ij}^{2}, \forall i,j \in N_{\mathrm{e}}, i<j, \\
& (a_k;e_j)^{\mathrm{T}} Z (a_k;e_j) - \alpha_{kj}^{+} + \alpha_{kj}^{-} = d_{kj}^{2}, \forall k,j \in N_{\mathrm{e}}, \\
& (0;e_{ij})^{\mathrm{T}} Z (0;e_{ij}) + \beta_{ij}^{-} \geq \underline{r}_{ij}^{2}, \forall i,j \in N_{\mathrm{l}}, i<j, \\
& (a_k;e_j)^{\mathrm{T}} Z (a_k;e_j) + \beta_{kj}^{-} \geq \underline{r}_{kj}^{2}, \forall k,j \in N_{\mathrm{l}}, \\
& (0;e_{ij})^{\mathrm{T}} Z (0;e_{ij}) - \beta_{ij}^{+} \leq \bar{r}_{ij}^{2}, \forall i,j \in N_{\mathrm{u}}, i<j, \\
& (a_k;e_j)^{\mathrm{T}} Z (a_k;e_j) - \beta_{kj}^{+} \leq \bar{r}_{kj}^{2}, \forall k,j \in N_{\mathrm{u}}, \\
& \alpha_{ij}^{+}, \alpha_{ij}^{-}, \alpha_{kj}^{+}, \alpha_{kj}^{-}, \beta_{ij}^{-}, \beta_{kj}^{-}, \beta_{ij}^{+}, \beta_{kj}^{+} \geq 0, \\
& Z \succeq 0.
\end{aligned}$$

Solving the linear or semidefinite program centrally, the time complexity is $O(k^2)$ for angle of arrival data, and $O(k^3)$ when radial (e.g., hop count) data are included, where k is the number of convex constraints needed to describe a network. Thus, the computation complexity of SDP is likely to preclude itself in practice.

4.3 Distributed Localization Approaches

4.3.1 Beacon-based Localization

4.3.1.1 Iterative Trilateration

Beacon-based localization approaches utilize the node-to-beacon distances. The distance between an unknown node and a beacon can be estimated using a basic

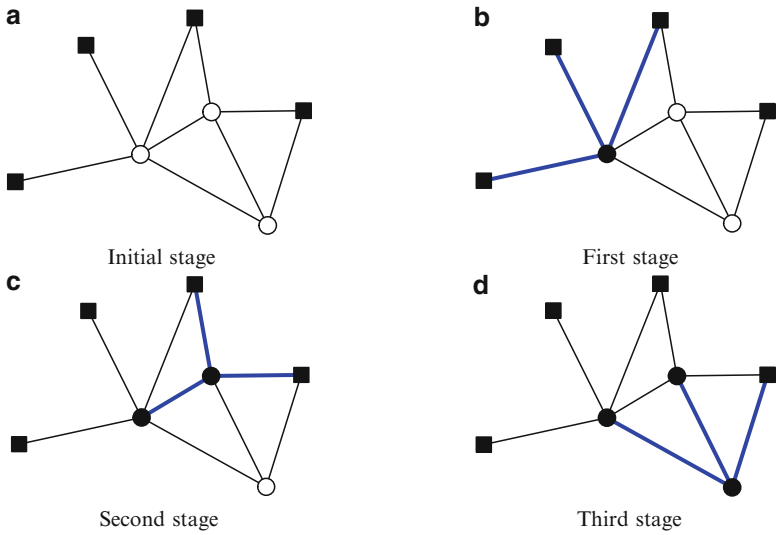

Fig. 4.1 Iterative localization. Bring a to d here (**a**) Initial stage; (**b**) first stage; (**c**) second stage; (**d**) third stage

distance-vector technique [58, 59]. Such a mechanism can be viewed as a top-down manner due to the progressive propagation of location information from beacons to entire networks.

