1 Overview

In the last lecture, we presented a distributed, linear time, anchor-free algorithm to localize the sensor nodes, followed by the NP-hardness proof of Unit Disk Graph embedding.

In this lecture, we demonstrate more on hardness results. Especially, we will introduce a localization algorithm with angle information. The basic idea of this algorithm is to use a practical anchor-free embedding scheme by solving a linear program. In the second part of this lecture, we focus on Location-based Routing in Sensor Networks such as Face Routing.

2 Location and Routing in Sensor Networks by Local Angle Information

2.1 What you can do & what you can not do by using angles?

We start from a simple example to have the intuition why we need angle information. Suppose we have had distance information, flipping will probably cause different embedding, especially for sparse graph like Figure 1. But if we have angle information, this probability will be eliminated.

![Figure 1: Flipping Example](image)

Lemma 1. If we know the angles between adjacent edges of a unit disk graph, we can determine all pairs of crossing edges in a valid embedding.
Proof. In particular, if two edges $AB$, $CD$ intersect with each other, there must be a node that is connected with all the other three nodes by the crossing lemma mentioned in the last lecture. Suppose $B$ is connected with the other three nodes. Then $AB$, $CD$ cross each other if and only if $AB$ is located inside the cone defined $\angle CBD < \pi$ and $A$, $B$ are on different sides of the line defined by $CD$.

First we can decide if $AB$ is located inside the cone defined by $\angle CBD < \pi$ easily by the angle information. Further, if $AB$ is located inside the cone defined by $\angle CBD$ and $A$, $B$ are on the same side of the line defined by $CD$, then $A$ is inside the triangle $BCD$. See Figure 2, then $A$ is connected to $B$, $C$, $D$ due to plane geometry. This situation can be identified since $BA$ must be outside the cone defined by $\angle CAD$.

Figure 2: How to determine crossing edges?

The above lemma implies that we can identify all crossing edges in a valid embedding with local angle information. However, it is not sufficient for us to decide a valid embedding. It turns out that the problem of finding a valid embedding for unit disk graph by using the connectivity and the local angle information is still hard. In fact, its even NP-hard to find a topologically equivalent embedding.

**Definition 2.** A topologically equivalent embedding $\varepsilon$ of a graph $G$ with angle information is an embedding of the vertices such that two edges cross in $\varepsilon$ if and only if they cross in a valid embedding. The angle between any two adjacent edges $uv$, $uw$ is as specified.

2.2 The hardness of UDG embedding with angles

Now we prove that UDG embedding with angles is NP-hard by a reduction from a 3SAT problem. A 3SAT problem consists of a set of Boolean variables and clauses such that each clause is composed of at most 3 literals, which are either negated (0) or unnegated (1).
The 3SAT problem is to find an assignment to the variables such that all the clauses are satisfied. A 3SAT instance $C$ can be formulated as a graph $G_C$ where vertices are the set of clauses and variables, and there is a path connecting a clause with a variable (or its negated version) if the variable appears in the clause. Figure 3 is an example of $G_C$. Such a graph can be drawn on a grid in polynomial time.

![Figure 3: Unit Disk Graph Representation for 3SAT problem](image)

Next, we focus on realizing the graph $G_C$ by a unit-disk graph with the angle constraint such that there is a topologically equivalent embedding if and only if the corresponding 3SAT problem is satisfied.

We first present a set of building blocks by using unit disk graphs.

**Spring.** A spring is a line segment with length between $l$ and $2l$. It can be realized by a set of $2l + 1$ nodes placed on a straight line such that there are only edges between adjacent pairs, as shown in Figure 4. In particular, each edge in a unit disk graph has length at most 1, so a chain of $2l + 1$ nodes have length at most $2l$. For 3 adjacent nodes $a$, $b$, $c$, since $a$ cannot communicate with $c$, their distance must be at least 1 away. Thus the chain is no shorter than $l$.

![Figure 4: The realization of a spring by unit disk graph](image)

**Amplifier.** An amplifier is a triangle with fixed inner angles. Thus the ratio between the
edge lengths of the triangle is fixed. For a number $l$ we can use an amplifier to get the number $l' = cl$ for any $c > 0$. An amplifier can be realized by a unit disk graph with pre-specified angles between adjacent edges.

Figure 5: The realization of an amplifier by unit disk graph

**0/1 block.** To represent negated or unnegated literals. Consider a unit disk graph, where the two "teeth" do not cross, there are only two types of valid embedding (Figure 6). In short, we construct a concave cycle with one top tooth and one bottom tooth. If we don’t allow the teeth to overlap, there are basically two ways to embed the concave cycle, either by putting the top tooth to the left of the bottom tooth, or the other way around. Figure 6 also points out the length ranges for blue and green segments in different cases.