One variant of this approach is the indirect usage of beacon nodes. Initially an unknown node, if possible, is located based on its neighbors by multilateration or other positioning techniques. After being aware of its location, it becomes a reference node to localize other unknown nodes in the subsequent process. This step continues iteratively, gradually changing unknown nodes to known ones. The process of iterative trilateration is illustrated in Fig. 4.1, in which squares are beacons, soft circles are unknown nodes, and solid circles are known nodes.

The advantage of this approach is that it only involves local information (information within neighborhood) when locating a node, leading to high efficiency in terms of communication. However, the use of localized unknown nodes as reference inherently introduces substantial cumulative errors, especially for the nodes far away from beacons. Some works [60, 61] characterize the error propagation in multihop localization approaches and make efforts to control error accumulation. The error control techniques will be discussed in detail in Chap. 6.

4.3.1.2 Finite Localization by Bilateration

Experimental studies show that trilateration-based algorithms require an average node degree beyond 10 for correctly localizing most of the nodes in a network [62]. When the average degree is below 8, the iterative trilateration will fail in most

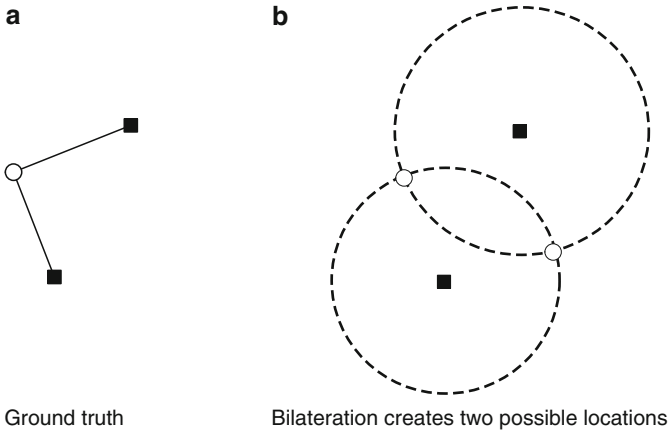

Fig. 4.2 Bilateration. Bring a and b here (**a**) Ground truth; (**b**) bilateration creates two possible locations

cases. To be more applicable for sparse networks, sweeps [63] partially relaxes the requirement of node dependence in iterative approaches. In contrast to the traditional "unique position computation," sweeps presents finite localization which locates a target node to a set of possible positions called candidate positions. Finite localization guarantees that the ground truth position of a node is one of its candidate positions. Further, sweeps adopts a new positioning scheme called bilateration, which computes the candidate positions of a node by utilizing the distance measurements of only two reference nodes. As shown in Fig. 4.2, bilateration produces two candidate positions (soft dots) for an unknown node and one of them (the left one) is the ground-truth position. Similar to multilateration, the finitely localized node, called swept node, can act as a reference node to localize other unknown nodes. The only difference is that all candidate positions of the swept node are enumerated for the location computation of the target node. Moreover, after each bilateration, sweeps checks the consistency among the candidate position sets and deletes the incompatible items. Under this mechanism, sweeps can locate a large proposition of theoretically localizable nodes in a network. However, the worst case computation grows exponentially in terms of the number of nodes.

The details of sweeps design are demonstrated through a typical network topology, as shown in Fig. 4.3 a, in which solid dots denote reference nodes and soft dotes denote unknown nodes. Clearly, traditional trilateration-based algorithms cannot localize any of the unknown nodes. In contrast, sweeps can locate v_4 that has the distance measurements to two swept nodes v_1 and v_3. This bilateration generates two candidate positions, as shown in Fig. 4.3 b1, b2. As node v_4 is finitely localized, it becomes a swept node. Then, node v_5 knows the distances to two swept nodes v_1 and v_4, so it can be finitely localized. Note that, all candidate positions of node v_4 are enumerated to compute the candidate positions of node v_5. Based on the two

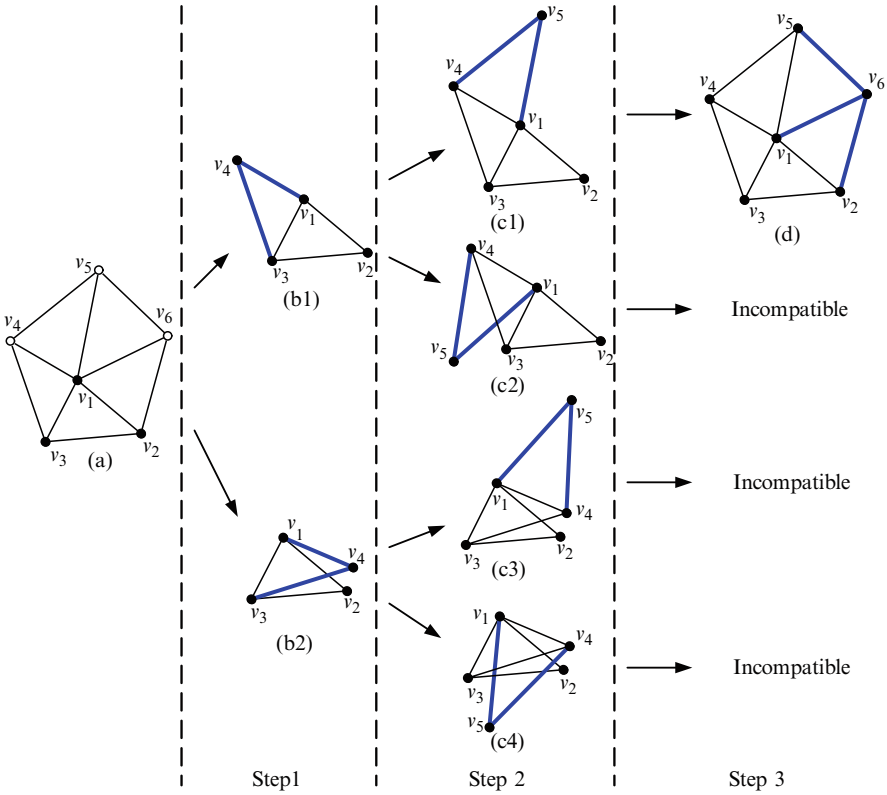

Fig. 4.3 Sweeps execution

candidate positions of node v_4, node v_5 has four candidate positions. The dependence relationship of the candidate positions is shown by arrows in Fig. 4.3. All the candidate positions of node v_5 are consistent with the distant measurements, as shown in Fig. 4.3 c1–c4. Finally, node v_6 has distance measurements to three swept nodes v_1, v_2, and v_5. Due to the consistency check of these distance measurements, only one of the candidate positions of node v_5 is compatible. Hence, all the incompatible candidate positions of node v_5 are pruned. Further, the related candidate positions (by the dependence tree) of node v_4 are also pruned. Eventually, all the nodes in this network are properly localized.

Sweeps can localize a kind of network called localizable bilateration network, formally defined as follows. A network has a bilateration ordering with anchors v_1, v_2, and v_3 if its nodes can be ordered as v_1, v_2, …, v_n so that v_1 and v_2 are adjacent, and each v_i, $i > 2$, is adjacent to at least two vertices v_j where $j < i$. A network is called a bilateration network if it has a bilateration ordering. Clearly, the wheel network illustrated in Fig. 4.3 a is a special case of the bilateration network.

From the example of the wheel network, it is concluded that the number of candidate positions can grow $O(2^n)$ in the worst case, where n is the number of nodes in the network. Hence, one of the drawbacks of sweeps is the computational complexity. Sweeps introduces two mechanisms to mitigate this problem. First, sweeps adopts an immediate consistency check after each bilateration to reduce the amount of the candidate positions. Second, sweeps reduces the growth of the candidate position set by choosing a particular sweep ordering called shell sweeps. Shell sweeps is a breadth-first sweep in which at each stage, the nodes having distance measurements to at least two already swept nodes are placed earlier in the ordering than all other nodes. Nevertheless, these mechanisms cannot reduce the worst case computational complexity, as the computational complexity is still $O(2^n)$ for a wheel network. Though it is nonpolynomial in the worst case, sweeps has acceptable average execution cost in a random deployed network.