Figure 6: Two valid embedding

Figure 7 shows a complete 0/1 block by unit disk graph. The concave cycle is bounded by $AEFGHDCKLIJB$, the top tooth is the part of the cycle $EFGH$, the bottom tooth is the part of the cycle $JILK$. Suppose the length of $AB = CD$ is $l$, we use amplifiers and propagators such that the length of $BC = DA = 11l/6$. There are two squares $EFGH$, $IJKL$ inside the rectangle $ABCD$. Both of them have side length $2l/3$. The two squares don’t have edges in between. Thus any embedding without incorrect crossings will have to embed the graph in two ways, either by putting the square $EFGH$ to the left of $IJKL$ or the other way around. In the first case, the length of the path $AE$ is no more than $l/2$, the
length of $HD$ is at least $2l/3$. In the second case, the length of the path $AE$ is at least $2l/3$, and the length of $HD$ is no more than $l/2$. The segments $AE$, $HD$, $BJ$, $KC$ are springs, thus their lengths can be stretched and shranked by a factor no more than 2.

0/1 blocks implement the literals(variables) in 3SAT problem. Then how about the clauses? A clause component puts constraints on the input variables. In particular, it put a total maximum length on the concatenation of springs whose lengths represent the assignments of input variables. Figure 8 is an example for a clause. The widths in it depend on the widths in other blocks related to it such as springs, 0/1 blocks and etc..

Figure 7: The only two embedding of a concave cycle without incorrect crossings for 0/1 block

3SAT clause: $(\overline{x_1} \lor x_2 \lor \overline{x_3})$

After knowing how to realize clauses, variables, we can put all these components together

After knowing how to realize clauses, variables, we can put all these components together
and get a realization of the graph $G_C$ for a 3SAT instance C. It means we provide a
reduction from a 3SAT problem to the problem of UDG embedding with angles between
adjacent edges. Since 3SAT problem is NP-hard, the problem of UDG embedding with
angles between adjacent edges is also NP-hard.

In summary, we can have the following conclusions:

1. It’s NP-hard to find a topologically equivalent embedding of a unit-disk graph with
   local angle constraints.
2. It’s NP-hard to find a valid embedding of a unit disk graph with local angle constraints.
3. It’s NP-hard to find an $\alpha$-approximate embedding of a unit-disk graph with local
   angle constraints, for $\alpha < \sqrt{2}$.

2.3 A localization algorithm with angle information

Embedding a unit-disk graph with local angles is NP-hard, and it is so even when the re-
striction is relaxed to be finding a topologically equivalent embedding. In practice, however,
we still hope to use the local angle information to find localization that well approximates
the true sensor network. In this section, we show that we can construct an embedding
method based on linear programming, which produces very good localization solutions; the
solutions lead to nearly optimal routing performance as well; we also demonstrate the ro-
 bustness of the embedding method to noisy measurements of angles and to more general
topological models of sensor networks. This shows that using local angle information to do
localization and routing is practically good for sensor networks.

We formulate the linear programming problem as follows: We include as many constraints
as possible such that the optimization remains a LP. We take the length of each edge $e$,
$l(e)$, as a variable. We fix a node as the origin, then arbitrarily pick an edge and make the
$x$-axis be parallel to it. By the fact that we know the angle between any two adjacent edges,
the absolute angle of every edge $e$ the counterclockwise angle between the positive $x$-axis
and $e$ can be uniquely determined. Thus, each node’s position $(x, y)$ is a linear function of
variables (Figure 9). Then a valid UDG embedding satisfies the following constraints.

1. Edge length constraint:

   \[ 0 \leq l(e) \leq 1 \]
2. Cycle constraint: for a cycle with edges \( \{c_1, c_2, \cdots, c_p\} \)

\[
\sum_{i=1}^{p} l(e_i) \cos \Theta_{e_i} = 0, \sum_{i=1}^{p} l(e_i) \sin \Theta_{e_i} = 0
\]

3. \( \exists \) edges \( AB, BC \), and no \( AC \)

\[
l(AB) + l(BC) > 1
\]

4. Crossing edge constraint:

\[
l(CD) \geq \left| xD \right| = l(AD) \frac{\sin \angle DAB}{\sin(\angle ADC + \angle DAB)}
\]

The above constraints serve as the linear constraints in our linear programming. A feasible solution to the LP gives us an embedding of the UDG. There are many ways to select the objective function; as a heuristic, we choose it to be maximizing the minimum length of all edges.

Although the LP doesn’t necessarily produces a valid embedding, it works well in practice. Figure 10 is several examples to use LP to produce embedding. We can clearly see the results are very good.

The simulation we have shown so far assumes that the angles are measured accurately. In practice measurement errors are inevitable. The modelling of sensor networks as UDG can be inaccurate, too, because network links can be lost due to noise, signal interference or obstacles, and the transmission ranges of directional antennas are not circles. A more realistic model for sensor networks is called quasi-unit disk graphs, where a pair of nodes have an edge for sure if their distance is no more than \( \alpha \leq 1 \), don’t have an edge if their...
Figure 10: The embedding result by LP
distance is more than 1 apart, and may or may not have an edge if their distance is between \( \alpha \) and 1. The experiment results show that the embedding algorithm by LP is robust to measurement errors and network models. Figure 11 is such an example.

Figure 11: The embedding result by LP with noise

The embedding solution based on linear programming indeed gives very good results, but there are many further work and open problems on using local information. For instance, the linear program here is a centralized algorithm, but in practice distributed localization methods are more desirable. We list some challenges from the points of algorithm and system respectively in the following.

Algorithm challenges:

1. Noisy measurements
2. Insufficient connectivity, continuous deformation
3. Hardness of embedding suppose we are given different extra information.

System challenges:

1. Physical layer imposes measurement challenges such as Multipath, shadowing, sensor imperfections, changes in propagation properties and more
2. Extensive computation aspects. For examples, Many formulations of localization problems, how do you solve the optimization problem? and how do you solve the problem in a distributed manner, under computation and storage constraints?
3. Networking and coordination issues. For instance, Nodes have to collaborate and communicate to solve the problem? If you are using it for routing, it means you don’t have routing support to solve the problem! How do you do it?

4. System Integration issues, including How do you build a whole system for localization? How do you integrate location services with other applications? Different implementation for each setup, sensor, integration issue.

### 3 Location-based Routing in Sensor Networks

In the second part of this lecture, we focus on Location-based Routing in Sensor Networks such as Greedy Routing and Face Routing.

An sensor network consists of mobile nodes equipped with sensor. If the source and the destination of a message are not within mutual communication range, the message can be relayed by intermediate nodes, a process known as routing. Routing protocols in communication networks obtain route information between pairs of nodes wishing to communicate. Normally, there are two categories of routing protocols. One is proactive protocol: the protocol maintains routing tables at each node that is updated as changes in the network topology are detected. The other is reactive protocol: routes are constructed on demand. No global routing table is maintained. Due to the high rate of topology changes, reactive protocols are more appropriate for ad hoc networks, for example, ad-hoc on demand distance vector routing (AODV) and Dynamic source routing (DSR). However, both of them depend on flooding for route discovery, which means they use some type of controlled packet duplication mechanism to ensure that all routers in the network know the current map (or the change) of the entire network’s topology.

Among many routing protocols have been proposed for mobile networks, geographical routing, being a simple and scalable routing scheme, has attracted a lot of interests in recent years. Compared with "Data-centric" routing, which means routing is frequently based on a nodes’ attributes and sensed data, rather than on pre-assigned network address, Geographical routing uses a node’s location to discover path to that route. Using geographical routing, we need to have several assumptions as follows first.

1. Nodes know their geographical location

2. Nodes know their 1-hop neighbors
3. Routing destinations are specified geographically (a location, or a geographical region)
4. Each packet can hold a small amount ($O(1)$) of routing information.
5. The connectivity graph is modeled as a unit disk graph.
6. The information that the source node has includes the location of the destination node, the location of itself and its 1-hop neighbors.

Many earlier geographical routing protocols adopt a so-called geographical forwarding scheme, i.e., in the routing process, the current node send the packet to the 1-hop neighbor that makes most progress towards the destination. This scheme is actually of purely greedy nature. And there are many ways to measure ‘progress’. For example, at each intermediate network node the message to be routed is forwarded to the neighbor closest to the destination in Euclidean distance, or to the neighbor with smallest angle towards the destination which is also called ‘compass routing’ (Figure 12). The benefit of this scheme is that no flooding is involved, and each node makes the decision on which node to forward the packet to based solely on the location of itself, the neighboring nodes, and the destination. Thus, the routing algorithms based on geographical forwarding scheme are local algorithms, which are lightweight, robust and distributed in nature.

![Figure 12: Greedy Progress](image)

However, geographical forwarding scheme has a serious limitation, it can not guarantee delivery. Because of local minimum phenomenon, a packet maybe get stuck at a node that does not have a closer neighbor to destination (Figure 13), even though the source and
destination are connected in the network. Even if compass routing, it may get in loops (Figure 14).