Besides the computational complexity, sweeps also suffers noisy ranging measurements. When the measurements contain errors, the consistency check scheme may prune all the candidate positions. Further, the location error in each step may accumulate severely and decay the result in several steps. sweeps has an extended version to handle noisy ranging measurements, while the revised version requires a strong geometric model of a network, the unit disk graph (UDG) model, and generates results with no guaranteed accuracy. Hence, error control is still an open problem for sweeps.

4.3.2 Coordinate System Stitching

4.3.2.1 Local Map Stitching

Coordinate system stitching is an alternative for localization and has attracted a lot of research efforts recently [58, 62, 64]. It works in a bottom-up manner, in which localization is originated in a group of local nodes in relative coordinates. By gradually merging local maps, it finally achieves entire network localization in global coordinates, illustrated in Fig. 4.4.

Coordinate system stitching typically works as follows:

1. Split the network into small overlapping subregions. Very often each subregion is simply a single node and its one-hop neighbors.
2. For each subregion, compute a "local map," which is essentially an embedding of the nodes in the subregion into a relative coordinate system.
3. Finally, merge subregions using a coordinate system registration procedure. Coordinate system registration finds a rigid transformation that maps points in one coordinate system to a different coordinate system. Thus, step 3 places all the subregions into a single global coordinate system. Many algorithms do this step suboptimally, since there is a closed-form, fast and least-squares optimal method of registering coordinate system.

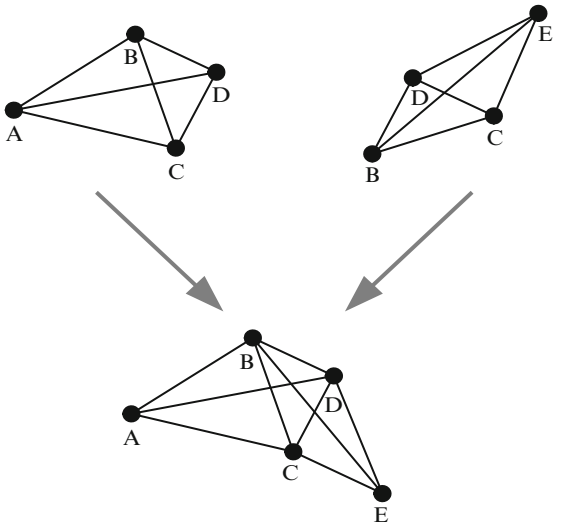

Fig. 4.4 Coordinate system stitching

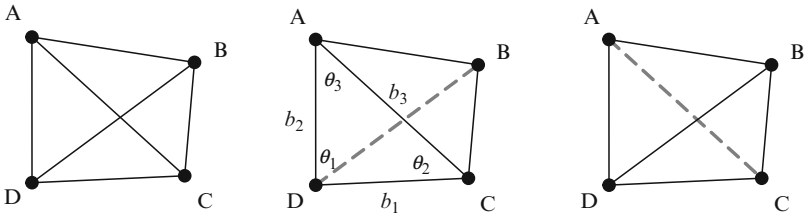

Fig. 4.5 Robust quadrilateral

Moore et al. [62] outline an approach that produces more robust local maps. Rather than using three arbitrary nodes, they use "robust quadrilateral" (robust quads) to define a local map. As shown in Fig. 4.5, a robust quad consists of four subtriangles ($\triangle$ABC, $\triangle$ADC, $\triangle$ABD, $\triangle$BCD) that satisfy

$$b \times \sin^2(\theta) > d_{\min},$$

where b is the length of the shortest side, θ is the smallest angle, and $d_{\min}$ is a predetermined parameter according to the level of measurement error. The idea is that the points of a robust quad can be placed correctly with respect to each other (i.e., without "flips"). Given zero mean Gaussian measurement error, Moore et al. demonstrate that the probability of a robust quadrilateral experiencing internal flips can be bounded by setting $d_{\min}$ appropriately. In effect, $d_{\min}$ filters out quads that have too much positional ambiguity. The appropriate level of filtering is based on the inaccuracy of distance measurements. Unfortunately, coordinate system stitching suffers from error propagation caused by local map stitching. Moore et al.

calculate the probability of their algorithm constructing correct local maps and present an error lower bound of the local map positions. Furthermore, their algorithm in many cases fails to locate orphan nodes, either because they could not be added to a local map or because their local map failed to overlap sufficiently with neighboring maps. Moore et al. claim that this is acceptable because the orphaned nodes are the nodes most likely to display high error. In addition, for many applications, missing localization information for a known set of nodes is preferential to incorrect information for an unknown set.