Figure 13: Send packets to the neighbor closest to the destination

Figure 14: Send packets to the neighbor with smallest angle towards the destination

One technique to deal with this problem, proposed by Bose et al. and independently by Karp and Kung, is to maintain a planar subgraph of the underlying connectivity. When a packet is stuck at a node, the protocol will route the packet around a face of the graph to get out of the local minimum (Face Routing). Karp and Kung also proposed a routing protocol, the Greedy Perimeter Stateless Routing (GPSR) protocol that guarantees the delivery of the packet if a path exists.

3.1 Face Routing

The prerequisite of Face Routing is to find a connected planar subgraph of Unit Disk Graph, which is a straight line graph with no crossing edges. The planar subgraph subdivides the plane into connected regions called faces. Given a vertex \( v \) on a face \( f \), the boundary of \( f \) can be traversed in the clockwise (keep left hand on the wall) direction until hit the straight...
line connecting source to destination, then switch to the next face. In this way, the routing algorithm gets out of the local minimum and guarantees the delivery. Figure 15 show the routing process using face routing algorithm.

We conclude the properties of Face Routing.

1. All necessary information is stored in the message such as source and destination positions, the node when it enters the perimeter mode, and the first edge on the current face.

2. Completely local. It means knowledge about direct neighbors’ positions is sufficient. And faces are implicit. Only local neighbor ordering around each node is needed.

We say Face routing guarantee delivery of a message if there is a path. However, what if the destination is disconnected? The answer is the perimeter routing in a face will get back
to where it enters the perimeter mode, then the algorithm think routing failure, i.e., no way
to the destination.

### 3.2 Planar Graph Subtraction

In Face Routing algorithm, we know routing algorithm is based on a planar subgraph of
unit disk graph, which preserves the connectivity of the graph and permits distributed
computation.

In order to understand more about planar graph, we introduce some definition related to
Delaunay triangulation first. Delaunay triangulation was first proposed by B. Delaunay in
1934. It has numerous applications since then. Voronoi diagram is the dual of delaunay
triangulation. It partitions the plane into cells such that all the points inside a cell have
the same closest point. If we connect the edge which their Voronoi cells are adjacent, we
will get the delaunay triangulation. Figure 16 gives the examples of Voronoi diagram and
its corresponding delaunay triangulation.

Delaunay triangulation has the following properties:

1. empty-circle property: he circumcircle of a Delaunay triangle is empty of other points.

2. The converse is also true: if all the triangles in a triangulation are locally Delaunay,
then the triangulation is a Delaunay triangulation.

3. Greedy routing on Delaunay triangulation never gets stuck. We can use Delaunay
triangulation as the planar subgraph in face routing. It’s easy to prove this claim.
4. For an arbitrary point set, the Delaunay triangulation may contain long edges. If the nodes are uniformly placed inside a unit disk, the longest Delaunay edge is $O((\log n/n)^{1/3})$. (proved by Kozma et.al. in PODC’04)

5. centralized construction.

Because of the last property of Delaunay triangulation, we hope we can construct the planar subgraphs in a distributed way which is more natural for sensor network. Thus, we will introduce relative neighborhood graph and the Gabriel graph in the next. Both of them can be constructed in a distributed way.

**Definition 3.** Relative Neighborhood Graph (RNG) contains an edge $uv$ if the lune is empty of other points.

**Definition 4.** Gabriel Graph (GG) contains an edge $uv$ if the disk with $uv$ as diameter is empty of other points.

![Figure 17: RNG (left) and GG (right)](image)

**Claim 5.** $\text{MST} \subseteq \text{RNG} \subseteq \text{GG} \subseteq \text{Delaunay}$

**Proof.** 1. RNG $\subseteq$ GG: if the lune is empty, then the disk with $uv$ as diameter is also empty.

2. GG $\subseteq$ Delaunay: the disk with $uv$ as diameter is empty, then $uv$ is a Delaunay edge.

3. MST $\subseteq$ RNG: Assume $uv$ in MST is not in RNG, there is a point $w$ inside the lune (See Figure 18). $|uw| > |uv|$, $|uw| > |vw|$. Now we delete $uv$ and partition the MST into two subtrees. Say $w$ is in the same component with $u$, then we can replace $uv$ by $wv$ and get a lighter tree. $\rightarrow$ contradiction.

From the above claim, we have that RNG and GG are planar (Delaunay is planar) and keep the connectivity (MST has the same connectivity of UDG). We show an example for UDG, GG and RNG in Figure 19.
Definition 6. A subgraph $G'$ of $G$ is an $\alpha$-spanner if the shortest path in $G'$ is bounded by a constant $\alpha$ times the shortest path length in $G$.

Both RNG and GG are not spanners because a short path may not exist! Then, there is a problem here: even if the planar subgraph contains a short path, can greedy routing and face routing find a short one?