Coordinate system stitching techniques are quite compelling. They are inherently distributed, since subregion and local map formation can trivially occur in a network and stitching is easily organized in an ad hoc manner.

4.3.2.2 Component Stitching

A more general form of coordinate system stitching is the component-based localization [65]. A component is a group of nodes that form a rigid structure (e.g., each node has finite candidate positions in the local coordinate system). Using globally rigid components (e.g., each node has a unique position in the local coordinate system) as basic units, the algorithm merges and localizes components through utilizing intercomponent distance measurements and anchors.

As shown in Fig. 4.6. three intercomponent distance measurements constrain the relative geometric relationship between two components A and B, both of which are adjacent to two anchors. From the perspective of each single node, none of the nodes has enough neighboring anchors (no less than two) to be finitely localized immediately. Traditional local-map-based algorithms will fail to localize this network, because the local maps (i.e., components A and B) do not contain anchors to convert the coordinate system. From the point of view of components, however, component A and component B can be merged into a bigger component, which is localizable by referring to the four anchors. Finally, all nodes in the two components are localized.

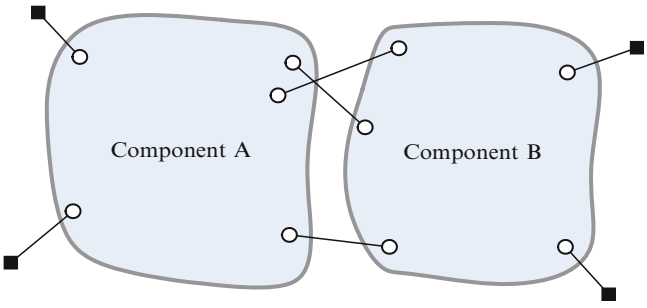

Fig. 4.6 Component-based localization

The concept of component and related terms are formally defined as follows. For a given network, a distance graph $G = (V, E)$ is constructed, where vertices denote nodes in the network and an edge (i,j) exists if nodes i,j can measure the mutual distance between them. Associated with each edge, a function $d(i,j)$: $E \rightarrow R$ is used to denote the distance value. Assume there are m anchors, labeled from 1 to m, and the left $n - m$ unknown nodes are labeled from $m+1$ to n, where n is the total number of nodes in the network. The ground truth position of node i is denoted as p_i. A realization of a network is a mapping from nodes to their 2D coordinates, P: $V \rightarrow R^2$, such that $P(i)=p_i$ for all $1 \leq i \leq m$ and $\|P(i) - P(j)\| = d(i,j)$ for all $(i,j) \in E$, where $\|P(i)-P(j)\|$ denotes the Euclidean distance of $P(i)$ and $P(j)$. Analogously, the concept of realization on the subgraph of G is defined, and the only difference is that the rotations, translations, and reflections of the mapping are treated as the same mapping when operating on a subgraph. Then, a node is localizable if and only if its image is unique for all realizations of G. A node is finitely localizable if and only if the cardinality of its image set is finite for all realizations of G. If a localization algorithm can generate a candidate position set that contains all the possible positions of a finitely localizable node, then the node is finitely localized by the algorithm. Given a distance graph G, a component is a set of nodes that has finite number of ways to be realized. A component is globally rigid if and only if there is a unique realization in a plane. Components are realized through both in-component anchors and the interconnected edges between the component and anchors. Hence, a component is realizable as long as it can determine its physical layout by the anchor information.

There are two versions of component-based localization algorithms: BCALL and CALL. As a basic version, BCALL operates on globally rigid components and unique realization of them, thus can terminate in polynomial time. BCALL has three major operations: component generation, component mergence, and component realization:

1. Component generation partitions the network into globally rigid components. Component generation follows similar procedures as generating local maps. The only difference is that a node can only join one component, such that components do not share any common node. BCALL initially selects a triangle as a component and generates the local coordinate system of the component according to the initial triangle. Other nodes then join the component and record their local coordinates through trilateration. By iterative trilateration, the newly generated component expands as large as possible. After component generation, each node either belongs to a component or becomes an isolated node.
2. After component generation, component mergence integrates nonrealizable components and isolated nodes into a larger component. As components do not share any common nodes, the mergence is performed through the interconnected edges between the two components. BCALL requires the resultant component to be globally rigid, thus there must be at least four independent interconnected edges connecting the two components. Here, independent means the edges guarantee the global rigidity of the result. For isolated nodes, it can be

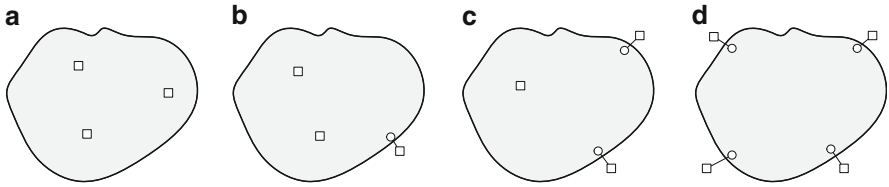

Fig. 4.7 Unique realization of components

merged to a component by trilateration. After merging, BCALL converts the local coordinate system of a component to that of the other one by solving overdetermined simultaneous equations. Component mergence is an iterative process. Some mergence can make other components or isolated nodes capable to merge into the resultant component. Component mergence process terminates when no such mergence can continue or the resultant component is realizable.

3. Component realization converts the local coordinate system of the realizable component to the physical positions. BCALL requires the realization to be unique, so the anchor information must uniquely determine the physical layout of the target component. As shown in Fig. 4.7, there are four ways to realize a component based on the number of in-component anchors:
   (a) The component contains at least three anchors
   (b) The component contains two anchors and a nonanchor node sharing an edge with a realized node
   (c) The component contains one anchor and two distinct nonanchor nodes sharing two edges with two distinct realized nodes
   (d) There are at least four independent edges connecting at least three distinct nodes in the component with at least three distinct realized nodes

CALL design is based on the concept of finite realization. CALL relaxes the requirements of the component mergence and realization from unique to finite states and adopts a consistency check step to prune the incompatible states. CALL follows similar steps as BCALL:

1. Follow the same procedure as BCALL to generate components.
2. Merge nonrealizable components and isolated nodes to generate larger components. CALL does not require the resultant component to be globally rigid. Instead, it only demands that nodes have finite candidate positions. Specifically, nodes are merged to components by bilateration. Components are merged, if the result has finite ways to be realized on a plane, thus there must be at least three interconnected edges connecting the two components.
3. Realize the components to a finite set of physical positions. Hence, the anchor information must finitely determine the physical layout of the target component. As shown in Fig. 4.8, there are three ways to finitely realize a component based on the number of in-component anchors:

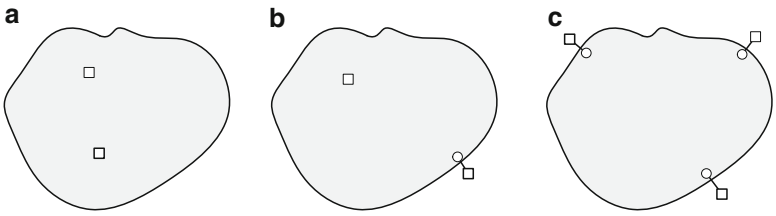

Fig. 4.8 Finite realization of components

(a) The component contains at least two anchors
(b) The component contains one anchor and a nonanchor node sharing an edge with a realized node
(c) There are at least three edges connecting the component with at least two distinct realized nodes, and there are at least two vertices associated with these edges in the component

The relaxations cause nodes to have several candidate positions. CALL inserts an extra substep in each of the above operations to check the consistency of the candidate positions. Each neighboring node pair checks the consistency of its candidate position sets by enumerating their items. Two items in each of the sets are defined as counterparts, if their distance is equal to the measured distance. A candidate position is incompatible, if it has no counterpart in the candidate position set of a neighboring node.

Using components can better share and integrate the anchor and ranging information, so the component-based localization algorithms are more applicable for sparse networks. Clearly, using fewer measurements leads the algorithm to be more sensitive for ranging errors.

4.4 Summary

We present a comparative study on existing range-based approaches with emphasis on beacon nodes, node density, accuracy, and cost.

4.4.1 Beacon Nodes

Beacon nodes (a.k.a. seeds or anchors) are necessary for localizing a network in the global coordinate system. Beacon nodes have no difference from ordinary network nodes except knowing their global locations as a priori. This knowledge can be hardcoded or acquired through some extra hardware like a GPS receiver.

Beacon configuration has significant impacts on localization. Existing works show that higher localization accuracy can be achieved if beacons are placed in a convex hull around the network. Placing additional beacons in the center of the network is also helpful. Thus, it is necessary for system designers to plan the beacon layout, as well as the amount of beacons, before deploying a network.

4.4.2 Node Density

Many localization algorithms are sensitive to node density. For instance, when the average degree is over 10, the network has a trilateration ordering with high probability, thus to suit the requirement of iterative trilateration. When designing or analyzing an algorithm, it is important to take the algorithm's implicit density assumptions into account, since high node density can sometimes be expensive and infeasible.

4.4.3 Accuracy

Given a localization algorithm, location accuracy shows how well the computed locations match with the physical positions of nodes. To be specific, location accuracy is defined as the expected Euclidean distance between the location estimate and the actual location of an unknown node, while location precision indicates the percentage of the results satisfying the predefined accuracy requirement.

Location accuracy trades off with location precision. If we relax the accuracy requirement, precision is increased and vice versa. Thus, we must put these two metrics in a common framework for comparison. We can fix location precision, say 95%, and evaluate the localization algorithms based on the corresponding accuracy performance.

Error propagation demonstrates how location accuracy varies with measurement error. Intuitively, localization error is linear with measurement error. However, it is not true for sequential localization algorithms, such as trilateration and bilateration. Nodes with large location errors would contaminate the location estimates of their neighbors. In this scenario, measurement error is no longer the only factor contributing to localization error.

4.4.4 Cost

In general, the cost of a localization system includes hardware cost and algorithm cost. Hardware cost refers to the ranging equipment. Different ranging equipments

Table 4.1 Comparative study of localization algorithms

Localization	Algorithm		Accuracy	Node Density	Beacon Amount	Cost		Error Propagation
						Comp.	Comm.	
Centralized	MDS		Median	Low	Low	High	High	Low
	SDP		High	High	Median	High	High	Low
Distributed	Beacon based	Trilateration	High	High	High	Low	Low	High[a]
		Bilateration	Low	Low	Median	High	Low	High
	Coordinate stitching	Local map	High	High	Low	Low	Median	High
		Component	Low	Low	Low	High	Median	High

[a] In case of iterative localization

provide different physical measurements, as discussed in Chap. 2. Basically, the more accurate measurements the equipments can provide, the more expensive they are. On the other hand, algorithm cost refers to the time and power consumption of computation and communication required by an algorithm. In general, distributed algorithms are more efficient than centralized ones, as they only produce local optimal solutions and exchange information locally.

After years of extensive study on this topic, many localization solutions are presented. Table 4.1 presents an overview of typical approaches in terms of accuracy, node density, beacon percentage, computation cost, communication cost, and error propagation. All approaches have their own merits and drawbacks, making them suitable for different applications. Hence, the design of a localization algorithm should sufficiently investigate application properties, as well as algorithm generality and flexibility. In present and foreseeable future study, obtaining a Pareto improvement is a major challenge. That is, increasing the performance of one of the metrics without degradation on others.